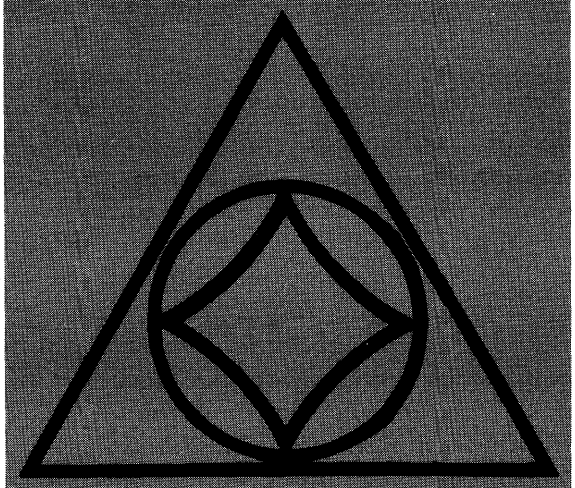


THE THOMPSON BOOK COMPANY
National Press Building
Washington, D.C. 20004

AFIPS

CONFERENCE PROCEEDINGS

VOLUME 33
PART TWO

1968

FALL JOINT COMPUTER CONFERENCE

December 9-11, 1968
San Francisco, California

The ideas and opinions expressed herein are solely those of the authors and are not necessarily representative of or endorsed by the 1968 Fall Joint Computer Conference Committee or the American Federation of Information Processing Societies.

Library of Congress Catalog Card Number 55-44701
THOMPSON BOOK COMPANY
National Press Building
Washington, D.C. 20004

Printed in the United States of America

CONTENTS

PART II

WRITEACOURSE:
An educational programming language*

by EARL HUNT and MARY ZOSEL

University of Washington
Seattle, Washington

The problem

Computer applications in education are becoming more and more prevalent. Perhaps the most talked about use of computers in the schools is to control the educational material presented to students ... the Computer Assisted Instruction (CAI) application. CAI requires that two problems be solved. Someone has to decide what material should be sent to a student, and when, and someone has to arrange that the computer actually do what is desired. The first problem, what should be done, is a topic for educators and psychologists. Our concern is with the second. How can we make CAI a convenient tool for the educator?

We will assume that the educator has access to an interactive system, but that system was *not* specifically designed for computer assisted instruction. Once the educator has determined the form of a lesson, he would like to be able to go to the typewriter, type in the instructions, and then leave the typewriter knowing when he returns with a student, the computer will be prepared to conduct the lesson. The problem is that the computer "understands" instructions only in a very restricted set of languages. The form of these languages has, for the most part, been dictated either by the internal design of the machine or by the requirements of mathematicians and statisticians who are, after all, the largest group of users of general purpose computers.

The language problem can be solved in several ways. The educator could, himself, become proficient in computer programming. This diverts his time from the problem he wishes to pursue. He could acquire a specially designed computing system which had languages and equipment suitable for his use. This alternative is extremely expensive (the equipment alone would rent for $100,000 a year or better) and is feasible only for large research projects. He could hire a computer programmer and tell him what the computer was supposed to do. This introduces another specialist into the research team, and has the disadvantage that the computer will then act as the programmer thought the educator wanted it to act. The educator may not discover a misunderstanding until after it has been built into the programming system, at which point it is hard to fix.

We advocate another alternative, placing in the general purpose computing system a language which is easy for the educator to use. This is the solution which was taken over ten years ago by mathematicians, when they were faced with the prospect of writing mathematics in a language which was designed for machine execution, rather than for problem statement. The great success of languages such as FORTRAN and ALGOL testifies to the feasibility of the approach. In the next ten years an educator's language may also be needed.

What should the characteristics of such a language be? By far the most important requirement is that the language should be natural for the teacher. Its syntax and semantics should conform to his writing habits. Insofar as possible, and there are limits on this, the form of the language should not be determined by the physical characteristics of the computer on which it will be used.

Readability is a second requirement, It will often be necessary for a person to understand a program he did not write. The structure of the programming language should be such that the basic plan of a program can be communicated without forcing the reader to master the intricacies of each line of code.

A judicious choice of a language can also ensure the

*This research was supported by the Air Force Office of Scientific Research, Office of Aerospace Research, United States Air Force, under AFOSR Grant No. AF-AFOSR-1311-67. Distribution of this document is unlimited.

availability of a computer. Any language which is not tied to the physical characteristics of a computer requires a translator. Pragmatically speaking, then, the language is defined by the translator program. Thus the educational language can be "inherited" by any machine for which its base language translator exists.

We are by no means the first to recognize the need for an educator's language. Several others have already been developed. The best known are probably IBM's COURSEWRITER[5] and System Development Corporation's PLANIT.[2] These languages are admirably suited for the particular computer configurations for which they were developed. For a variety of reasons, however, we believe that they fail to meet the criteria we have listed. Our principal criticism is that they either are too much influenced by the way a computer wishes to receive commands, instead of the way a person wishes to give them, or that they contain features which, although quite useful in themselves, would not be available except in specially designed computing systems.

The WRITEACOURSE Language.

We have developed an educational language, called WRITEACOURSE, which is consciously modeled after the ALGOL arithmetic programming language,[7] which it resembles in its syntactic structure. The basic unit of discourse is the *statement*, corresponding roughly to an English sentence. Statements are grouped into larger units called *lessons*, and lessons into *courses*, similar to the way a group of subroutines make up a program. Statements are composed of *instructions*. In WRITEACOURSE there are only ten instructions. Physically, they are English words, such as ADD and PRINT, which have been chosen to have a meaning as close as possible to their meaning in the natural language.

Limiting the commands of the language restricts us. There are actions which can be executed by a computer, but which are difficult to express in a restricted idiom. The initial users of WRITEACOURSE have not found this to be a great problem. They appear to be able to say almost everything they want to say without extensive training.

The WRITEACOURSE translation program has been written entirely in the PL/I programming language,[6] which we expect to be widely available in a few years. We assume that the particular configuration has an interactive computing capability, in which the user can exchange messages with a program from a remote station equipped with a typewriter or other keyboard device. By 1970 this sort of capability should be common in universities, at a price well within the reach of a modest research budget.

An earlier version of WRITEACOURSE[4] was defined for the Burroughs B5500 computer only, using the extended ALGOL provided for that machine.[1] Thus the early version was not machine independent in the sense that our present program is, although it would be a fairly straightforward task to adopt it to some other computer which had an ALGOL compiler.

Our approach should produce an easily maintained system. This is a very important point. Undoubtedly there will be errors in any system as complicated as a programming language. Also, different users will want to extend the language to suit their own purpose. Since the translation program is written in a commonly available, user oriented language, the educator will find that there are many people who can understand and alter it. This will be particularly true in universities, where Computer Science departments and computer centers will regularly offer undergraduate courses in PL/I programming.

A user's view of the language.

The purpose of developing WRITEACOURSE was to have a language which could be easily understood by educators. We can test this now by presenting a fragment of a WRITEACOURSE lesson. Hopefully it will be readable with only a minimal explanation.

The following statements are taken from a fragment of a WRITEACOURSE lesson. They appear exactly as they would be typed by an instructor, with the exception of the numbers in parentheses at the beginning of each line. These have been introduced for ease of reference in explaining the lesson.

```
(1) 20 PRINT "THE ANGLE OF INCIDENCE IS
       EQUAL TO THE ANGLE OF ..."
(2)    ACCEPT CHECK "REFLECTION" "RE-
       FRACTION" IF 1 CHECKS THEN GO TO 5|
(3)    IF 2 CHECKS THEN PRINT "NO, THE
       ANGLE OF REFRACTION DEPENDS ON
(4)    THE TYPE OF LENS"|
(5)    PRINT "TRY AGAIN" ACCEPT CHECK
       REFLECTION" IF 0 CHECKS THEN
(6)    PRINT "THE CORRECT ANSWER IS RE-
       FLECTION"|
(7) 5 PRINT "HERE IS THE NEXT QUESTION"|
```

What would happen when a student executed this lesson? The first statement (statement 20) to be executed is the statement beginning on line (1) and extending to the end of statement marker("|") on line (2). Statements always begin on a new line; otherwise, they may be typed in any way convenient. Line (1) would print the question THE ANGLE OF INCIDENCE IS EQUAL TO THE ANGLE OF ... on the computer-controlled typewriter. At line (2) the ACCEPT instruction would print an underscore ("_") on the next line. This would be a signal to the student indicating

that an answer was expected. At this point the paper in front of the student would look like this

```
THE ANGLE OF INCIDENCE IS EQUAL TO
  THE ANGLE OF ...
```

The computer would then wait for the student, who would type whatever he thought was an appropriate reply, then strike the carriage return key of the typewriter, indicating that he was through with his answer. The program would ACCEPT this answer, and CHECK it against indicated possible answers. Suppose the student had typed

```
REFRACTION
```

The CHECK command on line (2) would match this answer against the quoted statements "REFLECTION" and "REFRACTION." The quoted statements are called *check strings*. In this case the answer would be identical to the second check string, so we say that "2 CHECKS." At line (2), however, the question asked is, "DOES 1 CHECK?" This would only be true if the student had replied REFLECTION (the correct answer), in which case control would have been transferred to the statement named 5, at line (7) of the lesson, which continues with a new question.

However, I did not check, so the next commands to be executed are those on line (3), which begins a new, unnamed statement.[5] Lines (3) and (4) are straightforward. The computer asks if 2 CHECKS, which it does, since the student's reply was identical to the second check string.[6] Upon determining this, the computer types out the correcting response given on lines (3) and (4). Next the statement beginning on line (5) is executed. This prints another line, urging the student to try again, and an underscore (the ACCEPT of line (5)) telling him an answer is expected. The student will now have in front of him

```
THE ANGLE OF INCIDENCE IS EQUAL TO
THE ANGLE OF ... REFRACTION
NO, THE ANGLE OF REFRACTION DEPENDS
  ON THE TYPE OF LENS
TRY AGAIN
```

Assume that he replies correctly, printing REFLECTION. This will be read by the ACCEPT statement in line (5) and the immediately following CHECK statement will determine that 1 CHECKS is true. IF 0 CHECKS tests to see if nothing checked, i.e., 0 CHECKS is true if the student's answer does not match any of the check strings. In this case, the condition 0 CHECKS would be true for any answer other than REFLECTION. Looking at the final three lines of the conversation, we have

```
TRY AGAIN
REFLECTION
HERE IS THE NEXT QUESTION
```

But suppose that the student had not been so bright. The final lines could have read

```
TRY AGAIN
WHO KNOWS?
THE CORRECT ANSWER IS REFLECTION
HERE IS THE NEXT QUESTION
```

More sophisticated programming

The example just given was very simple. Using the computer's capabilities more fully, WRITEACOURSE makes possible the specification of a much more complex branching sequence. There is also a limited arithmetical capability. A set of counters (temporary variables) are provided to keep track of intermediate results. Counters can be used either to do arithmetic or to record the number of times a student takes a particular path through a course. This turns out to be a powerful device. We will give a few examples.[7]

Counters are named by preceding a number with the symbol "@." Thus @10 means "counter 10." Three commands are defined for counters, SET (counter number) TO (value), ADD (value) TO (counter number, and SUBTRACT (value) FROM (counter number). They have the obvious meaning.

```
SET @10 to 0
```

establishes 0 as the value of counter 10, while

```
ADD 5 TO @10
```

sets the value of counter 10 to 5 plus its original value. It takes little imagination to see that the counters can be used to keep scores on a student's responses, through the device exemplified by

```
IF 1 CHECKS THEN ADD 1 TO @7.
```

The value of a counter may also be printed. To do this the name of the counter is included in a PRINT command. When the command is executed, its current value will be printed. The statement

```
SET @8 TO 5 PRINT "THE VALUE OF 8 IS@8"
```

will print

```
THE VALUE OF 8 IS 5.
```

The content of a counter is a value, so arithmetic can be done on counters. ADD @2 TO @3 would set the value of counter 3 to the original value of counter 2 plus the value of counter 3.

There are actually three groups of counters. Counters

50–99 are *lesson counters*, their values are carried over from one use of a WRITEACOURSE lesson to another. There are several reasons for doing this. For instance, a counter can be used to keep track of the number of students executing a lesson, or the number of students who miss a particular question. Counters 1 to 49 are the *temporary counters*. They are set to zero when a student first signs in for a session with the computer. They are retained for that student, however, for the duration of the session even if he switches WRITEACOURSE lessons. Finally, Counter O is a special counter set by the computer's internal clock. It can be used to time a student's responses.

A set of Boolean IF statements are provided to check the value of a counter against another counter, or some constant value. The command IF @4 = 7 THEN GO TO 6 will cause a transfer to statement 6 only if counter 4 contains 7. The normal arithmetical relations of equality and ordered inequality are permitted.

Counter numbers may also be used for a computed GO TO. GO TO @2 is an instruction to go to the statement whose number is contained in counter 2. Of course, the instructor who writes this command must insure that counter 2 will contain the name of a statement whenever this command is executed.

Let us look at an example which uses some of these more complex commands.

```
(1)      SET @54, @41 TO 0 PRINT "WHAT DIS-
           COVERY
(2)      LEAD TO LASERS?" |
(3)   3  ACCEPT CHECK "MASER" "QUASER"
(4)      "CANDLES" IF 1 CHECKS THEN GO TO 6|
(5)      ADD 1 TO @41 IF 0 CHECKS THEN GO TO
           40 |
(6)      IF 2 CHECKS THEN PRINT "THAT IS IN
           ASTRONOMY."
(7)      GO TO 40 |
(8)      IF 3 CHECKS THEN PRINT "DO NOT BE
           SILLY." |
(9)  40  IF @41 < 3 THEN PRINT "TRY AGAIN"
           GO TO 3 |
(10)     ADD 1 TO @54 PRINT "THE ANSWER IS
           MASER." |
(11)  6  PRINT "HERE IS THE NEXT QUESTION"|
```

the following exchange might take place between the student and computer.

```
WHAT DISCOVERY LEAD TO LASERS?
QUASER
THAT IS IN ASTRONOMY.
TRY AGAIN
MASER
HERE IS THE NEXT QUESTION.
```

The first statement sets counters 54 and 41 to zero, then prints the basic question. Statement number 3 through statement number 40 establish a loop, which checks the student's answer for the correct answer or two anticipated wrong answers, prints an appropriate message for a wrong answer, then gives the student another chance. If the correct answer is detected (if 1 CHECKS in line (4)), the loop is broken by a transfer to statement 6. If a wrong answer is detected, the question is reasked. Counter 41 is used to keep track of the number of wrong answers. If three wrong answers are given, the correct answer is printed, and the program continues on. If this alternative occurs, however, the value of counter 54 is incremented by 1. Recall that counter 54 is one of the lesson counters, i.e., its value carries over from one user of the lesson to another. At some later time, then, an instructor could interrogate the lesson to see how many students had failed to answer this question in three or fewer tries.

Lessons and courses

Statements are grouped into *lessons*, and lessons into *courses*. Roughly, a lesson can be thought of as the number of WRITEACOURSE statements needed to carry on the computer's part of a computer-student interaction lasting about half an hour. Another important functional distinction is that a lesson is the WRITEACOURSE unit to which counters are attached. Thus if @54 appears in two different statements in the same lesson, it refers to the same counter. If the two statements are in different lessons, they refer to different counters. Note that this is not true for temporary counters, since they remain attached to a student for the duration a student-computer conversation. Thus if it is anticipated that a student will use more than one lesson during a single session, the results accumulated while the first lesson is active may be communicated to the second lesson via the temporary counters.

Lessons themselves are grouped into courses. Functionally, the chief distinction of a course is that it is possible to activate one lesson from within another, providing that the two lessons are in the same course. Suppose a student signs in, with the intention of taking a course in Romance Literature. He would begin by indicating that he wanted to work on the first lesson of this course. He would do this by replying, in response to a computer question, that he wished to work on LESSON1/LIT 47. LIT47 is assumed to be a course name, and LESSON1 a lesson of the course. Let us suppose that this lesson is going to discuss the novel *Don Quijote*. The instructor might want to check to make sure the student knew enough Spanish to understand some of phrases. This can be accomplished by the following statement.

```
(1)    1 PRINT "DO YOU WISH TO REVIEW
         SPANISH?"
(2)    ACCEPT CHECK "YES" IF 1 CHECKS
         THEN CALL SPREVUE/LIT47 |
```

If the last command on the second line is activated, it will suspend the current lesson now active (LESSON1/LIT47), and load the lesson SPREVUE/LIT47. Both lessons must be in the same course. Upon completion of SPREVUE/LIT47, control would be returned to the statement in LESSON1/LIT47 immediately after line (2).

The command LINK (lesson name) / (course name) will also change a student from one lesson to another within the same course. In this case, however, there is no automatic return to the calling lesson after the called lesson is completed. The normal use of LINK is to string together several lessons which the instructor wishes to have executed in sequence.

Using WRITEACOURSE

The steps in using WRITEACOURSE will now be described. The steps a student must go through to initiate a lesson have been kept to a minimum. He types XEQ and then supplies the lesson name and course name when requested. After a lesson is over, he may type XEQ and go through another lesson, or type STOP to terminate the session.

When an instructor constructs a lesson, the process is necessarily more involved. After calling the system the instructor sends the message /// COMPILE indicating a course is to be established or modified. (In general, the symbols "///" precede compiler commands.) If a new course is to be written, the order is sent.

```
///PROGRAM NEW lesson/course
```

The translator will then be ready to accept the lesson. Each statement is checked for syntax errors as it is received. If there is no error, the next statement is requested. Whenever an error is detected, a message is printed indicating where it occurred. After determining the corrected form, the instructor re-enters the statement, from the point of the error to the end. When the instructor wishes to stop working on the lesson, he types ///END. The lesson will be automatically stored in the computing system's files. If the instructor desires, he may order a check for undefined statement numbers referenced by GO TO instructions before the lesson is recorded.

The instructor may modify existing lessons or obtain a listing of lessons, using the commands ///ADD, ///DELETE and ///LIST.

System implementation.

WRITEACOURSE has been tested on an IBM 360/50 with a remote 2741 terminal. The translator was written in the RUSH4 subset of PL/I, provided by Allen-Babcock Computing (8). The only non-standard PL/I used is the timer function. WRITEACOURSE lessons are incrementally compiled into a decimal integer code, which is stored in a data file. The storage file for each course consists of 64 tracks of fixed format data with a block size of 252 bytes. The internal code is edited whenever a teacher makes a modification. The execution program interprets this code to produce the sequence of events planned by the instructor. The first block of code in a course contains the names and locations of the lessons in it. Each lesson occupies 38 blocks of the file, and is divided into five parts.

1. The instruction table, which contains the compiled decimal code with approximately one code word for each instruction in the lesson.
2. The statement number table, which contains the statement numbers with a pointer to the corresponding instructions.
3. The counters attached to the lesson.
4. The print tables, which contain all of the strings to be printed.
5. The print table index, which contains a pointer to the location of each string.

Since the source code is not saved, the compiled code must be used whenever the lesson is changed. To obtain a listing of the lesson, the code is interpreted, as if it were to be executed, and the source code is reconstructed. When a section of a lesson is deleted, the instruction table and the statement number table are closed up to eliminate the desired portion. The strings in the print tables are marked inactive, for later garbage collection. Code is added to a lesson by opening a hole in the instruction table and statement number table of the proper length, and then inserting the compiled code. New print strings are added to the end of the print tables.

A pointer is kept in each table to indicate the last entry in the table, so that new code can be added to the end of a lesson. Source code is input to the compiler one statement at a time. The compiler analyzes the statement instruction by instruction. If it detects any errors it requests that the user re-input the statement from the instruction containing the error to the end.

WRITEACOURSE is broken into several programs in order to fit within the limited computer space available in a time-shared system. The programs operate as overlay segments, with PL/I external variables used to communicate between them. The modular structure of

WRITEACOURSE should facilitate system additions or modifications. Figure 1 shows the basic overlay structure. The functions of each program are indicated in the figure.

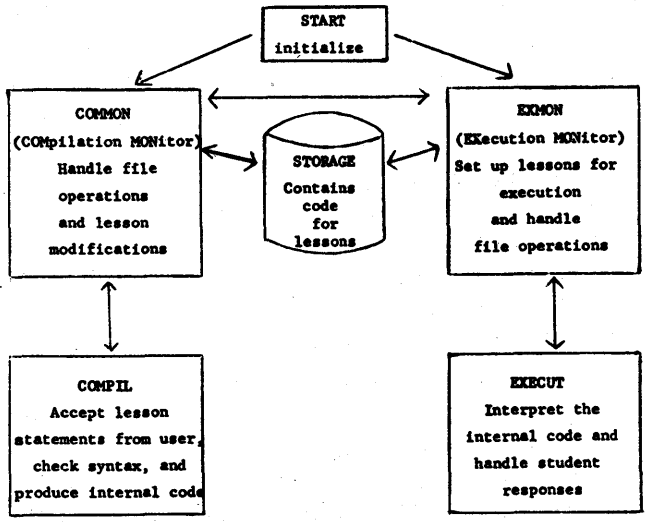


FIGURE 1

Status

The earlier ALGOL version of WRITEACOURSE has been successfully used by people with little programming experience. Although the current version, at the time of this writing, has not yet been put into general use, the programming is completed. A limited number of manuals describing the language details and use of the system are available from the Department of Psychology (Cognitive Capabilities Project), the University of Washington. Listings of the translator and manuals will be provided upon request and at cost.

REFERENCES

1 *Burroughs B5500 information processing systems extended Algol language manual*
Burroughs Corporation 1966

2 S L FEINGOLD C H FRYE
User's guide to PLANIT
System Development Corporation 1966

3 J FELDMAN D GRIES
Translator writing systems
Comm ACM Vol 11 Feb 1968

4 S HENDRICKSON E HUNT
The WRITEACOURSE language programming manual
Department of Psychology University of Washington 1967

5 *IBM 1500 operating system computer-assisted instruction coursewriter II*
Form CAI–4036–1 IBM

6 *IBM operating system/360 PL/I: language specifications*
Form C28–6571–2 IBM 1966

7 P NAUR (Editor)
Revised report on the algorithmic language Algol 60
Comm ACM Vol 6 pp 1–17 Jan 1963

8 *RUSH terminal user's manual*
Allen-Babcock Computing Inc 1966

FOOTNOTES

1. This research was supported by the Air Force Office of Scientific Research, Office of Aerospace Research, United States Air Force, under AFOSR Grant No. AF–AFOSR–1311–67. Distribution of this document is unlimited.

2. We wish to express our thanks to Sidney Hendrickson for his comments and work on an earlier version of the language.

3. There is an unfortunate ambiguity in the word "program", since it is used by educators to mean a sequence of interchanges between student and teacher, and by computer scientists to mean the sequence of commands issued to a computer. We shall use "lesson" when we mean "sequence of educational steps" and "program" when we mean "sequence of commands to be executed by a digital computer."

4. At this point the mind of people not familiar with modern computer technology tends to swim. It is possible to carry this process even further (3).

5. The statement had to end at line (2) because of the IF.. THEN command. The general rule is that when a question of the form IF *condition* THEN is asked, the commands between the word THEN and the next | are executed only if the condition is true. If it is false, as it is in this case, the command immediately following the |, i.e., the first command of the next statement, is executed.

6. More complicated matches are possible, which do not require exact identity. For instance, it is possible to ask if a check string is included anywhere in an answer, so that, in this case 2 would check if the answer had been IT IS REFRACTION.

7. A manual describing the language in detail is available.

A table driven complier for use with automatic test equipment

by ROLAND L. MATTISON and ROBERT T. MITCHELL

Radio Corporation of America
Burlington, Massachusetts

INTRODUCTION

When generating compilers for use with automatic test equipment (ATE), a substantial need arises for flexibility in both the source and object languages. Flexibility is desirable for two reasons: (1) The field of ATE construction is rapidly expanding[1] and (2) the hardware and support software design, development, and debug cycles are often going on simultaneously. In earlier, more standard compilers, the modifications and/or extensions of either language could easily create chaos for the systems programmer.

In an attempt to facilitate compiler implementation and growth, a table driven system, the Universal Test Equipment Compiler (UTEC), has been developed. As in other table driven systems,[2] the function of defining a source language has been dissociated from the actual translation mechanism. The source language is specified to the generator which creates a set of tables for subsequent use by the translator. A dual-purpose meta-language has been created for use in the system. This language is used to specify the syntax of a particular source language and the meaning to be imparted to the various allowable constructs of that language.

A typical ATE system consists of various programmable devices for applying stimuli to, and obtaining measurements from, the unit under test (UUT).[1,3]

A requirement peculiar to ATE compilers is the creation of a wire list specifying connections between the ATE and the UUT. An equipment designator has been included in the system to handle the wire list and to insure that the wire list remains fixed despite source program recompilations. This is necessary due to the cost incurred in the production of this wiring.

The wide range of computers currently used in ATE dictates that the output of UTEC be a symbolically addressed code which must then proceed through the second pass of a normal two pass assembler. Since this reduced assembler could be different for each type of ATE, it will be excluded from the following discussion.

The flow of information through the UTEC system is depicted in Figure 1. The source language specifications and translation logic are defined to UTEC using the meta-language and are fed into the generator. From this, the generator produces translation tables for use by the translator. The generator also accepts the ATE hardware configuration and produces equipment tables for the equipment designator. When a source program is input to UTEC for translation, the translator uses the translation tables and outputs an intermediate code ready for assembly. Whenever ATE equip-

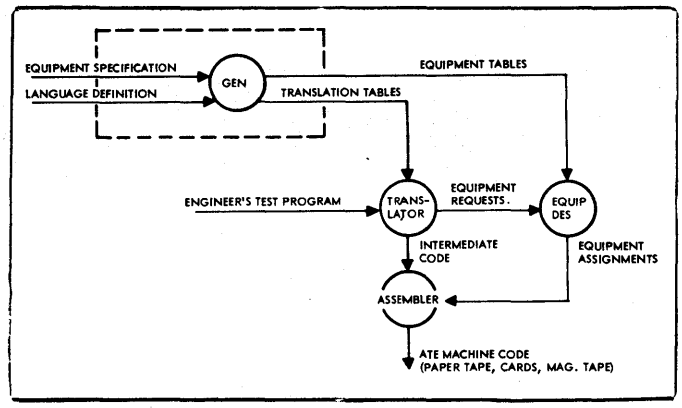


FIGURE 1—System flow block diagram

ment must be specified by the translator, it inserts a symbolic into the intermediate code, and requests the required equipment from an available equipment pool in the equipment tables. The request is tied to the intermediate code by the symbolic. The equipment designator now processes the equipment requests and, using the equipment tables, produces equipment assignments for each symbolic in the form of a symbol table.

The META-language

We now present a language, SYNSEM, (SYNtax and SEMantics) for explicitly defining a problem oriented language (POL).[4] SYNSEM itself is a twofold problem oriented language which (1) specifies the syntax of the POL's and (2) specifies the semantics of the allowable constructs in a POL. SYNSEM is therefore divided into two sublanguages: SYN for specifying syntax, and SEM for specifying semantics.

Problem oriented languages currently in use with ATE are tabular in format. The reason for this, and examples of such languages have been previously presented,[5,7] and, therefore, will not be considered here. Let it suffice to say that fixed fields are generally adhered to, with one field set aside for the function or verb and the remaining fields for modifiers of various types. Each verb-modifier complex is referred to as a source statement.

The goal of SYN is to allow format-syntax type information to be specified for each verb of the POL. This information is encoded into a table by the generator and will be used by the translator whenever the verb is used in a source program.

SYN is comprised of various disjoint subsets of any commonly used character set. Three of the subsets are given below:

NUMBERS = {A, B, C, D, E, F, G, H, I, M, N, O,}

LETTERS = {P, Q, R, U, V, W, Y}

MAIN UNITS = {K}

To specify a numeric modifier, a letter is chosen from NUMBERS and repeated so that the number of times the letter appears equals the maximum number of digits the modifier may contain. Alphabetic modifiers are handled in a similar way by choosing from LETTERS.

If desired, a MAIN UNITS modifier may be used with any verb. When the letter K is recognized by the generator, the 4 characters immediately following the K are taken as the MAIN UNITS and entered into a dictionary with the verb. MAIN UNITS are used to further distinguish the verb when more than one source statement uses the same verb.

As an example, consider the following SYN statement to specify the verb CONNECT with the modifier VDC:

```
CONNECT  AAA  KVDC  BBB  PPPP
```

The modifiers may be 2 numeric (A and B), 1 alphabetic (P), and the MAIN UNITS for this form of CONNECT is VDC.

Once SYN has been used to specify a given verb, a SEM "program" is written, later to be executed by the translator, which analyzes the verb-modifier relationship and generates the desired intermediate code for the source statement. The SEM language is composed of a number of semantic instructions, some of which are described below. A maximum of 750 such instructions can be used in any one SEM program. A statement in SEM consists of a semantic instruction followed by its possible modifiers. A three digit label is optional for all statements. A three digit branch is required with some instructions and optional with others. If a branch is given on optional instructions, it is considered unconditional. The SEM instructions are divided into three categories: (1) Code producing, (2) Modifier handling and (3) Control.

CODE, CVAR, CALPHA, and CSIGN are four of the code generating instructions. CODE tells the translator to output the characters which are literally specified with the CODE instruction.

```
CODE  3  PS1
```

will cause the three characters "PS1" to appear in the intermediate code. CVAR, CSIGN, and CALPHA each are used with an identifier which the translator references to find the data to be output. The identifiers with CVAR and CSIGN must be numeric.

```
CSIGN  NUM1
```

will generate a + or − depending on the sign of NUM1

```
CVAR  NUM1  4  2
```

will cause the value of NUM1 to be coded using

four characters with two implied decimal places. If NUM1=46.913, the characters 4691 will be coded.

```
CALPHA  W1  3
```

will cause the three leftmost characters of W1 to be coded.

TEST, RANGE, and the four arithmetic operators ADD, SUB, MUL, and DIV are some of the modifier handling SEM instructions. TEST causes the translator to compare a referenced quantity with a group of characters specified following the instruction. RANGE causes a check of a referenced quantity to see if it is numerically between two limits. Execution of either a TEST or RANGE instruction by the translator can cause a branch in program flow to a labeled SEM statement if the comparison fails.

```
TEST  A  3  AMP  40
```

causes a comparison of the three leftmost characters of A with the three characters AMP.

```
RANGE  B  20.0  30.0  40
```

causes a comparison of B to see if $20.0 \leq B \leq 30.0$. If the above comparisons are satisfied, the translator executes the next sequential instruction; otherwise, statement 40 will be processed next.

JUMP, SCWL, ROUTINE and EQUI are each SEM control instructions. JUMP is used with a SEM statement label and causes an unconditional transfer by the translator to the labeled statement. SCWL informs the translator that all of the intermediate code generated for a particular source statement must be saved with a label for future use. The SEM language is provided with a subroutine capability through the ROUTINE instruction. ROUTINE may be followed by a parameter list of from one to seven dummy parameters. The CALL instruction, followed by the actual parameters, is used to invoke a SEM subroutine. The EQUI instruction is used to cause the translator to generate a symbolic equipment request for the equipment designator. Its modifiers must be a unique symbolic, which will be placed in the intermediate code by the ECODE instruction, a type number referencing a particular pool of equipment, and a set of "connections" to which a specific piece of equipment from that pool should be wired.

A control card called REQUIRED is used between the two parts of the SYNSEM language and lists all required modifiers in the SYN portion.

The following example shows the combined use of the SYN and SEM languages to specify the verb CONNECT modified by VDC.

```
    CONNECT    AAAA    KVDC    JC    JD
    REQUIRED   AC
    NEWN       UN1
    CODE       1       S
    RANGE      A       0       50          10
    EQUI       5       UN1     C     D
    ECODE      UN1
    CVAR       A       4       2           20
10  RANGE      A       51      120         30
    EQUI       6       UN1     C     D
    ECODE      UN1
    CVAR       A       5       2
20  CODE       2       ES                  40
30  ERROR      A       OUT OF RANGE
40  END
```

The above program is suitable for input to the generator which would create the necessary table entries for later use by the translator after it sees the CONNECT VDC verb in a source program. For example, suppose the statement

```
CONNECT  4.2  VDC  J1-4  J3-5
```

was processed by the translator. The code produced by the previous definition would be

```
S;00005;0420ES
EQUIPMENT
SYMBOLIC
```

which would cause the ATE to connect 4.2 volts of direct current between points J1-4 and J3-5. The equipment symbolic number 00005 was produced by the SEM instruction NEWN.

In a compiler for use with ATE, there is another function of the meta-language, other than defining source statement syntax and semantics. It is to specify equipment available in any ATE configuration. This is done in UTEC by the two instructions, SYMBOL and DATA.

All equipment is divided into types, e.g., power supplies, signal generators, voltmeters, and each type is given a number to identify it. One SYMBOL instruction and as many DATA instructions as there are pieces of equipment in a type are used to define that type. The SYMBOL instruction tells how many pieces of equipment in the type, how many connection terminals each has, and the total table area required to store requests

for this type. Each DATA instruction gives an equipment name and the ATE connection terminals for it.

When a new POL or a modification to an existing POL is defined to UTEC by means of SYNSEM, the syntax of the language and its semantics are stored into tables by the generator. Since ease of language modification is a requirement, three tables have been implemented as linked lists.[6] The format list is used to hold the syntax specification for each verb in the language. The logic list is used to hold one entry for each SEM instruction specified in a verb definition. The logic modifier list holds SEM instruction modifiers which are not suitable for entry in the logic list. A dictionary is also used which contains the name of each function defined by SYNSEM as well as various pointers to the lists. Since UTEC is designed to handle POL's for automatic test equipment, it automatically controls the assignment of equipment and produces wire lists. A pair of equipment tables, the hardware name table and hardware usage table, are built by the generator to aid in these tasks.

The generator

The generator division of UTEC accepts the definitions of verb syntax and semantics written in the SYNSEM language, and assembles this information into all the necessary tables and lists. It also has the ability to delete and equate verbs in the lists, and to build the equipment tables. The generator is used whenever a DEFINE, EQUATE, DELETE, or EQUIPMENT control card is encountered and is divided into four corresponding sections as follows:

Define: Following the DEFINE control card, the SYNSEM language is used to define verbs. First the syntax of a verb is given using the SYN language. The verb is placed into the dictionary. The syntax specification is analyzed character by character, determining the type of each argument encountered. It counts the number of characters or uses a standard count allowed in each argument, and thus builds the format list. It also enters the symbolic character of each argument along with its format list position into an argument table for later reference by the REQUIRED control card and SEM language instructions. At the completion of analyzing the syntax, the argument table contains the one letter symbolic of each argument, in the order in which they will appear in the source statements. The REQUIRED control card containing the one letter symbolic of each required argument follows the SYN syntax specification. Each argument is found in the argument table and its format list position obtained. The format list is thus modified to indicate which arguments in the syntax are required with each usage of the verb. After the REQUIRED control card is processed, the generator must load the SEM program, which gives the semantics of the verb, into the logic and logic modifier lists. There are fourteen different formats for the thirty-six SEM instructions. There are fourteen corresponding routines in the generator to handle the building of the lists. For each instruction, the correct routine is called to set up the list entries. When a modifier of a verb is referenced by an instruction, the one character symbolic of the SYN language is given as a modifier to the SEM instruction. This character is looked up in the argument table and its integer position number is used for the logic list entry. (When a source statement is parsed by the translator, each argument is loaded into a table at the same position as is used for containing its SYN symbolic character in the generator.) Variables may be established in the SEM language by a symbolic name. This symbolic name is placed into the argument table after the symbolic SYN modifier characters, thus establishing a location for numeric reference in the logic lists entries and for storage use by the translator. The argument table provides storage only within a single definition, in that each new source statement starts using this table at its top, destroying symbolic names from previous source statements. The SEM instructions SX and TX provide storage locations for use throughout an entire source program compilation. A table exactly like the argument table is used, except that the location symbolic name is never destroyed, thus giving each definition access to the same location and providing for exchange of information between definitions. If a SEM instruction requires alphanumeric information, or if one entry in the logic list is not sufficient to contain all the necesary data for the instruction, a pointer to the logic modifier list is placed in the logic list entry, and as much space as necessary is used in the logic modifier list. A two digit op

code (1-36), for access by the translator; and a brance and link address, are always in a standard location in each logic list entry.

Equate: Many verbs in a particular POL developed for use with automatic test equipment are similar in syntax and semantics. For example, the source statement for connecting a stimulus to deliver volts is very similar to the statement for connecting kilo-volts or milli-volts. The equate section of the generator was therefore developed whereby two or more verbs may share the same definition, and therefore the same list area. The name of the verb to be equated is placed in the dictionary and all the pointers associated with the equated verb are used with the new one, thereby using the same definition. In order that the small differences of the two verbs can be taken into account, the CHFLG SEM instruction must be used in the original definition. This instruction requires two indicators which are stored in the dictionary. A definition always sets them to zero, but they may be set to any desired value by the language designer using the EQUATE option. The CHFLG op code can test the value of these indicators and thereby set up branching logic in the SEM language program to control the translation.

Delete: The generator section of UTEC maintains a list of available cells to which the DEFINE section looks as it makes the various list entries for a given definition. The purpose of the DELETE section is to remove previously defined verbs from the dictionary and to restore their various list entries to the list of available cells. This is done by changing the link at the bottom of the list of available cells to point to the top of the list entry for the deleted verb. This makes the last list entry for the deleted verb the bottom cell on the list of available cells.

Equipment: In this section, the generator builds the hardware name table and allocates area in the hardware usage table, both of which are used by the equipment designator. All equipment of each type which the system has available is defined in the SYNSEM language by the SYMBOL and DATA instructions. The generator calculates the area required in each table section, and sets up the pointers in the hardware name table and hardware usage table. The DATA cards contain the name of one piece of equipment along with its terminal connections in the allocated positions in the hardware name table.

Translator

The analysis and translation of source programs and the subsequent output of intermediate code is handled by the translator.

When analyzing a source statement, the verb is first checked against the list of defined verbs in the dictionary. When a match occurs, an attempt is made to verify any main units allowed with the verb. Once a verb and main units match is made, the associated dictionary entries are used as references to the format and the logic lists where information specified by the SYNSEM program for this source statement has been stored by the generator.

Using the format list as a guide, the translator parses each source statement and creates an argument table as it goes. A left to right scan of the source statement is initiated looking for a modifier of the type specified in the first format list entry. If the modifier is found, it is placed in the first position of the argument table. The scan continues looking for the next modifier as called for in the next format list entry and places it in the next available argument table position. An error condition exists if the scan fails to verify a modifier, unless that modifier is not required in which case a dash is placed in the argument table in the next position. The scan finishes when the entire format list for this verb has been considered and an argument table entry is present for each item in the list. Once the format scan has been completed, the translator turns its attention to the logic list where the algorithm for generating intermediate code has been stored for this source statement. Each entry in the logic list is a numeric representation of one of the SEM instructions. A two digit op code is extracted from each entry which identifies the particular SEM instruction requested. Once the instruction is known, the translator is able to completely dissect the logic list and modifier list entries for this instruction and perform the desired operation. All references by the instruction to the verb modifiers are made by simply referencing the argument table position for that modifier, as the parsing algorithm already has inserted the modifiers in the table. As an example, consider the following SYNSEM specification:

```
CONN    AAA    KVDC    JC    JD
  .
  .
  .
RANGE   A   10.0   20.0   100
```

The generator creates the argument table as shown in Figure 2.

1	A
2	K
3	C
4	D

FIGURE 2—Argument table of generator

Since the character A is in position 1, this position number is used in the logic list when the RANGE instruction is processed by the generator. When the source statement:

CONN 14.6 VDC J101-42 J16-33

is parsed by the translator, the argument table is filled as shown in Figure 3.

1	14.6
2	VDC
3	J101-42
4	J16-33

FIGURE 3—Argument table of translator

When the translator discovers the RANGE instruction number in the logic list, it decodes a reference to position one in the argument table for the number it is to test. In the example, the number 14.6 is checked to determine if it is between 10.0 and 20.0.

Each logic list entry provides the translator with the position of the next instruction to be considered, or in the case of conditional instructions, the translator must pick the next instruction from two or three choices after it performs the current instruction.

The translator continues through the logic list until the END op code is discovered, at which time it has completed its analysis and code generation for the source statement under consideration. The next statement is read and the entire process repeats. When the translator reads the END verb it turns the intermediate code generated for the program over to the assembler for final object code production.

Equipment designator

Each time the translator processes a source statement which requires the use of ATE equipment, an entry on a tape is generated by means of the SEM instructions EQUI or PREAS. This tape is called the request tape. The translator itself has no ability to select equipment from the available equipment pool in order to satisfy the needs of the source statement. The SEM language program used to translate these source statements requiring equipment first generates a unique symbolic number which will be used by it to symbolically refer to an equipment name in the intermediate code it produces. It then determines the type of equipment required by the source statement and generates the request tape entry. Each such entry generated tells the type of equipment desired and the symbolic number used to identify it, and tells how the terminals of that equipment should be connected. In the case when a specific piece of equipment must be used in a particular manner, the translator also processes an equipment preassignment by producing a request tape entry which gives the specific name of a device and tells how it is to be connected.

The function of the equipment designator is to read and process the entries on the request tape produced by the translator. It attempts to match an equipment name of the correct type to each symbolic number and produce information describing how each piece of equipment is to be connected. In the assembly of the intermediate code, each symbolic number is replaced by the matching equipment name as provided by the equipment designator.

The equipment designator operates using two tables: the hardware name table and the hardware usage table. The hardware usage table is divided into two sections for each equipment type: the hardware assignment section and the hardware request section.

The hardware assignment section for each equipment type contains one assignment indicator and one row for each piece of equipment of that type. The indicator gives the status of the equipment while the row contains references to the connections made to this equipment.

The hardware request section for each equipment type can contain a number of requests for equipment of that type. Each request section entry is made up of an indicator and row like those

in the assignment section, plus a half-word which is used to hold the unique symbolic number for the request. The number of entries allowed in the request section for a particular type of equipment is specified in the SYNSEM equipment definition.

The requests processed by the designator fall into two classes: (1) Those which name specific pieces of equipment, and (2) Those which symbolically seek an assignment of any piece of equipment of a specified type. When fulfilling requests, two passes are made over the request tape with the items in classes one and two being handled on passes one and two respectively.

On pass one, the designator simply reads the requests, and in the hardware assignment section, sets the indicator for the named piece of equipment and fills the rows with the connection references. Before the first pass, all indicators reflect an equipment available status. After pass one, the indicators of the equipment named in pass one are set to indicate one of two states: (1) Hard preassigned-specified equipment may only be used as stated. (2) Update preassigned-specified equipment should be used as stated if possible, but may be used differently if needed. This preassignment is automatically generated at the end of each compilation for each piece of equipment used. It then is submitted on the following run to insure that the same wire connections will be generated whenever possible, even when changes are made in a source program.

On pass two, the designator tries to assign one piece of equipment to each symbolic request. In addition, it creates the matching list to be used by the assembler when processing the symbolic references in the intermediate code.

Each request causes a scan of the hardware assignment section for the type of equipment requested. If the connections of the request match those of a piece of equipment already used, the request is matched with that equipment. If the connections of the request do not match those of any already used, a new piece is assigned to match this request. If all the equipment of the type requested has been used, the request is put into the hardware request section and saved. When the entire request tape has been read in pass two, the designator is finished unless some unfulfilled requests remain in the request section. If unfulfilled requests do exist, the designator scans the assignment section for all equipment which was update preassigned but not used in this compilation. It resets the indicators of these equipments to reflect an available status. An attempt is then made to assign the unfulfilled requests to the equipment made available. If the request still cannot be satisfied, it remains in the hardware request section. Finally, a wire connection list is produced from the hardware assignment section giving all the equipment used in the compilation and how it is to be connected. How the equipment was used in relation to a possible previous compilation is also stated. Error conditions are produced based on entries remaining in the hardware request section. New update preassignments are also generated for use if the program is to be changed and recompiled, so that a similar wire list can be produced.

CONCLUSION

At this time, UTEC has been completely written and checked out using FORTRAN IV, and a language developed for use with one type of automatic test equipment (LCSS) currently being produced by RCA has been implemented using UTEC.

The implementation of another language for a second type equipment is being considered at this time.

It is interesting to note that after having defined the language to UTEC, the users could evaluate the quality of the language and its usefulness, and suggest changes and improvements.

These changes were easily incorporated into the language almost daily during a shakedown period, thus allowing them to be tested within days after they were conceived. The overall effect was to stimulate ideas for improvement. Thus, a much more effective language than that originally specified was developed.

REFERENCES

1 B J EVANZIA
Automatic test equipment: a million dollar screwdriver
Electronics August 23 1965

2 P Z INGERMAN
A syntax-oriented translator
Academic Press 1966 ch 1 pp 13–19

3 V MAYPER
Programming for automated checkout—Part I
Datamation April 1965 Vol 11 No 4 pp 28–32

4 V MAYPER
Programming for automated checkout Part—II
Datamation May 1965 Vol 11 No 5 pp 42–46

5 B L RYLE
The atoll checkout language
Datamation April 1965 Vol II No 4 pp 33–35
6 M V WILKES
Lists and why they are useful
Proc ACM 19th Natl Conf August 1964 Phila Pa
7 B H SCHEFF
Simple user oriented compiler source language for programming automatic test equipment
Communication of the ACM April 1966 pp 258–266

On the basis for ELF—An extensible language facility*

by T. E. CHEATHAM, JR., A. FISCHER and P. JORRAND

Computer Associates, Inc.
Wakefield, Massachusetts

INTRODUCTION

There are two basic premises which underlie the development of ELF. The first of these is that there exists a need for a wide variety of programming languages; indeed, our progress in the understanding and application of computers will demand an ever widening variety of languages. There are, in fact, "scientific" problems, "data processing" problems, "information retrieval" problems, "symbol manipulation" problems, "text handling" problems, and so on. From the point of view of a computer user who is working in one or more of these areas there are certain *units of data* with which he would like to transact and there are certain *unit operations* which he would like to perform on these data. The user will be able to make effective use of a computer only when the language facilities provided allow him to work toward a desired result in terms of data and operations which he chooses as being a natural representation of his conception of the problem solution. That is, it is not enough to have a language facility which is *formally sufficient* to allow the user to solve his problem; indeed, most available programming languages are, to within certain size limitations, universal languages. Rather, the facility must be natural for him to use in the solution of his particular problem.

The second premise underlying our work is that the environment in which programs are prepared, debugged, operated, documented and maintained is changing and that the language facilities currently available do not properly reflect these changes. We are speaking, of course, of the advent of computer-based files and of interactive computer systems which permit the user to be more intimately involved with his program than was possible with a batch system. A modern language system must be developed with this kind of environment in mind, but should still be adaptable to the older environment.

Let us now explore briefly the implications of these two premises and examine some alternative approaches to providing an appropriate language facility.

The "classical" approach to providing a large variety of languages has been that of developing languages and their translators—and often even their operating environments—independently. However, it seems clear that the cost of creating and maintaining an ever increasing number of language systems is not tolerable. Somehow we must both provide the variety of facilities but, at the same time, also reduce the number of different systems. It would seem that there are two extreme approaches to the problem of developing a language facility which provides all things to all men. These are referred to as the *shell* approach and the *core* approach. The shell approach calls for the construction of one universal language which contains all the facilities required for every class of users. PL/I with the "compile-time" facility is probably the best current example of a shell language. In contrast, the core approach calls for the development of a small "core" language which, by itself, is probably not appropriate for any class of user, but which contains facilities for self-extension. A particular class of users then extends the core language to create a language which is appropriate for their problems. There are, to our knowledge, three current languages which are, to some extent, core languages: ALGOL-D, GPL, and ALGOL-68.

The shell approach does have a certain appeal. Like the modern supermarket it promises us a great variety of both ordinary and sophisticated products. But the overhead inherent in utilizing such a system is rather large. As in the supermarket the user must pay for both the space to contain the products he is not using and the extra time to access the desired product. Perhaps a more important difficulty inherent in the shell approach is

*This work was supported, in part, by the National Aeronautics and Space Administration under contract No. 12-563

that whenever a meaning is prescribed for a construction, that same meaning is forced upon all users, even though the construction might reasonably mean several things. In PL/I this has lead to such anomalies as: both of the boolean expressions $5 < 6 < 7$ and $7 < 6 < 5$, are true; the interpretation A*B where A and B are matrices is the matrix whose $(i,j)^{th}$ element is the product of the $(i,j)^{th}$ elements of A and B. It is not that these kinds of interpretations are "bad"—the point is that they are built-in and unchangeable. No matter what meaning one might like for $7 < 6 < 5$ (we like *false*) or for $(i,j)^{th}$ element of A*B, (we like the inner product of the i^{th} row of A and of the j^{th} column of B), that meaning provided by the designers is now fixed. One must revert to procedures if he wishes to introduce new operators or to detour around the built-in operators when he needs to vary the meaning of those originally provided. And this becomes even more cumbersome when, as in PL/I, procedures can produce only scalar results. We would maintain that our reasons for rejecting the shell approach are not based on speculation; the difficulties currently being experienced with the implementation and utilization of *full* PL/I provide ample evidence.

Thus it is our contention that the most reasonable approach to providing the desired variety of language facilities is that of providing an extensible language supported by an appropriate compiling system. We do not, however, suggest that we can now devise a single universal core language which will adequately provide for the needs of the whole programming community; the diversity in "styles" of languages and translation mechanisms will probably always be sufficient to encourage several language facilities. ELF, which is the subject of this paper, provides a facility in the "style" of such languages as ALGOL–60, PL/I, and COBOL.

Now let us discuss the second premise, concerning the environment in which we envision programming being done. Our basic assumption here is that the programmer does not approach the computer with a deck of cards or magnetic tape which constitute a complete and independent run: a "run" deck which would commence with control cards, followed by his problem and then by his data, and which would result in the system accepting these, compiling his problem, running it against his data, and finally burying him in dumps or some other visible output. Rather, the programmer's unit transactions should be thought of as acts of updating some file. He might insert a few corrections to his program text, might call for some incremental change to some executable form of his program, and then might let his program run.

If he is working in an interactive system, he might maintain intimate control over the proceedings, responding to messages as they occur instead of having to wait for the final results before he can exert any control.

We do not suggest that ELF is a solution to the problem of providing a language for the effective use of a modern time-shared* system with permanent users' files. Indeed, there is really very little experience now accumulated in using such facilities, as most of the language facilities now in use on the available systems were developed as "batch" languages. It is to be hoped that work such as that now underway at Carnegie-Mellon under Perlis' direction will provide some guidance in this area.[11]

We do suggest, however, that we can now devise an extensible language facility in such a manner that it is cognizant of an avilable filing system and provides for interactive control; we will discuss our point of view on the relation of the language to the system in a later section.

The remainder of this paper is divided into four sections. In the next section we will discuss the overall design criteria which have guided the development of the language. Following this, we will present an overview of the language with the object of providing the reader with a general feeling for the language as well as for the translating and executing mechanisms which we envision. Following this we will discuss the kinds of features and facilities which will be in the language; the purpose of this section is to justify and describe certain constructions proposed for the programming language component of ELF. The final section is devoted to a summary and conclusion.

Design criteria

Perhaps the most eloquent defense of the overall design criteria to which we have tried to adhere was given in the 1966 Turing lecture by A. J. Perlis.[10] There Perlis framed the problem as that of providing for systematic variability in a language. All acceptable languages provide for constant as well as for variable operand values. However, a great deal more variability must be provided if a language is to be extensible. There must be means of providing for variability in the types of quantities with which we deal, in the operations on these quantities, in programs or procedures, in the syntax of programs, in regimes of control, in the binding of programs to other programs and data, in the means of accessing data, in the employment of the various storage and input/output resources afforded by the system, and so on. However, we must provide for this variability very carefully so that we retain the necessary control over the efficiency of use of the com-

*That is, interactive: how the intimacy between the user and the system is arranged does not concern us.

puter, or else our result will be a purely academic exercise.

In our design of ELF we have looked to a number of "users" as sources of constraint; unless the language facility is properly matched to its users, it will not be an effective tool. These "users" include the programmers who will read and write in the language, the computer which will execute programs, the compiler or translator which will prepare executable programs, and the operating system which will provide the environment for the preparation and execution of programs. In addition, we feel that there are two other important sources of constraint: the traditions established by current languages, and the practicality of the language. Let us now briefly discuss the nature of the constraints which each of these various sources imposes.

Programmers

People have to learn and use the language. Indeed we hope that people will even *read* programs in the language in addition to writing them. However, we find that different people have rather different ideas about the form in which a program should be cast. Most serious programmers adhere to the basic expression forms where these forms are appropriate—using the infix, prefix, and postfix operators plus parentheses which have resulted from the years of development of mathematical notation. The form of program text which is not inherently "expression-like" is, of course, not so well established. We note here, however, that the usual "out" for introducing new operations into a language—the use of functions or procedures—does not provide an adequate notation for the majority of operations. If the number of arguments required exceeds three or four, the user has difficulty in associating the "meaning" of an argument with its position in the argument list and he might be considerably better off with some keywords to help him focus on what is what. Also, if the nesting of function calls gets to be more than two or three deep, the "LISP-unreadability" problem becomes serious. We would also note that, for the user, an important criterion is that he should not have to introduce and deal with constructs which are unnecessary to the solution of his problems. The arithmetic expression form provides a facility which is both natural to a large class of users, and which also very effectively hides the setting up of temporary storage for intermediate results. Similarly the various renderings of McCarthy's conditional expressions as well as the iteration or looping facilities which appear in many programming languages have, as a secondary effect, that of eliminating the need of introducing temporaries or labels which are used only once (see Refs. 2 and 9 for interesting discussions of this point).

Computers

The abilities of current and projected computers must also be viewed as a source of constraint. That is, we should try to "match" the basic types and operations in the language with those available in "standard" computers (and here we have reference to CPUs, not the whole "system"). Thus, for example, although our mathematical natures might encourage us to define only integer quantities and operations as primitive, and obtain floating point quantities and operations by extension, this would surely be foolish when we are faced with computers which by-and-large have floating point quantities and operations as primitives. Similarly, we reject the notion of quantities and operations drawn from set theory as primitive in the language because of the wide variety of implementation strategies which might be employed in providing for these. Such quantities and operations should be introduced via extensions. We must presume that the facilities available in current computers mirror, to some extent, the basic facilities which the users require.

Compilers

The past several years have witnessed the emergence of a considerable body of experience and technology in compiling programs. Unfortunately, most recent language developments seem to ignore this technology and demand new and ever more difficult and expensive translating mechanisms. We have attempted to reverse this trend and to adhere rather strictly to the technology available—to provide a language which can be effectively and efficiently translated and for which the known techniques of code generation and optimization will apply.

Operating systems

The constraints which might be imposed by the peripheral devices and operating systems which are to be used must be noted. Thus, the means for encoding messages to and from the computer are rather strictly dependent upon the devices (and software) which are available; our adherence to a conventional string language with reasonably conventional characters is dictated by this consideration. The control structure inherent in modern computers must also be kept in mind; for example, the notion of "interrupt" is basic in most computer systems and our language facilities should reflect this. Further, the availability of various kinds of storage having varying degrees of accessibility plus the needs of the operating system to allocate the storage and other resources of the system must not be ignored.

Tradition

The "tradition" which has been established by such languages as ALGOL-60, PL/I, COBOL, and LISP and which is being established by ALGOL-68, GPL, and ALGOL–D should be considered as a source of constraint. That is, it does not seen reasonable to re-invent and re-cast the facilities available in those languages just to be different. Our departures from the facilities available there should be well thought out and well justified. It will be clear that we have in fact departed in more-or-less significant ways from all these languages; we hope that our arguments for doing this are convincing.

Practicability

The final source of constraint which we have tried to observe is that of practicability. It is our intention that the language be as efficient and useable as any of the conventional programming languages. In adhering to this constraint we have failed in many ways to reach all the goals of variability which Perlis prescribed. Thus, ELF provides a language which has the kinds of variability which we can imagine being handled with reasonable efficiency. Another generation of language development will be desirable when we better understand other kinds of variability and can devise mechanisms for handling them efficiently.

Overview of the extensible language facility

There are a number of facets of ELF that are of interest. In this section we will look at three of these. First we will look briefly at the base language (BASEL). Following this we will consider a compiler for BASEL, and the ways in which it might be extended. Finally we will consider the interface between the language system and the operating environment.

The base language

BASEL has four kinds of primitive program elements:

a Names identify the objects that a program manipulates.
b. Operators manipulate the objects. These include the assignment operator, operators such as plus and less than, and procedure calls.
c. Control statements are used to specify the order in which the expressions are executed.
d. Declarations are used to define objects and operators, and give them names.

These program elements are embedded in a block structure which is a generalization of that in ALGOL-60.

In the next section we will discuss the elements of the language in somewhat more detail; for the moment it will suffice to think of the language as similar to ALGOL-60 but with provisions for new data types and operations (or, if ALGOL-68 is familar to you, similar to ALGOL-68).

A BASEL compiler

Second we want to explore the compiling mechanism which we have in mind. Although one does not conventionally talk about compiling techniques in describing a language, we believe that it is helpful in this case. Our choice of language constructs, notations, and mechanisms has been strongly influenced by what we feel can be readily handled by the current compiling technology. Thus understanding our view of the kinds of compiling mechanisms envisioned is rather important. For present purposes we want to think of the compiler for the language as consisting of several "components," including: a lexical analyzer, a syntactic analyzer, a parse interpreter, and a user controlled optimizer, plus other components for generating machine code and filing it, or for interpretively executing some "internal" representation of the program text, and so on. We shall have no particular interest in these latter components here; let us now consider the other components.

Lexical analyzer

The lexical analyzer will be responsible for isolating, identifying, and appropriately converting the source input (e.g., typed characters) thus producing a stream of "token descriptors" representing constructs at the level of "identifier," "literal," "operator," "delimiter," and so on. We anticipate that, although the lexical analyzer will be "table driven" by tables derived from a grammar which specifies the structure of the tokens, these tables will not ordinarily be changed or extended by the average user and we will thus think of them as fixed.

Syntactic analyzer and parse interpreter

We intend that the syntactic analyzer be essentially an operator precedence analyzer. An operator precedence analyzer is, of course, one of the simplest and most efficient kinds of syntactic analyzers available. Operator precedence analysis works only on a rather restricted set of languages. However, as Floyd demonstrated in his original paper on this method ,[4] ALGOL-60 is close to being an operator precedence language; further, those changes Floyd proposed to the original syntax rules for ALGOL-60 and to certain constructions in the language in order to make it operator precedence

did no real violence to the language but actually made it cleaner and more symmetric. Thus, a language does not necessarily suffer in richness of style because it was designed with this method of analysis in mind. Another important reason for the choice of this method is that those properties of the operators which, properly encoded, are required to "drive" such an anlyzer are exactly the properties which the user has in mind when he specifies an operator, namely, the precedence, in the sense of order of evaluation of that operator relative to other operators.

It will be convenient to think of the operators available in the language as including binary infix (e.g. '+' or '<'), unary prefix (e.g. [1]–'), unary suffix (e.g. '!'), unary outfix (e.g. '| ... |'), n-ary "distributed" (e.g. '*if ... then ... else*' or [1]*increment ... by ...*'), and [11]functional" (e.g. 'MAX (...,...)' or 'SIN()'). Each operator (actually each fixed "part" of each operator) will enjoy one of four relations with respect to all other operators (or parts of operators), namely: takes precedence, yields precedence, has equal precedence, or none. The user will introduce a new operator (syntactically) by specifying the precedence of each of its parts relative to the precedence of operators already available. It will generally be the case that a given operator will be defined for operands of a variety of data types; the "syntactic analyzer" will isolate a phrase—an operator plus its operands—by using the precedence relations, and then the *parse interpreter* will then determine the "meaning" or "interpretation" of the phrase in accordance with specifications which are either built-in (e.g. with '+' operating on two integers) or supplied as extentions by the user (e.g. with '+' operating on two quaternions). One of the "dispositions" which the parse interpreter might make of some phrase is to place the operands of that phrase into some previously given (macro) "skeleton" and re-submit the resulting text for syntactic analysis. This will provide what are essentially the "lexical macro" and "syntactic macro" facilities proposed in. [1]

User controlled optimizer

There are certain operators which require a larger context than the phrase in which they occur for their interpretation, particularly if one has a goal of producing optimal coding and either does not have, or perfers not to overburden, a code optimizer. An example here would be the coding of the multiplication of two conformable matrices in the three contexts:

$$A * C \qquad (A + B) * C \qquad (A + B) * (C + D)$$

Thus, it may be that one might desire an algorithm for matrix multiplication which required only one temporary scalar for the first case, a temporary row for the second, and a full temporary matrix only for the third. On the other hand, one might use temporary rows for all three cases, giving up storage efficiency in the first case and computation efficiency in the third. The point is that there are cases in which the determination of the appropriate means for performing some operation depends upon some context. The *user controlled optimization* phase provides for this. It would also be in this stage that the user would have the ability to tinker with such things as the allocation of storage, the means of access (e.g., via some hardware or software "paging" scheme) to certain quantities, and so on.

Briefly, we think of the parse interpreter as constructing what in effect is a computation tree representation of the analyzed program text; each node of this tree would be labelled with the data type of the value it represents. The *user controlled optimizer* may then be thought of as a mechansim which "walks" over this computation tree, inspects context as appropriate, and re-organizes and re-constructs portions of this tree. The mechanism has certain similarities with those proposed in ALGOL–D, but with a control and sequencing strategy similar to that of the GSL component of the CGS system (see Refs. 12, 13 and 15)

Interface view

Now let us briefly consider the interaction of the language with the environment in which programs are constructed, debugged, and executed. First, we want to emphasize that we would expect the language to include the means for the kinds of communication with the operating environment which are typically handled via "control" or "job" statements as well as the kinds of communications which have to do with "editing." We presume that there is a filing system which contains such things as: (1) program text which we might want to incorporate into the input stream to the compiler during some run, (2) specifications of modes of programs or procedures which our current program might want to reference, (3) modifications or extensions to the compiler which we might want incorporated for processing our current program text, (4) an "executor" which can execute programs as they are represented following the interpretation of the parse and user-controlled optimization, (5) data which has been previously input or generated and then filed, (6) and so on. Clearly there must also be means for placing any of these items in the filing system. Thus, a "run" or "session" might be one in which we input a number of extensions to the language including new data types and operations over them, with the result that they are filed in such a fashion that we can later call the compiler, mentioning that it is to include these extensions. Another type of

session might be the input of program statements with the expectation that they be executed directly. Another might be the input and editing of a program with the expectation of filing it for later execution. That is, a "unit transaction" within the ELF system will typically utilize material previously developed and filed, and result in material to be filed. There will be a number of forms which this material might take, ranging from or certain programs within the compiler at the other. So long as this philosophy of operation is understood, we will not go into further details here; we intend to spell out more details of the linguistic forms and possible system mechanisms to attend to problems in this area in a subsequent paper.

The base language

In this section we will present the basic concepts and mechanisms which are introduced in BASEL, the base language component of ELF. It is not our purpose to provide an introduction or primer for the language. (One must see Ref. 3 for this.) Rather, we hope that our discussion will, as it were, "soften the blow" and make the details of BASEL appear reasonable. Thus we will suggest the kinds and means of variability which BASEL permits. We might also note that we are not attempting to justify all the concepts and notions in the language. This section is divided into two parts: First we will discuss the fundamental notion of value and show how it is handled in BASEL, comparing BASEL to other languages. Then we will present the declarations and expressions which manipulate values.

Values

One of the basic characteristics of a programming language is the variety of values it can handle, and the way in which these may be manipulated (that is, read in, stored, named, declared, operated upon, passed as parameters or returned as results of functions and written out.) The larger the variety of values a language can manipulate, the more "powerful" that language is; the more uniformly these values are treated, the easier that language will be to write, compile and extend.

We will use FORTRAN II to demonstrate what we mean by "a value." FORTRAN handles only real values and integer values in a general way. One may read and write alphabetic values, but must treat them as if they were integers. Also, FORTRAN has arrays, but no concept of an array value which may be treated in all ways as a unit.

We can distinguish two aspects of a value: (We have been strongly influenced here by ALGOL-68.)

— its data type or mode (that is, the "class" it belongs to, that aspect which determines the contexts in which it is meaningful and the operations which apply to it.)

— its meaning (that is, its interpretation, the object which it represents.)

Example:

The number 3.14159 has the mode *real* and its meaning is an approximation to the number pi.

One must be able to store values, and thus we are led to think about variables. We see that we can treat variables like values, having both mode and meaning.

Example:

A real variable has the mode *loc real* (a location in which to store a real value) and its meaning is the address of that location.

We have made a distinction here between the *meaning* of a variable and its *value* (that is, the value stored in it.) We can refer to either (as will be described later) and speak of the mode of either. This makes possible a simple and uniform treatment of pointers. We see that a pointer is simply a variable in which the meaning (address) of another variable is stored.

Example:

A pointer to a real variable has the mode *loc loc real* and its meaning is the address of a box in which we can store the meaning of a real variable (a *loc real*).

Continuing this line of thought we see that we can describe the mode of any pointer in terms of the mode of the thing it points to, which permits a compiler to decide whether it is meaningful to use that pointer in any given context.

The aggregate-value is another generalization of the concept of a value to which the notions of mode and meaning must also apply. We must go further than PL/I did. The PL/I declaration

```
DECLARE   1   A
              2  A1  INTEGER
              2  A2  FLOAT
              ...
```

declares an instance of an aggregate, but there is no way of speaking of the mode of that aggregate independently of this instance. One result of this is that such aggregates may not be returned as the results of functions. This difficulty also arises with arrays in ALGOL-60 and FORTRAN.

So in addition to a set of "basic modes" (like *real*) we need a set of "mode constructors" which can be used to combine the basic mode descriptors into descriptions of more complicated values.

Basic modes

We have adopted the following set of modes as basic:

integer value	written	*int*
real value		*real*
boolean value		*bool*
character value		*char*

Mode constructors

Variables:

If $\mathfrak{M}$ is a mode, then *loc* $\mathfrak{M}$ is the mode of a variable which can store a value of mode $\mathfrak{M}$. (*"loc"* in BASEL serves the same purposes as *"ref"* in ALGOL-68. The differences will be explained later.)

Aggregates:

An aggregate is a sequence of values. BASEL has three ways of describing these.

tuple: A tuple is an ordered list of values. These are used primarily as the actual parameter list in a function call, but they can also be computed, constructed and stored in variables.
If $\mathfrak{M}_1, \mathfrak{M}_2, \ldots \mathfrak{M}_n$ are modes, then
tuple ($\mathfrak{M}_1, \mathfrak{M}_2, \ldots \mathfrak{M}_n$) is the mode of an aggregate-value whose parts are n values of these modes.

Examples:

tuple (*real*, *int*, *char*)
tuple (*loc real*, *real*)

row: A row is a homogeneous aggregate. We treat this case separately for two reasons: first, rows are particularly simple to implement, and second, this is the only way to describe a variable-length aggregate.
If n is an integer value and $\mathfrak{M}$ is a mode, then *row* n *of* $\mathfrak{M}$ is the mode of a homogeneous series of n values, each of mode $\mathfrak{M}$. These are numbered from 1 to n and may be accessed by number. Note that this is a more elementary notion than the *array* of ALGOL-60.

Examples:

row 3 *of real*	
row 6 *of loc char*	(This might be used to describe a character string variable.)
row any of int	(This describes a variable length row of integers.

struct: The mode constructor *struct* attaches names to the elements of an aggregate, permitting these parts to be accessed by name. If $\mathfrak{M}_1, \mathfrak{M}_2, \cdots, \mathfrak{M}_n$ are modes and $e_1, e_2, \cdots e_n$ are identifiers, then *struct* ($\mathfrak{M}_1 e_1, \mathfrak{M}_2 e_2, \cdots \mathfrak{M}_n e_n$) is the mode of an aggregate-value with n named parts. (This mode constructor is exactly the same as in ALGOL-68.)

Examples:

struct (*real* r, *real* i)
might be used to describe a complex number.
struct (*loc int* level, *row* 50 *of loc bool* elem)
might be used to describe a push-down-stack which can hold up to 50 boolean values. The integer variable 'level' would store the index of the current top of the stack.

Procedures:

A procedure is a parameterized description of a value. (This value is usually specified by giving an algorithm by which to compute it.) In BASEL, procedures are also considered to be values, which may be stored, passed as parameters, etc. Because of this we are interested in the mode of procedures. Since procedure calls are used in expressions, the procedure's mode must embody information about its domain and range, in order for the compiler to ensure the meaningfulness of expressions involving procedures. Therefore we define the mode of a procedure as follows: if $\mathfrak{M}_1, \mathfrak{M}_2, \cdots \mathfrak{M}_n$ and $\mathfrak{M}$ are modes then *proc* ($\mathfrak{M}_1, \mathfrak{M}_2, \cdots \mathfrak{M}_n$) $\mathfrak{M}$ is the mode of a procedure which takes n parameters of modes $\mathfrak{M}_1, \mathfrak{M}_2, \cdots$, etc. and returns a result of mode $\mathfrak{M}$. If the procedure takes no parameters, empty parentheses are written. If it returns no result, the word *"none"* is used in place of the mode of the result.

Examples:

proc (*real*) *real*	describes the mode of of the sine function.
proc (*int*) *none*	might be used to describe a procedure which

	opens the file designated by the integer code.
proc () *struc* (*int*, *int*)	could describe a procedure which returns the current time of day expressed as two integers representing hours and hundredths of hours.
proc () *none*	could be used to describe a DUMP procedure.

Dynamically varying modes

Occasionally it is useful to permit a variable to store values of more than one mode, or to permit an expression to produce a value whose mode depends on the data. The mode-descriptor operator *'union'* is used to build the description of a mode which is not completely fixed, but can be one of a fixed finite set of modes. The mode of a value stored in a *'loc union'* can vary dynamically among the fixed set of modes specified by the *'union'*. The mode of the value of an expression can vary if that expression contains a conditional whose *'then'* and *'else'* clauses have values of different modes. If $\mathfrak{M}_1,\ldots, \mathfrak{M}_n$ are modes than *union* ($\mathfrak{M}_1,\ldots \mathfrak{M}_n$) describes an object whose meaning may have any one of the given modes; which one may not be known until run time, and may change.

Meanings

Certain values of the basic modes have pre-defined representations. These are:

the non-negative integers	written 0, 1, 2, ···
the non-negative reals	0., 3.96, .5, 3.7 *e*-2
the boolean values	*true*, *false*
the character values	'A' ···, 'Z', '1', '2', ···, ',','+', ···

Because these are conventional representations of their meanings, they are sometimes called 'literals'. Through declarations, the programmer can give other names to these values. In fact, it is possible to name any value that can be computed.

Since an unbounded number of modes can be defined in BASEL, the language must provide a consistent way of denoting a value of any mode.

The conventions we have chosen are:

A value of mode *tuple* ($\mathfrak{M}_1, \cdots, \mathfrak{M}_n$) is denoted by writing [v_1, v_1, ··· v_n]. Each v_1 denotes a value of mode $\mathfrak{M}_i$.

A row or a structure is denoted by writing a *row* or *struct* mode followed by a tuple with the appropriate number of elements, each of which has the appropriate mode.

A procedure of mode proc ($\mathfrak{M}_1, {}_2, \cdots, \mathfrak{M}_n$) $\mathfrak{M}$ is written: *proc* ($\mathfrak{M}_1 f_1, {}_2 f_2, \cdots, \mathfrak{M}_n f_n$) (E)

where the f_i are the names given to the dummy parameters, and E denotes an expression whose value has mode $\mathfrak{M}$. If the procedure returns no result, then E must denote an expression with no value.

Finally, if $\mathfrak{M}$ is a mode then

a $\mathfrak{M}$ creates a value of mode $\mathfrak{M}$.

Examples:

Mode	*A Value of that Mode*
tuple (*int*, *int*, *int*)	[1, 5 + 7, 4]
tuple (*real*)	[sine X]
row 4 *of bool*	*row* 4 *of bool* [*true*, *true*, *true*, *true*]
struct (*real* r, *real* i)	complex [0.5, 2.] ("complex" has been declared as a synonym for the mode *struct* (*real* r, *real* i).)
proc (*int*, *int*) *int*	*proc* (*int* A, *int* B) (if A > B *then* A *else* B)
loc int	*a loc int* (This causes an integer variable to be created. Its contents are not initialized. This expression then denotes the new address.)

Declarations, expressions and programs

Declarations

A declaration attaches a name to a value, thereby creating an object. In BASEL, any value which can be computed or denoted may be named. BASEL has four kinds of declarations, which we will consider in the following order:

- mode declarations
- data declarations
- operator declarations
- meaning definitions

Mode declarations

A mode declaration defines a name for a mode. This name then acts as a synonym for that mode.

Examples

let complex *be struct* (*real* re, *real* im)

This describes a complex number. Having given a name to the mode expression, the following expressions become exactly equivalent:

struct (*real* re, *real* im) [1.4, .2]
complex [1.4, .2]

let tree *be struct* (*char* top, *row* 2 *of loc* tree son)

This describes a recursive tree structure, in which each node has two sons.

Data Declarations

As with mode declarations, a data declaration defines a name for a value. Within the scope of this declaration that name and the value it denotes are completely synonymous.

Examples

let pi *be* 3.14

This declares "pi" to be a name for the real value "3.14". Notice that pi is a name for a constant, not a variable. One would never be able to store another value in pi.

let X *be a loc real*

As mentioned above, the expression "*a loc real*" causes a new *loc real* to be created, and denotes the resulting address. This declaration makes X a synonym for that address.

let MAX *be proc* (*int* i, *int* j) (*if* i > j *then* i *else* j)

This declares that MAX is a name for a procedure which computes the maximum of its two integer arguments.

Operators

In BASEL an operator is simple a collection of related procedures plus some information on how a call on that operator is to be written. This syntactic information is declared with an *operator declaration*, written as follows:

let ⟨operator name⟩ *be* ⟨shape⟩ *prec* ⟨precedence relation⟩ where the ⟨shape⟩ may be

prefix
suffix
infixL (left associative)
infixR (right associative)
infix (non associative)

and the precedence relation specifies that the precedence of the new operator is either =, (equal to), > (just greater than) or < (just less than) that of some previously defined operator.

Examples:

let ↑ *be infixR prec* > *
let ! *be suffix prec* > *
let sin *be prefix prec* > !

If these three declarations were given in this order, the resulting precedence relations would be:

prec of ↑ > prec of sin > prec of ! > prec of *

After an operator has been declared one or more *meaning definitions* are used to associate procedures with that operator. These are written:

let ⟨operator name⟩ *mean* ⟨procedure-value⟩

Example:

let + *mean proc* (*bool* a, *bool* b) (*if* a *then* ¬ b *else* b)

This definition states that "+" is to be meaningful when written between two boolean values, and is to denote "exclusive or" in that context. A call on this operator would be written: x + y, where x and y are boolean values. When the procedure is executed "x" is bound to dummy parameter "a" and "y" is bound to "b."

The term "generic" is commonly used to describe an operator that can have different meanings, depending on the modes of its operands. A particular meaning is chosen in a given context by matching the modes of the operands to the modes of the formal parameters in one of the procedures attached to the operator. This "match" is made according to the conventions described later for procedure calls.

Expressions

Program Structure

In BASEL a *program* is a *compound expression*.
A *compound expression* has a *body*, which is written between *"begin"* and *"end"* or between "("and")".
The *body* itself is a series of *blocks* separated by commas.

A *block* begins with a *declaration part* and ends with an *expression part.*

The *declaration part* is a series of declarations each terminated by a ";". This part may be omitted.

The *expression part* is a series of *clauses* separated by "*exit*" s.

A clause is a series of *expressions* separated by ";"s.

Finally, the syntax of an expression corresponds to conventional usage. Statements such as "*go to*", "*for*", "*if*", compound expression, and the empty statement are also considered to be expressions. A compound expression may be used anywhere a name is legal.

The evaluation of a compound expression

A compound expression (or a program) is evaluated by serially evaluating its blocks. At most one of these blocks may have a value, and this value, if any, is taken as the value of the compound expression.

A block is evaluated by

(1) Elaborating its declarations, in order. The scope of a declared name or operator meaning is the smallest block which contains that declaration.

(2) Evaluating the expression part until the first "*exit*" is encountered. The expressions in this part are evaluated in order except, of course, any "*go to*" commands are obeyed. The value of a block is the value, if any, of the last expression evaluated in it.

The value of each kind of expression is defined as follows:

The *go to* statement and the empty statement have no value.

A conditional is evaluated in the conventional way, its value is the value, if any, of the selected expression.

The value of a "*for*" statement is the series of values produced by the repeated evaluated of its scope.

Since the use of an operator is considered to be a call on one of the procedures associated with it, in all other cases an expression is nothing but a set of procedure calls ordered with respect to the precedence of the operations involved. Its value is the value, if any, returned by the last procedure call.

Procedure calls

A procedure call is written as a procedure-valued expression followed by a tuple-valued expression. The call is evaluated in four steps:

1. The tuple expression is evaluated resulting in the tuple of values, $[v_1, v_2, \ldots, v_n]$. Note that these values may be of any mode including *loc* modes and *proc* modes. Thus we may have calls by value, address or procedure.
2. The procedure expression is evaluated resulting in the procedure: proc $(\mathfrak{M}_1 f_1, \mathfrak{M}_2 f_2, \cdots, \mathfrak{M}_n f_n)$ (E).
3. The parameters are bound. This has the same effect as replacing the procedure call by the following compound expression:
   (*let* f_1 *be* v_1; $\cdots$; ; *let* f_n *be* v_n; E)
4. This compound expression is evaluated.

In order for a procedure call to be legal, the modes of the values v_i must "match" the modes of the dummy parameters. For the basic modes, the definition of "match" is the natural one; "match" only becomes complicated when "*union*" is involved. The precise definition follows:

A mode P of an actual parameter "matches" a mode Q of a dummy parameter (we will write $P \subset Q$)if

— P is *union* $(\mathfrak{M}_1, \cdots, \mathfrak{M}_n)$ and $\forall_i$, $\mathfrak{M}_1 \subset Q$

or Q is *union* $(\mathfrak{M}_1, \cdots, \mathfrak{M}_n)$ and $\exists_i$ such that $P \subset \mathfrak{M}_i$

or P is *int*, *bool*, *real* or *char* and Q is respectively *int*, *bool*, *real* or *char*.

or P is *loc* $\mathfrak{M}$ and Q is *loc* $\mathfrak{M}'$ with $\mathfrak{M} \subset \mathfrak{M}'$ and $\mathfrak{M}' \subset \mathfrak{M}$.

or P is *tuple* $(\mathfrak{M}_1, \cdots, \mathfrak{M}_n)$ or *struct* $(\mathfrak{M}_1 e_1, \cdots, \mathfrak{M}_n e_n)$ and Q is respectively *tuple* $(\mathfrak{M}'_1, \cdots, \mathfrak{M}'_n)$ or *struct* $(\mathfrak{M}'_1 e_1, \cdots, \mathfrak{M}'_n e_n)$ where $\forall_i$, $\mathfrak{M}_i \subset \mathfrak{M}_i'$.

or P is *proc* $(\mathfrak{M}_1, \cdots, \mathfrak{M}_n)$ $\mathfrak{M}$ and Q is *proc* $(\mathfrak{M}'_1, \cdots, \mathfrak{M}'_n)$ $\mathfrak{M}'$ where $\mathfrak{M} \subset \mathfrak{M}'$, and $\forall_i$ $\mathfrak{M}_i' \subset \mathfrak{M}_i$.

Some other aspects of BASEL

Assignment

The infix operator $\rightarrow$ is used to denote assignment. The assignment of a value v, of any mode $\mathfrak{M}$, to a variable V of mode *loc* $\mathfrak{M}$ is written $v \rightarrow V$.

The assignment $I \rightarrow J$ where I and J are both integer variables (*loc int*) is meaningless, since both I and J stand for their meanings, which are addresses. One must write: *val* $I \rightarrow J$, which explicitly fetches a value from I and stores it in J. In BASEL there is no automatic built-in mechanism to fetch the value of a variable or to do any of the other transformations performed by the "COERCION" mechanism of ALGOL–68. Of course, one can

freely define extended meanings for "→" and other operators, which would make the expression "I + 1 → I" meaningful.

Allocation

We have already mentioned that the expression a "$\mathfrak{M}$", where $\mathfrak{M}$ is a mode, causes a value of mode $\mathfrak{M}$ to be created. Thus allocation is caused when $\mathfrak{M}$ is "*loc* $\mathfrak{M}$'". If $\mathfrak{M}'$ is in turn a "*loc*" a second location is not created. This is a major difference between *loc* in BASEL and *ref* in ALGOL-68 where the declaration:

ref ref real Y = *ref real* ()

causes two locations to be allocated, the first one pointing to the second one, which in turn can store a *real*.

In addition to the block-entry-time allocation caused by declarations of variables, BASEL also permits dynamic allocation: if V is a value of mode $\mathfrak{M}$, then *aloc* V stores V in a newly created location, and returns its address as a result. This result, of course, has mode *loc* $\mathfrak{M}$.

Using a "union"

With "*union*", BASEL allows dynamically varying modes. But in order to use an object of "*union*" mode it is often necessary to check the current status of the mode. This is done with a "*when*" conditional expression, defined as follows:

when ⟨name of the object⟩ *is* $\mathfrak{M}$ *then* E_1 *else* E_2

where $\mathfrak{M}$ must "match" the declared mode $\mathfrak{M}'$ of the object.

In such expression, all uses of the object in E_1 are treated exactly as if the object's mode were $\mathfrak{M}$ rather than $\mathfrak{M}'$.

SUMMARY

BASEL derives its simplicity and power from a general and unifying notion of data objects. Data objects in BASEL comprise not only the usual integers, reals, booleans, and character strings, but procedures, pointers, and aggregates as well. Aggregates may be built out of any objects whatsoever, so that arrays of arrays and arrays of procedures are but special cases. Similarly, any object may be pointed at, so that procedure variables and aggregate variables are treated just like integer variables.

This generality in the treatment of objects greatly decreases the number of special constructs which need appear in the language. Any object may be named using a declaration, so that separate statement types are not needed to name procedures, variables, arrays, and numeric constants (the latter is not allowed in ALGOL-60). Similarly, a single parameter passing mechanism serves to allow call by value, call by address, and call by name (procedure) for all three are simply BASEL data objects. Arrays and procedures may be passed to other procedures as arguments and may be returned by them as results. Location valued procedures are likewise permitted, allowing the user for example to define new subscripting functions. The left and right sides of an assignment statement may be evaluated using exactly the same set of rules; the compiler need not distinguish between them.

To enable the wide class of data objects to be used effectively, BASEL allows the user to extend the meaning of existing operators as well as to define new ones. Thus, the user is free to define (or leave undefined) as he chooses such strange combinations as the concatenation of a real and an integer or the square root of a character.

Unlike many other generalized languages, BASEL is designed to be compiled with as little as possible left to be interpreted at run time. Efficient compilation is enabled by the mode information which the programmer specifies (as he is required to in most other higher level languages). Only when complete mode information is not known until run time is the type checking done interpretively (as indeed then it must).

We hope that BASEL has helped to illustrate that power and generality are not necessarily bought at the cost of complexity—indeed, careful generalization can provide power and at the same time make a language much simpler, both to learn and to implement.

REFERENCES

1 T E CHEATHAM JR
The introduction of definitional facilities into higher level programming languages
Second Edition Proceedings of the AFIPS Fall Joint Computer Conference San Francisco November 1966 Vol 29 Washington DC Spartan 1966 pp 623–637

2 E W DIJKSTRA
Letter to the editor
Communications of the ACM Vol 11 March 1968 pp147–148

3 A E FISCHER P JORRAND
BASEL The base language for an extensible language facility
Mass Computer Associates Inc CA–6806–1311 June 1968

4 R W FLOYD
Syntactic analysis and operator precedence
Journal of the ACM 10 July 1963 pp 316–333

5 J V GARWICK J R BELL L D KRIDER
The GPL language

Control Data Corporation Palo Alto California Programming Technical Report TER–05, 1967

6 J V GARWICK
A general purpose language (GPL)
Forsvarets Forskningsinstitut Norwegian Defence Research Establishment Intern Report S–32

7 B A GALLER A J PERLIS
A proposal for definitions in ALGOL
Communications of the ACM Vol 10 April 1967 pp 204–219

8 G F LEONARD J R GOODROE
An environment for an operating system
Proceedings of the ACM 19th National Conference Philadelphia Pennsylvania 1964 New York ACM 1964 pp E2 3–1–E2 3–11

9 P J LANDIN
The next 700 programming languages
Communications of the ACM Vol 9 March 1966 pp 157–166

10 A J PERLIS
The synthesis of algorithmic systems
First ACM Turing Lecture Journal of the ACM Vol 14 January 1967 pp 1–9

11 A J PERLIS
Private Communication

12 R M SHAPIRO S WARSHALL
A general purpose table driven compiler
Proceedings of the AFIPS Spring Joint Computer Conference Washington DC April 1964 Balitmore Spartan 1964 pp 59–65

13 R M SHAPIRO L J ZAND
A description of the compiler generator system
Massachusetts Computer Associates Inc Wakefield Mass CA–6306–0112 June 1963

14 A VAN WIJNGAARDEN B J MAILLOUX J E L PECK
A draft proposal for the algorithmic language ALGOL 68
IFIP Working Group 2 1 MR 92 January 1968

15 N WIRTH H WEBER
EULER: A generalization of ALGOL and its formal definition Part I
Communications of the ACM Vol 9 January 1966 pp 3–9
Part II
Communications of the ACM Vol 9 February 1966 pp 89–99

Associative processing for general purpose computers through the use of modified memories*

by HAROLD S. STONE

Stanford Research Institute
Menlo Park, California

INTRODUCTION

Tbe concept of the content-addressable memory has been a popular one for study in recent years,[1,2,3,4] but relatively few real systems have used content-addressable memories successfully. This has been partly for economic reasons—the cost of early designs of content-addressable memories has been very high—and partly because it is a difficult problem to embed a content-addressable memory into a processing system to increase system effectiveness for a large class of problems.

In this paper, we describe a relatively inexpensive modification to the memory access circuitry of a general purpose computer that will permit it to perform some of the operations that can be performed in a content-addressable memory. The major restriction is that the memory must be a 2½D memory.[5,6] The modification results in a new access mode to memory, one which permits a bit slice from a number of different words in memory to be accessed. Memory can be viewed as a collection of N × N arrays of bits such that an access can be made to either the i^{th} row or the i^{th} column of an array. Although this capability is somewhat limited by comparison to the capability of large content-addressed memories as they are normally conceived, it is ideally suited to that class of problems that requires both conventional and associative processing. A good example of this type of problem is the Gauss-Jordan algorithm for matrix-inversion which involves a search through a matrix for the numerically greatest element.

In the next section we further describe the functional behavior of the modified memory and illustrate its use through a series of examples. The subject of the third section is the modification of a 2½D memory in order to permit row-column access. The last section contains an evaluation of the technique and a summary.

*This research was supported by the Office of Naval Research, Information Systems Branch, under contract Nonr-4833(00) with Stanford Research Institute.

Associative processing with row-column operations

The functional behavior that is described in this section is not novel; it was first described nearly a decade ago. The reason for its resurrection is due to the ease with which it can be implemented in present technology using techniques described in the next section.

The basic idea is illustrated in Figure 1. Memory is viewed to be partitioned into multiple arrays of bits, each of size N × N, where N is a power of two. Memory accesses can be made in one of two modes, row mode or column mode. In either mode, a particular array in memory is selected by the high order bits of an effective address while the least significant $\log_2$ N bits select either a row or a column. This behavior is essentially the same as that of the horizontal-vertical computer described by Shooman.[7,8] Row selection is equivalent to word selection in a conventional computer. The column selection mode has no counterpart in conventional computers and is the mode that supports a limited form of associative processing.

There is a major difference between Shooman's vertical-horizontal processor and the idea that is developed in this paper. Shooman conceives of two separate collections of registers and processing logic to be used in his processor, one for vertical processing and one for horizontal processing. An implementation of Shooman's idea is described in Ref. 12. What we describe here uses just one collection of registers and logic. This characteristic comes about because in the modified memory all data is transferred between memory and a common data register for both row and column operations.

To be more specific, Figure 2 shows an N × N array of bits and shows the contents of the memory data register

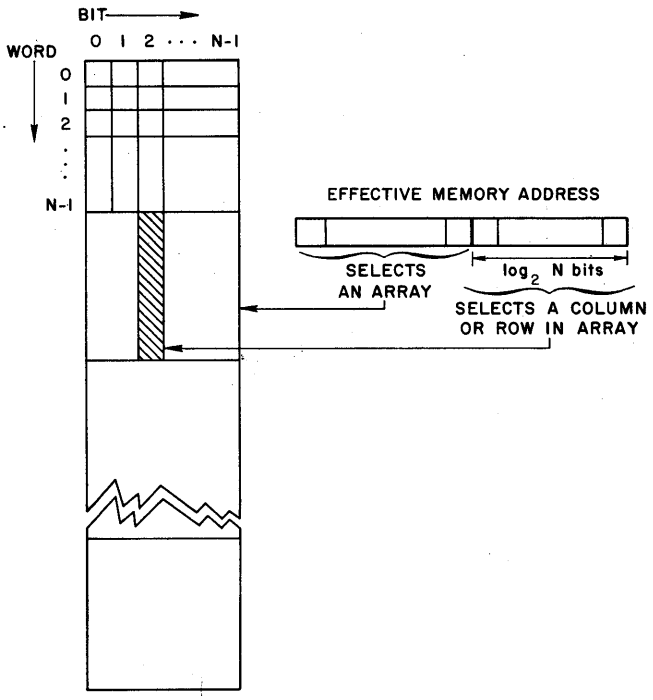


FIGURE 1—The logical organization of memory. The mode of access is determined by the instruction

both after reading the k^{th} column and after reading the the i^{th} row. Note carefully that rows are placed in the register such that the first bit, $b_{i,o}$, is at the *left* end of the register and that the first bit of a column, $b_{o,k}$, is placed at the *right* end of the register.

To illustrate how row-column accessing is useful for associative processing, we now consider several examples. The notation that is used in the examples needs a brief explanation. Symbols written in capital letters, such as X and Y, are symbolic addresses. Subscripted letters such as R_3 denote hardware registers which are part of a central processor. The symbols "$\leftarrow$" and "$\Leftarrow$" denote row and column operations respectively. Symbolic addresses that are followed by bracketed symbols such as X[I] denote indexed addresses such that X[I] is the address obtained by adding the contents of memory location I to the address of X.

Thus, we have the primitive operations:

$R_1 \leftarrow X;$	Fetch a row and place it in R_1.
$Y \leftarrow R_1;$	Store R_1 as a row at address Y.
$R_2 \Leftarrow X;$	Fetch a column addressed by X and store in R_2.
$Y \Leftarrow R_2;$	Store the contents of R_2 as a column at address Y.

Let X and Y be the base addresses of two N × N bit 0, 1 matrices in memory. Then rows of X can be stored as columns of Y by iterating the instructions below for different values of the index variables.

$$R_1 \leftarrow X[I];$$
$$Y[J] \Leftarrow R_1;$$

If I is equal to J and the loop is repeated for I = 1, 2,. . ., N, Y will contain a copy of X rotated a quarter turn. If I = N-J, then Y will contain the transpose of X where the transpose is taken about the minor diagonal.

To aid in search operations, we introduce the machine function NORMALIZE. The NORMALIZE instruction left shifts a specified register until a "1" appears in the left-most position or until N shifts are performed if there are no "1"'s in the register. A count of the number of shifts is placed in another designated register. In our symbolic notation we use the form

$$R_2 \leftarrow \text{NORMALIZE } (R_1);$$

to mean that R_1 is normalized and the shift count is placed in R^2.

As an illustration of the use of the column operation for searching consider the problem of scanning a status vector array to find a vector with a "1" in the i^{th} bit position. The pair of instructions that perform the search are

$$R_1 \Leftarrow \text{STATUS [I]};$$
$$R_2 \leftarrow \text{NORMALIZE } (R_1);$$

R_2 contains N-j where j is the index of a vector with a "1" bit in the i^{th} position. (For programming convenience, it would be wise to implement the NORMALIZE command so that the result left in R_2 would be j instead of N-j.)

Now consider the problem of searching an array for a vector that contains a particular field matching a specified pattern. The technique that we use is to perform an iterative sequence of column operations on the pattern field. In the instruction sequence below, register R_1 holds the pattern, R_2 holds columns fetched from memory, and R_3 holds a 0, 1 vector that contains a 1 bit in a bit position if the corresponding word in the array has matched the pattern on all preceding operations. The sequence is initialized so that the left-most bit of the pattern lies in the left-most bit of R_1, and R_3 is initialized to all "1"'s. "NOT" and "AND" are full register logical operators.

$R_2 \leftleftarrows X[I]$; Fetch the next column of the pattern;

IF LEFTBIT(R_1) = 0 THEN

$R_2 \leftarrow$ NOT R_2;

$R_3 \leftarrow R_3$ AND R_2; Mask out bits in R_3 that disagree with pattern for this iteration.

$R_1 \leftarrow$ LEFTSHIFT(R_1);

The sequence of instructions must be repeated for values of I ranging over the index field. After the final iteration, R_3 contains a "1" in positions that correspond to words with fields that match the pattern. A NORMALIZE command can be used to obtain the addresses of words that satisfy the search criterion.

Note that the search procedure effectively looks at N words but the number of fetches required is equal to length of the pattern. Hence, for short patterns, considerably fewer than N memory fetches will serve to search N words. Patterns that occupy a full word can be processed about equally well in row mode or in column mode.

Searches need not be made on "equality" matches. A slight modification of the instructions above is all that is required to implement a threshold search. We use one more register, R_4, that contains "1"s in positions that correspond to words with fields greater than the pattern. R_1, R_2 and R_3 are used as before. R_4 is initialized to all "0"s prior to executing the sequences below.

$R_2 \leftleftarrows X[I]$; Fetch next column of the pattern;

IF LEFTBIT(R_1) = 0 THEN

BEGIN $R_4 \leftarrow R_4$ OR (R_3 AND R_2);

$R_2 \leftarrow$ NOT R_2;

END;

$R_3 \leftarrow R_3$ AND R_2;

$R_1 \leftarrow$ LEFTSHIFT(R_1);

The statement immediately after the "BEGIN" updates the vector in R_4 such that a word is greater than the pattern if it had been greater on the previous iteration or had been equal before and is greater on the current iteration. At the close of a sequence of iterations of the program steps given above, the vectors in R_3 and R_4 uniquely identify all words that are either equal to the pattern or greater than the pattern. All of the remaining words are clearly less than the pattern. Sets that satisfy any of the relations "=", "≠", ">", "≥", "<", and "≤" can be obtained easily by simple operations on the vectors in R_3 and R_4. "Between limits" or "outside limits" searches can be done by using three registers for each limit, i.e., one register to hold the limit pattern, and two that function as counterparts of R_3 and R_4.

A number of other operations are also possible with row column accessing. Many of these are given in Shooman[7]. The examples given here and in Reference 7 should suffice to illustrate the power of idea. It is appropriate at this point to consider the implementation.

The memory modification

The memory modification can be developed from a discussion of the functional requirements of row-column processing and the constraints of conventional memory technology. We begin by examining the N × N matrix in Figure 2.

In a conventional memory, the bits in the matrix are stored so that the columns of the matrix lie on separate and distinct sense lines of the memory. Because of constraints of conventional memory technology, during any memory cycle no more than one bit per sense line can be read or written. For row operations, selection circuitry activates all bits of a specified row, each of which lies on a different sense line. The physical portion of a core memory that is threaded by a sense line will be called a *bit plane* in the following material. For both 2½D and 3D memory organizations, the entities that we call bit planes correspond to physical planes of memory stacks, but the correspondence is usually not true for 2D memories.

For column operations, the technology constraint is severely restrictive. With physical memory organized

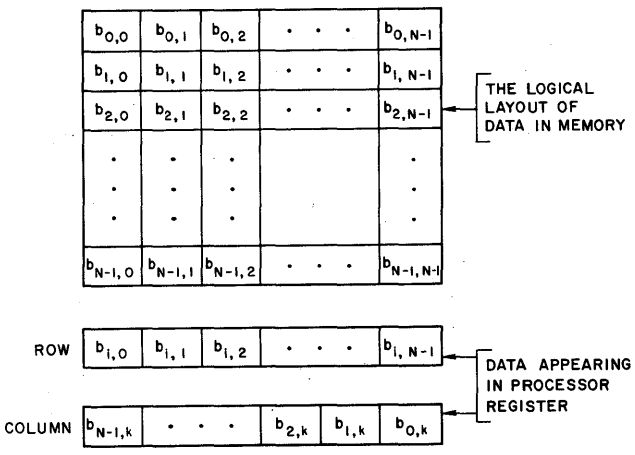


FIGURE 2—The result of fetching a row and a column from an N × N array

as shown in Figure 2, columns of arrays will lie wholly within bit planes, and thus no more than one bit per column could be accessed during any memory cycle under the assumed constraint. In order to overcome the constraints of memory technology, memory can be organized as shown in Figure 3. Each row in the array corresponds to a row in the array shown in Figure 2, but in Figure 3, the matrix is stored such that the i^{th} row is cyclically shifted to the left by i bit positions. Careful examination of the figure shows that both rows and columns of the matrix have the property that exactly one bit lies in each plane. The bits are scattered and shifted, however, so that the memory access circuitry must be constructed to take this into account.

Figure 4 shows how row and column selections must function when data is held in a skewed fashion. In

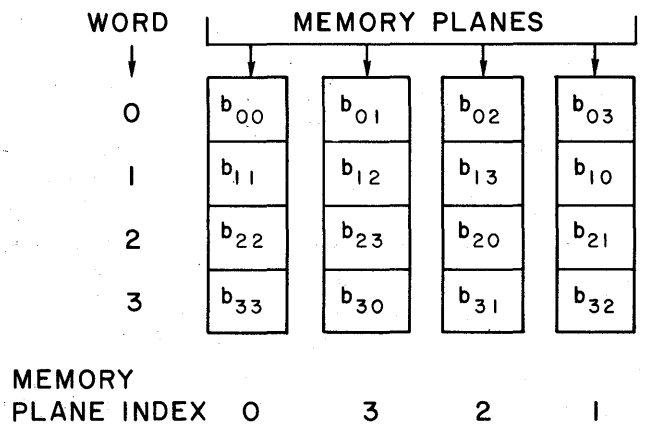


FIGURE 3—Physical layout of a 4 × 4 array in memory

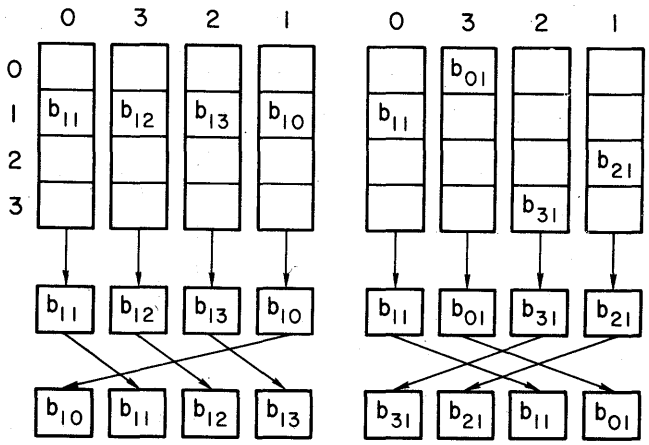


FIGURE 4—Examples of a row and column access to a 4 × 4 array

FIGURE 4a—Access to row 1 followed by a right shift of 1 bit

FIGURE 4b—Access to column 1 followed by a right shift of 2 bits

Figure 4a, the first row is read into a data register, and must be cyclically shifted right one bit to place in a standard format. In general, an operation on the i^{th} row requires a right cyclical shift of i bits after a read cycle, and a left cyclical shift of i bits before a write cycle.

Column operations are somewhat more complex. Examination of Figure 4b shows that N different words must be accessed to obtain all of the N bits in one column. Specifically, the memory planes must be able to support simultaneous selection of bits in different rows. We shall return to this point later. The planes in Figure 4 are numbered from left to right as 0, N, N-1,. . .,1. The selection of the i^{th} column is such that the j^{th} plane must select the bit in word (i+j) mod N. After access, the word must be cyclically shifted to the right by N-i-1 bits. The skewed storage technique has been used in the Illiac IV design,[9] where Illiac IV memory modules correspond to bit planes here, and full word operands in Illiac IV correspond to bits.

Thus far, the discussion has been functional in nature. Now we must take a closer look at memory technology. Among the conventional memory organizations only one can easily be adapted to permit accesses to different bits in different bit planes. This memory organization is the so-called "2½D" memory, and its organization is shown in Figure 5.

In the figure, it is shown that addresses are split into two components, X and Y, and are separately decoded. There is a single set of X drivers for all planes, but there is an individual set of Y drivers for each bit plane. The several sets of Y drivers act in unison for selection purposes in the normal mode of operation. It is precisely the multiplicity of Y drivers that permits the memory access circuitry to be modified to support the column mode of operation.

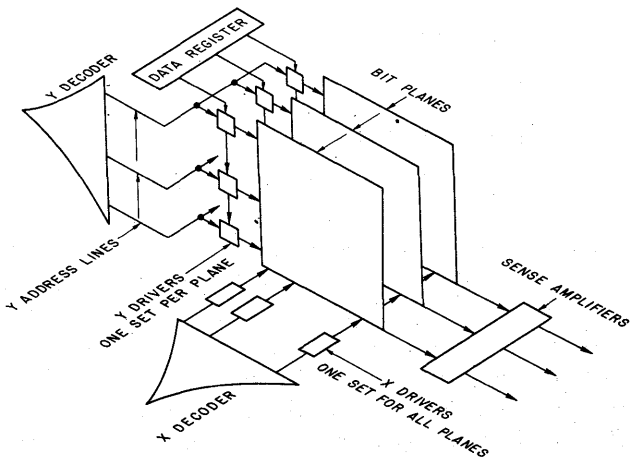


FIGURE 5—The organization of a 2½D memory system

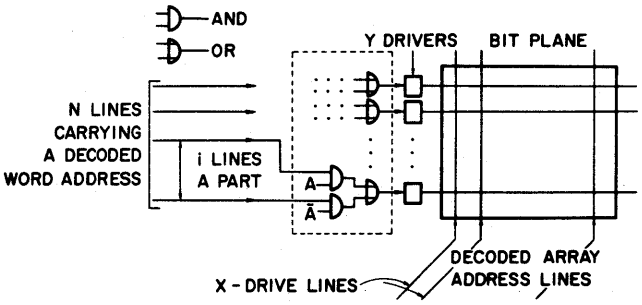


FIGURE 6—Modifications required for the i^{th} memory plane. The modifications are enclosed by the dashed rectangle

For column mode operations, it is necessary to modify the drive circuitry slightly in order to place independent control in each plane for the selection of the Y driver to be energized. The requirements are that each plane energizes either the driver corresponding to the decoded Y address (row mode) or the driver corresponding to the sum of decoded Y address and bit plane index (column mode).

Since the plane index is fixed for each plane, it is not necessary to use an adder in each plane to implement associative access mode. Consider, for example, the schematic diagram of the memory drive circuitry for a single bit plane as shown in Figure 6. In this figure, the X drive lines link all bit planes and effectively select a particular N × N bit array for interrogration. The Y drive lines select either a row or a column from the N × N array. The circuit in Figure 6 is intended to be identical to that in Figure 5 except for the two level logic circuit shown in dashed rectangle.*

The two level logic circuit has the following property. In row mode, signal A is false, and the output of the Y decoder is fed directly to corresponding drive lines in the selection matrix. Note that all planes act identically in this condition, so that each plane returns a bit from the same word address. When signal A is true, the output of the Y decoder is displaced cyclically by an amount i in the i^{th} plane. Thus, each plane reports a bit that belongs to a different word of the N × N array, and in fact reports the bit indicated by Figure 4.

The signal displacement in column mode is "end-around." That is, if the decoded address is the j^{th} address, then line (i+j) mod N will be activated in the i^{th} plane.

*Actually, X and Y axis selection circuits of 2½D memories are more complex than is indicated in Figures 5 and 6. In most implementations, the Y address lines are partitioned into two sets which are separately decoded. The decoder outputs are connected in a cross-bar arrangement and the Y lines that thread cores are located at the crosspoints. The modifications shown in Figure 6 is easily adapted to the cross-bar method of selection.

In terms of logic circuitry, the two level circuit shown in Figure 6 is the only modification that is necessary for the access circuitry.

One possible adverse effect of the two level logic circuit is a small increase in the total memory cycle time. This increase is expected to be less than one percent in magnetic storage technologies, and is likely to be more than balanced by an increase in memory effectiveness. However, in newer technologies, such as integrated circuit memories, the increase in cycle time may be somewhat larger.

We turn our attention now to the problem of cyclically shifting data prior to entry to the memory and after retrieval from memory. The two shifts are called preshifts and postshifts, respectively. To solve the shifting problem note that the amount of a preshift or a postshift can be derived from the mode and the effective memory address. Let the least significant $\log_2$ N bits of an effective address be called the *s* index. (*s* for *shift*). Then, the shift direction and amount is given by the table below.

TABLE I

Mode	*Preshift amount*	*Postshift amount*
row	s bits left	s bits right
column	N − s − 1 bits left	N − s − 1 bits right

Note that the value of N-*s*-1 can be computed by taking the 1's complement of *s* in a register with $\log_2$ N bits. Consequently, it is extremely simple to compute the shift amount because it is equal to the lower address bits in row mode or the 1's complement of these bits in column mode.

A good candidate for the shift circuit is the barrel shifter shown in Figure 7. Shifters of this type have been

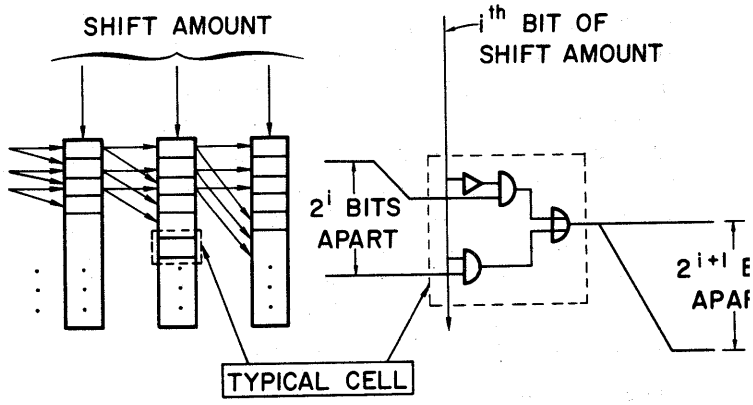


FIGURE 7—A barrel shift register with a detailed view of a typical cell

implemented in several commercial computers. A shifter with $\log_2$ N stages can cyclically shift N bit words by any amount from 0 to N-1. Each stage shifts either 0 or 2^i bits depending on the i^{th} bit in the binary representation of the shift amount. For our purposes, it is most likely that two shifters are needed, one for postshifting and one for preshifting. The two shifters should be placed on the data paths between memory and processor, and not between the memory data register and the memory. The reason for the placement on the data paths is that there is a possibility for overlapping the shift operation with other operations, whereas if the shifters were placed between data register and drivers or sense amplifiers, the effect would be to increase the memory cycle.

A good example of masking the effect of shifting time through overlap of operations occurs for write cycles. Normally, write cycles are preceded by clear cycles to clear memory in the bit positions that are to receive new data. The clear cycle can proceed as soon as an address is available. While the clear cycle is active, data can be shifted into the proper format and be ready for storage when the write cycle begins. Postshifting after read cycles will tend to increase the time required to access data but it will not increase the memory cycle time.

The barrel shifters thus are used to translate between a standard data format for the processor and one of N different storage formats. Because data within the processor are held in a standardized form, manipulation of data within the processor can be done in conventional ways. Conventional load and store commands can be issued from the central processor without regard for the fact that data might be cyclically rotated during the transfer. In essence, the physical form of data storage is completely invisible to the processor.

What has been gained, of course, with the barrel shifter, the modified memory, and the modified storage format, is the capability for performing associative operations in a general purpose computer. The net cost of the memory modification is two-barrel shift registers, and some logic in the Y address circuitry. The cost is undoubtedly a small fraction of the cost of the memory and processor. There may be some impairment of performance because of increased access time, but this is unlikely to be significant because the delays through a microelectronic barrel shift register are very small compared to the cycle time of a core memory.

SUMMARY AND CONCLUSIONS

The performance benefits to be derived from row-column addressing depend greatly on the effectiveness of column mode operations in reducing the number of memory accesses. The greatest benefit of column mode addressing is for those operations in which a small portion of a large number of words in memory must be accessed. Scaling and magnitude searches fall in this category. Take, for example, IBM System/360 long floating point words with 7-bit exponents and 56-bit mantissas. Scaling and magnitude searches over 64-word groups can be done with approximately eight accesses by accessing the exponent field of the groups in column mode. This is essentially how one would conduct the search for a pivot element in a Gauss-Jordan reduction.

Since the word-length effectively determines the number of different words that are accessed in a column operation, it is desirable to have as large a word length as possible. In present technology, it is feasible to implement memories with up to 64 or 128 bits/word. It appears that reasonably good performance improvement is possible with 64-bit words, so that a useful implementation of row-column accessing is possible within present technology.

Some analysis is required to determine the utility of column accessing for common operations such as symbol table searching and sorting. Hash-addressing used in combination with column mode access is one possible method for performing table searching. Hash-addressing normally involves a search when conflicts occur[10,11] and this search might be speeded with column access. The biggest improvement would come when tables are nearly full, at which time there is a high probability that a search will take place after the computation of the hash-code.

Ultimately, the performance improvement to be derived from the techniques that have been described in this paper are determined by the applications. While we cannot predict what benefits might accrue at this time, the fact that row-column accessing can probably be achieved for little cost within present technology indicates that the benefits of an implementation of the technique will almost certainly outweigh the cost of implementation.

REFERENCES

1 A KAPLAN
A search memory subsystem for a general purpose computer
AFIPS Proc of the 1963 FJCC Vol 24 Spartan Books Baltimore Md pp 193–200

2 R G EWING P M DAVIES
An associative processor
AFIPS Proc of the 1964 FJCC Vol 26 Spartan Books Baltimore Md pp 147–158

3 B T McKEEVER
The associative memory structure
AFIPS Proc of the 1965 FJCC Vol 27 Part 1 Spartan Books Baltimore Md pp 371–388

4 A G HANLON

Content-addressable and associative memory systems—a survey
IEEE TEC Vol EC–15 No 4 pp 509–521 August 1966

5 T J GILLIGAN
2–1/2D high speed memory systems—past present and future
IEEE TEC Vol EC–15 No 4 pp 475–485 August 1966

6 J R BROWN JR
First and second order ferrite memory core characteristics and their relationship to system performance
IEEE TEC VOL EC–15 No 4 pp 485–501 August 1966

7 W SHOOMAN
Parallel computing with vertical data
Proceedings of the EJCC Vol 18 december 1960

8 W SHOOMAN
Orthogonal computer
US Patent 3 277 449 October 4 1966

9 D L SLOTNICK
Unconventional systems
AFIPS Proc of the 1967 SJCC Thompson Book Co Washington DC pp 477–481

10 W D MAURER
An improved hash code for scatter storage
CACM Volume 11 Number 1 pp 35–38 January 1968

11 R MORRIS
Scatter storage techniques
CACM Volume 11 Number 1 pp 38–43 January 1968

12 P A HARDING M W ROLUND
A 2-1/2 D core search memory
Fall Joint Computer Conference December 1968

Addressing patterns and memory handling algorithms

by SHERRY S. SISSON* and
MICHAEL J. FLYNN**

Northwestern University
Evanston, Illinois

INTRODUCTION

One of the principal problems facing the designer of a high performance computer system is the efficient handling of memory. In arranging a memory system the designer, lacking knowledge of the programs to be run, usually selects "worst case" assumptions concerning addressing patterns. This paper is an attempt to compare a set of selected disparate programs and analyze their actual addressing traces with respect to the various memory algorithms.

In high performance systems an increase in the bandwidth* of memory must be achieved to insure an ample supply of instructions and operands to the central processing unit. More efficient program storage organizations are needed to decrease effective memory access time to a time compatible with the processor cycle time.

Three methods by which storage bandwidth can be improved are: (1) making use of a high speed immediate storage by transferring blocks of words between local and main storage as required (variations of this method have been called paging), (2) interleaving independent memory units in order to obtain a faster effective access time, and (3) using a high speed virtual memory (look-ahead unit) in the central processing unit in order to obtain an optimum instruction or data flow. This method anticipates the actual information requirements of the system.

The purpose of this project is to evaluate these system organizations by studying recorded address traces of executed programs. Blocking, interleaving, and looking ahead configurations were evaluated and compared with similar results presented in the literature. This work was done in two parts.

1. The first step was to write a simulator which chronologically recorded on tape storage demands requested by a test program as this program was being executed. This simulator was then used to generate addressing pattern tapes of three representative test programs. The memory requests of each test program were written on tape in the order used by the program and were flagged to indicate whether they were data or instruction requests.
2. The second step consisted of statistical analysis of the data. This study was done by adapting various program organizations to the data generated by Part 1. Three statistical programs were written. All of them gathered information for: a) the instruction stream** b) the data stream c) instruction and data stream combined. These statistical programs determined the frequency of run lengths, number of jumps within a look-ahead unit for various level sizes, and the interference occurring for various interleaving configurations.

The results of the data analysis can be used to give an indication as to how blocking, interleaving, and look-ahead organizations improve the speed of execution of actual programs. First, a

*Present Address: Bell Telephone Laboratories
Naperville, Illinois 60540

**Author's time supported in part by U.S. Atomic Energy Commission, Argoune National Laboratory, Argoune, Illinois

*Storage bandwidth is the retrieval rate of words from memory.[5]

**Stream is defined to be a sequence of data or instructions as seen by the machine during the execution of a program.[5]

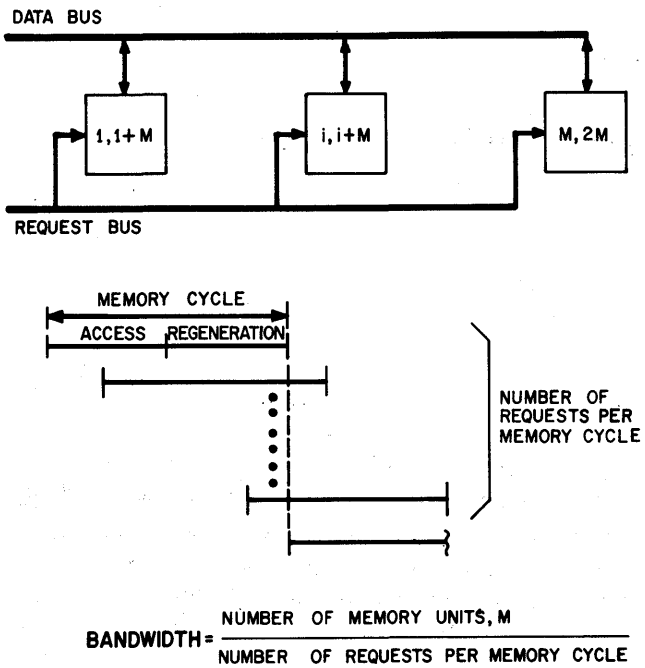


FIGURE 1

general review of program storage organizations will be presented, followed by descriptions of the simulator, test programs, and analysis programs. The final sections present the results of the adaptions of program storage organizations to the test program address traces, and conclusions.

Organization of storage (memory) systems

Concept of interleaved storage

The concept of interleaved storage is developed from the idea of using M independent memory units instead of one main memory. The words of a program and its data are distributed successively among the memories modulo M such that memory 1 contains addresses 1, 1 + M, 1 + 2M, etc., as shown in Figure 1. By increasing M for a given computer, the memory bandwidth is increased because the number of accessing conflicts is decreased. Recall that bandwidth is the service rate of the main memory. This bandwidth must process fetching of instructions and operands, storage of results, and input/output demands.

In his paper, Flores[4] studied the limiting effect of the number of memory banks on computer response time and developed an equation relating a waiting time factor to relative memory cycle time based on a queuing model. His work was based on the assumption that the demands are independent of response; therefore, the demand distribution is stationary. As a result, the probability of the arrival of a demand request in a small fixed time interval is the same as any other period. Only the worst case condition of random memory addressing was considered. In reality, programs execute demands somewhere between the extremes of sequential and random. The interleaving results of this project used actual program addressing patterns. From these results, one can get an idea as to the effect of the assumption of random storage demand. The results are presented and contrasted with Flores' predicted results.

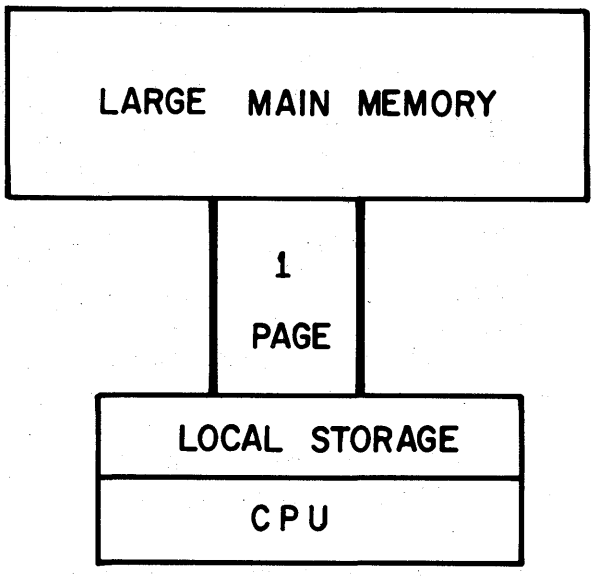


FIGURE 2

Blocking and look-ahead design

Gibson's block-oriented system

Another approach to the bandwidth problem is a block-oriented design presented by Gibson.[6] He proposed a block storage method where a request to main memory moves a block of words to a local store. Statistical results are presented on how big a block should be and what the size of local storage should be. Of course, the optimum configuration at any one time depends on the program being executed; thus, one must find an overall best configuration for the system.

Gibson's simulations results show that for a local storage size of 2048 words the number of accesses to main storage is relatively independent of the block size. This result indicates that the block size should be kept as small as possible. If large blocks are used, then the local store size

should be increased in order to get good performance compared to conventional storage. The local storage replacement algorithm used does have some effect on the usefulness of local storage, while the size of the program being executed has little effect on local storage characteristics. As a reference the paging results of this project will be compared to Gibson's block storage results.

Look-ahead systems

Look-ahead systems are employed in large-scale computers in order to increase program execution speed and to smooth fluctuations in memory demand. The Stretch[3] system included a look-ahead unit in its design for these reasons. Other systems incorporating a look-ahead feature are the IBM System 360 Model 91[1] and CDC 6600.

Look-ahead, as the name implies, is anticipatory in nature unlike the block transfer method which operates on demand. Instruction look-ahead is counter driven by the expectation of instructions to lie in a strict sequence. Data look-ahead is driven by the expected instruction stream. Figure 3 shows the types of look-ahead units that were applied to the address traces. A comparison of the performance of these look-ahead models will be presented as part of the results.

Simulator description

Simulator program description

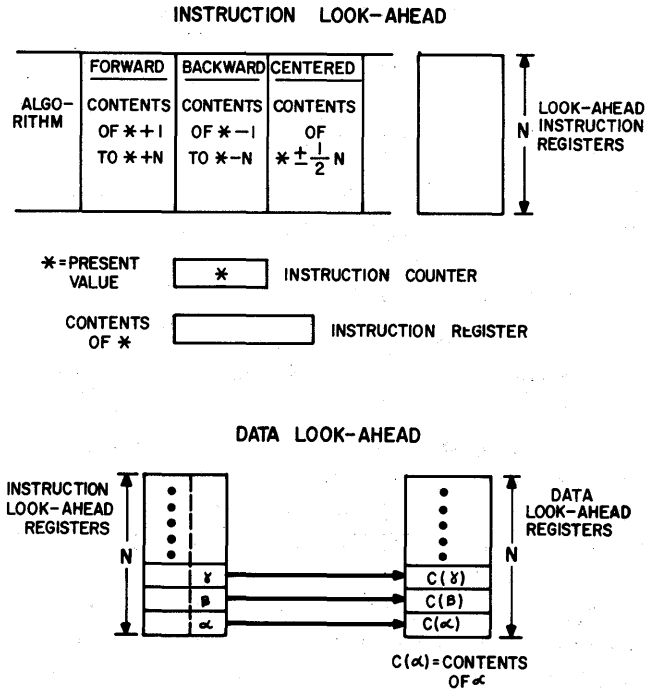


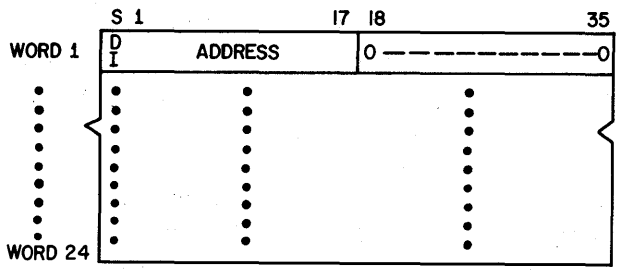
FIGURE 3

ADDRESS TAPE FORMAT

THE TAPES CONTAIN BINARY RECORDS, 24 WORDS PER RECORD.

THE ADDRESSES ARE WRITTEN ON TAPE IN ORDER OF USE DURING EXECUTION OF THE TEST PROGRAM

RECORD FORMAT:



DI = 1 IF ADDRESS IS IN DATA STREAM
= 0 IF ADDRESS IS IN INSTRUCTION STREAM

FIGURE 4

This program was written in FAP, the IBM 7094 assembly language, for the purpose of monitoring the execution of other FAP programs. The simulator keeps an instruction location counter for the test program being monitored. This location counter is used by the simulator to read the next instruction of the test program and record the location on tape. If an instruction requires an operand, the absolute address of the operand is recorded on tape following the instruction address. Any index modification of the operand is done before recording the address. One level of indirect addressing can be handled, with all absolute addresses requested during the indirect addressing process recorded on tape. These addresses are written on tape in the order requested by the test program and are flagged to indicate whether it was an instruction stream or data stream address. The format of the tapes is shown in Figure 4.

Some difficulties were encountered in writing the simulator. The execute command (XEC) was used to run a test program under control of the simulator. The major problem with this method of control is that when XEC executes a transfer, the simulator loses control to the transferred location. This problem was partly solved in the following manner: Before execution of a transfer, the simulator inserts at the transferred location a transfer to a location in the simulator program and sets a flag to indicate that this was a transfer

operation. After execution of the transfer instruction (conditional or unconditional), the simulator, replaces the original instruction into the test program. If the transfer was taken, the instruction location counter is updated to the transferred location. This method works except when control is transferred to a protected area in core during an input/output sequence.

The input/output problem was resolved by placing any test program I/O code in the simulator and placing an unconditional transfer to this code in the appropriate test program location. As the simulator is executing the test program, it looks for a transfer to I/O code before executing an instruction. When it finds one, the I/O code is executed. The simulator then reads the next instruction of the test program and continues execution. Because the simulator performs the I/O, no addresses used during I/O execution are recorded on tape.

Test program descriptions

This simulator was used to simulate and record the accessing patterns of three test programs. In an attempt to get a representative sample of patterns, programs with varied characteristics were run.

Differential equation solution

The first program was an iteration problem which solved the Van der Pol differential equation using Hamming's method. The Van der Pol equation,

$$\frac{d^2x}{dt^2} - \varepsilon(1 - x^2)\frac{dx}{dt} + x = 0$$

is used to describe non-oscillatory systems. Input parameters were given to the program for ε, h, and the first four x and y points. The next 96 points were calculated using the input parameters and Hamming's predictor-corrector equations (see Ralston[7] p. 189):

$$\text{Predictor: } y^0_{n+1} = y_{n-3} + (4h/3)(2y'_n - y'_{n-1} + 2y'_{n-2})$$

$$\text{Corrector: } y^{i+1}_{n+1} = (1/8)(9y_n - y_{n-2}) + (3h/8)(y^i{}'_{n+1} + 2y'_n - y'_{n-1})$$

Where: y_{0n+1} is the estimated initial value of y_{n+1},

y'_n is the derivative of y_n,

$y^i{}'_n$ is the ith approximation to y_{n+1}.

The corrector formula was reapplied at each point until the change was less than 0.0001. Each x,y point plus the number of times the corrector was applied at each step were printed at the end of the program. This program passed through a few distinctive loops many times as each new point was predicted and the corrector formula applied. The simulation of this program caused 83856 accessing requests to be recorded on tape.

Data processing problem

A data processing program was run as the second test program. Data cards are initially read and stored sequentially in storage. Each card contains an employee name (last name first), number, and codes indicating marital status and sex. After all the cards are read into storage, the following steps are performed.

1. The cards are sorted so that the employee numbers are in ascending order.
2. The input information is used to prefix each employee name by MR., MRS., or MISS. Each prefixed name is reversed to last name last before going to the next employee name. Several logical and shift commands were user to prefix and reverse a name and place it left adjusted in storage.
3. Any leading zeroes on the employee number are replaced with blanks.
4. The employee names, each with the proper prefix and followed by the employee's number, are printed in numerical order.

When executed by the simulator, this program requested 190608 accesses to main storage.

Machine simulation

The last test program was a program which simulated the language of a nonexistent machine on the IBM 7094. It interpreted control cards and simulated the instructions of the nonexistent machine. The main program read and interpreted the control cards which included such commands as IDMP, LOADER, PRINT, SNAP, and START. After the START control card was read, the program transferred to a decoding routine which read and decoded the instructions of the nonexistent machine. Each instruction was simulated by

a command subroutine. A very small program, consisting of a few control cards and instructions, was written in this new language for the test program to simulate. The test program requested 1392 addresses when it was executed, enough to get an idea of the characteristics of this program. There was more input/output in this program than in the other two test programs. Recall that memory requests in an input/output section of code are not recorded by the simulator, thus a portion of the test program access pattern is not considered in the program analysis.

Analysis program descriptions

When executed, the following programs made three passes of an address tape, one for each of the three addressing streams.

Run length program

The first program written recorded run length information. A run length is defined as the number of addresses increasing in value by one over the previous address in an addressing pattern. When an address breaks the sequential pattern, this address is taken as the start of a new run length. A table is printed showing the run lengths that occurred in the program and the number of times each run length occurred. Also recorded are the total number of instruction, data, and combined stream references used by the test program. This information can give an idea of the size and number of loops occurring in a program, and of the sequential nature of a program.

Blocking and look-ahead program

Blocking and look-ahead characteristics were obtained by the second analysis program. The program as written with the assumption that only one page can reside in local memory (readily available to the processor) at a time. For a given page size, the number of new pages requested by a test program is printed. Along with the page count, the number of addresses used by a test program assuming single word access is printed. It should be noted that the blocking analysis program considered accesses to memory, not distinguishing between fetches and stores. The look-ahead section looked for three situations for a given number of levels (N).

1. It considered that N levels followed the current address (forward look-ahead).
2. N was considered to be the number of sequential addresses preceding the current address (backward look-ahead).
3. The levels were considered to be centered with N/2 addresses preceding the current address and N/2 following.

The analysis program printed the total number of jumps occurring within N for each situation (forward, backward, and centered). The blocking and look-ahead data were collected first on the combined stream and then on the instruction stream and data stream separately.

Interleaved storage program

The last program tabulated information on interleaved storage. It required three input parameters, relative memory cycle time, relative processor cycle time, and the number of memory banks used. Several items were tabulated for each set of input parameters, including the following.

1 A count was made of the *number of interferences* that occurred. An interference occurs when service is requested of a memory bank which is still busy with a previous request.
2. The *total wait time* due to memory bank interference was recorded; the wait time being the total time in cpu cycles that the cpu had to wait because the requested memory bank was busy.
3. The *total processing time* was tabulated. This time included one cpu cycle for each memory access plus the wait time.
4. The *number of accesses* to memory was printed.
5. The *occupancy ratio* (busy ratio divided by the number of banks) was calculated. The busy ratio is the ratio of memory cycle time to processor request rate.

Increasing the number of banks for a given relative memory cycle time proportionately decreased the occupancy ratio. Results were obtained on all three addressing streams. When the instruction stream was used alone, the data stream and any of its effects on processing time were ignored. Therefore, the results are worst case interference figures for the given relative processor and memory cycle times.

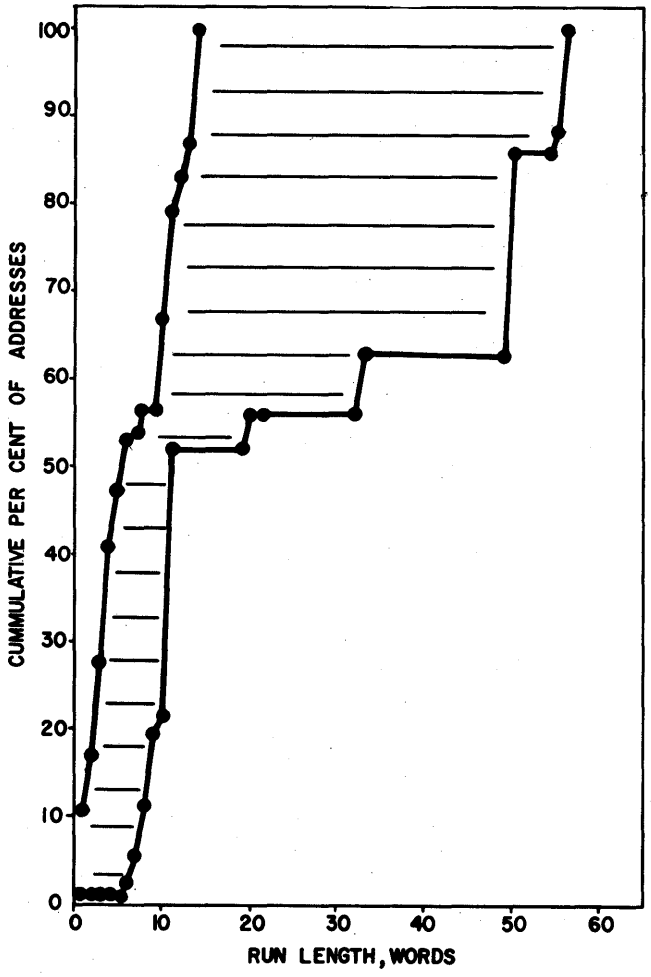


FIGURE 5

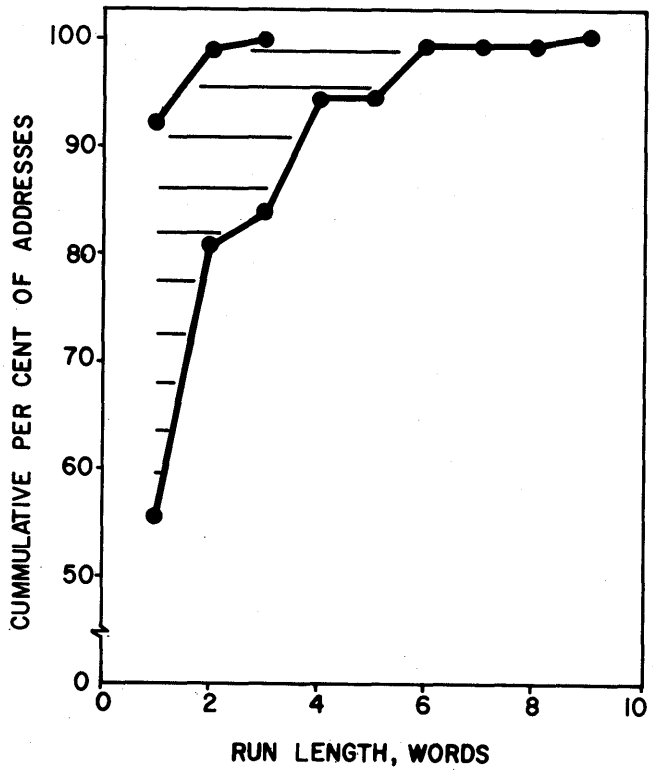


FIGURE 6

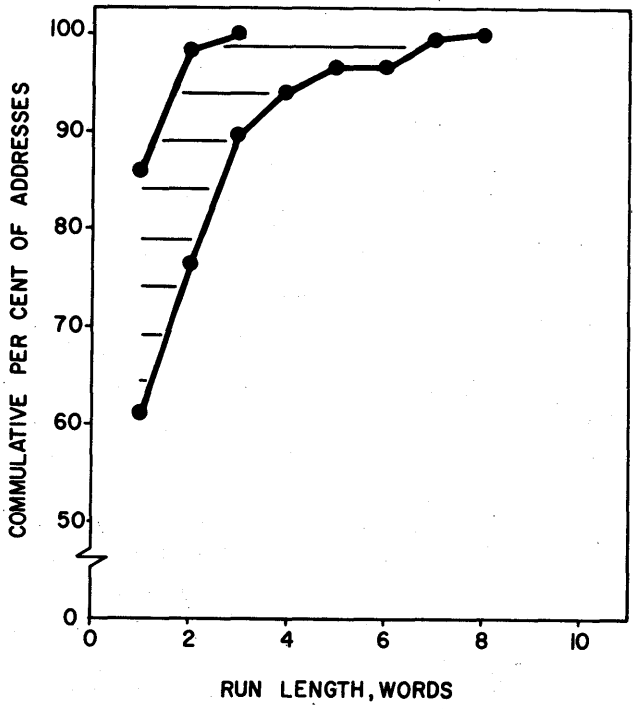


FIGURE 7

Results

Run length results

The run length results in Figures 5 - 7 show the sequential nature of the three addressing streams fore each test program. They are a plot of the percentage of addresses which are of a given run length size or less. The figures show the worst and best case range of results. From observation of these figures, one can see that the instruction stream run lengths are significantly longer than either the data or combined stream run lengths. Because a run length is a number of sequential addresses, this longer run length of the instruction stream indicates that the instruction stream is more sequential than either the data or combined streams. The difference is more marked in program 1 where the mean instruction stream run length is 14.4 words while the data and combined stream mean run lengths are just over 1 word.

The wide range of instruction stream run lengths generated by program 1 and 2 is reflected by their large standard deviations of 13.7 and 8.8, respectively.

MEAN RUN LENGTH, WORDS

	COMB	INST	DATA
TEST PROGRAM 1	1.08	14.44	1.05
TEST PROGRAM 2	1.09	4.52	1.04
TEST PROGRAM 3	1.33	3.78	1.38

RUN LENGTH VARIANCE
(STANDARD DEVIATION IN PARENTHESES)

	COMB	INST	DATA
TEST PROGRAM 1	0.08(0.29)	187.5(13.7)	0.07(0.27)
TEST PROGRAM 2	0.13(0.37)	77.1(8.8)	0.09(0.30)
TEST PROGRAM 3	0.73(0.86)	13.7(3.7)	0.39(0.63)

TABLE 1

For all three programs, the data run lengths are short with means ranging from only 1.04 to 1.40 words. This almost random referencing of data is primarily due to the initial data layout. For example, program 1 sequenced through large tables; but in between referencing a location in the table, constants and temporary storage locations were referenced. Perhaps the single data stream should be considered as two streams, one referencing the tables, and another referencing constants and temporary storage locations. The short combined stream runs strongly reflect the data characteristics.

In general, one can observe from these figures that the instruction stream is significantly more sequential than either the data or combined stream. This difference gives an indication that the instruction and data streams should be treated as separate memory areas in a computer organization.

Look-ahead results

Another way to observe the sequential nature of program accessing patterns is to investigate its branching characteristics. A look-ahead unit anticipates the sequential nature of the instruction stream alone. An instruction operand is obtained from memory as a part of the look-ahead process and is saved along with the instruction at a single level of the unit. One problem with a look-ahead unit is that its effectiveness is diminished when a program branches to a location outside the unit.

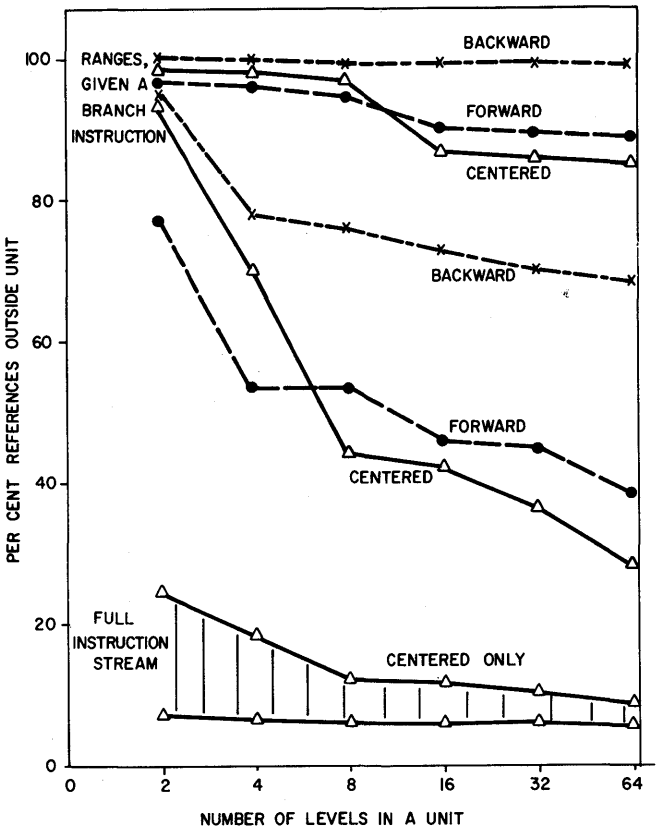


FIGURE 8

Results in Figure 8 depict the probability of an instruction reference to a point outside a look-ahead unit for a given size. The best and worst case results for each look-ahead model are plotted. Results were also obtained on instruction stream

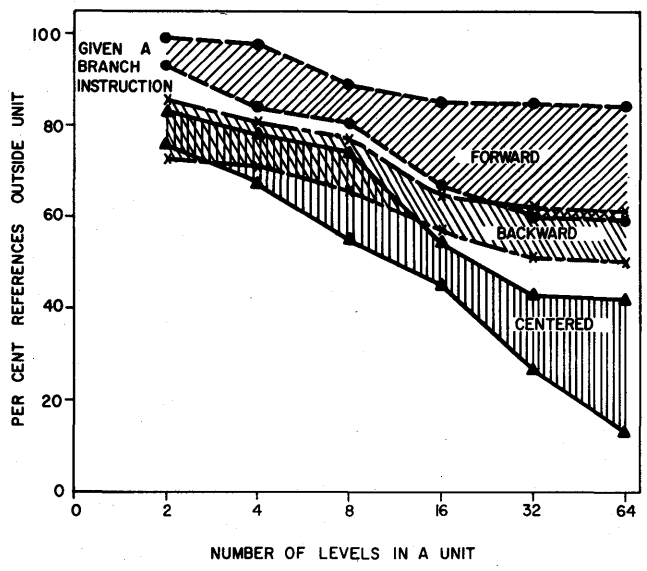


FIGURE 9

branching to instructions just previously executed (backward look-ahead) and on branching to instructions in both the forward and backward direction. For all test programs the three different look-ahead configurations are plotted on the same figure so that the configurations may be compared. Similar results are also plotted for the data stream in Figure 9.

Instruction stream

Test program 1 demonstrates the best look-ahead characteristics because of its highly sequential accessing pattern. For any of the look-ahead configurations, program 1 had a probability of less than 0.07 of not finding any given instruction within the look-ahead. However, this program was also the least sensitive to look-ahead size with the probability of not finding a given instruction within the unit decreasing by only 0.01 up through the 64 level unit size.

The most significant improvement in the probability of finding a reference within a unit came when expanding from 2 to 4 levels for the forward and backward cases. For example, for the best forward unit results the probability of a branch instruction transferring out of 4 forward levels is 0.54, down 0.23 from 2 levels. The same probability at 4 levels in the backward unit was somewhat higher at 0.78, down 0.17 from 2 levels.

Each level plotted on the centered look-ahead reflects the combination of the forward and backward results of the next lower level. The branching improvement decreases rapidly for the first 8 levels, then decreases more slowly for the rest of the range.

Branching conclusions

Based on the results in Figure 8, a centered unit would be most desirable, but this configuration may also be the most difficult to implement. A unit of at least 8 levels is needed before the centered look-ahead performs any better than the forward configuration. If a smaller unit were required, then the forward look-ahead would provide the best performance. A timing simulation study was made for the Stretch computer to get information as to the effect of a forward look-ahead unit on relative program execution speed.[8] The Stretch results indicate that the biggest improvement in speed came when expanding to 4 levels. The same characteristic is displayed by the test program branching patterns. This comparison indicates that program branching behavior is a direct indication of look-ahead applicability.

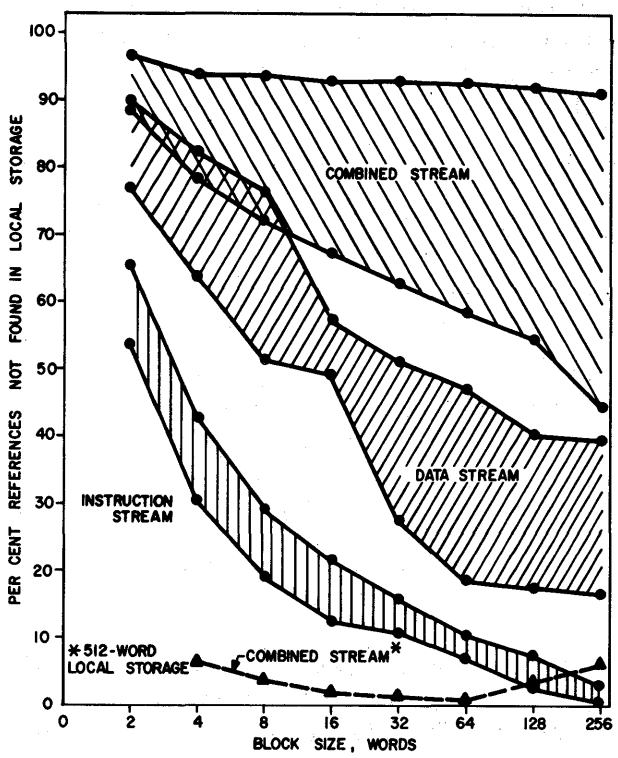


FIGURE 10

Blocking results

Another method for improving program execution speed is by use of blocking techniques. The first results were obtained by the paging analysis program with the assumption of only a one-page local storage. From previous observations of the run length curves, one would expect paging to be most applicable to the more sequential instruction stream. Figure 10 reveals, as expected, that the instruction stream always has a smaller probability of not finding a word in local storage than either of the other streams. It takes only a 4-word page to bring the instruction stream probability down to less than 0.5 while the data stream needs a 32-word page to achieve the same performance. A 64-word page brings the probability of not finding an instruction reference down to less than 0.1. However, at this large size, the number of references made to local storage represents a small fraction of the number of words transferred to local storage. A large number of words are now being transferred from main memory to local

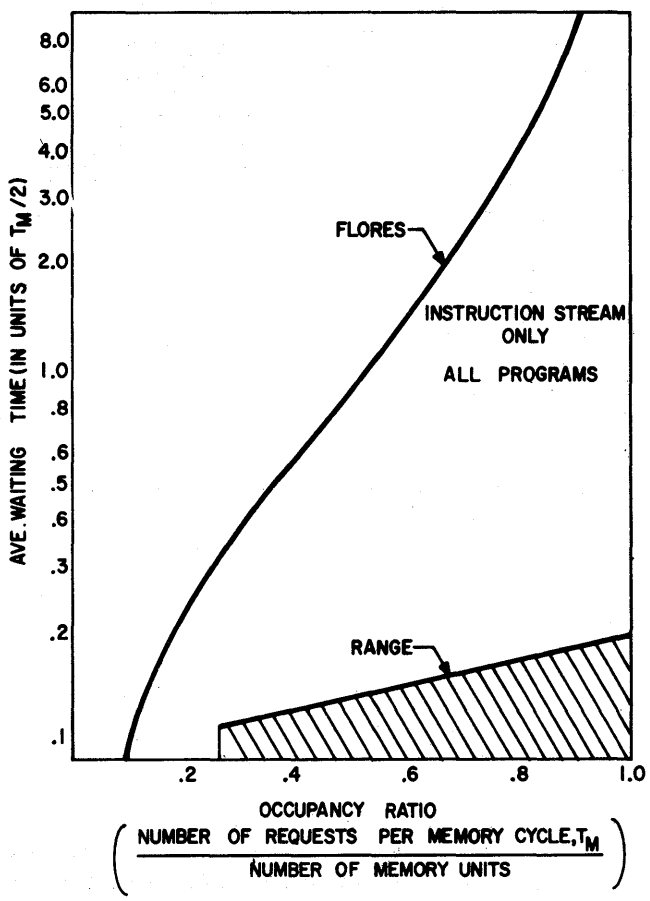


FIGURE 11

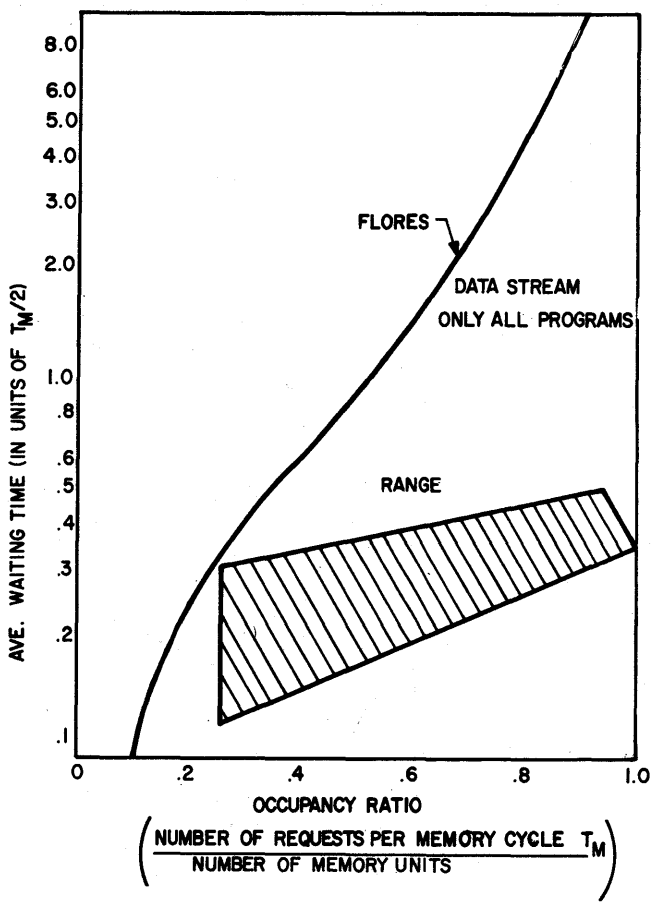


FIGURE 12

storage that are of no use to the program. Observation of Figure 10 also points out that program size is not a determining factor of paging characteristics.

When only one page is saved in local storage, the combined stream demonstrates very poor blocking characteristics. Gibson [6] states that, theoretically, local storage should be able to save at least four pages, one to accommodate the instruction stream, two for source data operand areas, and one for the output data operand area. Gibson made a paging study using a 2048-word local store and the combined addressing stream.

The effect of a larger local storage on test program 3 can be observed in Figure 10.[8] A 512-word local storage, administered on a first page in first page out basis, was used to obtain this data. With a 16-word page the probability of not finding a reference in local storage decreases from 0.68 with a one-page storage (16 words) to 0.02 with a 512-word storage, an improvement of well over one order of magnitude. The number of references outside local storage is less than the references required by the central processing unit for smaller page sizes, similar to the 2048-word local storage characteristics.

Interleaved storage results

The last type of program organization adapted to the addressing patterns was the interleaved storage model. Figures 11, 12, 13 are plots of the average waiting time, expressed as a multiple of memory access time (Memory cycle, T_M divided by two), versus occupancy ratio. These figures show the best to worst case range of results. The waiting time is the average additional amount of time, above the access time, that is required to process a request due to conflicting requests. All three addressing streams are shown, along with the results of Flores.[4]

The data and combined stream delay times are always less than half of T_M but longer than the instruction delays. The curves show that the combined stream is no more memory limited than the data stream alone. The randomness of the

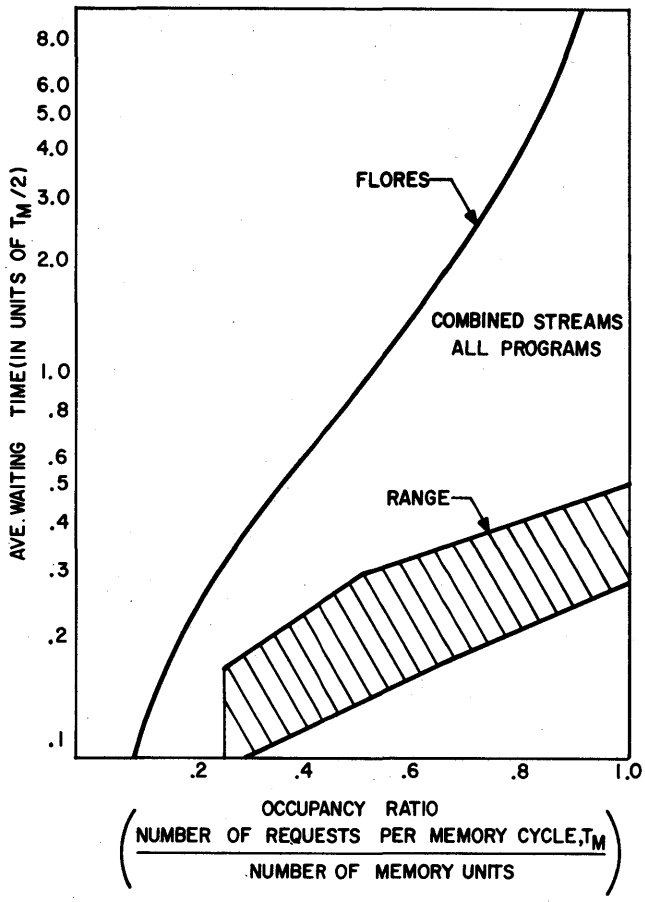


FIGURE 13

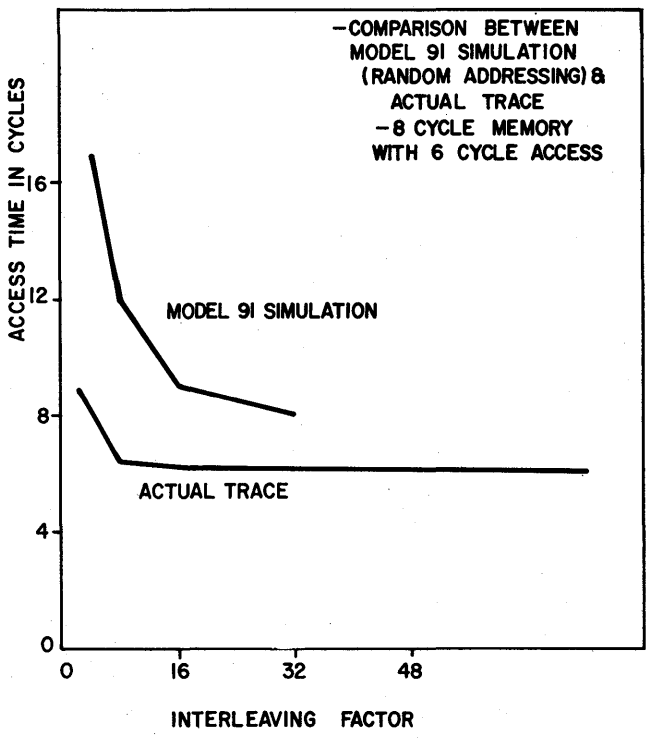


FIGURE 4

data stream causes the greater blocking times. One way to make the data stream wait times more compatible with those of the instruction stream would be to separate the data stream into three operand areas as enumerated in the previous section.

Flores' curve, shown in the figures for comparison purposes, was derived from an open loop queuing model assuming a uniform random accessing pattern. It should be used only as a limiting factor in considering program execution speeds and not for determining the optimum waiting time for a system. Comparison of the test program addressing patterns with Flores' random accessing pattern reveals that fewer banks are needed to obtain a reasonably low memory waiting time for the test program patterns. With an occupancy ratio of 0.5, the average waiting time is less than one-half of the wait with random accessing. Increasing the occupancy ratio to 1.0 causes an insignificant increase in the waiting time of the test program addressing patterns as compared to Flores' curve. It should be noted that the test program waiting times are slightly optimistic because interferences generated by input/output sequences were not included in the accessing pattern.

The relative improvement of the accessing rate that occurs as interleaving and memory speed are improved can be demonstrated in another manner. Figure 14 depicts the average access time in cpu cycles as a function of interleaving. Notice that the amount of interleaving yields an exponentially diminishing improvement. Similar results were obtained from a simulation of the IBM System 360 Model 91 storage system.[2] The Model 91 simulation also used a random addressing pattern, which resulted in longer average access times than those of the test programs.

CONCLUSION

The objective in the design of a general-purpose computer is a system which executes programs as fast as possible, and is technically and economically feasible. The analysis of dynamic address traces provides a quantitative measure of system performance which can be used to guide computer system design. This project examined program accessing patterns, looking for characteristics

which would lead to more efficient storage utilization by programs. Final evaluation of the optimum method of organization depends on hardware costs for the various configurations which were not considered in this study.

The major methods of organization are blocking, interleaved storage, and look-ahead. Results indicate that separation of the instruction and data streams improves program performance. The results of a one-page local storage show that the test program performance is much improved with a page size of 4 or more words. However, the results presented using a larger local storage reveal that performance is improved by more than an order of magnitude when more than one page is saved in local storage. Interleaved storage performance is good with the combined stream, but could be significantly improved by using separate streams. Reasonably low access waiting times are produced by using enough memory units to offset the ratio of memory to processor cycle times.

A look-ahead unit would be more applicable to an interleaved storage organization than an organization with local storage. A centered look-ahead handles program branching more efficiently than the other configurations at a size of 8 or more levels. If a smaller look-ahead is to be considered for a system, a forward look-ahead would provide the best performance.

REFERENCES

1 D W ANDERSON F J SPARACIO R M TANASULO
The Model 91: Machine philosophy and instruction handling
IBM J Res and Dev January 1967

2 L J BOLAND G D GRANITO A M MARCOTTE B U MESSINA J W SMITH
The model 91 storage system
IBM J Res and Dev January 1967

3 W BUCHHOLZ Ed
Planning a computer system
New York McGraw-Hill 1962

4 I FLORES
Derivation of a waiting time factor for a multiple band memory
J ACM Vol 11 pp 265–282 July 1964

5 M J FLYNN
Very high-speed computing systems
Proc IEEE Dec 1966 pp 1901–1909

6 D GIBSON
Considerations in block-oriented systems design
AFIPS Conf Proc Vol 30 pp 75–80

7 A RALSTON
A first course in numerical analysis
New York McGraw-Hill 1965

8 S SISSON
Statistical analysis of computer accessing systems
Thesis for the MSEE degree Electrical Engr Dept Northwestern Univ Evanston Illinois June 1968

Design of a 100-nanosecond read-cycle NDRO plated-wire memory

by TAKASHI ISHIDATE

Nippon Electric Co., Ltd.,
Kawasaki, Japan

INTRODUCTION

Plated-wire memories are now attaining a promising position in main memories. UNIVAC claims that, in the non-destructive readout (NDRO), the cost of peripheral circuit can be reduced, since peripheral circuits are useful enough to maintain multiple words. However, the NDRO is most effective only for slower memories with a comparatively small number of interface bits. The main memories such as 72-bit-per-word 100-nsec read-cycle memory system cannot be improved by the use of the NDRO as recommended by UNIVAC.

This paper deals with the best use of the NDRO techniques in the high speed operation of memory system.

Reasons for adopting NDRO

The advantage and disadvantage of NDRO operation of plated-wire memory are as follows.

Advantages:

(1) The NDRO memory is more reliable as compared with the DRO memory against temporary errors. In the DRO memory, stored information will be definitely destroyed, if the readout signal is detected in the wrong sense or if there is any malfunction in the recirculation loop.
(2) The read cycle time is short. There is neither rewrite time nor recovery time of digit noises in the NDRO memory.
(3) If the word current for writing is compatible with that for reading, we can store information exclusively into desired bits of a word, while reading the remaining bits. This gives rise to efficient use of the memory.
(4) From the similarity between NDRO memory and electronically changeable read-only memory (ROM), most of the NDRO techinques can be uti lized in ROM.

Disadvantages:

(1) The output signal level obtained by NDRO mode is generally lower than that obtained by DRO mode, because, in the NDRO operation, the readout word field is kept considerably below the amplitude which brings the magnetic vectors along the hard axis of thin magnetic films. Low output signals make the sense circuit more complicated.
(2) When the thin film is operated by NDRO mode, the domain wall is apt to creep, and the yield of good plated wires fall, thus increasing the cost of memory planes.
(3) The read cycle time of NDRO is short. But the time required to write information increases due to the limited word current for writing.
(4) The difference between read cycle time and write cycle time is not favorable to the control unit of a processor to access the memory.

Design objectives

The following items were selected for the design, anticipating the demand of performance characteristics in the 1970's.

(1) Unit capacity of the memory is 16,384 words of 72 bits each.
(2) The memory must have a read cycle time of 100 nanoseconds, a read access time of 70 nanoseconds, and a write cycle time of 200 nanoseconds.
(3) Input and output signal levels must be compatible with those of CMLs.

Timing

The time should be determined for each functional block before the detail system design is given. The timing waveforms are shown in Figure 1.

Read operation is repeated at a rate of 10 M cycles

per second. The time delay for the address decoding and for the propagation of word current from the current source to the memory plane is estimated as 30 nanoseconds.

The peak of readout signal appears 10 nanoseconds after the mid-point of the rise of the word current.

The maximum propagation time of the signal in the memory plane is assumed to be 15 nanoseconds. The delayed signal is shown with broken lines.

Each of the circuit delays admitted, in the sense amplifier, in the polarity detecting strobe circuit, and in the output buffer, is 5 nanoseconds.

For write operation, 200 nanoseconds are allotted. The word current has two timings. The timings are called "Word Phase I" and "Word Phase II."

Digit current consists of a pair of pulses, one positive and the other negative. The first timing is called "Digit Phase I" and the second "Digit Phase II."

The digit current timings shown by broken lines mean that the digit driver advances the digit current in the same amount as the propagation delay in the memory plane.

This technique insures the precise time relation of the word and digit current for any address.

System design

The system design should be accomplished taking engineering feasibility and cost into consideration.

Some of the possible combinations of word and bit

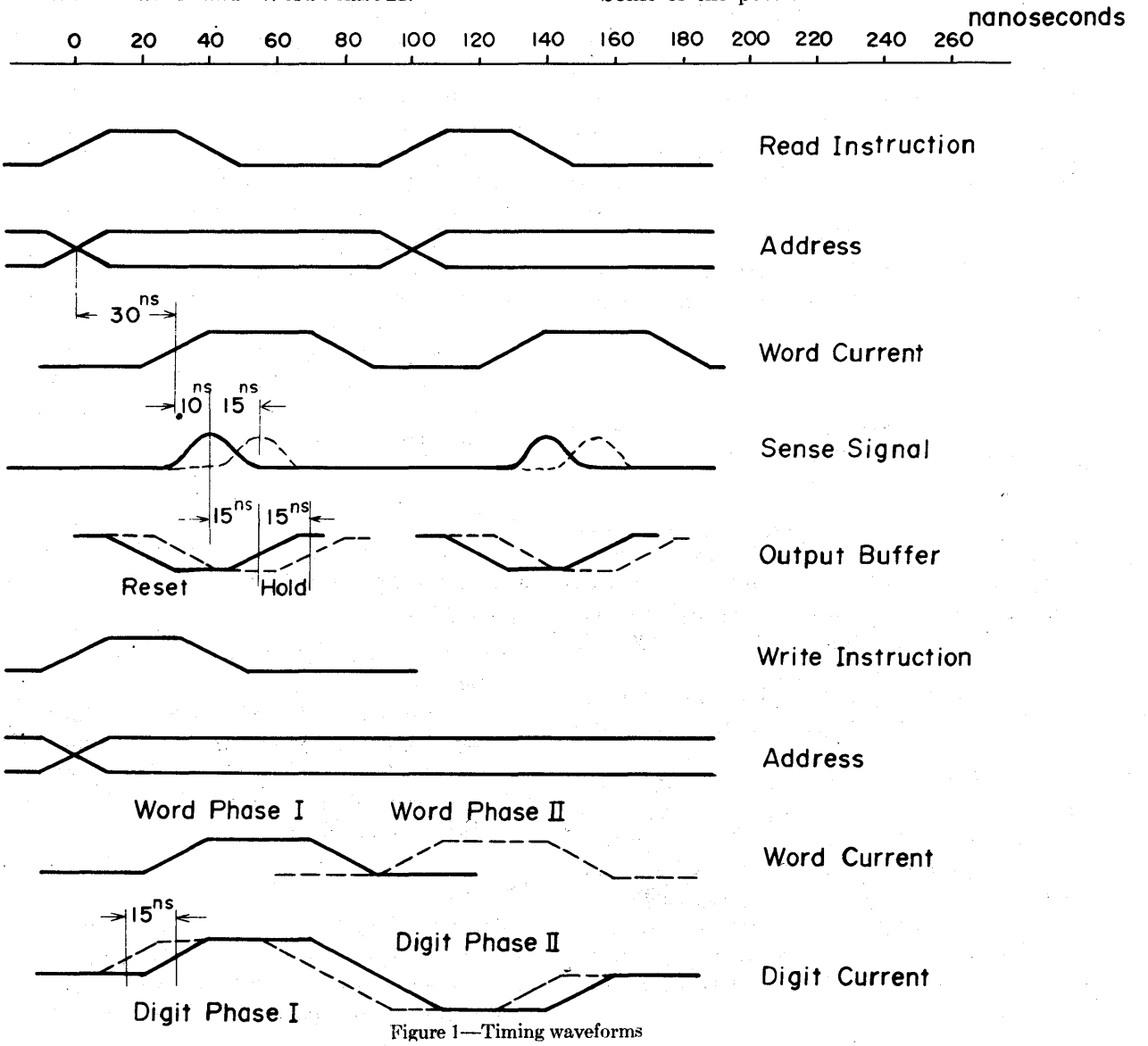


Figure 1—Timing waveforms

sizes are shown in Figure 2. If the costs of pheripheral circuit are as in Table 1, the total peripheral costs become as follows:

[A] $17,100
[B] $13,400
[C] $9,600

Word Driver		Sense Amp./Digit Driver		Matrix-Switch	
72 bits	$1	16 K words	$10	.....	.....
144 bits	$1.5	8 K words	$7.5	8 K words	$1.5
288 bits	$2	4 K words	$5	4 K words	$1

TABLE 1—Assumed costs per unit of peripheral circuits

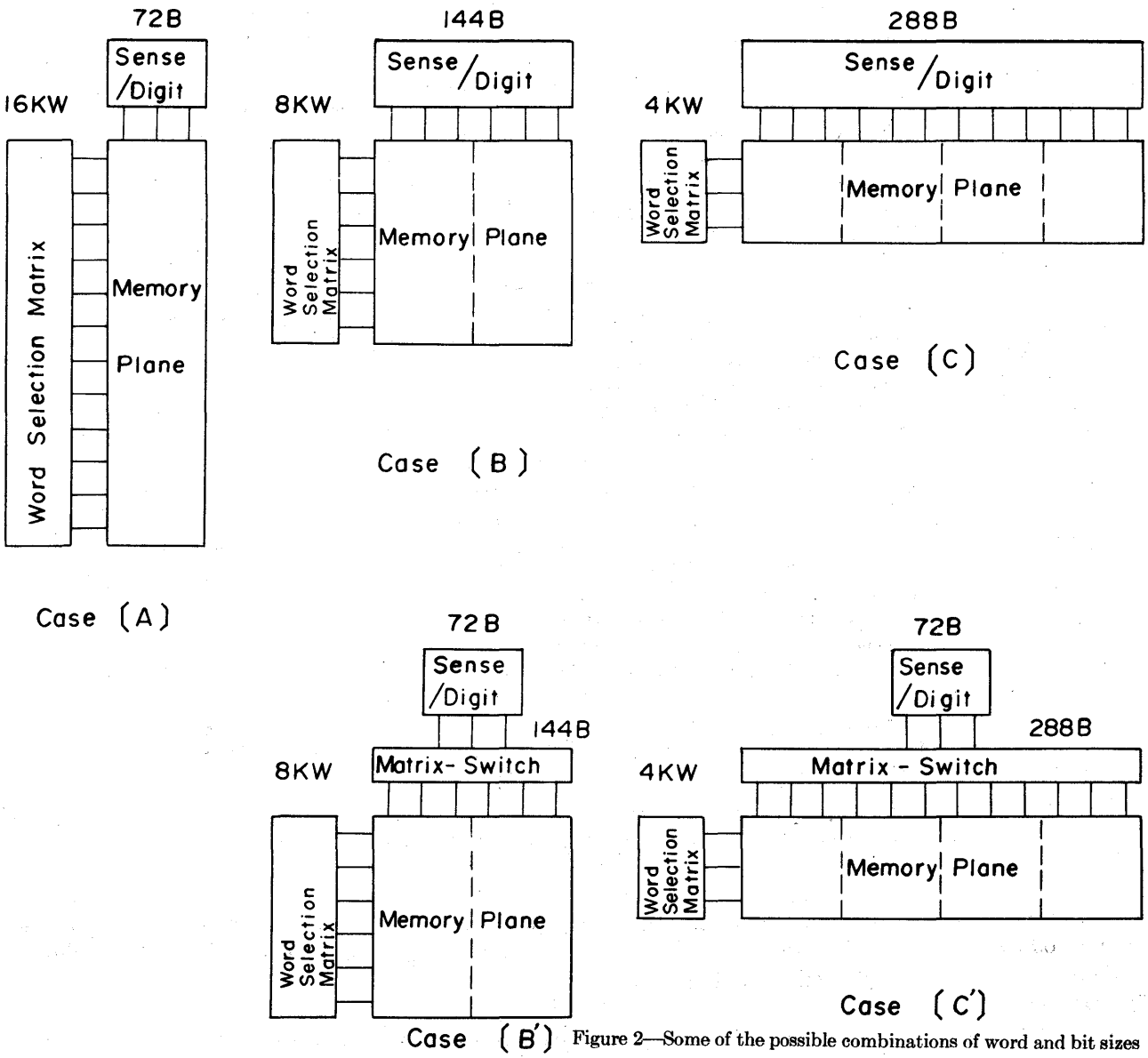


Figure 2—Some of the possible combinations of word and bit sizes

If the selection-matrix as used by UNIVAC is introduced into [B] and [C], the total costs are as follows:

[B'] $13,000

[C'] $ 8,800

Thus we see that the selection-matrix method is not very effective. The cost down, less than ten percent, will not be sufficient to compensate the considerable circuit delay in the matrix. This method could advantageously be used with the byte-by-byte machines. Thus, we chose the case [C].

1 Intersection/bit or 2 Intersections/bit

Information is stored in a restricted area of the plated wire, which is surrounded by word line. Intersection of plated wire and word line is sufficient to store one bit of information. However, we have two-intersections-per-bit-system in the plated-wire memory, reminding us of two-cores-per-bit system in the ferrite core memory.

Features of one intersection-per-bit system and two intersections-per-bit system are compared in Table 2.

Items	1 Intersection/Bit System	2 Intersections/Bit System
Cross-talk of adjacent lines	(1) By making the distance between the wire and ground plane as small as possible, a bit pitch of 1 mm is feasible. (2) If the terminating resistors are connected between adjacent lines cross-talks can be made considerably small.	(1) It is possible to make the wire pitch smaller. However if the space occupied by a pair of bit lines is considered this system has not always a higher density. (2) The comparatively wide space left between the word line and the ground plane increases the effectiveness of word field to the cylindrical film.
Capacitive Noise from Word lines	To reject common mode noises, conventional transformers or baluns must be used.	The common mode noise on the word line is eliminated at the differential sense amplifier.
The Effect of Bit Currents on the Sense Amplifier	Some of the bit connection system requires common mode rejection means such as transformers, because the maximum common mode voltage swing to the input of conventional linear IC's is about 1 volt which is less than the voltage generally occurring.	As a matter of principle, no voltage change at the sense amplifier occurs.
Memory Cell Cost	The cost per bit of a memory cell is low.	The cost is about twice as much as that in the one-intersection-per-bit system.
Length of the Word Line	(1) The length of a word line is short so that the overall size of the plane is small. (2) Since the back voltage of the word line is smaller with regard to the same word current, the design of an IC word driver is easier.	Difficulties arise in the drive circuit.
Readout Signal	Wiring and stacking of the plane must be done carefully in order to avoid the noise from outside the plane.	The signal-to-noise ratio is increased because two readout signals from each intersection are added.

TABLE 2.—Comparison of 1 intersection/bit system and 2 intersections/bit system.

The smaller the back voltage of the word line, the easier the introduction of the IC word driver, and one-intersection-per-bit system is adopted for a 100 nsec read-cycle memory.

The connection of digit lines is shown in Figure 3. This connection prevents the common mode voltage of a digit driver from being applied to the sense amplifier.

Bipolar digit drive system

The use of bipolar digit current to write the informa-

tion is popular in NDRO memory systems.[1,2]

The advantages of the bipolar drive are as follows:

(1) A larger output voltage can be obtained, because even if the information is stored with a large digit drive current, the information in the other location is hardly destroyed.
(2) As a matter of principle, no dc level shift occurs on the digit line. This allows the dc coupling of a sense amplifier to digit lines.
(3) A transformer can be used in the digit drive circuit. It can make the circuit simple and reduces the cost.
(4) As mentioned in (1), the threshold of digit current beyond which the creep of walls happens can be increased. The increase is favorable to improve the yield of usable plated wires.

It was found that the bipolar digit drive system has following merits.

If two phases of timing are provided for the word current, one for the even planes and the other for the odd planes, it will become unnecessary to invert the polarity of readout signals or digit currents according to the plane of the selected word.

As shown in Figure 1, the information is written by the coincidence of "World Phase I and "Digit Phase I for the even planes. For odd planes, the information is written by "Word Phase II" and "Digit Phase II."

Since the polarities of the digit current in "Digit Phase I" and "Digit Phase II" are always opposite, the odd planes and even planes are written automatically in the opposite sense by the same digit current pair.

A schematic diagram of the digit logic which controls digit driver is given in Figure 4.

Control of word timing or digit timing

When the propagtion delay of signals in the memory plane is not negligible compared with the duration of readout waveforms, it is not efficient to add up two waveforms by a differential amplifier; the one directly travels to one of the input terminals of the amplifier; the other travels once to the opposite end of the memory plane and then returns to the other input terminal of the amplifier through a balancing digit line.

The time difference of two waveforms to be added is so significant that the peak amplitude of the resultant waveform is greatly influenced by the position of signal source in the memory plane.

However, if the direct waveform is used exclusively as a signal, the change in the peak amplitude of the signal with the selected address will be slight and dependent only on the attenuation of signals in the digit line. It should be noted that the delay of signal waveforms becomes remarkable.

Methods to compensate the propagation delay of signals by adjusting time relations of pulses have been discussed in many papers. There are two main methods. One is achieved by controlling the timing of word currents against a fixed timing of digit circuitry,[4] and the other employs strobe pulses and digit drive currents with controlled timings against a constant word timing.[5,6]

In the word control system, the delay time of the readout signal, spent after the address information has been given, is always adjusted to the worst case.

In the digit control system, the readout signal is obtained as soon as possible. The digit control system will

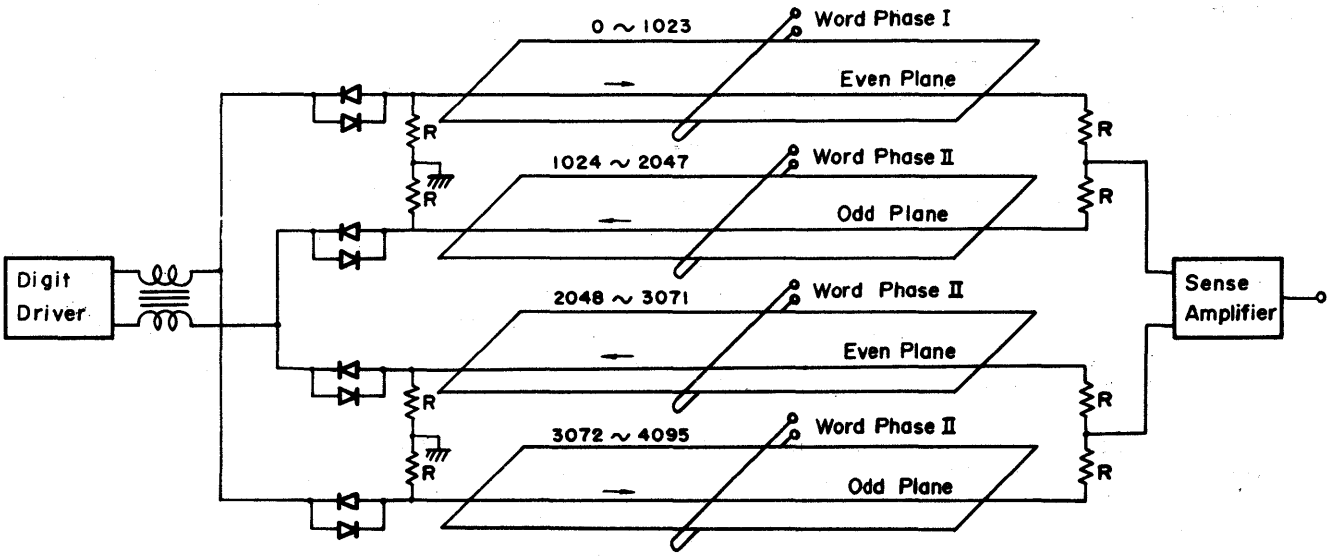


Figure 3—Connection of digit lines

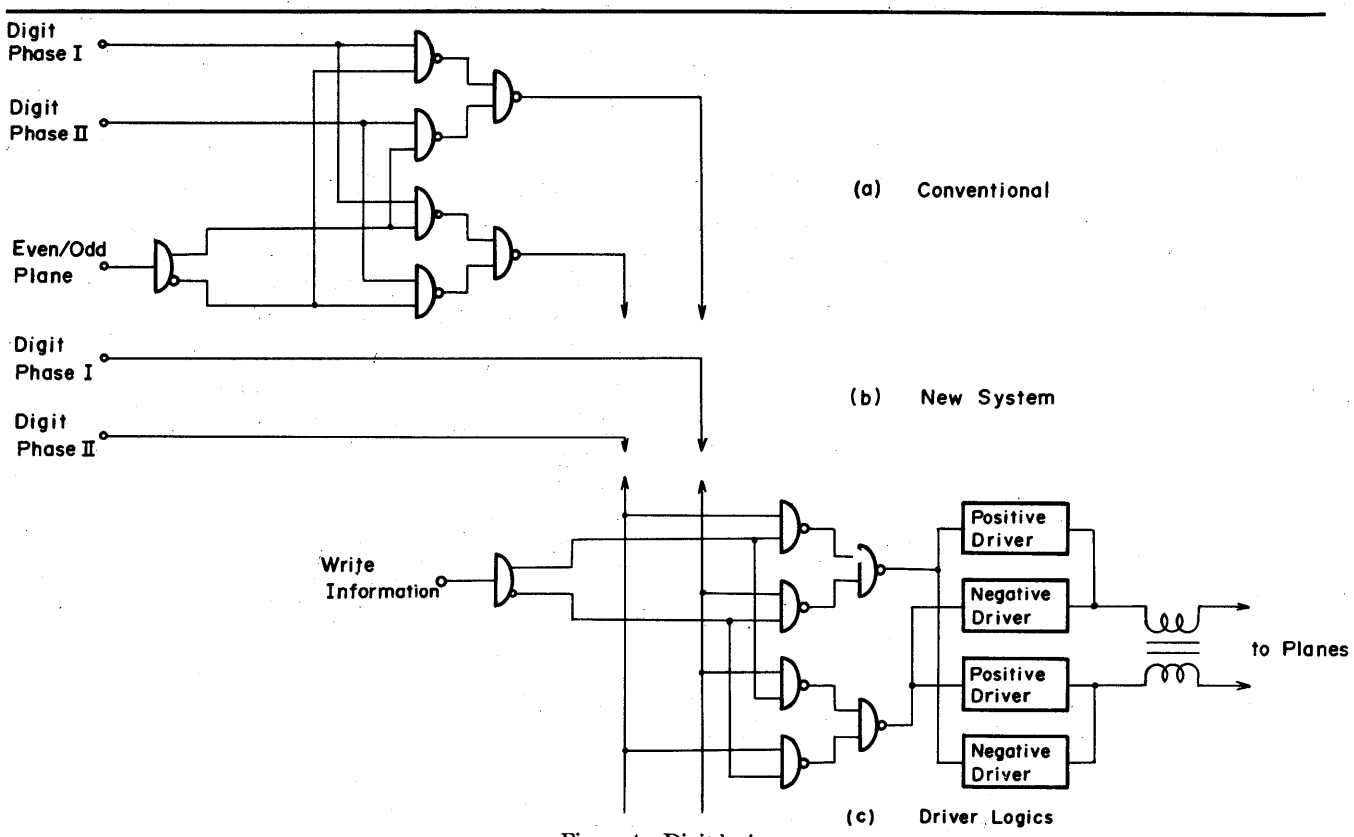


Figure 4—Digit logics

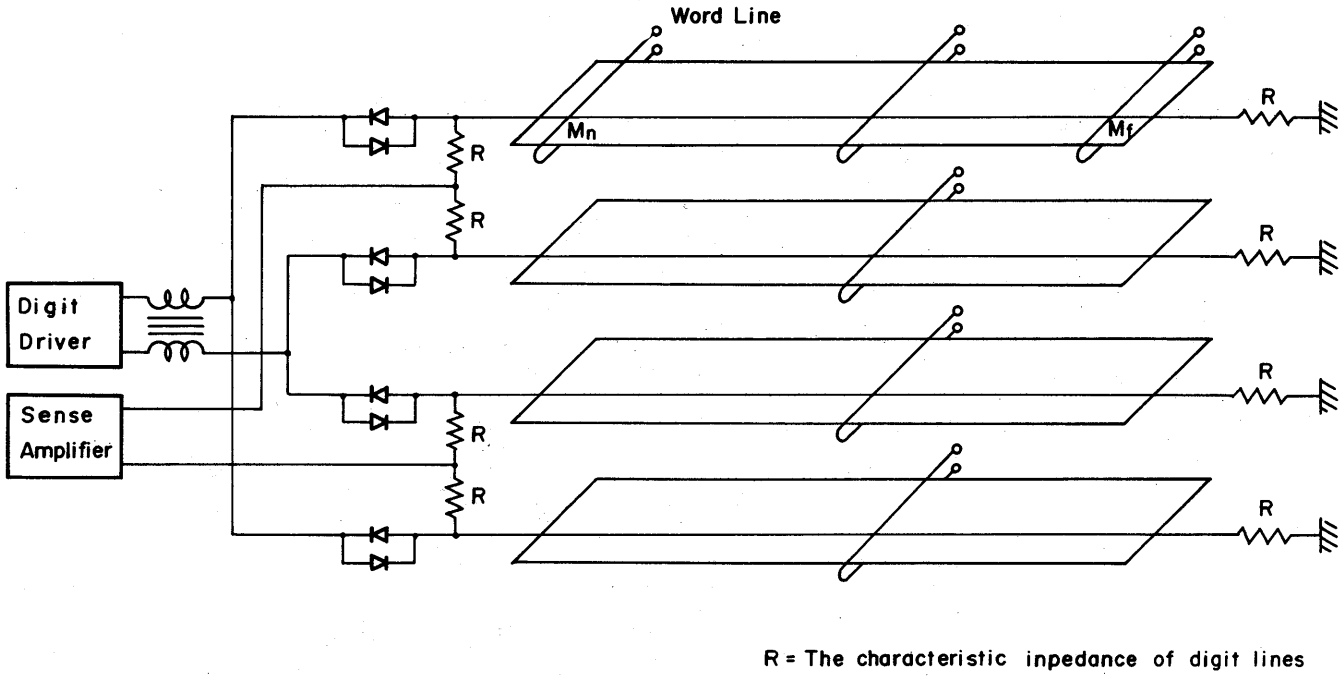


Figure 5—A digit driver and a sense amplifier are connected on the same side of the memory plane

work most satisfactorily when used in the connection shown in Figure 5, where the digit driver and sense amplifier are connected with the memory plane on the same side. Features of both methods are listed in Table 3.

Items	Word Control	Digit Control
Word Timing	When the selected memory location is far from (near to) the sense amplifier, the word timing occurs early (late).	Fixed.
Strobe Timing	Fixed	When the selected memory location is far from (near to) the sense amplifier, the strobe timing occurs late (early).
Digit Timing	Fixed	When the selected memory location is far from (near to) the digit driver, the digit timing occurs early (late)

TABLE 3—Comparison of word control system and digit control system

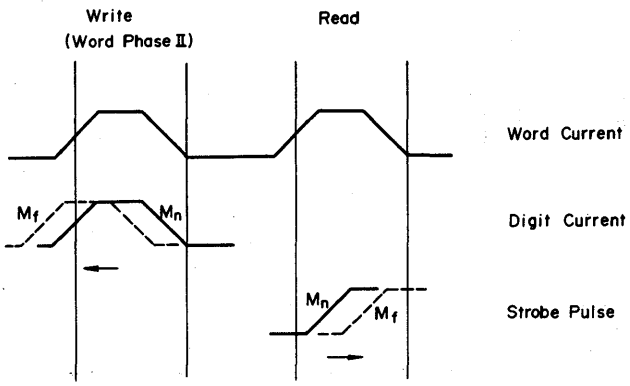


Figure 6—Timing waveforms in the digit control system

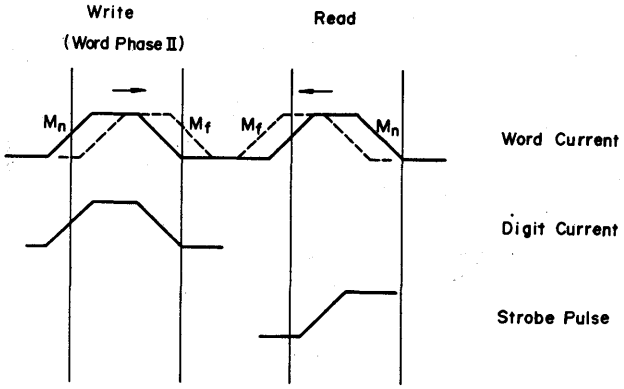


Figure 7—Timing waveforms in the word control system

Let the memory location nearest to the digit circuitry be M_n and the farthest M_f in Figure 5, then an example of the time relation using the digit control system is as shown in Figure 6.

In the cases such as write-after-write, read-after-read, and write-after-read, the time separation of succeeding timings may be smaller than the typical value. However, it is not so serious as will be shown in the case of the word control system.

Figure 7 shows another time relation of word and digit timings when the word control system is employed. The timings of digit circuitry are fixed irrespective of the selected memory location.

As shown in Figure 7, the time separation of succeeding word currents becomes very small in the worst case.

The reason why the situation is worse with the word control system is that the time relation is adjusted only by the word current, causing a significant approach of leading and trailing word currents.

The timing generator for a total system is shown in Figure 8. The delay circuits connected with address decoding logics are used to control the bit timings in accordance with the memory address selected.

Strobe circuit

A two-stage monolithic differential sense amplifier is followed by a modified CML circuit which detects the polarity of readout signals at the strobe timing, and also holds it as long as the strobe pulse is present.

As shown in Figure 9, the polarity detecting strobe circuit consists of a current switch made of transistors Q_1 and Q_2, feed-back loops made of transistors Q_3 and Q_4, steering transistors Q_5 and Q_6, and an emitter Q_7 which makes the current switch inactive when the strobe pulse is absent.

At the moment the base level of transistors Q_7 is shifted from -0.8 volt to -1.6 volt, i.e., the strobe pulse is

given, the current switch is turned on. Since both the transistors Q_1 and Q_2 cannot conduct simultaneously, one of them assumes "on" state answering to an imbalance, however small, present between the base levels of input transistors Q_3 and Q_4.

Word seleztion matrix

Diode-steered transformer matrix[7] and transistor matrix[3] can be selected as word selection matrices.

The feature of the two methods are compared in Table 4.

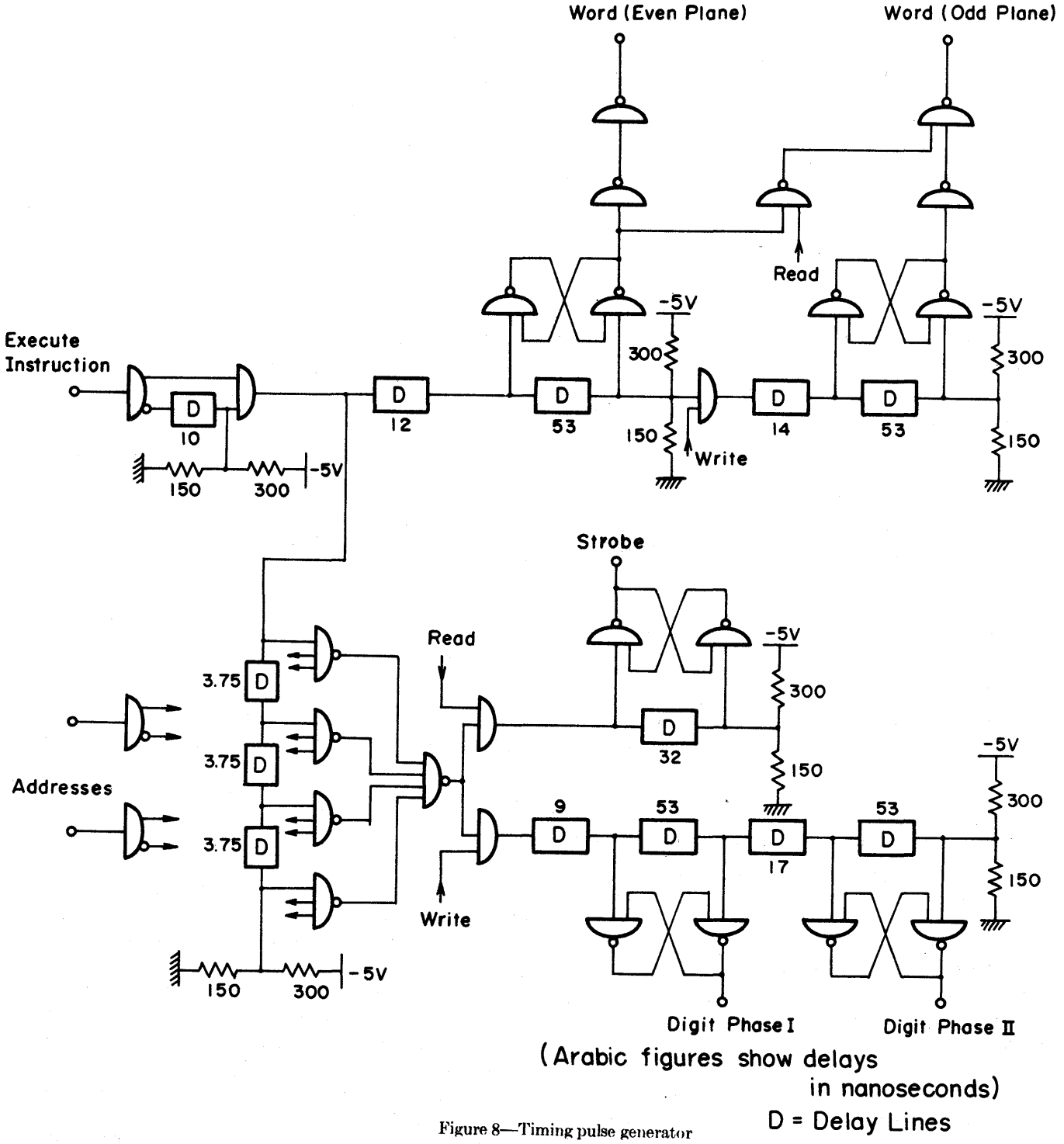
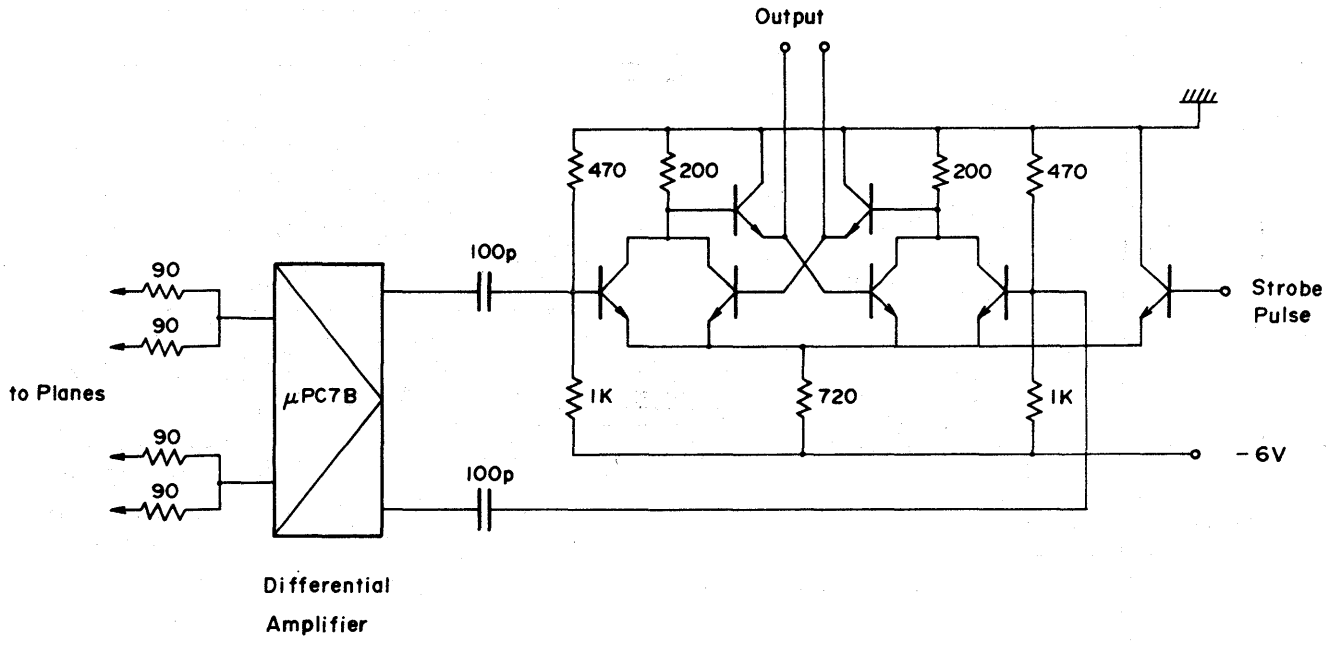


Figure 8—Timing pulse generator

Items	Diode-steered Transformer Matrix	Transistor Matrix
Switch Lines	(1) The current capacity must be large enough to supply the word current. (2) The characteristic impedance of the bus line must be as low as possible to minimize the voltage shift of the bus line. A large part of the power will be wasted in the terminating register. (3) The voltage swing must be greater than the peak amplitude of the back voltage on the word line.	(1) The current required is $1/\beta$ times the word current. (2) The characteristic impedance of the bus line can be made higher. (3) The voltage swing is a function of transistor parameters. It is generally smaller than the peak amplitude of the back voltages on the word line.
Problems in the current Switch Lines	No particular problems exist.	Since the bus line is apt to ring, a damping resistor must be added.
Problems in the Ground Line	There is hardly any current in the ground plane, because the transformer prevents common mode current.	Instead of the voltage source bus line the ground or power line supplies the current to the word line. Accordingly the impedance of the ground or power line must be kept as low as possible.
Problems in the Half-Selected Word Current	(1) The balance in the transformer windings must be improved to prevent the common mode voltage on the primary winding from inducing a differential mode current in the secondary winding. (2) Diodes with less junction capacity must be used.	(1) Transistors with less collector-to-base capacity are recommended. (2) The voltage swing of the base driver (voltage switch) must be designed to be small.

TABLE 4.—Comparison of word selection matrics

Figure 9—Polarity detecting strobe circuit



When production is taken into consideration, a transistor matrix is preferred in the NDRO plated-wire memory. A blockdiagram of word selection circuits is shown in Figure 10.

Experimental results

A cross-sectional model of the system was constructed, and it worked as expected.

The memory stack consists of four modules each containing eight planes. Each plane has 128 word lines by 288 plated-wire digit lines.

The memory wire is prepared by electroplating permalloy onto a bronze phosphide wire 0.13 mm in diameter. The plated wires are spaced at 1.0 mm centers and the word lines are spaced at 1.5 mm centers.

The back EMF of the word line for 300 milliamperword current with a rise time of 20 nanoseconds is approximately 14 volts. A 40 milliampere digit current gives rise to a typical output signal of 10 millivolts.

The basic logic element exclusively used in the experimental model is a current-mode-logic circuit μ PB 80 which has two four-input gates. Typical propagation delay of μ PB 80 is 3 nanoseconds.

A monlithic integrated differential amplifier μPC7B is used for the sense amplifier. It has a bandwidth of 40 MHz and a voltage gain of 40 dB.

Transistor selection matrices and digit drivers are temporarily constructed by discrete components.

The propagation delay and the attenuation per 1-K words were measured. The results obtained are 14 nanoseconds and 1.5 dB.

Adjustment of terminating resistors was necessary to minimize the digit noises.

The use of diodes, connected back-to-back in series with the digit line, was satisfactory.

The minority carrier strage in the diodes and the junction capacity gave rise to a slight ringing on the digit line.

CONCLUSION

It has been pointed out that the delays in address decoders, sense amplifiers and strobe circuits are the princi-

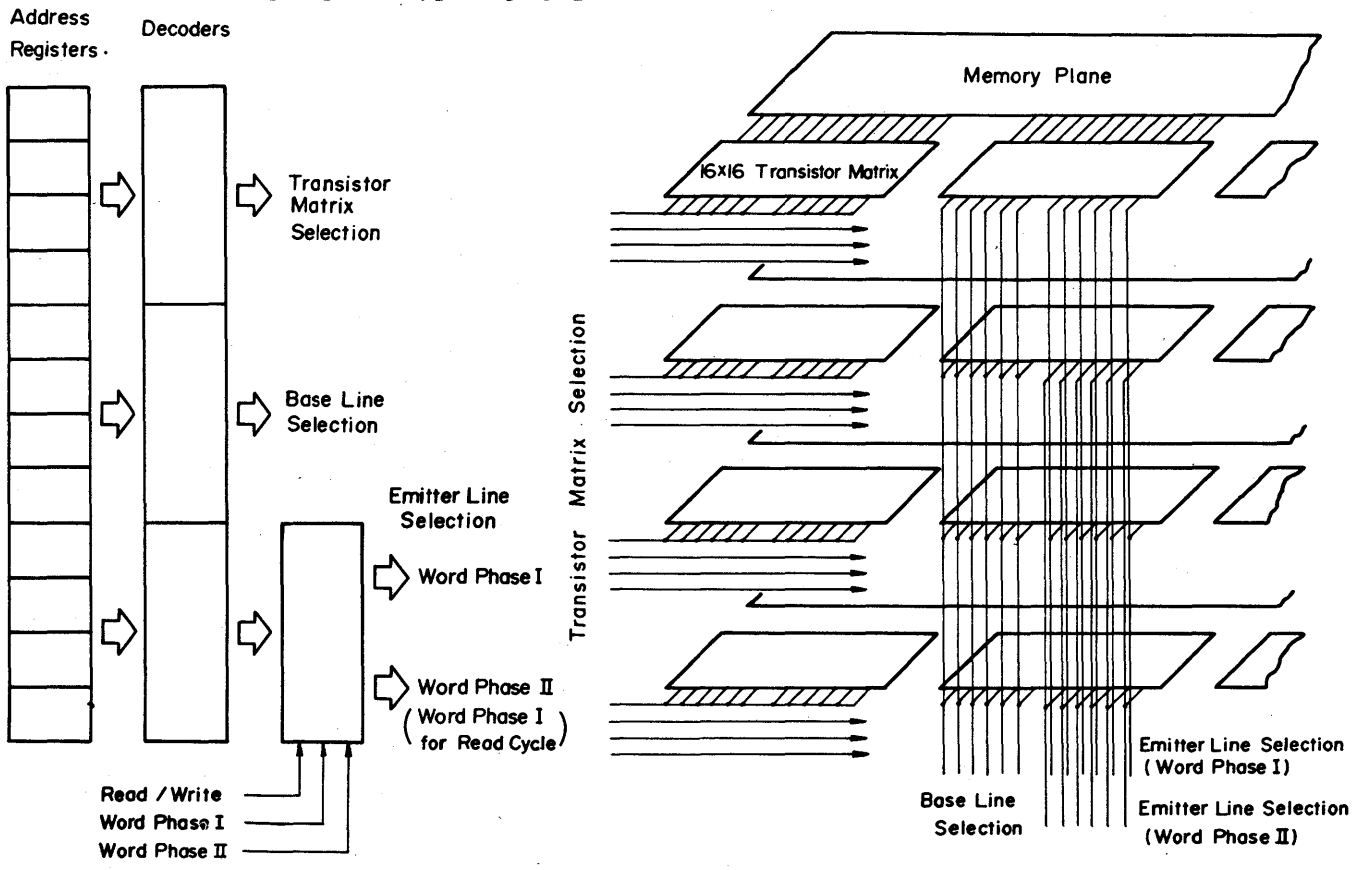


Figure 10—Blockdiagram of word selection circuits

pal factors determining the read cycle time of the memory.

Small-scaled transistor selection matrices in combination with CML decoders were employed, and a decoding delay of 30 nanoseconds was obtained.

A polarity detecting strobe circuit was developed with a modified CML circuit. The strobing delay observed was less than 5 nanoseconds.

The method to compensate the propagation delays in the memory plane was discussed. It was shown that it is desirable to control the timing of digit circuitry.

ACKNOWLEDGMENT

This paper is a part of work done jointly by many research engineers of the Central Research Laboratories of Nippon Electric Co., Ltd.

The author would like to express his cordial thanks to Dr. I. Someya and Dr. Y. Sasaki for their support and guidance, and also to his colleagues, particularly Messrs T. Furuoya and H. Murakami for their cooperation in the construction of his memory system.

REFERENCES

1 C F CHONG R MOSENKIS D K HANSON
Engineering design of a mass random access plated wire memory
Proc FJCC 363 1967

2 J P McCALLISTER C F CHONG
A 500-nanosecond main computer memory utilizing plated-wire elements
Proc FJCC 305 1966

3 B A KAUFMAN P B ELLINGER H J KUNO
A rotationally switched ROD memory with a 100-nanosecond cycle time
Proc FJCC 293 1966

4 S A MEDDAUGH K L PEARSON
A 200-nanosecond thin film main memory system
Proc FJCC 281 1966

5 T ISHIDATE
Circuit techniques for one hundred nanosecond thin film memory
Colloque International sur les Techniques des Memoires Editions Chiron Paris 1966 p 671

6 T ISHIDATE
Delay compensation concept in very high speed memories
NEC Res and Dev 9 129 1967

7 E E BITTMANN
A 16 K-word, 2-Mc magnetic thin film memory
Proc FJCC 93 1964

High speed, high-current word-matrix using charge-storage diodes for rail selection

by S. WAABEN and P. CARMODY

Bell Telephone Laboratories, Incorporated
Murray Hill, New Jersey

INTRODUCTION

Diode matrices used to select the path of an unidirectional or bidirectional matrix current are well known.[1-3] Conventional matrices use a low storage diode crosspoint for unidirectional current and a charge-storage diode crosspoint for bidirectional current. For typical magnetic memory cells the required currents approach 1 ampere. Also, for magnetic thin film memories the required word current duty cycle is small, typically 30 ns out of a store cycle time of several hundred nanoseconds. To conduct such currents, the required silicon area for a diode is almost one order of magnitude less than that required for a transistor. Since the cost of a semiconductor device is strongly dependent on the silicon area used, diode matrices are therefore commonly used for the economical selective drive of magnetic memory stacks. For many memory system configurations, because of the significant cost of high current transistor matrix selection switches, the cost per word line of matrix rail selection is comparable to that of the individual word selection diode. Large matrices are therefore commonly employed to share the switch cost among many matrix crosspoints. The penalty is more stray impedance and system noise as well as difficulty of reliable assembly of large arrays. It will be shown how this rail selection switch function can be implemented advantageously by a circuit combination of a low cost, high-current charge-storage diode and an inexpensive 100-200 mA current transistor. No transformers are needed. Furthermore, the usual number of rail selection transistors can typically be halved by a tandem diode matrix arrangement.

In this paper the basic schemes are presented. The design tradeoffs are then given and discussed. Experimental results are shown.

Basic charge-storage diode rail selection

Figure 1 is the schematic of a diode matrix of 32 word rails and 32 diode rails for a plated-wire store which will be used as an example. The distributed word rail loading capacitance is 32×32 pF ≈ 1000 pF. The distributed diode rail capacitance is 32×3 pF $\cong 100$ pF. Resistors for biasing the matrix diodes in such a manner that nonselected diodes will remain backbiased are also shown. Note that inherent in all diode matrix selection schemes the matrix rail selection switches must carry the current of the selected path.

To select the crosspoint of a word rail and a diode rail, one of the 32 word rails is changed from zero volt to E_1 volt by turning the charging current I_{ch1} on. If I_{ch1} is constant then the word rail voltage rises linearly to E_1 volt. When the rail voltage reaches E_1, CSD_1 goes into forward conduction and the rail voltage is thus clamped to E_1. From this point on, charge is accumulated in CSD_1 by the continued flow of I_{ch1}. All matrix diodes remain backbiased since the diode rail voltage is more positive than the clamped word rail voltage.

Similarly, the closure of a diode rail selection switch generates a current flow through CSD_2. Charge has now been accumulated in the two matrix rail charge-storage diodes, CSD_1 and CSD_2. The charging currents I_{ch1} and I_{ch2} are supplied via two selected low current transistors. In the reverse conduction phase of the charge-storage diode, the voltage drop across the diode is the junction voltage across the forward biased junction of a reversely conducting diode minus the IR drop across the body resistance. Therefore, by closing a common control switch transistor to ground potential, current will flow in the selected path as long as there is charge available in the electrically floating charge-storage diodes. Notice that the current handling and turn-on and turn-off requirements on the matrix rail selection switches are relaxed and decoupled from the high-speed, high-current requirements of the selected matrix path. Rise time, amplitude and duration of the current pulse are determined by the circuit parameters and the common control circuitry described below.

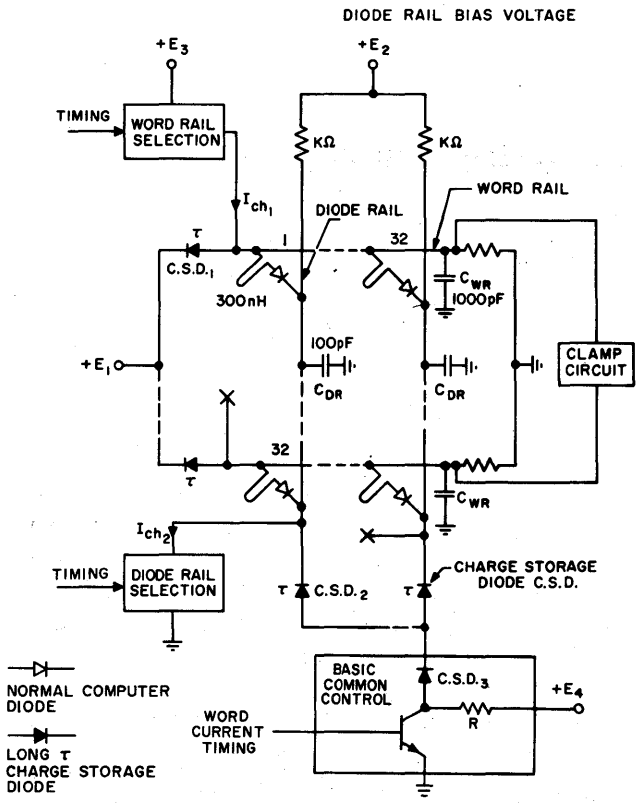


FIGURE 1—A basic 32×32 diode matrix with charge-storage-diodes for matrix rail selection

As an extension on the basic scheme the number of rail selection switch drivers can be reduced by a circuit arrangement where groups of the rail selection diodes are shared by common switch drivers. Figure 2 illustrates this principle as applied to the diode rails. A 16 word rail by 64 diode rail matrix is shown selected by 16 + 8 + 8 = 32 medium current transistor switches. This should be compared to 80 high-current switches for a classical 16 X 64 matrix. Alternatively, 64 switches would be required for a square matrix covering the same 1024 matrix crosspoints. It can be seen that the charge-storage diodes are also arranged in matrix form which we shall refer to as being in tandem with the original matrix.

The practicality of the basic scheme presented above depends on a tradeoff between particular circuit parameters and requirements. A brief analysis of typical circuit performance will be presented next.

Required charges

Word current charge area

The nominal plated-wire store word current pulses shown in Fig. 3 require the following amounts of charge:

$$Q_{NDRO} = \int_0^t i\,dt = \tfrac{1}{2}\cdot 0.4\cdot 40\cdot 10^{-9} = 4\text{ nC}$$

$$Q_{DRO} = \tfrac{1}{2}\cdot 1\cdot 50\cdot 10^{-9} = 25\text{ nC}$$

This amount of charge must be supplied by CSD_1, CSD_2 and the common control circuit. The following arithmetic indicates the relative sensitivities of the parameters involved. For a desired word current peak I_p and a constant rising current slope $s = \frac{I_p}{t_{rise}}$ the required charge Q is given by: $I_p = \sqrt{2Qs}$. Also since $s = \frac{E}{L}$, where E is the fixed driving voltage minus the semiconductor junction drops of a selected path, and L is the driven word loop impedance, it follows that

$$I_p = \sqrt{2Q\frac{E}{L}}.$$

Consequently for a fixed available Q,a10 percent variation of E or L will result in 5 percent variation of I peak. This reduced sensitivity to variations in L is significant because L may vary from word loop to word loop while Q and E are more readily controlled in a memory systems environment. The charges of the matrix capacitance will be discussed next.

Matrix capacitance charges

To change the voltages at the matrix terminals charge must be supplied to or drained from the matrix rail capacitances. Here only the charges at the word rail will be discussed.

The word rail terminal of the word line must for pulses with equal rise and fall times supply twice the basic 25 nC for the DRO and 4 nC for NDRO. At the word rail terminal there are three possible sources of charge available to supply the word current (see Figures 1 and 4):

(a) The rail selection switch.
(b) The charge-storage diode CSD_1.
(c) The word rail capacitance.

To attain a linearly rising word current the voltage on the word rail must remain constant during the rise time of the word current pulse. Therefore the role of CSD_1 is twofold (I) to limit and clamp the charging of the selected word rail capacitance to a well-defined voltage and (II) to implement a low impedance charge reservoir at a fixed votage level during the rising portion of the word current. This buffer can in principle be either the CSD or the large word rail capacitance of 1000 pF. However, such a charge drain from the word rail capaci-

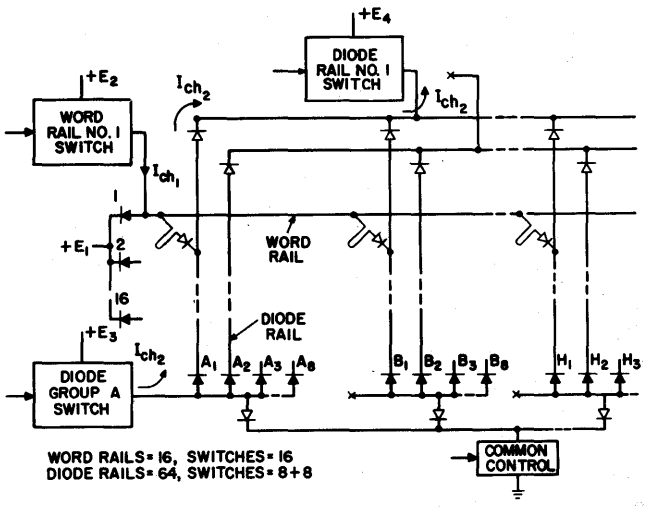


FIGURE 2—Tandem diode matrix with charge-storage-diodes. There are 16 word rails and 64 diode rails. The word rails are operated as indicated in Fig. 1. The diode rail charge-storage-diodes are arranged in a diode matrix of eight groups of diodes (A, B, . . ., H) each of eight diodes. After charging, a selected crosspoint is driven by the closure of the common control circuit.

tance will produce access noise during read. It follows implicitly from the discussion above that in the case of the 8 nC necessary at the word rail terminal for NDRO that the charge storage feature of the word rail diode is in some ways incidental. For NDRO operation in particular the matrix access time is dominated by the charging time, say 75 ns, of the word rail capacitance of 1000 pF charged to 15 volts at the corresponding charging current of 0.2 A. The finite lifetime of the carriers in the CSD does, however, set a practical limit to how small a rail selection current one can realize and still achieve a 1 ampere DRO pulse. The next section will summarize the charge-storage-diode phenomena briefly.

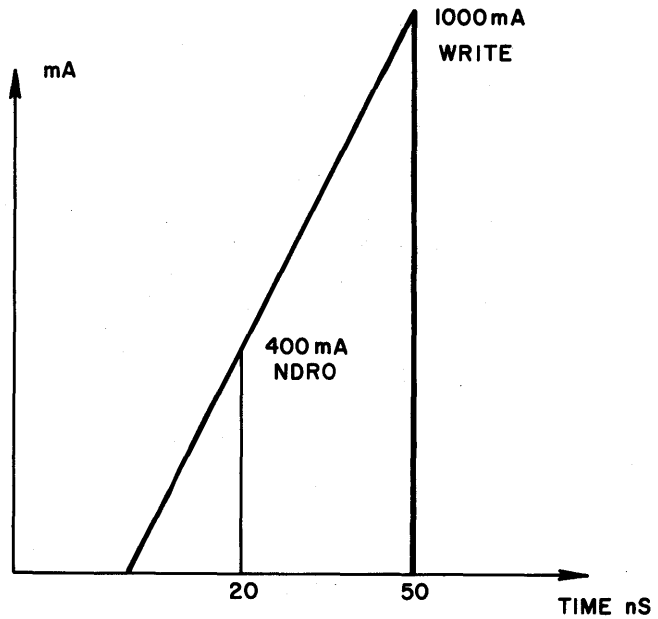
Charge-storage diode properties

The continuity equation for charge describes the charge flow through a diode:

$$\frac{dQ}{dt} + \frac{Q}{\tau} = i(t)$$

The term $\frac{Q}{\tau}$ is the amount of charge disappearing by recombination in the diode and the other terms expresses the conservation of charge. Assume a current I_F is conducted in the forward direction for a period of time t_F. Charge is accumulated in the diode. The efficiency E of this charge reservoir measured at time t_F is:

$$E = \frac{\text{Charge Available}}{\text{Charge Supplied}} = \frac{I_F\tau(1 - e^{-t_F/\tau})}{I_F \cdot t_F}.$$



$$Q_{NDRO} = \int_0^t i \cdot dt = \frac{1}{2} \cdot 04 \cdot 20 \cdot 10^{-9} = 4\,n\,COUL$$

$$Q_{WRITE} = \int_0^t i \cdot dt = \frac{1}{2} \cdot 1 \cdot 50 \cdot 10^{-9} = 25\,n\,COUL$$

FIGURE 3—Current, time and charge for nominal word currents

It follows that the longer the lifetime τ the higher the efficiency. The necessary charging time is given by:

$$t_F = -\tau \ln\left(1 - \frac{Q}{I_F\tau}\right)$$

Figure 5 shows typical calculated tradeoffs using this expression.

Notice that limiting the peak current with a common control charge-storage diode eases the charge uniformity requirement on the many rail selection diodes. The rail diode requirement is therefore only single ended; namely, more charge is required by the rail diodes than the amount in the common control diode. A simple common control circuit for a 400 mA peak NDRO pulse is implemented using a short lifetime diode, thus securing a maximum peak word current which is insensitive to the duration of the charging pulse. The 1.0 A peak DRO pulse is conveniently implemented using a longer lifetime diode for common control.

Rail selection switches

The rail selection switches can be implemented in a variety of ways. The "optimum" choice depends heav-

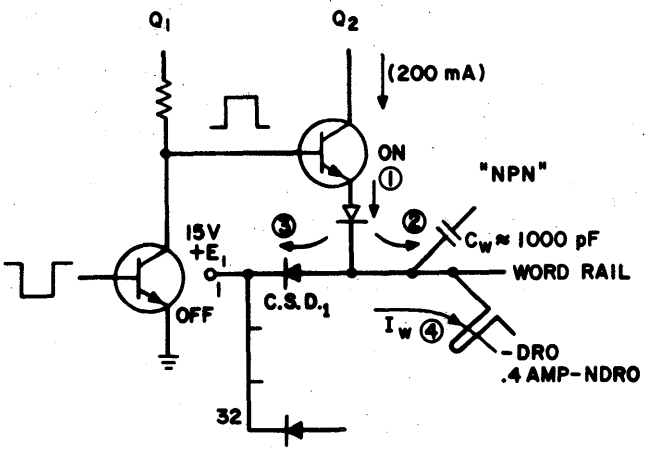
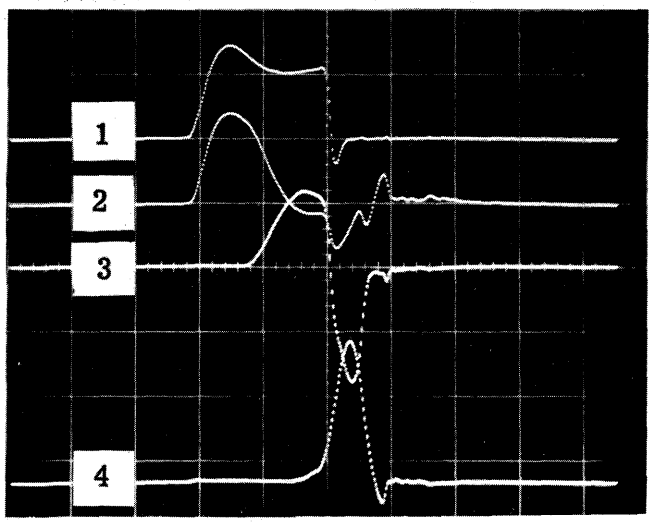


FIGURE 4—A word rail charging circuit. The current branches are indicated.



50 nS, 200 mA/DIV

FIGURE 6—The branch currents monitored at the word rail terminal shown diagrammatically in Fig. 4. Vertical scale I = 200 mA/div. Horizontal scale 50 ns/div.

ily upon the state of the art of the integrated semiconductor technology.

Figure 4 is the schematic of a simple word rail selection switch implemented with a charge storage diode and a medium current transistor. For selection Q_1 is cut off and Q_2 is driven into conduction. The word rail voltage is now rising as the current flowing through the emitter of Q_2 charges the 1000 pF. Eventually the CSD, goes into forward conduction and the word rail voltage is clamped to E_1. The charge flow through the emitter of Q_1 is now accumulated in the CSD. In this manner a selected word rail resides at +15 volts and will remain short circuited to this voltage as long as there is charge available in the CSD. A Darlington pair circuit reduces the required selection switch standby current drain by one order of magnitude. It is also worth mentioning that such a circuit provides isolation between the high current access circuit and logic circuitry. This feature is important for access noise minimization.

The schematic of a charge-storage diode diode-rail

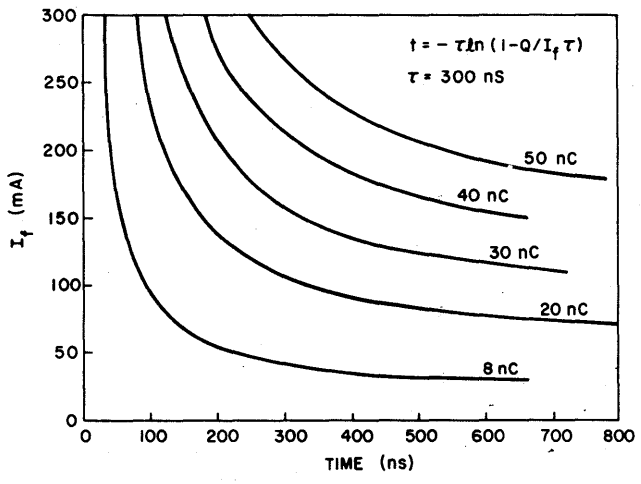
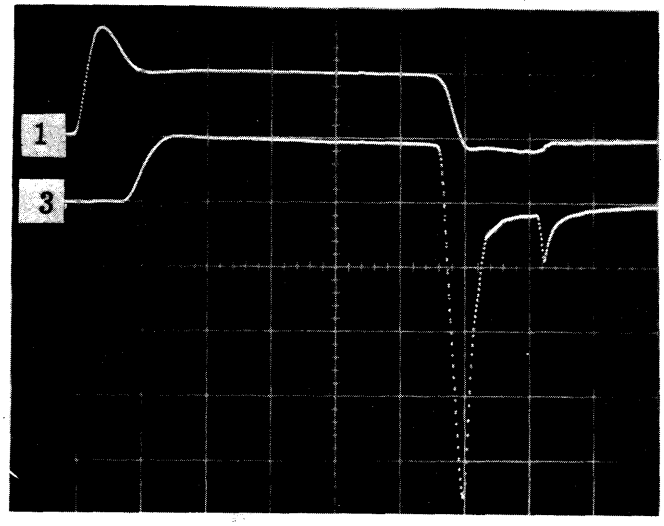


FIGURE 5—Typical calculated tradeoffs from:

$$t_F = -\tau \ln\left(1 - \frac{Q}{\tau . I_F}\right)$$



125 nS, 200 mA/DIV

FIGURE 7—Traces 1 and 3 of Fig. 6 extended for DRO operation. Horizontal scale 125 ns/div.

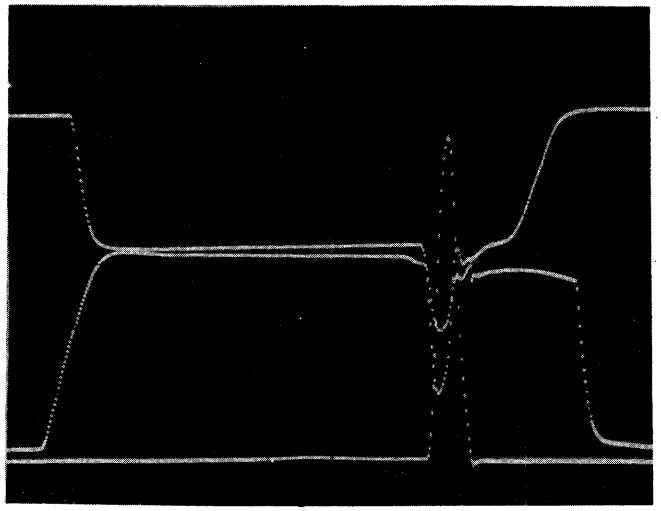


FIGURE 8—The voltages at a pair of selected diode and word rails: 5 volt/div, 125 ns/div. The DRO current is also shown, 200 mA/div.

selection switch is similar to that of the word rail. The function of this switch is similarly to provide a short circuit at the matrix rail for at least as long a time as current is driven through a selected loop.

Experimental results

Both the simple and the tandem diode matrices have been implemented. Oscilloscope traces for the simple matrix are shown here. The traces for the tandem matrix are similar but more complex. The stored charge required can be produced by a fairly large range of combinations of charging current amplitudes and durations. As an example, Figure. 6, trace 1, shows the charging current from the word rail selection switch. The NDRO word current is shown as trace 4. Trace 2 is the current charging the word rail capacitance. When the word rail voltage rises to the turn-on level of CSD_1 this current decreases to zero. The charging and reverse current of the word rail diode is shown as trace 3.

To generate the word current pulse the following operations take place in the common control circuit. First a 100 Ω resistive discharge path is switched on. This path starts discharging the 100 pF stray capacitance of the diode rail slowly. Thus the word current rises with a rounded slope and less access noise is generated. The full word current is reached by switching CSD_3 to ground.

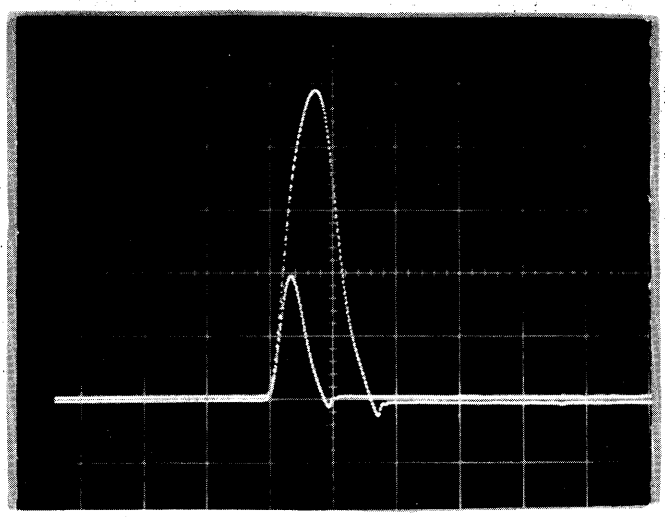


FIGURE 9—The DRO and NDRO pulses are shown superimposed. Vertical, 200 mA/div. Horizontal, 50 ns/div.

DRO word current is generated by increasing the duration of the charging of the diodes. Figure 7 shows traces 1 and 3 of Figure 6 extended to accumulate sufficient charge for the 1 ampere DRO pulse. The word and diode rail voltages during DRO operation are shown in Figure 8. The upper trace is the diode rail voltage, which is initially at the excessive bias value of 25 volts. The lower trace shows the word rail voltage initially residing at ground level. For selection the word rail voltage rises to the clamped level of E_1 and the diode rail voltage drops to a value low enough to permit charging of CSD_2 and CSD_3 while still maintaining a reverse bias on the selected access diode. This bias condition prevails until the common control circuit is activated. As mentioned above the word current ceases when the common control diode CSD_3 is depleted of charge. The rail voltages are subsequently recovered to the initial conditions. Figure 9 shows the NDRO and DRO pulses superimposed for comparison. A diode τ of approximately 300 ns is calculated from the waveforms shown.

CONCLUSION

Charge-storage-diodes in combination with 100-200 mA selection transistors can readily be arranged for diode matrix rail selection of five to ten times larger currents. The limit of the reduction of the current handling requirement on the rail selection transistors is determined by (1) the available time for charging of the diodes and (2) the lifetime of the carriers in the charge-storage-

diodes. The switching speed requirements on the transistors are also relaxed since the matrix current pulse is not conducted via the rail selection transistors. Furthermore, the number of rail selection transistors can typically be halved by charging the rail selection by shared driver transistors.

ACKNOWLEDGMENTS

It is a pleasure to thank T. R. Finch for active interest and constructive comments.

REFERENCES

1 A MELMED R SHEVLIN
Diode steered magnetic core memory
IRE Trans EC–8 p 474 Dec 1959

2 T R FINCH A H BOBECK
The waffle iron store
ISSCC Digest of Technical Papers 1963 pp 12–13

3 G A DODSON J A RUFF
Charge storage diode for memory applications
ISSCC Digest of Technical Papers 1964 pp 104–105

Automatic checkout of a large hybrid computer system

by JESSE C. RICHARDS

Lockheed Missiles and Space Company
Sunnyvale, California

INTRODUCTION

This paper describes the Automatic Preventive Maintenance (APM) Program used on the hybrid computer system installed in 1967 at Lockheed Missiles and Space Company, Sunnyvale, California. This is one of the largest hybrid computer systems in the world. It has a multi-processor digital computer (CDC 6400) with five remote entry display units (CDC 211), 32K core storage, and a large storage disc. Because of these capabilities the APM program has many unique operating features. The size and the cost of the equipment make it imperative to keep downtime to a minimum. To do this the system is subjected to many layers of manual and automatic checks. This paper will concentrate on the APM tests of the analog to digital linkage equipment and the analog computer section with the emphasis on the analog computer section. The digital computer section of the hybrid computer is maintained by the vendor and therefore is not included in the discussions of the APM program.

Background

When LMSC ordered its hybrid computer in April 1966, an integral part of the procurement was a software preventive maintenance (APM) program. The digital computer section had a diagnostic routine furnished by the vendor. The APM program had to be defined for the four analog computers and the two linkage equipments, Intracoms. Shortly after the contract for the hybrid computer system was awarded, meetings were held with vendor representatives to outline the AMP tests desired and to define the goals. The initial set of APM tests were designated Phase I tests. The Phase I tests were to be simple mechanizations and represent a limited investment in programming time and in checkout patchboards. A complete evaluation of the Phase I tests would determine the future direction of the APM tests.

Phase I APM guidelines

Our overall thinking on what APM tests should do can be summarized by the following formula.

$$T = (C + A + I) + R \tag{1}$$

Where:

- T = Total time to prepare computers for hybrid operation
- C = Time to run tests
- A = Time to analyze test results
- I = Time to isolate the malfunction after analyzing test results
- R = Time to change, adjust or repair faulty units

The APM tests were designed to minimize the terms enclosed within the brackets. The overall guidelines established for the Phase I APM tests were as follows:

Speed The total time of testing was to be a maximum of one-half hour for each of the four analog computers or two hours for the entire hybrid system.

Simple Mechanization The tests were to check individual units as opposed to checking a long chain of units, in order that a malfunction could be easily pinpointed to a particular unit. This made the digital programs required for the various tests simpler to write and debug.

Stand Alone Each test would run independently so that the tests could run in any order.

No plugging in of Cables or Auxiliary Equipment The tests would be run without altering the equipment con-

figuration. No cables were to be disconnected nor were auxiliary black boxes to be plugged into the computer.

Simultaneous testing Up to four analog computers could be checked simultaneously. Unused analog computers could also be checked out while a hybrid problem was running.

Minimum Equipment Tie-up Where possible the test would tie up only the particular machine under test.

DVM Readings The DVM would normally be tested first and then would be used as the measurement standard. Only one DVM reading of an output would be taken rather than taking a number of readings and averaging them.

Format of printout Printout was to be of the exception type; that is, only bad units would be listed, and thus the results would be easy to interpret.

Hybrid System Software Tests were to use standard system software and the normal system configurations, thus testing the system in the way that it would be used.

Ease of Operation Tests were to be easy to operate and require a minimum of special training to perform.

Hands Off A test would run to completion without pauses for operator intervention.

Patchboards Patchboards would be wired with normal patch cords and not "hard wired." Special equipment would not be mounted on the patchboard. No effort would be made to load all tests on one or two patchboards.

HINDSIGHT ADDITIONS

Printout

Where a test of a unit such as a multiplier involves a series of thirteen tests, there are thirteen possible failures. Rather than list bad units under each of the tests, the printout lists the units in a column and the various test results in a row next to each unit. This presents a much better one glance picture of a particular unit. Figure 1 shows a sample of our present printout on multipliers.

Display unit control

As originally delivered, the tests ran from card decks. Everything was converted to be run from the remote display units. The APM tests all reside on the mass storage device (disc) and can be copied into the central memory for execution at any time.

System description

Figure 2 is a block diagram of the hybrid computer configuration. The list of equipment and a brief system description is included in APPENDIX (A).

Hybrid computer tests

The APM tests of a hybrid computer system fall into three major categories. They are:

Discrete tests

Discrete tests are tests of logic units, such as flip-flops, Nand gates and binary-decimal up/down counters, or tests of units that have a predetermined logic output pattern for a logic input pattern. Tests of the general purpose logic and linkage equipment fall into this category. These tests are performed very efficiently under digital computer control. In a hybrid computer system many of these tests can be performed only under digital computer control. These tests originate and terminate in the digital computer without being transformed out of the discrete world.

Hybrid unit tests

Units that are not exclusively digital or analog fall into this category, such as analog-to-digital converters and digital-to-analog converters. The inputs to these units originate on one type of computer and terminate on the other. Consequently these tests are less adaptable to digital checkout than the discrete tests.

Analog computer units tests

Analog computer units, such as multipliers, integrators, amplifiers, servo set potentiometers and resolvers, operate in a linear-parallel fashion and are the least adaptable to digital computer check out. These units have to operate correctly over a $\pm$ full scale voltage range, a frequency range up to 100 khz at accuracies to within $\pm$ 10 MV and under various load conditions and modes. The present generation of analog computers are not designed with good APM features nor has much work been done on providing standard APM software. For these reasons, the analog computer section of the hybrid computer system poses a real challenge to an effective APM system.[1]

Overall maintenance philosophy

All computers are checked out between 12:30 and 7:00 a.m. All units are checked in place and in a standard configuration. Only those units testing bad are removed. No units are ever removed from the computer for periodic checking.

+X =	+00.00	+20.00	−20.00	. . .	−99.00	−99.00	+99.00
+Y =	+00.00	+30.00	+30.00	. . .	+99.00	−99.00	−99.99
0	0.00	0.00	0.00	. . .	0.00	0.00	0.00
1	0.00	0.00	0.00	. . .	0.00	0.00	0.00
.	.	.	.	. . .	.	.	.
.	.	.	.	. . .	.	.	.
454	0.00	0.00	0.00	. . .	0.00	0.00	0.00
455	0.00	0.70	0.80	. . .	0.00	0.10	0.00

FIGURE 1—Abbreviated example of multiplier test printout all errors below tolerance printout is zero

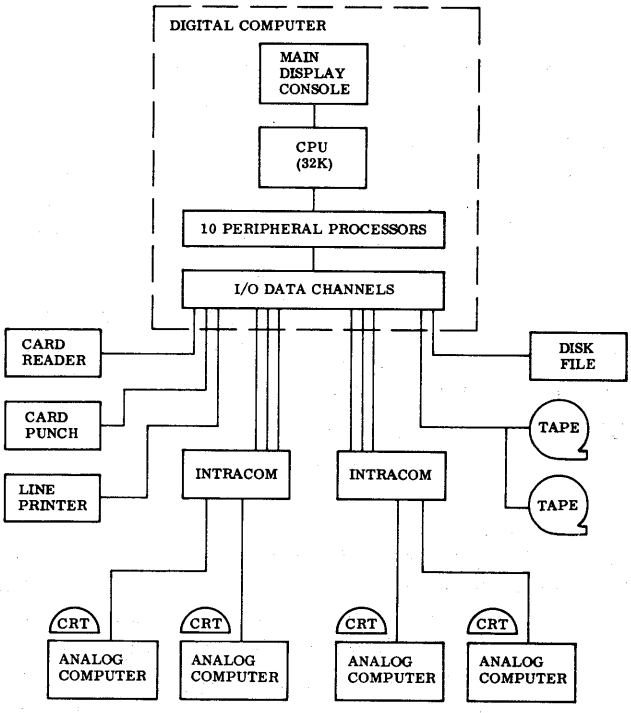
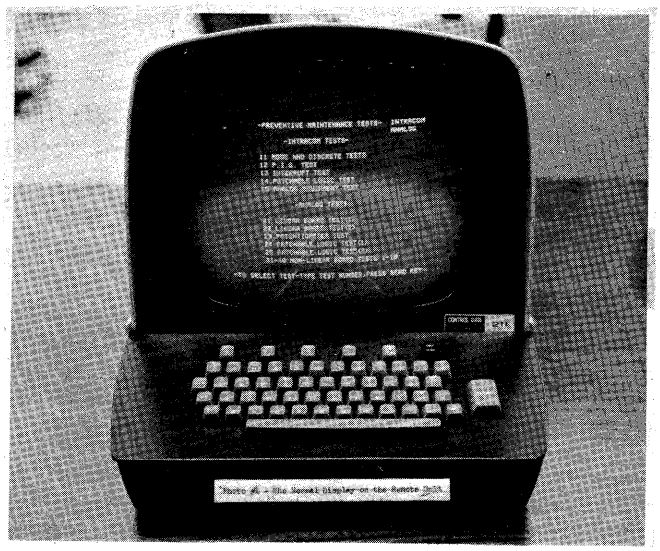


FIGURE 2—LMSC hybrid computer system

It is assumed that if the equipment is operational at 7:00 a.m., it will perform properly for the next two shifts (16 hrs.) (The customer then runs an automatic patchboard verification program and a check run to verify proper computer performance.) An accurate count is kept of the units adjusted and replaced on APM vs problem time. The ratio of these two numbers is considered a good measure of the APM effectiveness.

Method of performing automatic PM tests

The method of performing the APM tests is unique and may be of interest to some Readers. A Remote Display Entry Unit (Display Unit) with a 1000 character display capacity and a keyboard is located at each analog computer. The display unit keyboard, through software programs, allows complete control of all digital computer programs. Photograph #1 is a picture of



this unit. All APM tests are run from these display units, therefore the technician never has to leave the test area except to get patchboards and listings. The display unit has a normal display shown in Figure 3. To start the APM tests, the technician selects E on display unit keyboard. Now the display shown in Figure 4 is presented on the display unit scope face. Assume Test 21 is to be run; the technician types 21 on the keyboard (as per instructions on the remote unit scope face). This copies the digital program required for that test from the disc into central memory, and the display

1. Type Task X, X = Code Letter
2. Press SEND Key

Code	Task
A	Directory of Tasks
B	Job Control
C	Variable Display and Modification
D	Program I/O
E	Preventive Maintenance
F	Source Modification
G	Unassigned
H	Hybrid Utility
I	Dayfile Scan
J	Engineering Aid

* = Task assigned at this station (Max. of 5)

FIGURE 3—(The normal CDC 211 display)

PREVENTIVE MAINTENANCETESTS—INTRACOM ANALOG

—INTRACOM TESTS—

11 Mode and Discrete Tests
12 P.I.G. Test
13 Interrupt Test
14 Patchable Logic Test
15 Analog Equipment Test

—ANALOG TESTS—

21 Linear Board Test (1)
22 Linear Board Test (2)
23 Potentiometer Test
24 Patchable Logic Test (1)
25 Patchable Logic Test (2)
31–40 Non-linear Board Tests 1–10

—To select test—type test numbers, Press SEND Key—

FIGURE 4—This display appears when "E shown in Figure above is selected.

21 LINEAR OBARD TEST (1) INTRACOM ANALOG W

Test Covers DVM, Reference DAC, ANA Subsystem, power supplies. All amplifier inputs and outputs in summer configuration

** REMINDERS **

1. Mount analog board, PMLINEAR
2. Press digital computer button on analog
3. Press analog controller button on Intracom
4. No logic or intracom board required
5. Drop logic board.

—To start test—type GO, press SEND Key;—To return to initial display—Type 0, Press SEND.

FIGURE 5—The CDC 211 display when test 21 is selected.

shown in Figure 5 appears on the scope face. The technician performs the tasks described under reminders, then types GO into the display unit keyboard. The digital program now performs the test and prints a hard copy on the line printer and lists the three worst cases on the remote display. Figures 6A, 6B, 6C show the

********* ANALOG LINEAR TEST 1 *********
POWER SUPPLY CHECKS

Power Supply Address	Value Expected	Value Read	Error	Tolerance Specified
500(6A)	100,00V	100,00V	.00V	.02V
501(6A)	−100,00V	−100,00V	.00V	.02V
.	.	.	.	.
.	.	.	.	.
556(5Y)	100,00V	−5,79V	.21V	.30V
557(5Y)	−00,00V	−100.00V	.00V	.02V

* = Non-Adjustable

FIGURE 6A—Abbreviated printout of test 21 power supply test

******* ANALOG LINEAR TEST 1 *******
REFERENCE DAC ERRORS

Amplifier Address	Value Expected	Value Read	Error
200	11.11V	0.00V	11.11V
201	11.11V	0.00V	11.11V
.	.	.	.
.	.	.	.
207	−99.99V	0.00V	99.99V
210	−99.99V	0.00V	99.99V

FIGURE 6B—Abbreviated printout of test 21 reference DAC errors. (no test board on)

****** ANALOG LINEAR TEST 1 ******
SUMMER TEST ERRORS

Amplifier Address	Value Expected	Value Read	Error
0	47.83V	0.00V	47.83V
1	47.83V	.01V	47.82V
.	.	.	.
.	.	.	.
166	47.83V	0.00V	47.83V
167	47.83V	.01V	47.82V

First half of linear board test is complete.

FIGURE 6C—Abbreviated printout of test 21 summer errors (no test board on)

hard copy of the test results and Figure 7 shows the display that appears on the remote display. In a normal well maintained machine, the errors on each test will be less than three, therefore, the technician can look at the remote display and, depending on his schedule, make the repair immediately or run all of the PM tests and make the repairs all at once.

This sequence of selecting a test on the remote dis-

21 LINEAR BOARD TEST (1) INTRACOM ANALOG W

Test Completed
143 Reference DAC errors
1 Power Supply Errors
120 Summer Errors

WORST CASE SUMMARY

TEST	ERROR	ADDR	EXPECTED	MEASURED
REF DAC	236.52V	A206	−99.99V	−136.53V
PWR SUPL	1.14V	P503	−150.00V	−151.14V
Summer	185.32V	A 48	47.83V	−137.49V

—See printer listing for additional details—
—To return to initial display-type 0, press Send—
10.04.17. END

FIGURE 7—Remote display of the errors on test 21 (no test board on)

play, performing the reminders listed on the display, and running the test is continued until all tests are run.

Because the CDC 6400 is a time-sharing multi-processing digital computer, tests can be run on more than one analog computer or Intracom simultaneously. At the present time, the APM tests can be run on the complete hybrid computer system (four analog computers, and two Intracoms) in approximately one hour.

After performing all of the tests and correcting the malfunctions, the individual tests can be rerun as necessary to insure proper performance.

Test summary

Table I is a summary of the APM tests.

Test #		Time for test
	Intracom—Linkage Tests	
11	READ-WRITE discretes and parallel READ-WRITE. Generates 66 patterns on the WRITE discretes and checks for correct pattern on the READ discretes.	3 sec.
12	Precision Interval Generator (PIG): Initializes the PIG and verifies proper initial count. Checks for monotonic countdown. Exercises all three modes, RESET, HOLD and RUN. Repeats test 10 times.	20 secs.
13	Interrupt test: Checks all interrupts for proper operation in all three modes.	5 secs.
14	Intracom patchable logic: Tests all counters, Nand gates, flip flops and inverters. Tests counters in Binary and BCD in both up and down counting.	22 secs
15	Digital to Analog (DAC) and Analog to Digital converters (ADC) test: Sets all DACs and reads with the DVM to measure DAC accuracy. Uses DACs as inputs to ADC channels to measure ADC accuracy.	50 secs
16	DAC Ramp Test: All DACs are incremented one bit at a time. One DAC is nulled against all other DACs on the analog board and the error is displayed on the strip chart recorders.	2 min.
16A	DAC–ADC Test Static: One master DAC is incremented one bit at a time to ± full scale. This is used as the input to 16ADC channels. The ADC readings are used to set 16 DACs. Each of these DACs is nulled against the master DAC. The outputs of each null error amplifier is displayed on strip chart recorders.	2 min.
16B	DAC–ADC Test Dynamic: This is the same as Test 16A but alternate ADC channels are opposite polarity. This tests the ADC performance for all combinations of voltage jumps on alternate channels.	2 min.
17	ADC TRACK and HOLD: Tests all track and hold units for accuracy in TRACK and tests for drift in HOLD.	30 secs
21	Checks reference DAC at 9 settings, DVM, all analog power supplies and 120 amplifiers in summer modes. (Uses all amplifier inputs and outputs.)	12 secs.
22	Tests 120 amplifiers in HIGH-GAIN mode, checks 60 combination amplifiers and all D/A switches. Checks all computer modes for proper operation.	12 secs.
23	Potentiometer Test: Sets all 144 servo set potentiometers to a value selected from a random number table. The potentiometers are patched to amplifier inputs and the amplifiers are read out to verify connections through the patchbay and patchboard.	2 min. 30 sec.
24,25	Analog Computer Logic: Test the same as Test 14, but performed on the analog computer patchable logic.	
31	Integrators: Sixty integrators rate tested in four time scales. Tests for drift and hold. Performs derivative check on all integrator inputs. Checks initial condition input for accuracy. Tests all limiters by measuring slope of the limit.	2 min. 15 secs.
32*	Multiplier Test: Tests multiplier accuracy with thirteen different sets of input voltages.	2 min. 50 secs.
33,34,35,36	Multiplier Test: Tests multipliers in DIVIDE, SUMMER, SQUARE, and SQUARE ROOT mode.	63 secs.
37	Resolver Test: Tests resolvers accuracy with ten different input values.	15 secs.
38,39,40	Resolver Tests: Resolvers tested in 1R, ROTATION, and INVERSE modes.	8 secs.

TABLE I—APM Test summary

*Tests 32 through 40 are run non-stop under digital control or may be performed individually.

Total patchboards required, 4 analog, 4 Intracom and 4 analog logic.

Total time required to check one analog computer—10 minutes and 5 seconds. Total time to check 1 Intracom, 4 min.

Test discussion

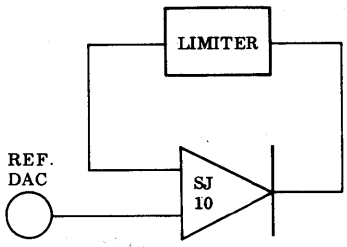
A brief description of the APM tests is given in Table I. The tests are being constantly revised and updated. Some of the difficulties encountered in a few of the tests may give some insight into problems involved in writing APM tests for analog computer units.

The servo potentiometer test (Test 23, Table I) could hardly be simpler. In the initial test, each of the 144 servo potentiometers was set to values ranging from .005 to .9950. For example, Potentiometer 1 would be set to .005, Potentiometer 2 would be set to .0100 etc. This method produced two unforeseen difficulties. If the tests were run more than once, all of the potentiometers would be set correctly as a result of the first test and therefore subsequent tests would not exercise the system.

The second fault was that each potentiometer was checked at the same point each day. The potentiometers would pass this test but fail to set properly at some other point. This problem could have been solved by setting each potentiometer to several settings but this would cause extra wear on the servo potentiometers and the time for performing the test would be too long. To overcome these problems a revision was incorporated that set each potentiometer once from a random number table.

Originally no automatic tests were performed on the limiters. They were considered troublefree and automatic testing was not considered necessary. Manual checks were performed once a week. When downtime due to bad limiters became a problem, the APM test shown in Figure 8 was added. The limiters have a maximum set value of 115 volts. Measuring two points above this value and calculating slope, gives a fair indication of limiter operation that is independent of the actual limiter setting. This test has solved the limiter downtime problem.

The resolver test, Figure 9, is performed by setting the servo potentiometer at several discrete points and



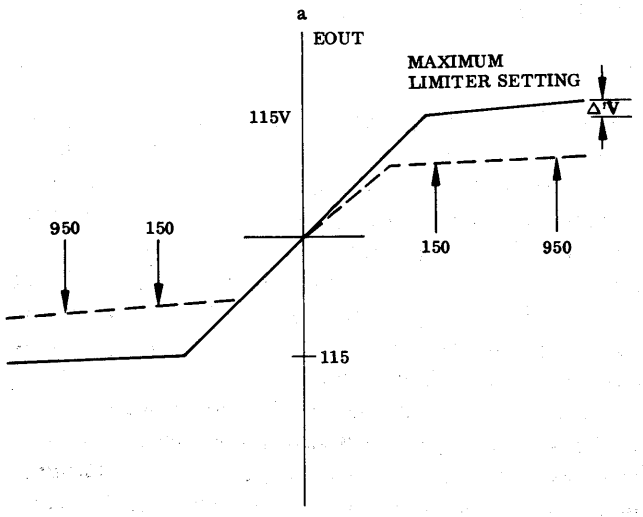


FIGURE 8—Mechanization of limiter test

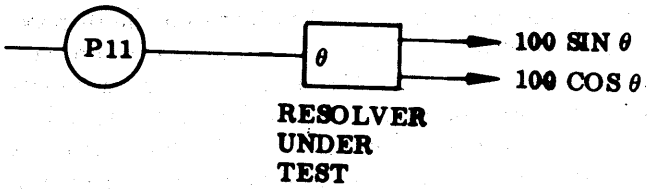


FIGURE 9—Mechanization of resolver test

reading the resolver outputs with a DVM to determine resolver accuracy. The accuracy of the servo potentiometer is ±20MV and the DVM is ±30MV. In addition, the resolver blows up the input error.

Using the output 100 Sin θ as an example of this

$$d(100\ Sin\ \cdot) = 100\ Cos\ \theta\ d\theta \qquad (2)$$

$$\text{The scaling is } 28.65\ \frac{\text{VOLTS}}{\text{RADIAN}}$$

Therefore: $\dfrac{dV}{d\theta} = 28.65$

And $d\theta = \dfrac{dV}{28.65}$

Therefore:

$$100\ Cos\ \theta d\theta = 100\ Cos\ \theta\ \frac{dV}{28.65}$$

$$= 3.48\ Cos\ \theta dV$$

at $\theta = 0$

$$= 3.48\ dV\ .$$

Therefore near zero the error in the Sin output is

$$\text{Error} = 3.48\ \Delta V + \text{DVM error} + \text{UNIT error.}$$

With the errors shown above, the error in the output is 99.6MV plus the unit error. To get meaningful test results this would have to be reduced to 5 to 10 MV maximum error.

Converting the multiplier test from manual to an APM test demonstrates some of the problems encountered in automating a test. The APM test now used is essentially the same as the resolver test with servo set potentiometers providing the x and y inputs. The error on the output is

$$X\Delta Y + Y\Delta X + \text{DVM error} + \text{UNIT error} \qquad (3)$$

Where ΔX and ΔY are INPUT errors.

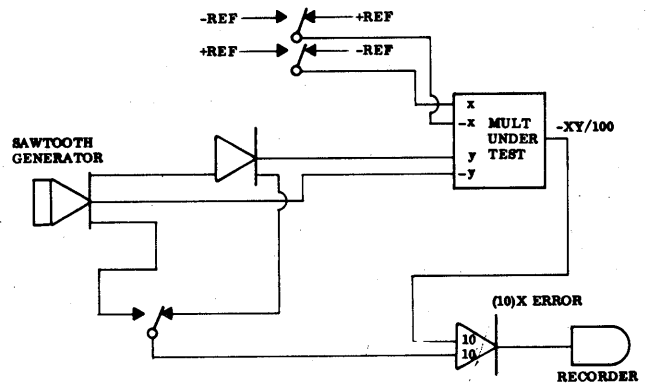


FIGURE 10—Mechanization of the manual multiplier test

Figure 10 is the mechanization for an excellent manual test that is accurate and relatively fast. It would have been very desirable to convert this test to automate the printout, so that only excessive errors were printed out. To test the ninety multipliers in each analog computer in a similar fashion would require mounting stepper switches on the analog patchboards to reduce the number of outputs to twenty-four channels or less. If this were done, the ADC channels could be used to read out the error detector circuits. This may be the best technical approach but so far our efforts have been concentrated on improving the accuracy in the present APM test. Although the stepper switch approach would solve the multiplier test accuracy problems, it would not help on the resolvers and other tests where accuracy is a problem. A more general fix is desirable.

The ADC-DAC tests are an example of how a simple test can grow. The original APM test tested the DACS for accuracy. Then set the DACS to various voltages. Each DAC was then used as an input to an ADC channel. All ADC channels were scanned one thousand times and a histogram was printed out. It was impossible by looking at the test data to isolate troubles. To determine static calibration accuracy tests a test was added that tested all ADC channels at 31 points where all inputs were the same. This separated the problems into static accuracy and dynamic switching problems. Further problems were encountered in the proper addressing of ADC channels and with the multiplexer switching. To isolate these problems another test was added that rippled a zero through all DAC-ADC channels then rippled 90 volts through all channels.

Troubles still persisted in the ADC area so the ramp test was added. This test is described in the summary under tests 16, 16A and 16B. This test checks 20 DACS and 16 ADC channels at 2^{15} points. A printout of the possible errors on the line printer would require 16×2^{15} printouts. Using the strip recorders is an efficient method compressing all of this data. In addition, it allows the one way test of the DACS.

The integrator rate test essentially integrates a known voltage for a known period of time repeating this for each time scale. The measurement error in the integrator rate test ignoring drift is:

$$\text{Measurement error} = (\Delta VT + \Delta TV)\,K + \text{DVM ERROR} \quad (4)$$

Where:

$$K = \frac{1}{C}$$

C = Integrator capacity in UF
V = Input voltage
T = Time of integration
ΔV = Error in input voltage
ΔT = Error in time of integration

In order to eliminate software timing errors, the timing for this test is generated on the analog computer logic board where counters are initiated by discrete lines from the digital computer. The counter is clocked at a 100KC rate. The input voltage comes from the Reference DAC, the most accurate voltage source on the analog computer.

Another problem area of the integrator rate test is that time scale selection relays are welded if the time scale is changed in any mode but POT SET and can be welded then, if large charges remain on the integrating capacitors. To guard against this, fail safe logic has been incorporated to prevent time scale changing except under the proper conditions.

Advantages of phase I APM tests

Although the above account sounds bleak, the APM tests are useful. Some of the advantages of the APM tests that have been verified in the last year of operation are:

Guaranteed Checkout A printed record of each test including the date and time the test was performed insures that the test was performed. An imperfect automatic test done in exactly the same way each night is better then a perfect manual test improperly performed.

Speed The large volume of tests performed and recorded could only be done under digital control. More units are checked in more ways than can be done manually.

Only Method For many of the tests there is no method to perform the tests manually and consequently the

test must be done under digital control. All hybrid and Intracom tests fall in this category.

Reduced Drudgery Most PM tests are repetitive, routine, and tedious when performed manually. Digital computer control performs this part of the job better than the technician and allows the technician to use his energies on the troubleshooting part of his job.

System Usage The tests use both hardware and software in the same manner that a customer does.

Flexibility The test programs can be modified with minor wiring changes on the test patchboards.

Low Downtime The downtime of the system for 16 hour per day operation is very low. This is due to many reasons, but the APMs are definitely a contributing factor.

Shortcomings of phase I tests

The major shortcomings of the APM program is in the analog computer area. The analog computer as delivered is not designed for APM. The following are the major shortcomings:

Accuracy The readout using the analog computer DVM is only accurate to .01% ±10MV. System noise adds an additional 10MV of uncertainty therefore ±30MV is the best that can be obtained under general conditions. Potentiometer settings which are used as inputs on some tests can have errors of ±20MV. A large number of units are tested in parallel, consequently the potentiometer outputs may go through several relays and amplifiers before becoming the input to the unit under test. The accumulation of all these errors can be several times the specifications of the unit under test.

Frequency Response The analog computers as furnished have no means of automatically making frequency response tests of the different analog units. To do this would require some method of generating frequencies of interest and measuring their relative amplitude.

Noise The present tests cannot detect high noise levels. Although noise may give a high DVM reading, this is not a reliable noise test. Hardware to do this is readily available.

Manual Tests The APM tests must be supplemented by manual tests to overcome the above listed shortcomings.

Manual Backup Manual backup tests are required for the analog computers. These can be run with a digital computer down. This allows an all analog computer problem to be run.

Future goals

Several modifications are under way to improve the Phase I APM tests. The major emphasis will be to eliminate the shortcomings listed above.

In order to improve accuracy, the following is being done:

Multiple DVM Readout Instead of reading the DVM once, the DVM will be read 4 to 5 times and the readings will be averaged. This will provide an improvement in accuracy of the DVM readings.[2]

Precision voltage sources

Two programmable remote control voltage dividers are being installed. These will provide two voltages on each analog computer accurate to ±2MV. This will provide more accurate input voltages and will allow nulling techniques to be used on the tested units.

Frequency generators and peak reading detectors

The addition of these two units will allow automatic frequency response and noise tests of the various computing units.

With these improvements installed, a complete check of all units in the analog computer to manufacturer's specifications can be performed each night.

APM cost and effectiveness

The cost of the hybrid system is approximately $500 an hour for an 80 hour week. Downtime of 1¼% or 1 hour per week costs $500 per week or $26,000 per year. Over the five year expected life of the equipment this amounts to $130,000 total cost. The cost of the APM programs to date is approximately 1½ years of programming time plus 12 checkout boards. It is felt that the expenditure to date for the APM will reduce downtime over the five year period enough to return the APM costs many times over.

The present downtime in the hybrid system is approximately 5%. This places an upper bound on the increment of improvement that could be obtained by future expenditures on the APM program. The system with infinite maintenance has a minimum downtime figure that can be attained because of limits in reliability and maintainability factors. This figure appears to be approximately 2%. These cost factors influence the amount of future investment that will be placed in the APM program.

SUMMARY

The APM testing of large hybrid computers is neces-

sary and feasible. The large volume of equipment and the high cost of downtime make a "complete" check out necessary. This can only be done by utilizing a digital control checkout. The shortcomings in the present APM program especially in the analog computer area can be overcome. Close customer-vendor cooperation could produce very efficient general purpose APM programs for hybrid systems that would reduce acceptance costs, long term maintenance costs, and downtime costs.

ACKNOWLEDGMENTS:

Douglas Theis and John Casaccia of Astrodata/Comcor programmed and patched the original APM tests. Much of the software work at LMSC has been done by Kenneth Bedient, Roger Hollingworth, David Lee and Stanley Ross. The test ideas are the combined ideas of all the people associated with the hybrid lab at LMSC and vendor representatives.

REFERENCES

1 T K SEEHUUS
Hybrid computer techniques to aid computer maintenance
SIMULATION Vol 7 Number 5 p 231

2 K A KORN
Hybrid computer techniques for measuring statistics from quantized data
SIMULATION Vol 4 Number 4 p 229

APPENDIX A

Intracom

An Intracom is a device which interfaces analog computers with the digital computer. There are two Intracoms in the LMSC system. Each Intracom is divided into three controllers as follows:

I/O channel controller
Interrupt channel controller
Analog computer controller

Each controller has its own 6400 channel for digital communication. The I/O channel controller is further divided into six subsystems as follows:

I/O interrupt subsystem—contains 24 real-time priority interrupts.
Discrete subsystem—contains 68 discrete read lines and 68 discrete write lines.
D/A subsystem—contains 40 double-buffered digital-to-analog converters.
A/D subsystem—contains an analog-to-digital converter with 32 input channels, plus 8 independent sample/hold controllers.
P.I.G. subsystem—contains a precision interval generator (real-time counter).
Intracom mode subsystem—contains logic for slaving and controlling the modes of the Intracom and analog computers.

The interrupt channel controller contains logic for handling the 24 interrupt lines for processing by the hybrid monitor program. These are the same 24 interrupt lines that are mentioned in the preceding I/O channel controller description.

The analog computer controller contains logic for the following interface with the analog computers:

Setting or reading a mode.
Setting or reading an address register.
Reading a DVM
Setting a reference DAC.
Outputting a CI-5000 BCD code.
Reading status conditions of the analog computer.

Each Intracom also contains a patchable logic group containing the following logic elements:

36 general-purpose counters.
72 general-purpose flip-flops.
12 delay flops.
48 five-input NAND-Gates.
180 two-input NAND-Gates.
72 inverters.
48 general-purpose indicators.
24 function switches.

Analog computer

There are four CI-5000 analog computers in the system. Normally two analog computers are connected to each Intracom. Each analog computer has the following components:

60 integrators (with derivative check feature). All have 4 time scale
60 summing amplifiers.
48 hybrid operated digital attenuator devices. (HODAD'S*)
144 servo-set potentiometers.
32 hand-set potentiometers.
3 continuous resolvers.
90 multipliers, of which 24 are associated with resolvers.
288 analog trunk lines (24 are D/A, 24 are A/D).
44 inverters associated with multipliers.
22 card-set function generators.
1 card-set surface generator.
24 relays (DPDT).
30 comparators.
30 hand-set feedback limiters.

*Trademark

16 function switches (8 are SPTT, 8 are DPTT).
3 recorders.
1 noise generator.
1 oscilloscope.
144 logic trunk lines and associated line drivers.
32 general-purpose flip-flops.
9 delay flops.
24 four-input NAND-Gates.
24 two-input NAND-Gates.
24 logic inverters.
4 four-bit counters.
4 four-bit shift registers.
24 lamp drivers.
8 logic switches.
3 clocks.
1 cycle counter.
24 holes on analog patchboard for ADC channels.
24 holes on analog patchboard for DAC's.

Hybrid diagnostic techniques

by THOMAS K. SEEHUUS and WILLIAM A. HARMON

Boeing Company
Huntsville, Alabama

and

WILLIAM MAASBERG

International Business Machines
Huntsville, Alabama

INTRODUCTION

The Automated Maintenance Procedures at the Boeing Company's Huntsville Hybrid Facility were the result of a joint development effort by the Boeing Company and International Business Machines. (IBM). The analog and linkage portions are the responsibility of Boeing while the Digital System is maintained by IBM.

The combined effort has resulted in highly efficient Diagnostic Maintenance procedures that have given us a profitable degree of reliability for all elements of the Hybrid System.

Equipment configuration

The stringent requirements for diagnostics resulted from the complexity and size of our Hybird Installation. Although many requirements have changed as portions of the system have been updated, our maintenance philosophy has remained the same.[1,2] The present equipment complement of the Boeing Huntsville Hybrid System is listed in Figure 1. The major components of the system are four Applied Dynamic (ADI) analog computers, sixty channels of digital to analog conversion (DAC), fifty multiplexed channels of analog to digital conversion (ADC) and an IBM 360 Model 44 Digital Computing System.

The real time monitor currently operating on the 360/44 Hybrid Computer at Boeing Huntsville was developed jointly by Boeing and IBM almost two years ago. It was decided that some sort of system test capability was also needed to aid in debugging the software and assure a maximum level of hardware reliability.

ADI ANALOG COMPLEMENT (4 CONSOLES)

1024	OPERATIONAL AMPLIFIERS (256 SUMMER-INTEGRATORS, 256 SUMMERS, 512 INVERTERS.
640	SERVOSET POTENTIOMETERS
160	HANDSET POTENTIOMETERS
172	ELECTRONIC QUARTER-SQUARE MULTIPLIERS
128	VARIABLE DIODE FUNCTION GENERATORS
40	FIXED SIN-COS DIODE FUNCTION GENERATORS
48	FIXED X^2 DIODE FUNCTION GENERATORS
16	FIXED LOG X DIODE FUNCTION GENERATORS
80	HARD LIMITERS
112	COMPARATORS
48	TRACK TRANSFER UNITS
4	HIGH-SPEED REP OP CONTROL UNITS

LINKAGE EQUIPMENT COMPLEMENT

50	ANALOG TO DIGITAL CONVERTER CHANNELS
60	DIGITAL TO ANALOG CONVERTER CHANNELS
6	DISCRETE OUT WORDS (32 BIT)
6	DISCRETE IN WORDS (32 BIT)
32	PRIORITY INTERRUPT LINES
1	ANALOG CONTROL CHANNEL

DIGITAL COMPLEMENT (IBM)

1	IBM DIGITAL COMPUTER, 360 MOD/44
3	MAGNETIC TAPE UNITS, 9-TRACK (2401)
1	MAGNETIC TAPE UNIT, 7-TRACK (2401)
1	LINE PRINTER (1403)
1	OUTPUT PRINTER (1053)
1	CONSOLE TYPEWRITER (1052)
2	VISUAL DISPLAY UNITS (2260)
4	I/O DATA ADAPTERS (2701)

Figure 1—Equipment complement

Basically one must be able to detect a malfunction and rapidly isolate it to a specific portion of either the software or hardware. Consequently, a System Test (SYSTST) program was developed.

The SYSTST program has been very successful, both during debugging of the initial Real Time Monitor and as aid in checking out improvements. The program is

readily available to application programmers to confirm suspected hardware malfunctions. Thus, the problem areas are immediately identified as either the application program or hardware. Corrective action is then initiated. Should the problem be hardware, the evidence collected by the SYSTST program is given to either the IBM Customer Engineers or the Boeing Hybrid Operations Personnel, who conduct detailed diagnostics and tests to locate and correct the malfunction.

360/44 Hybrid System Test Program (SYSTST)

A. SYSTST philosophy

SYSTST was written to fulfill the following primary objectives:

1. Aid in initial debugging of the 360/44 BPS Hybrid Monitor System
2. Aid in maintaining and improving the system
3. Test the Real Time I/O for a basic level of operation and accuracy

A secondary objective was to develop in SYSTST a basic test framework which could be easily modified, expanded, and developed into a rigorous test of the 360/44 HBPS software and hardware.

B. SYSTST organization

Although the individual tests were written to satisfy primary objectives, the final organization of SYSTST was dictated by the secondary objectives of flexibility and expandability. Consequently, SYSTST consists of one main program and a number of test subprograms. The main program contains the operator communications routines necessary for test completion and diagnostic action as a result of error. The test subprograms are entered from and returned to the basic program, as illustrated in Figure 2.

At the present there are six test subprograms which accomplish the following tasks:

1. Test display system
2. Test real time tape, single unit
3. Test real time tapes, all units
4. Test analog I/O from foreground
5. Test interval timers
6. Test analog I/O, interval timers, and tapes in real time

SYSTST is readily capable of expansion in either of two ways:

1. Expansion and refinement of the present test subprograms

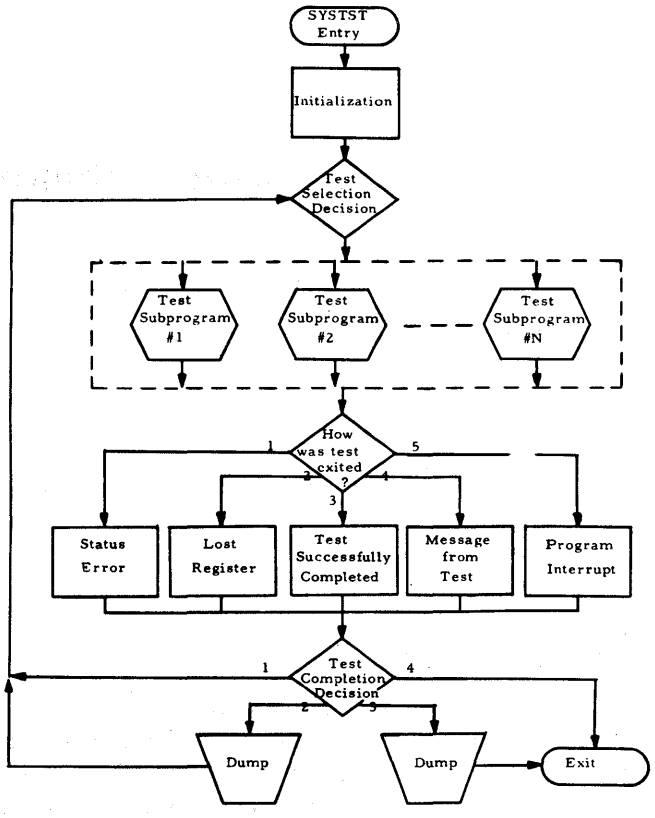


Figure 2—SYSTST flow chart

2. Addition of any number of new test subprograms

C. Diagnostics and error messages

It is readily apparent in Figure 2 that there are several basic error or diagnostic messages as well as provision for handling messages which are unique to a given test. These test exit messages are further described below:

1. STATUS ERROR

If during the course of a test, any I/O request results in an error or unusual condition indication from the I/O device, the test will be terminated, the test subprogram exited, and the following diagnostic message displayed:

STATUS ERROR

1. STATEMENT: CALL XXXXXX 2. LOGICAL UNIT: X

3. STATUS: X 4. I/O OLD PSW: XXXX XXXX XXXXXXXX

5. CSW: XXXXXXXX XXXX XXXX

2. LOST REGISTER

Each test subprogram is responsible for checking all the registers (except those used by the FORTRAN compiler) each time a call statement is made for a real time function. If a register is lost, the test is terminated, the test subprogram exited, and the following diagnostic message displayed:

REGISTER LOST DURING EXECUTION OF: CALL NNNNNN

REGS: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

LOST: XXXXXXXXXX X X X X X X

(X=0 indicates register not lost, X=1 indicates register lost. Registers 0, 1, 13, 14 and 15 are normally employed in Fortran branching and linking and, consequently, are not checked. Any additional registers used by the compiler in setting up a specific call cannot be checked.)

3. TEST SUCCESSFULLY COMPLETED

The above diagnostic message is felt to be sufficiently self-explanatory.

4. ERROR OR DIAGNOSTIC MESSAGE FROM TEST

Use of this SYSTST return path is entirely at the discretion of the test subprogram writer. This option is used whenever the programmer wants to terminate his test subprogram and idicate a unique reason, or whenever he wishes to output a unique message upon successful test completion. It should be noted that a programmer may print or display any data or messages; he may also request any operator response desired during the course of his test subprogram.

5. PROGRAM INTERRUPT

Any errors which would normally result in a program interruption and the resultant abnormal end of job (ABEOJ), will cause immediate test termination, return to SYSTST MAIN, and display of the following message:

PROGRAM INTERRUPTION

INTERRUPTION CODE: XX INSTRUCTION COUNTER: XXXXXX

D. SYSTST OPERATION

SYSTST OPERATION is relatively easy and self-explanatory, as the main program and each of the test subprograms were written with the following philosophies:

1. The operator will communicate the SYSTST only through the display system (although some test data may be output on the printer, depending on the test subprogram).
2. The operator will be kept informed of test progess.
3. The operator will be given explicit instructions when a decision and/or response is required.
4. The operator will initiate no unrequested entries, except to request a hard copy of the display (which he may do at any time by depressing the shift and print keys).
5. Any test subprogram may be entered in any order and any number of times. The cumulative numbers of entries and successful completions are displayed each time a test selection decision is requested.

Analog and linkage diagnostics

The maintenance and calibration of analog computer components and linkage equipment is accomplished by the use of Hybrid Diagnostics. The use of diagnostics for analog computer maintenance is a relatively new procedure and initially their effectiveness and ease of application were not fully realized. It was found in developing the diagnostics that most of the antiquated manual checks and calibrations could be adapted to digital control. In addition, some components that previously could not be adequately tested can now be checked under digital control.

Another result of using Automated Diagnostics is that operational tests and calibration checks require much less time, making possible more frequent testing. Figure 3 shows a comparison of the time required to run manual and diagnostic tests. The use of diagnostics enable maintenance personnel to apply more time to areas previously neglected. In addition to time savings, a thorough maintenance program can be established that does not entirely depend upon the proficiency of the maintennance personnel. An automatic check is consistently thorough and accurate.

The Hybrid Diagnostics used at the Boeing Huntsville Facility are grouped into two types: general opera-

TYPE OF CHECK	MANUAL CHECK MINUTES	DIAGNOSTIC CHECK MINUTES
STATIC CHECK	60	3
INTEGRATOR RATE	60	2
INTEGRATOR DRIFT	120	2
COMPARATOR GAP	30	1
MULTIPLIER	60	1
NON-LINEAR	120	8

Figure 3—Comparison of diagnostics

tion and calibration. General operation diagnostics check the overall operation of equipment that requires no calibration, such as logic devices. The calibration diagnostic is capable of checking accuracy as well as general operation. The calibration diagnostic can be used, in many cases, to aid the technician performing calibration or repair.

The analog and linkage diagnostics have been written as subroutines of a main calling program. Each diagnostic need not be loaded separately when going from one to another. Figure 4 is a flow chart of the diagnostics. The diagnostic subroutines and calling program reside on magnetic tape in core format. They are loaded in approximately five seconds, after which they are accessible from the IBM 2260 Display Unit. Figure 5 is a photograph of the 2260 displaying the program options. Following are descriptions of the analog and linkage diagnostics.

Non-linear preventive maintenance diagnostic

Basic observations

Our approach is based upon the contention that all non-linear components have one thing in common: they produce a predetermined non-linear function when they receive a linear or ramp function as an input. Therefore any non-linear device may be tested using a highly accurate ramp synchronous with a highly accurate predetermined non-linear function. This enables one to test all members of the non-linear family as relatively simple functions.

The ability to generate a ramp function synchronous with a desired non-linear function became the key to a successful diagnostic test. This capability is available in the digital portion of our hybrid system, but the conversion equipment does not meet the accuracy requirements for specification testing. At this point we perceived the need for two digital to analog converters of high accuracy.

Design objectives for high accuracy DAC

A. Accuracy equal to or better than .005%
B. An integral part of our present hybrid system
C. Ease of maintenance and calibration
D. Controlled manually or automatically

The high accuracy DAC shown in Figure 1 met all design objectives. It requires 16 bi-polar amplifiers, 1 bi-polar integrator, 15 electronic switches, 2 potentiometers, and 3 free 100 K resistors. Fifteen bits plus sign was selected because this DAC has the accuracy required. We did not go to sixteen bits because the least significant bit of a sixteen bit DAC is well within the noise level of our system.

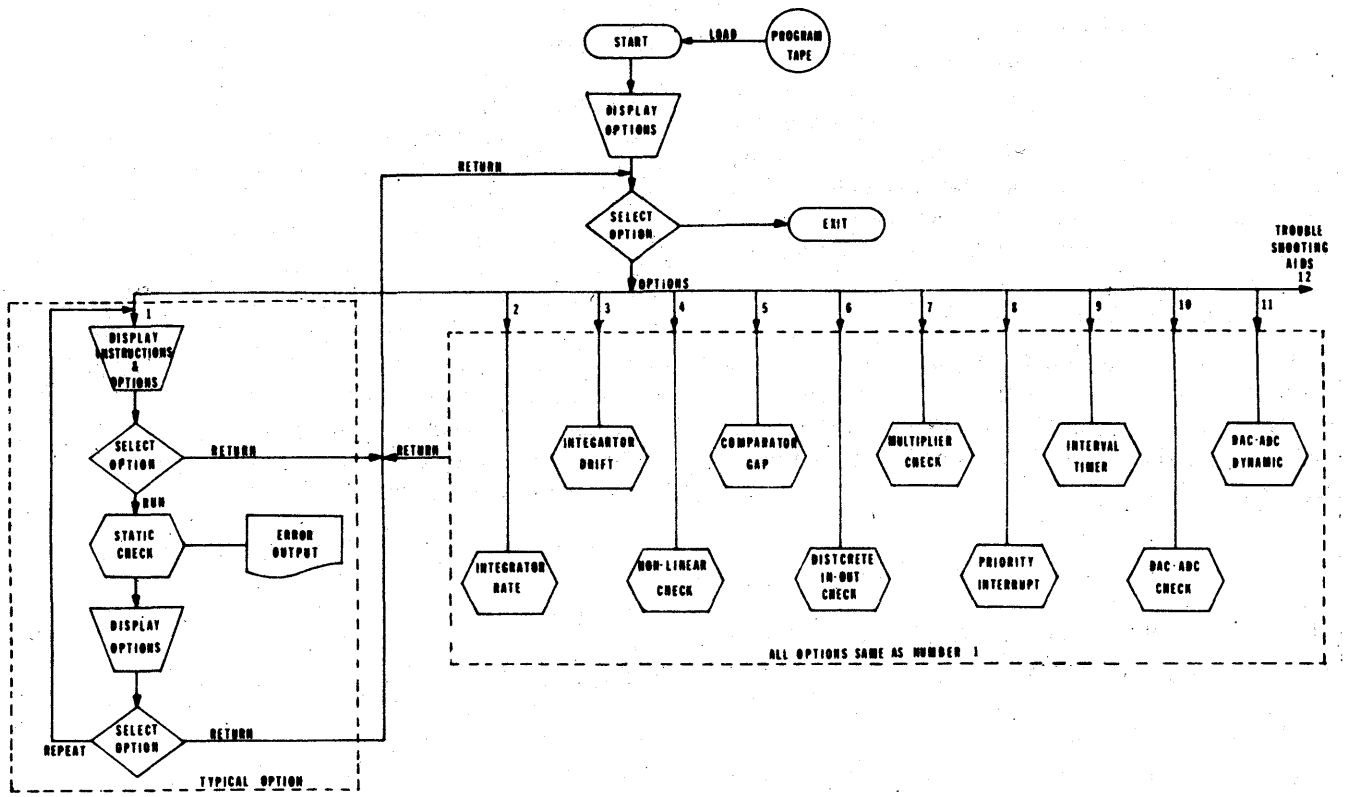


Figure 4—Diagnostic flow chart

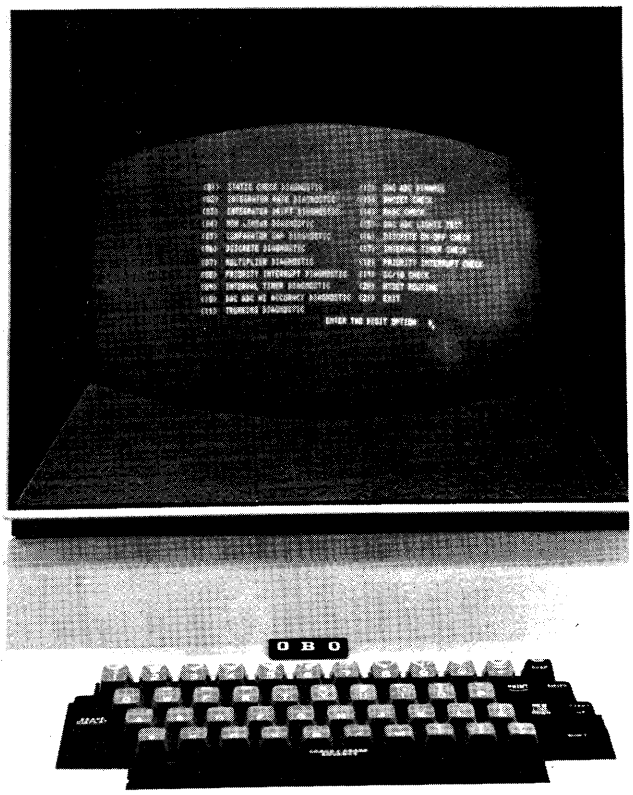


Figure 5—2260 display

The high accuracy DAC operates as follows: The plus volt reference is applied to the first summer on the left. Plus 50 volts is available at the input to electronic switch number one and is halved at the input to each succeeding electronic switch. The electronic switches may be operated manually at the analog console or by discrete lines from the digital. Each electronic switch has a 100 K resistor that is used as an input resistor to the summer at the top of Figure 6. The correct polarity of voltage is observed throughout the string by the use of bi-polar amplifiers. The negative output of the upper summer is fed into a gain of one, while the positive output determines the initial condition of the integrator. The sign bit of the digital to analog conversion determines whether the integrator is in initial condition or operate. If the sign is positive, the integrator will be in I C, with a negative sign in operate. The integrator is in times 100 mode to impart a filtering effect. This limits frequency response but keeps the DAC within design specifications.

The plus and minus 100 volt adjustments are necessary to compensate for the slight inaccuracies of the various components. They are adjusted in the following manner:

1. Throw all bits except the sign bit on
2. Set the +100 v adjustment to +99.997 volts

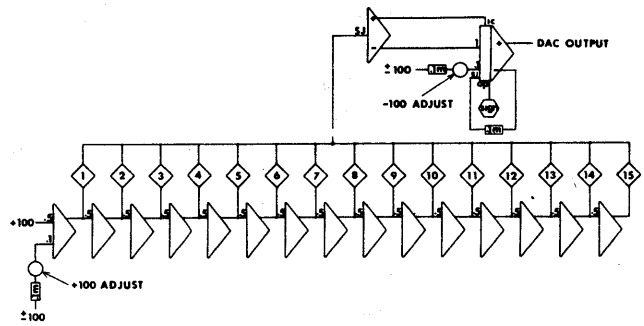


Figure 6—High accuracy DAC

3. Throw sign bit on
4. Set −100 v adjust to −99.997 volts

Caution: The plus 100 adjustment must be set first.

The accuracy of the DAC was verified in the following manner: the DAC test was conducted using the technique and equipment shown in Figure 7. The DAC is the input to one side of a null meter with a calibrated test voltage the other input.

The bit by bit error plot shown in Figure 8 verified

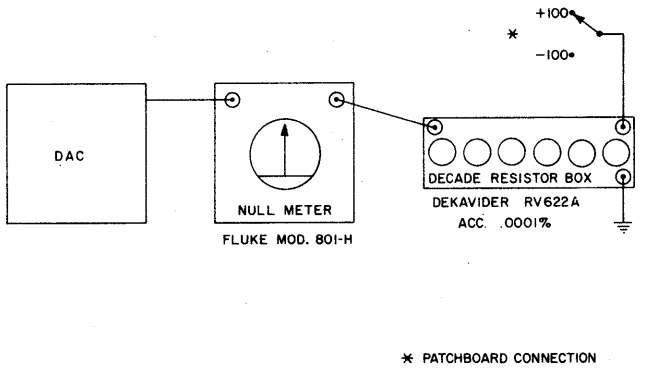


Figure 7—DAC test configuration

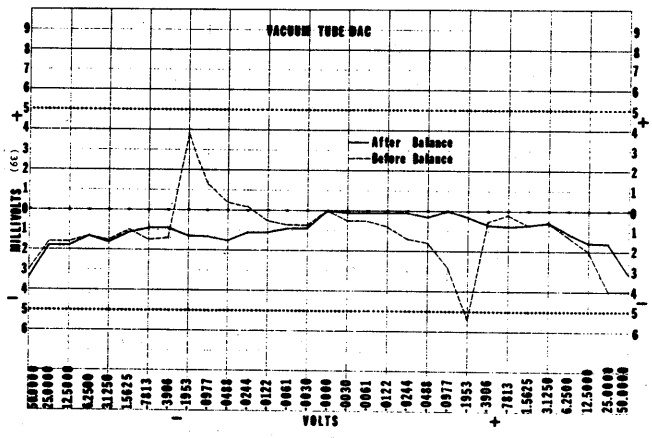


Figure 8—Vacuum tube DAC (Balance)

that our design specifications could be met on a vacuum tube analog after a careful balance of all amplifiers.

The bit by bit error plot shown in Figure 9 was taken on a different analog console after a complete balance of amplifiers, in addition to setting the plus and minus 100 v adjustments.

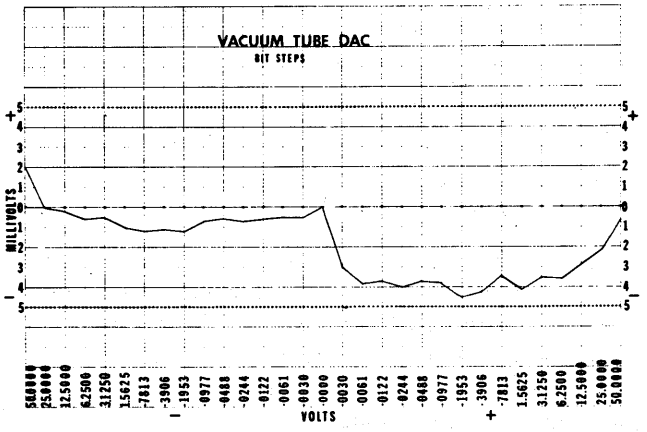


Figure 9—Vacuum tube DAC (Bit steps)

The plot shown in Figure 10 is an error plot in 10 volt steps from +100 v to—100 volts. This also falls within design specifications. Figures 11 and 12 were taken on a solid state console without balancing the amplifiers. The plus and minus 100 v adjustments were made, however.

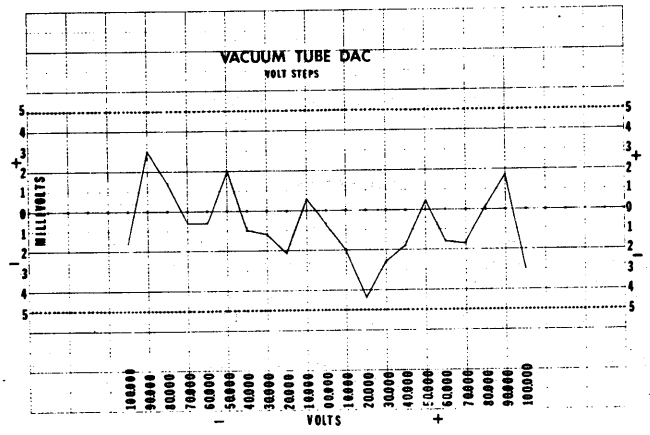


Figure 10—Vacuum tube DAC (Volt steps)

The preceding tests have given us confidence that the DAC will meet accuracy requirements on any well-maintained console.

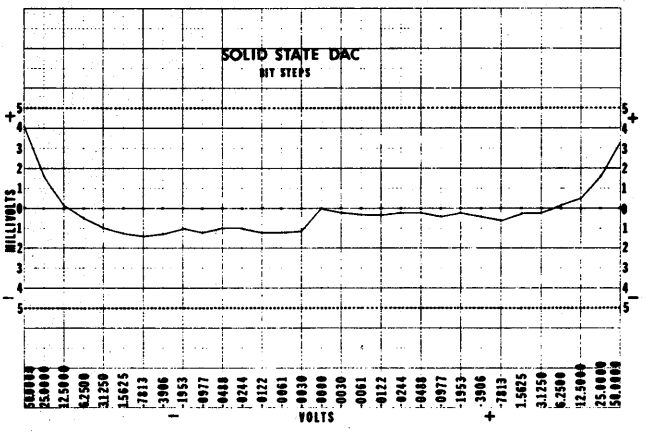


Figure 11—Solid state DAC (Bit steps)

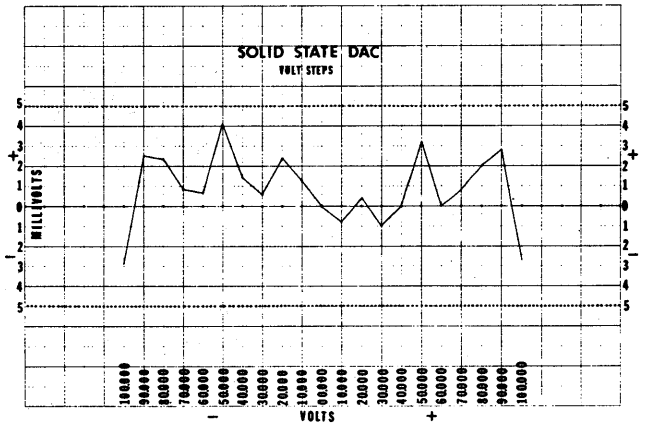


Figure 12—Solid state DAC (Volt steps)

Testing non-linear components

The test configuration shown in Figure 13 is the method used for daily checks. During the development stages the output of the error amplifier was plotted together with plots of desired function and the actual

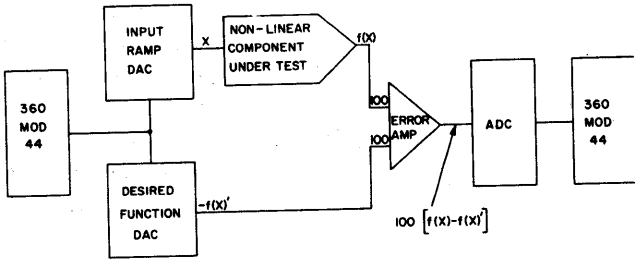


Figure 13—Non-linear test configuration

function. The first tests conducted were on an X^2 card. The results of these tests are shown in Figure 14. The plot took three hybrid runs. On the first run the desired curve was plotted, the second was a plot of the actual output of the X^2 card, and the third, a plot of the error between the two.

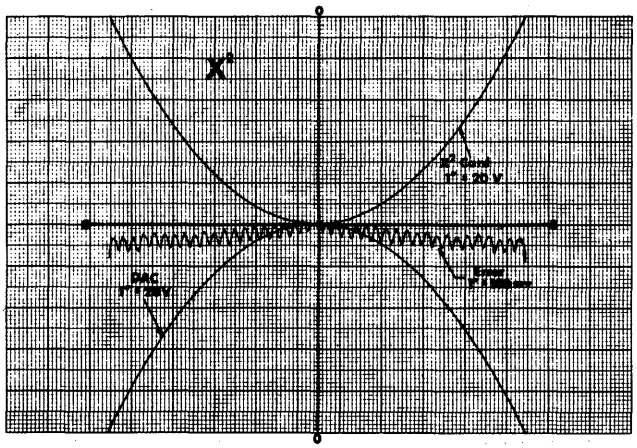


Figure 14—X^2 plot

Our next step in the verification on the testing procedures was to repeat our previously described plotting and testing for other members of the non-linear family. Figures 15, 16 and 17 show the results. All plots shown were generated using a ramp input to the tested non-linear device in 0.5 volt steps.

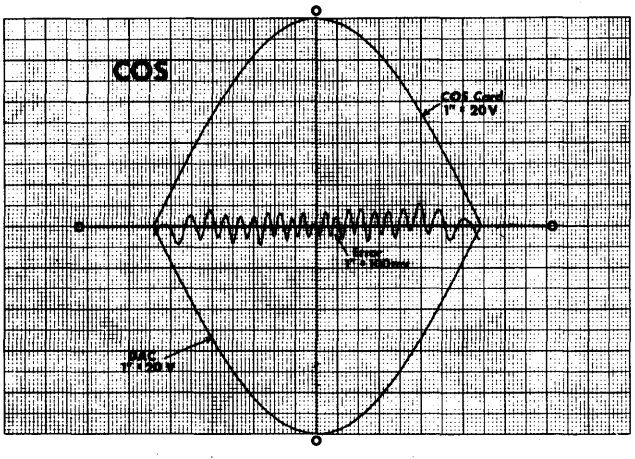


Figure 15—Cosine plot

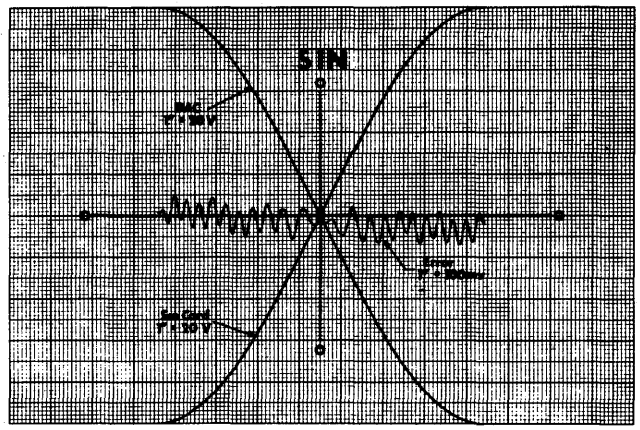


Figure 16—Sine plot

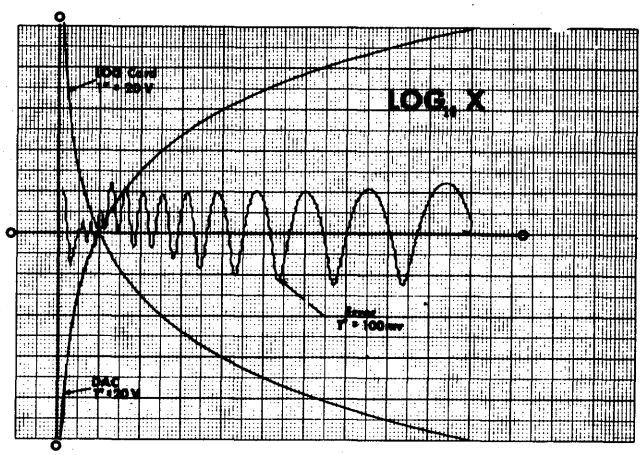


Figure 17—Log plot

Diagnostic technique

The program implements two of the highly accurate DAC's, one to provide a comparison curve (Figure 13). The outputs of the non-linear devices are summed with the comparison curve (opposite polarity into 100 gain amplifiers). The amplifier outputs, which represent 100 times the non-linear device error, are connected to ADC's. The following steps describe the method used to check Sine, Cosine, and Logarithmic devices:

1. First the circuitry and components used to test non-linear devices are checked by the program to insure proper operation. In the event of error, a pause results that allows the operator to investigate prior to program excution.
2. Error amplifiers associated with non-linear devices under test are enabled by descrete logic.
3. The input ramp is stepped from—100 to +100 volts, in one-half-volt increments and is applied to the non-linear device under test. At each increment

the program uses the value of the ramp to calculate the value of the comparision curve. Each ADC is read at every increment and the error magnitude is compared to the previous reading. The larger of the errors is saved.

4. Following the last one-half-volt increment, the largest error reading for each non-linear device is compared to that device's tolerance. Out-of-tolerance readings are recorded together with the address of the non-linear device and error amplifier.

The non-linear diagnostic program has an option for continuous operation. The ramp and comparison curve are operated continuously with the 100 gain amplifier outputs observed on an oscilloscope or recorder. The diagnostic may be used to analyze errors and perform necessary calibration or repair.

Total run time for the non-linear diagnostic program is approximately eight minutes.

The settling time, and therefore the frequency response of the high accuracy DAC, limits our testing to very low frequencies. We found no way to overcome this deficiency. The use of the integrator as the output amplifier was required to keep the noise level of the DAC within the 5 millivolt design specification. This problem should be rectified in the future and dynamic testing of nonlinear components using the method described will indeed become a reality.

Multiplier preventive maintenance diagnostic

The objective of the Multiplier Diagnostic Program is to insure that all multipliers function within their specified tolerance. The analog boards used in this diagnostic may be used manually to analyze errors and make necessary calibrations. All multipliers in an analog console may be checked simultaneously by the following steps:

1. A ramp varying between—100 and +100 volts is applied to the X input of all multipliers. A constant—100 volts is supplied to the Y input. Multiplier outputs are summed with the inverted ramp (+100 volts to—100 volts) by means of 100 gain amplifiers. The amplifier outputs, which represent 100 times multiplier error, are connected to ADC's.
2. The ADC's are continuously read as the input ramp varies from —100 volts to +100 volts. A comparison of the averages of selected readings is made and the largest average is saved. When the ramp ends, the X input and Y input to the multipliers are exchanged under program control. The ramp is restarted and the same procedure is used to obtain the largest average reading.
3. The largest average reading for each multiplier is compared to a predetermined tolerance. Out-of-tolerance readings for X or Y inputs are recorded together with the multiplier and error amplifier address.

The above check may be used manually by controlling the exchange of inputs with analog push buttons. The ramp is allowed to run continuously and the 100 gain amplifier outputs may be observed on an oscilloscope or a recorder. Total run time for Multiplier Diagnostic Program is approximately one minute.

Static preventive maintenance diagnostic

The Static Diagnostic Program will check the following equipment on an analog console:

1. Every input and output on all operational amplifiers
2. All servo set potentiometers for proper operation
3. All analog trunks
4. Mode control logic
5. The analog addressing system
6. All reference and electronic switches
7. Hand set potentiometer fuses

Permanently wired patchboards are used in conjunction with the Static Diagnostic to check the above components.

The Static Diagnostic requires three minutes hybrid time. To isolate bad components the computer modes and addressing are controlled manually. The actual value of the suspected components is read on the console digital volt meter, and appropriate action is taken using conventional trouble shooting techniques.

Integrator rate preventive maintenance diagnostic

The Integrator Rate Diagnostic is used to check all time scales as well as relay and electronic switching. The programming sequence is described in the following steps:

1. The integrators are allowed to integrate at one volt per second (Xl) for eight seconds. The integrators are then switched, via relays, to hold. All integrators are checked for the proper reading and if they are in error by more than a predetermined amount, the address, actual reading, specified value, and calculated error are recorded.
2. The above step is repeated for all additional time scales. (X10, X100, X1000)
3. Electronic switching is checked using the X10 time scale by the procedure outlined in step one.

Total hybrid time for the Integrator Rate Test is two minutes.

Integrator drift preventive maintenance diagnostic

The objective of the Drift Diagnostic is to periodically check all integrators for excessive drift. Drift is checked in both operate and hold using the X100 capacitor. While the integrators are being checked, they continue to drift. To prevent erroneous data, each integrator's drift rate is computed individually with respect to time.

Comparator gap preventive maintenance diagnostic

The Comparator Gap Diagnostic determines the input value at which the comparators are switched. These values are used to calculate gap and offset. Permanently wired patched boards are maintained with the following configuration: All comparators have their logic-one outputs tied to the hold control line of integrators. The associated integrators are input by a control integrator. The control integrator is also patched to a one-thousandth gain amplifier, whose output is the input to all comparators. The hybrid test stores the voltages at which the integrators are placed into hold. These recorded values are used to compute the gap and offset. The comparator address, on value, off value, and offset are recorded.

The total hybrid run time for the Comparator Gap Diagnostic is one minute.

Discrete preventive maintenance diagnostic

The objective of the Discrete P. M. program is to insure that each Discrete-out and Discrete-in bit is properly aligned and is capable of representing both logic levels. This objective is accomplished in the usual manner by trunking Discrete-out Units to corresponding Discrete-in Units. The digital computer then sets each Discrete-out bit and checks that the corresponding Discrete-in bit is on. The majority of time required for the completion of the Discrete P. M. program is a function of the amount of print-out. Total time is normally less than ten seconds.

Priority interrupt preventive maintenance diagnostic

The objective of the Priority Interrupt Diagnostic is to insure that all available priority interrupt lines will provide the desired number of interrupts at the proper priority level. There are eleven priority interrupt lines available which supply the capability of interrupting the computer program on one of eleven discrete levels of interrupt. A logic-one on any priority interrupt line will cause an interrupt request. This request will be honored immediately, if none of the following conditions exist:

1. An equal or higher priority interrupt still has control of the computer program.
2. The requested level of interrupt is temporarily disabled because the system is executing some nonreentrant code.
3. The request level of interrupt was never enabled by the program.

The Priority Interrupt Diagnostic checks the individual priority interrupt lines in the following steps:

1. The program enables all available priority interrupt lines. The seven highest level interrupts are trunked to seven discrete-out lines. The remaining four interrupt lines are hard wired to the overflows of interval timers and the end of conversion of the ADC's.
2. The program sets the seven discretes to a logic-one, starts the timers, and reads the ADC's.
3. The program analyzes the order in which the lines were serviced and checks that each line was serviced only once. Any error in the order or any multiple servicing of the interrupt lines will result in the recording of the improper line.
4. The program applies a signal to the first priority interrupt line and checks that the proper level of interrupt occurs only once. The remaining lines are checked in the same manner. Any error will result in the recording of the improper line.

Total run time for the Priority Interrupt Diagnostic is less than one second.

Interval timer preventive maintenance diagnostic

The Interval Timer Diagnostic provides a means for checking both interval timers in their four modes of operation. These modes provide a means for setting the timer register to a particular time which will provide a priority interrupt at the end of the desired interval (overflow).

The following is a description of the four modes of operation:

1. Upon selection of mode one, the desired time is immediately loaded into the timer register, regardless of the timer state. The timer will then stop after the desired interval has elapsed.
2. The selection of the second mode will cause the time register to be loaded with a new value immediately upon overflow. The new value will automatically be reloaded at subsequent overflows.
3. The third mode causes the timer register to be loaded immediately with the desired time interval, regardless of the timer state. The new value will automatically be reloaded at subsequent overflows.
4. The selection of the fourth mode results in an immediate stop of the timer.

Integrator drift preventive maintenance diagnostic

The objective of the Drift Diagnostic is to periodically check all integrators for excessive drift. Drift is checked in both operate and hold using the X100 capacitor. While the integrators are being checked, they continue to drift. To prevent erroneous data, each integrator's drift rate is computed individually with respect to time.

Comparator gap preventive maintenance diagnostic

The Comparator Gap Diagnostic determines the input value at which the comparators are switched. These values are used to calculate gap and offset. Permanently wired patched boards are maintained with the following configuration: All comparators have their logic-one outputs tied to the hold control line of integrators. The associated integrators are input by a control integrator. The control integrator is also patched to a one-thousandth gain amplifier, whose output is the input to all comparators. The hybrid test stores the voltages at which the integrators are placed into hold. These recorded values are used to compute the gap and offset. The comparator address, on value, off value, and offset are recorded.

The total hybrid run time for the Comparator Gap Diagnostic is one minute.

Discrete preventive maintenance diagnostic

The objective of the Discrete P. M. program is to insure that each Discrete-out and Discrete-in bit is properly aligned and is capable of representing both logic levels. This objective is accomplished in the usual manner by trunking Discrete-out Units to corresponding Discrete-in Units. The digital computer then sets each Discrete-out bit and checks that the corresponding Discrete-in bit is on. The majority of time required for the completion of the Discrete P. M. program is a function of the amount of print-out. Total time is normally less than ten seconds.

Priority interrupt preventive maintenance diagnostic

The objective of the Priority Interrupt Diagnostic is to insure that all available priority interrupt lines will provide the desired number of interrupts at the proper priority level. There are eleven priority interrupt lines available which supply the capability of interrupting the computer program on one of eleven discrete levels of interrupt. A logic-one on any priority interrupt line will cause an interrupt request. This request will be honored immediately, if none of the following conditions exist:

1. An equal or higher priority interrupt still has control of the computer program.
2. The requested level of interrupt is temporarily disabled because the system is executing some nonreentrant code.
3. The request level of interrupt was never enabled by the program.

The Priority Interrupt Diagnostic checks the individual priority interrupt lines in the following steps:

1. The program enables all available priority interrupt lines. The seven highest level interrupts are trunked to seven discrete-out lines. The remaining four interrupt lines are hard wired to the overflows of interval timers and the end of conversion of the ADC's.
2. The program sets the seven discretes to a logic-one, starts the timers, and reads the ADC's.
3. The program analyzes the order in which the lines were serviced and checks that each line was serviced only once. Any error in the order or any multiple servicing of the interrupt lines will result in the recording of the improper line.
4. The program applies a signal to the first priority interrupt line and checks that the proper level of interrupt occurs only once. The remaining lines are checked in the same manner. Any error will result in the recording of the improper line.

Total run time for the Priority Interrupt Diagnostic is less than one second.

Interval timer preventive maintenance diagnostic

The Interval Timer Diagnostic provides a means for checking both interval timers in their four modes of operation. These modes provide a means for setting the timer register to a particular time which will provide a priority interrupt at the end of the desired interval (overflow).

The following is a description of the four modes of operation:

1. Upon selection of mode one, the desired time is immediately loaded into the timer register, regardless of the timer state. The timer will then stop after the desired interval has elapsed.
2. The selection of the second mode will cause the time register to be loaded with a new value immediately upon overflow. The new value will automatically be reloaded at subsequent overflows.
3. The third mode causes the timer register to be loaded immediately with the desired time interval, regardless of the timer state. The new value will automatically be reloaded at subsequent overflows.
4. The selection of the fourth mode results in an immediate stop of the timer.

The timer modes, registers, and priority interrupts are checked in the following manner:

1. The program checks the first and fourth modes of operation by using them to check each bit of the timer register. This is accomplished by setting the register's highest order bit, stopping the timer, and recording the contents of the register. The highest order bit is then turned off and the next lower order bit is turned on. The contents of the register are recorded. This procedure continues until all bits have been checked.
2. The program checks the third mode of operations by using it to load the timer register with a known value. After the timer overflows, the value reloaded into the register is recorded. This cycle is continued for two additional overflows to insure that the values loaded at each overflow remain the same.
3. The last overflow, resulting from step (2), changes the value in the timer register under control of the second mode. After the register is loaded, its contents are recorded. The register value is recorded for two additional overflows. The proper operation of the second mode is assured if the register values are identical and have been changed from step (2).

Through the above steps the priority interrupts caused by interval timer overflows are checked for proper sequencing.

The total run time for the Interval Timer Diagnostic is less than one second.

DAC—ADC preventive maintenance diagnostic

A complete analysis of the operation of digital to analog converters and the analog to digital converters requires the use of two diagnostic programs. One program is a static check of each magnitude bit to insure proper operation and value. The other program is a dynamic check to reveal any intermittence in overall operation or crosstalk. The following is a description of these programs:

Static diagnostic

DAC

The Static Diagnostic uses the digital to analog converter, located within an analog console, for a standard. This console-DAC is controlled by the program and its accuracy is maintained to within $\pm$ .003 volts.

A single unit of linkage-DAC's contains 30 individual digital to analog converter channels. These channels may be checked simultaneously in the following manner:

1. The least significant bit of the console-DAC and the 30 linkage-DAC's is turned on.
2. A comparison is made between the output of the Console-DAC and the 30 linkage-DAC's by means of 30 error amplifiers with gains of 1000.
3. The outputs of the error amplifiers are read by analog to digital converters and are recorded.
4. The least significant bit is turned off and the next higher order bit is turned on. This error is recorded and procedure continues until all bits have been checked.
5. After checking the highest order bit, all magnitude bits are turned on and the linkage-DAC's are checked at full scale positive.
6. The sign bits of all the DAC's are turned on and the above steps are repeated until the error for full scale negative has been recorded.

The relays and amplifiers used in this diagnostic are checked by the program before execution. In the event of an error, a pause results that allows the operator to investigate the malfunction prior to program execution. Total time required is one minute.

ADC

The analog to digital converter static diagnostic also uses the console-DAC as a standard. A single unit of linkage-ADC's contains 25 individual analog to digital converters which may be checked simultaneously in the following manner:

1. The program makes use of the same analog equipment by switching out the linkage DAC's and changing the 1000 gain amplifiers to unity gain.
2. The least significant bit of the console-DAC is turned on and fed to 25 unity gain amplifiers.
3. The outputs of these amplifiers are read by the linkage-ADC's. The program then compares these values with the original console-DAC value and records the difference.
4. The console-DAC is stepped in the same manner as in the linkage-DAC test until the errors for each bit, both positive and negative, are recorded.

The total time for execution of the linkage-ADC Static Diagnostic is less than one-half minute.

Dynamic diagnostic

DAC–ADC

The Dynamic Check provides a method for checking dynamic response of the linkage equipment. Any intermittence of operation may be found quickly and isolated by various program options. The dynamic check supplies a varying voltage to an ADC, samples the ADC at a definite time interval, and fires the DAC to the sampled ADC value.

The check is accomplished by combinations of the following options:

1. The inputs to the first and second ADC units are program coupled to corresponding outputs of the *first* and *second* DAC units, respectively.
2. The inputs to the first and second ADC units are program coupled to corresponding outputs of the *second* and *first* DAC units, respectively.
3. Any input of either ADC unit can be selected to control the outputs of both DAC units.

The program is completely flexible in that any combination of units and their synchronization may be selected. An option is available to use the interval timer to control the timer interval. The synchronization of the DAC's and the ADC's may be controlled by the interval timer, an external signal, or by the program. Detection of any intermittence is accomplished by applying a signal of varying amplitude to the desired ADC and observing the output of the desired DAC with a recorder.

This test may be used continuously to locate a malfunction.

Present diagnostic schedule

The scheduled frequency of Diagnostic Routine execution can be seen in Figure 18. We must admit that the schedule is the result of experience (four years) rather than a highly technical evaluation. The schedule evolved from a trial and error method but has proven itself over the past year to be more than adequate for our needs. The high reliability of our hybrid system has proven the worth of both the diagnostics and the frequency of use.

PERFORMED DAILY	PERFORMED WEEKLY
STATIC CHECK	INTEGRATOR DRIFT
INTEGRATOR RATE	NON-LINEAR
DISCRETE	MULTIPLIER
INTERVAL TIMER	TRUNKING
DAC-ADC HI-ACCURACY	
PRIORITY INTERRUPT	

Figure 18—Diagnostic scheduled frequency

Trouble shooting aids

Figure 19 shows the balance of the options available in our Preventive Maintenance Program. These options are used as needed for trouble shooting and in some cases, to aid in automated calibration techniques.

DAC-ADC DYNAMIC CHECK	INTERVAL TIMER CHECK
DAC-SET CHECK	PRIORITY INTERRUPT CHECK
READ ADC-DISPLAY CHECK	SUB CHANNEL TEN CHECK
DAC-ADC LIGHTS TEST	HYBRID SET-UP ROUTINE
DISCRETE ON-OFF CHECK	

Figure 19—Trouble shooting aids

SUMMARY

In the process of preparing this paper we were forced to review our efforts over the past four years, and we feel that some of these reflections may be of interest. As we stated some years ago,[2] the approach we used was forced upon us by both the relatively new design of our equipment and the fact that we were embarking upon the establishment of a new technology; which, if not new to the world, was at least new to us. We began with a staff of nine, which consisted of three degreed engineers experienced in programming and design, plus six technicians. Only two members of the technician force were experienced in analog. Looking back, we would not have had it any other way. We found, in a hybrid environment, it was easier to establish new habits of work performance than it was to break old ones.

We designed our diagnostic routines in a serial manner and began using them as they became operational. As each new test was designed, we attempted to check components not covered by previous tests. In retrospect, we feel that this was and is a valid approach. Not all diagnostics used today were anticipated in the beginning. Some we designed as the need became apparent. We have tried to remain flexible in our approach to diagnostics and have redesigned and updated as our needs and equipment varied. The diagnostics outlined in this document have been stable for the past year. We see no need for further revision, unless our equipment is changed.

It is our firm belief that the time and money spent on developing this system for maintaining our hybrid equipment has been well worthwhile. Our prime objective has been and is to provide our customers with maximum operation time coupled with a minimum maintenance effort. In this goal we have succeeded. The Boeing Huntsville Facility operates 18 hours per day, six days a week (sometimes seven); the operation is supported by a six hour maintenance period on third shift. Our down time during the past year has averaged six hours per month on the analog and linkage equipment plus ten hours per month for the digital portion.

Down time is defined as that time between the notification of a maintenance technician and the appropriate repair.

We have found the Diagnostic Routines invaluable as a trouble shooting aid when the applications personnel have trouble with their simulations. The application engineers over the years have grown to trust our diagnostics and use them as they would a core dump to isolate problem areas. This we feel has been our greatest compliment.

REFERENCES

1 G H GALE
Boeing—Huntsville hybrid system
Simulation Vol 5 No 4 1965
2 T K SEEHUUS
Hybrid computer techniques to aid computer maintenance
Simulation Vol 7 No 5 November 1966
3 T K SEEHUUS M T BENNET
A hybrid oriented method for testing analog non linear equipment
Presented at a joint meeting of the Southeast and Midwest Simulation Councils January 1968 Miami Florida

Demand paging in perspective

by C. J. KUEHNER *and* B. RANDELL

International Business Machines Corporation
Yorktown Heights, New York

INTRODUCTION

The method of storage allocation known as "demand paging," first introduced by the designers of the Atlas computer, [1] has a very appealing conceptual simplicity. Furthermore, the possibility that this technique could be used to allow programmers to ignore the problems of organizing information flow between working storage and one or more levels of backing storage was, to say the least, very tempting.

However demand paging has its limitations. Some were foreseen—several years ago Dennis and Glaser[2] pointed out that page-turning was potentially disastrous if applied to the wrong class of information. This however did not prevent the implementing of systems which in at least their initial versions have had grave performance problems arising from excessive paging. Such problems have caused much work on various more-or-less ad hoc attempts to improve system performance. The purpose of this paper is to survey the various techniques that have been proposed or used in an attempt to improve the performance of demand paging systems. By this means it is hoped to make clear the relationships between the various apparently completely separate techniques, and to aid the understanding of both the potentials and the problems of demand paging.

Demand paging

The two essential characteristics of demand paging are clearly conveyed by its title:

(i) Information is *demanded* by a program at the moment it is needed, without any prior warning. The "demand" is in fact implicit, and arises out of an attempt to use information not currently in working storage.

(ii) Information is transferred to and from working storage in units of a *page*. Such pages are of equal size (typically 1024 words). Correspondingly, the working storage is regarded as being logically equi-partitioned into *page frames*.

These characteristics together imply one of the benefits of demand paging, i.e., the removal from the programmer of the burden of explicity stating what information is to be transferred to and from working storage, and when these transfers are to take place. However, as should become clear in the main part of this paper, it is also these two characteristics which are the source of the performance problems actually experienced in some systems,[3] and predicted by several analytic[4] and simulation studies.[5]

The typical symptom of these performance problems is a low CPU utilization, together with a high utilization of the channels between backing and working storage, arising from an excessive amount of page-turning (excessive, that is, in relation to the amount of processing achieved). A system exhibiting such behavior is often said to be "paging itself to death."

These inefficiencies derive from two fundamental causes. Firstly a program, while awaiting satisfaction of a page demand, will continue to consume system resources—notably working storage. This is discussed in reference [6] and illustrated graphically in Figure 1. It is clear that if page transfers are very frequent, or consume a lot of time, much of the resorce utilization will be non-productive. The second cause is somewhat more subtle, and is due to the non-linear relationship which has been found to hold between program efficiency (e.g. Q average number of instructions executed between page demands), and the number of page frames allocated to the program.[7] This relationship is illustrated in Figure 2. It will be seen that although a program will continue to run quite well with a somewhat smaller set of pages than the totality it references, its performance falls off rapidly when the set of pages is reduced beyond a certain point, often termed the *parachor* of the program.[8] If a system is such that programs are often forced to operate with less than their parachor of pages in working storage, the system performance is likely to be far from acceptable.

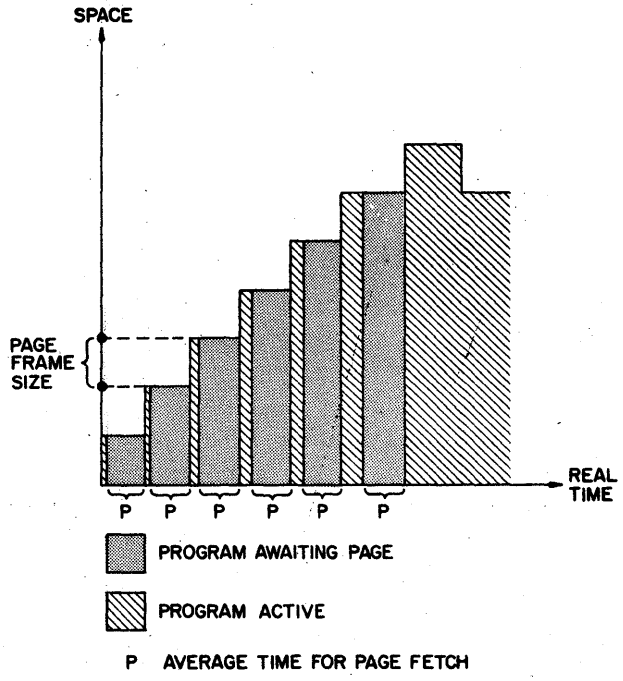


FIGURE 1—Storage utilization with demand paging

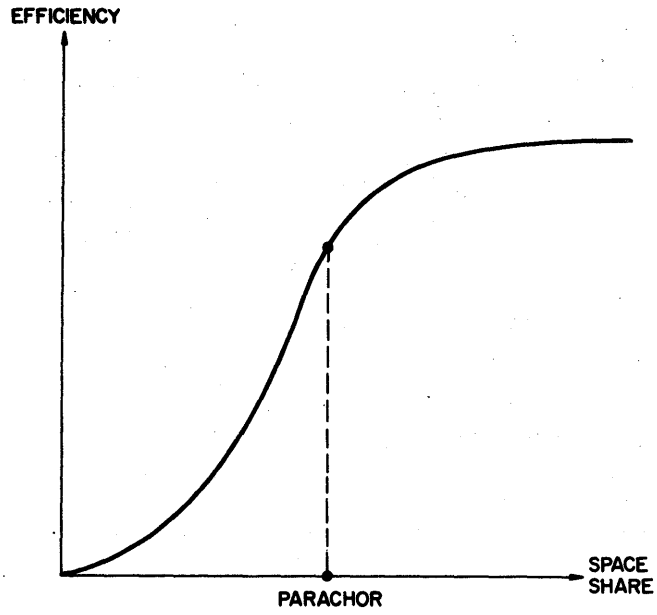


FIGURE 2—Efficiency versus space sharing

Three basic factors are involved in determining whether a demand paging system will succumb to the fate of being paged to death. These are:

(i) The system hardware characteristics
(ii) The program load
(iii) The operating system strategies

Of these factors the first is in some senses the most important, because an attack based on this area is by far the most likely to be successful in reducing the paging rate to an acceptable level. For example a drastic increase in the size of the working storage will overcome all but the most perverse of operating system strategies. However, adding more working storage capacity indefinitely is of course a trivial solution. The difficult question is what is the minimum amount of working storage which must be added in combination with various backing store designs to produce the required level of performance at minimum cost? Nielsen [8] has shown the effect of unbalanced system designs in which optimization of any but the critical system element has had little effect on overall system performance. A description of techniques available for determining the components of this balance in specific systems is beyond the scope of this paper,[8,9] however several general remarks may be in order.

Belady [7] has shown the effect upon system performance of reducing the delay associated with obtaining information from backing stores. From such an analysis it is clear that as one reduces the delay in obtaining a required page, one can obtain the same performance level with a significantly smaller amount of available working storage. Such results suggest the use of bulk core store as the solution to the performance problem. [10] However, at this time it is difficult to justify the use of such devices on a cost/performance basis in any but the most demanding of circumstances due to their relatively high cost. Conversely, the same results may be interpreted to show that flailing arm disks are a poor choice as backing storage in demand-paging environments. What effectively occurs in systems employing these devices is that the core latency effect shown in Figure 1 prohibits efficient operation unless a vast amount of working storage capacity is available. Consequently even though such devices are cheap, the performance obtainable does not yield a viable cost/performance ratio.

If one is free to vary the hardware complement, it can almost be guaranteed that an acceptable level of performance may be obtained. Such hardware solutions, however, are in some senses a brute force approach aimed at providing an environment which is insensitive to the difficulties caused by program load and operating system strategies. On the other hand, attacks on the factors of program load and operating system strategies can be considered as attempts to extend the area of practicality of demand paging. Therefore the remainder of this paper is concerned solely with the various proposed software solutions to the problems of demand paging.

Prior to entering into this discussion however, two notes of caution should be injected. First is the quesiton of cost. The cost of hardware solutions is easily de-

terminable in terms of rental or purchase dollars. This is not the case with software solutions in general. Although one would like to compare specific solutions *of equal cost*, it is not possible without a knowledge of the cost and skill of the software talent available to a specific installation. Second, the implementation of one, two or all of the techniques described below does not guarantee the transformation from an unacceptable to an acceptable level of system performance. In fact, a priori, it is very difficult to predict the magnitude of the performance improvement which would be obtained on a specific system by implementing one or several of these techniques.

Program load

The instructions and data whose storage and transmission form the load on the system resources such as storage and channels are comprised of both user programs and system programs. In fact certain parts of the operating system are users of the very resources that they have the task of allocating.

This load has properties of both volume and structure, both of which can be the source of problems. The effect of the volume of data and instructions on limited storage and channel resources should be readily understandable. The effects due to the structure of the program load are much more subtle. There is one easily-determined measure relating to the structure of a given program load, namely the *working set size*.[11] The working set of a program is the collection of distinct pages to which storage references were made during a given execution interval. The largest working set size of a program is then the number of pages occupied by the program and its data. At the opposite end of the spectrum a single page is also a working set, but for an extremely short execution interval. (To complete the execution of even a single instruction will normally require more than one page.) Working set size is hence a function of the interval during which storage references are observed. Obviously a plot of average working set size against length of execution interval observed is monotonically increasing. However, it should be remembered that for a given interval of observation different stages during execution of a program may well result in extremely different working set sizes and compositions.

Unfortunately the working set size measure is far abstracted from the underlying characteristics of program load structure which affect its value. These characteristics include:

(i) programming style
(ii) degree of modularity of code
(iii) layout of data

The various techniques for increasing the efficiency of paging systems described in this section all have as their goal a reduction primarily in the average value of the size of the working set and also in the volume of the program load. It is convenient to discuss separately techniques which can be supplied to an existing program load, and those of relevance during the initial design of programs for a paging system.

Program load improvement without recoding

The prospect of modifying the instruction flow and data layouts of an existing complex program load in order to reduce the average working set size is not exactly attractive. Hence some attempts have been made to achieve such a reduction by simply repacking the modules (which may be greater or less than a page in extent) that make up the program layout in virtual memory. Comeau[12] has reported on a brief series of experiments in which the modules which constituted an entire operating system were presented to the system loader in various different orders. One rearrangement was found that reduced the rate of page transfers by over 80% from that caused by the original version of the operating system.

This technique is obviously very simple to carry out, assuming one knows how to choose a good reordering of the modules. Any rational choice requires at least some information about the dynamic interaction of program modules and the pattern of referencing activity. Possible classes of information include:

(i) frequency of reference to the various modules
(ii) sequence of references to the modules
(iii) frequency of reference to the various pages
(iv) sequence of references to pages

Such information could be obtained during trial runs of the operating system. Two problems immediately come to mind. Firstly, there is no guarantee of consistency in data gathered from different runs. Secondly, the data regarding modules concerned with the actual paging function will vary as the working set composition changes, due to different packing arrangements. However these are probably minor problems compared with that of computing an optimal, or even near-optimal module ordering from the obtainable reference information.

Summing up, in the present state-of-the-art, any but the most minor attempts at re-packing are probably best regarded as last-ditch efforts at recovering from inadequate hardware, operating system strategies, and/or programming style.

Program load improvement via redesign and recoding

The style of program design required to obtain good

performance from a paging system in the face of inadequate hardware or operating system design is quite different from the conventional. In such circumstances adherence, by both programmers and compilers, to a set of programming commandments is as important as it is distasteful. The distastefulness arises from the need for a continual awareness of the part of the programmer of the (at least approximate) position of page boundaries in relation to his instruction and data layouts.

Necessary commandments include:

(i) Do not reference a wide variety of pages in rapid succession. It is better to localize activity for a reasonable interval, and then move on to a different locality, rather than to intermix references to the much larger combined locality.

(ii) Minimize space requirements for instruction and data storage only insofar as this permits adherence to commandment (i). (The assumption here is that it is worth trading increased backing storage utilization for decreased paging activity).

(iii) Avoid excessive modularity of programs. Program modularity, although very helpful if one must introduce changes and modifications to existing codes, can reduce execution performance substantially. If an operating system is composed of literally hundreds of program modules, any but the most trivial of work requests necessitates tens or hundreds of control transfers. Such transfers are costly in terms of unproductive CPU time but even more critically, in the worst case, each transfer could require that a distinct page of information be referenced. It should be clear, therefore, that a distinct tradeoff exists between the effective static level of program modularity and the amount of dynamic control transfers which directly result.

(iv) Design data layouts to take account of the order in which data items are to be processed. For example, if an nxn matrix (where n is slightly larger than the page size) is to be scanned repeatedly by rows then storing it by columns would be unwise, to say the least. (For a detailed discussion see McKellar and Coffman.[13])

The more frequently a particular program is to be used, the more important is adherence to these rules. Therefore of necessity systems programmers must be particularly circumspect. It should also be noted that adherence to the rules must be maintained during program modification (in the case of extensive and repeated modification this can be quite difficult).

However there have been some quite successful attempts to redesign fairly simple algorithms in order to improve their paging performance. For example the papers by Cohen[14] and by Bobrow and Murphy[15] describe list processing systems in which the dynamic storage allocation of list elements has been designed to group elements which are likely to be referenced in quick succession into the same page. Each system has thus enabled very large list processing applications to be run, with an efficiency surprisingly close to that attained by small applications which fit entirely within core storage (for example Cohen reports that only a factor of 3 in speed was lost, even though the average access to backing store was 10^4 slower than access to core storage). Another study which gives evidence of the performance improvements that can be obtained, particularly when a program has a very limited number of page frames allocated to it, is that by Brawn and Gustavson.[16] This report describes an extensive set of experiments investigating the effect of programming style on performance obtainable from the M44/44X experimental demand paging system. The experiments involving improvement of a sorting algorithm are particularly interesting. A series of changes were made to the original straightforward implementation of the algorithm. Each change affected the sequence, but not the number, of page references. The changes were simple to implement, but involved considerable thought about the logic of the program and its behavior in a paging environment. It was found that the amount of real core storage needed for reasonable performance was reduced by a factor of over six.

Operating system strategies

If the hardware and the program load are such that programs can usually have more than their parachor of pages in working storage while they are executing, then quite simple operating system strategies will suffice. However in general very careful thought must be given to the problem of achieving a high probability that there is always a task in working storage readv for CPU activity, and avoiding running programs in an excessively space-squeezed environment. This is difficult enough in a simple multiprogramming environment, but is even worse in a time sharing environment. In such circumstances there is the added need to give all users frequent service, so that simple requests can be answered quickly. Hence execution requirements which turn out to be lengthy will be subject to time slicing and, in all probability, to being paged in and out of working storage repeatedly.

It is convenient to consider the functions of an operating system which are relevant to this paper as being *scheduling*, *allocation*, and *dispatching*. Here scheduling is understood to be the maintenance of an ordered list of the jobs that are competing for the ser-

vices of the allocator. The allocator controls the allocation of working storage between such jobs from this list as it chooses to service. Finally, should there be more than one job ready and waiting for a CPU when one becomes available, the dispatcher makes the choice. Ideally a design should guarantee the harmonious cooperation of these three mechanisms under a wide variety of load conditions. Unfortunately even untried theoretical approaches to this goal are few in number, one of the most developed being that of Denning.[11]

However we are getting beyond the scope of this paper. The rest of this section is a discussion of various more or less isolated approaches to improving the strategies, and hopefully the performance, of demand paging systems. The techniques to be described can be divided into those concerned primarily with scheduling and those concerned with allocation.

Scheduling

In paging systems experiencing performance problems, the scheduling function must be viewed in a somewhat different light than one normally finds in the literature, where scheduling algorithms are compared and optimized in isolation of the system being scheduled. In such cases scheduling is considered the dominant system controlling function and allocation is reduced to providing space for jobs as the scheduler dictates. The result of such an approach in a system containing a resource class more critical than processing time is, in general, a significant degradation of performance. The problem is that few "schedulers" sample the resources of the system and base their decisions on the results. Instead they attempt to inject and remove jobs from execution on the basis of arbitrary pre-set bounds for time-slice, quantum time, amount of occupied storage, etc. The effect upon performance in a demand-paging system of cycling a job into and out of execution is critical. One can easily see why this may be so from an analysis of Figure 1.

The key to improving performance via scheduling in such systems is to subordinate the scheduling function to that of allocation. Jobs should be introduced and removed from the system on the basis of the state of the system resources by the scheduler under the control of the allocator (with few if any exceptions). Once a job has been introduced into the system and acquires its parachor of pages, one should be hesitant to remove it and lose the investment unless it has been in execution for a lengthy period of time. The period of time-slice should be an upper bound exception and not the general rule of operation.

The above philosophy of scheduling is concerned with when, and under whose control, items should be entered or removed from the list of jobs competing for working storage. There remains the problem of which jobs should be selected for adding to the list when circumstances make this desirable. The two scheduling techniques that have been proposed for improving the performance of demand paging systems, which are described below, are concerned with this latter problem area.

A fairly simple and frequently proposed technique of reducing the amount of paging is to arrange that requests for use of the same program are batched together. This can be quite successful, particularly in those time-sharing systems where many users are making continual use of a substantial program, such as a compiler, or a text editing and filing program. In a very large system the fact that the similar requests are batched together might not even be apparent to the user, the delays caused by waiting for a batch to be accumalated being quite small and perhaps even being outweighed by the effect that reducing the amount of paging has on system efficiency.

A more autocratic proposal[17] is to dynamically limit the programmer response rate. The theory is that although a system should respond to a programmer's request immediately, it will be good for the system, and in all probability the programmer, if his rate of response to the system is controlled. One simple method would be to lock the terminal keyboard after a system response, for a period of time in some way proportional to the amount of system resources used in providing the response. This is intended both to limit the load in the system, and to encourage users to think before they type. The acceptability or quantitative performance improvement that such a scheme would yield remains to be proven.

Allocation

As mentioned earlier, it is considered the function of the allocator to decide when to allow the scheduler to introduce new jobs into contention for working storage space and to decide which requests for working storage it should attempt to service. This enables the allocator to control the level of multiprogramming (i.e., number of independent contenders) in working storage during any time interval. If the modifications to the existing operating system modules would be too extensive to incorporate this scheduling-allocation procedure, one can introduce a new almost autonomous mechanism to the original operating system to perform such a function. Such a mechanism, termed the "load-leveler," was in fact added to the M44/44X system. The scheme chosen for the load leveler was to periodically examine recent CPU utilization and paging rate, and when necessary request the scheduler to temporarily remove jobs from the list of those being serviced by the allocator.

The jobs which are temporarily set aside are chosen from among those which are not experiencing frequent interaction.

The need to move a job which has reached the end of its time slice out of working storage temporarily can be a source of considerable difficulty. Such a job may have painstakingly accumulated the ration of page frames that it needs for effective progress. If on next being eligible for a time slice it must demand pages one-at-a-time, its initial progress will be minimal. (This is borne out by the simulation results given by Fine et al.[5]) The technique known as "pre-paging" (Oppenheimer and Weizer[18] is intended to avoid this situation. Prepaging involves fetching a set of pages, which are believed to be the working set of a job, into working storage before the job is allocated to a CPU. The obvious difficulty with prepaging is deciding what to prepage (i.e. Q the composition of the working set). The problem which involves predicting future program activity, is very similar to that of designing a page replacement algorithm.[19] A straightforward approach is to prepage the last (n) pages referenced during the program's previous time slice. A major factor in choosing the magnitude of (n) should be the speed of the backing store. One theory is that the longer it takes to fetch a page on demand, the larger (n) should be. Although this will increase the probability of fetching unwanted pages, it should avoid many subsequent page demands and reduce the overall space-time product for the execution of this job.

A logical extension of pre-paging is "paging by function." This involves the use of special program directives indicating that a certain set of pages will all be required for a certain function to be performed. These pages will then all be brought to working storage at once, rather than one by one, on demand. A corresponding directive can be used as a form of "block page release," indicating that all of the indicated pages have been finished with, and can be replaced. A possible source of the information needed to use these program directives intelligently might be the sort of program module interaction data discussed in an earlier section.

Lessons to be learned

It is convenient to attempt to summarize the above discussion by considering separately the effects of the demand and the paging concepts.

The idea of having a running program notify the system of its need for a particular resource when it has reached the state of not being able to make any further progress until that resource is granted to it is at one end of the spectrum of possible resource allocation strategies. The other extreme is to allocate the entire resource requirement to a program from the beginning of its life time in the system. (This of course requires complete knowledge of all the resources that a program will need.) In the former case inefficiencies arise from non-productive resource utilization while a program is awaiting satisfaction of its demands. In the latter case the inefficiency arises from giving a program resources for a longer period than it requires thereby making them unavailable to other users. Corresponding to the above comments on resource allocation, a similar discussion applies to the spectrum of strategies possible for controlling the deallocation of resources.[11,19]

A correct choice of allocation and deallocation strategies for a particular system involves an appreciation of the cost of non-productive resource utilization, the time taken to satisfy a resource demand, the frequency with which such demands are likely to be made, and the probability of getting accurate advance notification of resource needs.

It should be noted that this problem is not peculiar to paging systems, or even indeed to the particular resource of storage space. However experience has shown that a wrong choice of strategy can have drastic effects in a dynamic storage allocation system. This would appear to be due to the fact that in most systems working storage is a very critical resource, and also because of the non-linear relationship that holds between program efficiency and space allotment. (Experimental evidence for this relationship comes from paging systems, but there is no reason to believe that it may not also apply to dynamic allocation systems which allocate differing sizes of blocks of working storage.)

On the subject of paging, it is clear that one of the most obvious difficulties that arise in the design of a paging system is that of choosing a page size. From the viewpoint of avoiding the transfer to working storage of a large amount of information which may not be used, a small page size is desirable. On the other hand if too small a page size is chosen the overhead caused by very large page tables will be excessive. It seems clear that if the basic hardware and operating system strategies are adequate for the program load, an adequate choice of page size can be made, and a very simple page replacement technique for working storage allocation will suffice. The problem arises when this is not the case, and it is necessary for programmers to maintain a continual awareness of the underlying paging environment while designing or modifying programs. In these circumstances the page size becomes a straight jacket and, if only for reasons of programmer convenience, a multiplicity of sizes of storage areas would be preferable. There is in fact some evidence[20] to suggest that such an environment permits higher utilization of working storage than one in which only a single page size is permitted. However so much depends on programming

style and methods of compilation that a definitive comparison is extremely difficult.

CONCLUSIONS

The software techniques for improving the performance of demand paging systems described above all attempt in their various ways to perform one or more of the following functions:

(i) provide advance warning of page demands
(ii) reduce the working set of programs, and hence their parachor
(iii) ensure that programs are not excessively space-squeezed (ideally that each program is given just over its parachor of pages)
(iv) provide explicit identification of the working set.

When one attempts to abstract the common denominator of these techniques it becomes clear that they are not just attempts to "tune" the basic demand-paging philosophy, they in fact attempt *to negate both the "demand" and the "paging" characteristics.* Prepaging, for example, attempts to avoid the "demand" page exception with its attendant overhead and delay, as well as avoiding "paging" by attempting to select a *meaningful portion* of the program to preload. The result is a storage management system which is a mixture of strict paging and slow swapping.

It is important therefore to remember that demand paging is but one of the many possible techniques of dynamic storage allocation. As with any strategy its range of efficient use is limited. The simplicity of working with uniform units of allocation and the potential efficiency of bringing into working storage only those units of information which are actually referenced are the strong points of demand paging. The demand paging systems so far implemented have in general shown that these advantages are purchased at the cost of rather more working storage, and/or faster backing storage, than have heretofore been considered necessary for the throughput and response times that have been obtained. A final conclusion on the merits and limitations of demand paging must however await the accumulation of more experience of their use in actual user environments.

ACKNOWLEDGMENTS

The authors would like to thank L. A. Belady, M. Lehman and F. Zurcher for numerous discussions and suggestions which aided in the preparation of this paper.

REFERENCES

1 T KILBURN D B G EDWARDS M J LANIGAN F H SUMNER
One-level storage system
IRE Transactions on Electronic Computers 11 2 1962 pp 223–235

2 J B DENNIS E L GLASER
The structure of on-line information processing systems
Proc 2nd Congress on Information Systems Sciences Spartan Books Washington DC 1965 pp 5–14

3 J N BAIRSTOW
Time-sharing
Electronic Design 16 9 1968 pp C1–C22

4 J L SMITH
Multiprogramming under a page on demand strategy
Comm ACM 10 10 1967 pp 636–646

5 G H FINE C W JACKSON P V McISAAC
Dynamic program behavior under paging
Proc 21st ACM National Conference Thompson Book Co Washington DC 1966 pp 223–228

6 B RANDELL C J KUEHNER
Dynamic storage allocation systems
Comm ACM 11 5 1968 pp 297–306

7 L A BELADY C J KUEHNER
Dynamic space sharing in computer systems
Report RC 2064 IBM Thomas J Watson Research Center Yorktown Heights New York May 1968

8 N R NIELSEN
The simulation of time sharing systems
Comm ACM 10 7 1967 pp 397–412

9 J E SHEMER G A SHIPPEY
Statistical analysis of paged and segmented computer systems
General Electric Company Technical Report Phoenix Arizona April 1966

10 H C LAUER
Bulk core in a 360/67 time-sharing system
AFIPS Conference Proceedings Vol 31 1967 FJCC Washington DC 1967 pp 601–609

11 P J DENNING
The working set model for program behavior
Comm ACM 11 5 1968 pp 323–333

12 L W COMEAU
A study of the effect of user program optimization in a paging system
ACM Symposium on Operating System Principles Gatlinburg Tennessee Oct 1–4 1967

13 A C McKELLAR E G COFFMAN
The organization of matrices and matrix operations in a paged multiprogramming environment
Technical Report No 59 Dept of Elec Eng Computer Sciences Laboratory Princeton University February 1968

14 J A COHEN
Use of fast and slow memories in list processing languages
Comm ACM 10 2 1967 pp 82–86

15 D G BOBROW D L MURPHY
Structure of a LISP system using two-level storage
Comm ACM 10 3 1967 pp 155–159

16 B BRAWN F GUSTAVSON
An evaluation of program performance on the M44/44X system
Part 1 Report RC 2083 IBM T J Watson Research Center Yorktown Heights New York May 1968

17 R L PATRICK
Time-sharing tally sheet
Datamation 13 11 1967 pp 42–47

18 G OPPENHEIMER N WEIZER
Resource management for a medium scale time sharing operating system
Comm ACM 11 5 1968 pp 313–322

19 L A BELADY

A study of replacement algorithms for a virtual storage computer
IBM Systems J 5 2 1966 pp 78–101

20 B RANDELL

A note on storage fragmentation and program segmentation
Report RC 2102 IBM T J Watson Research Center Yorktown Heights New York May 1968

Program behavior in a paging environment

by BARBARA S. BRAWN and FRANCES G. GUSTAVSON

IBM Watson Research Center
Yorktown Heights, New York

Study objectives

This paper is the result of a study conducted on the M44/44X system, an experimental time-shared paging system designed and implemented at IBM Research in Yorktown, New York. The system was in operation serving up to sixteen users simultaneously from early 1966 until May 1968. Conceived as a research project to implement the virtual machine concept, the system has provided a good deal of information relating to the feasibility of that concept.[1] The aim of this study is to investigate the concept more thoroughly from a user's viewpoint and to try to answer some important questions related to program behavior in a paging environment. As an experimental system, the M44/44X provided an excellent vehicle for the purposes of this study, and the study itself forms some basis for an evaluation of the system.

It is recognized by the authors that the results and conclusions presented here are to a great extent characterized by a particular configuration of a particular paging system, and as such do not constitute an exhaustive evaluation of paging systems or the virtual machine concept. Nonetheless, we feel that the implications of the conclusions reached here are of consequence to other system implementations involving paging.

Conventional vs automatic memory management

There has been much written about the benefits and/or disadvantages of paging machines and the virtual machine concept.[2,3,4] However, little data have been obtained which sheds a realistic light on the relative merits of such a system compared to a conventionally designed system. From a programming point of view there is little question that any technique which obviates the necessity for costly pre-planning of memory management is inherently desirable. The question that arises is—given such a technique, how efficiently is the automatic management carried out?

From a user's point of view this can simply mean—how long does it take to run a program which relies on the automatic memory management, and is this time comparable to the time it would take to run the program if it were written in a conventional way where the burden of memory management is the programmer's responsibility. It is this user's viewpoint that forms one focal point for this study.

The role of the programmer

Perhaps the most important aspect of the study concerns the role of the programmer. How does the role of the virtual machine programmer differ from that of the conventional programmer? For a conventional system the role of the programmer is well defined—the performance (i.e., running time) of his program is usually a direct result of his ability to make efficient use of system resources. How much he is willing to compromise efficiency for the sake of ease of programming may depend on how often the program is to be run. In any case, the decision rests with him. (There of course exist many applications where his choice of programming style or ability have little effect on performance; this case is of little interest to our study.)

When faced with the problem of insufficient machine resources to accommodate a direct solution of his problem, the conventional programmer is left with no choice but to use some procedure which is inherently a more complex programming task. The quality of the procedure he chooses may

have a dramatic effect on performance but it is at least a consistent effect and often quantifiable in advance. In any event, because conventional systems have been around a long time, there are many guidelines available to the programmer for achieving acceptable performance if he should wish to do so.

The role of the virtual machine programmer is not nearly so well defined. One of the original attributes claimed for the virtual machine concept was that it relieved the programmer from consideration of the environment on which his program was to be run. Thus he need not concern himself with machine limitations. As was pointed out previously, the question is—given that the programmer does in fact ignore all environmental considerations, what kind of efficiency results? Assuming that the answer to this question is sometimes undesirable, that is, running time is unacceptably long, another question arises. Can the programmer do anything about it? Clearly it is difficult to conceive of his being able to reorganize his program in such a way as to assure improved performance if he has no knowledge of the environment nor takes it into consideration when effecting such changes. Thus if the premise of freedom from environmental considerations is to be strictly adhered to, there can be no way for the programmer to consciously improve performance.

Should this premise be compromised to allow the programmer to influence performance through exercising knowledge of the system environment? This study assumes that this should indeed be the case and shows that there is much to be gained and little to be lost. It should be emphasized, however, that the original premise *need not* be compromised at all in so much as it would, of course, not be *necessary* for the programmer to ever assume the responsibility of having knowledge of the environment (unlike in the case of the conventional programmer faced with insufficient machine resources). It would only enable him to have better assurance of acceptable performance if he chose to do so.

Clearly, the many interesting questions concerning the role of the virtual machine programmer and his effect on performance are worthy of pursuit. We feel that the measurements obtained in this study of program behavior in a paged environment provide a valuable insight to such questions and serve as motivation for further consideration of them.

Test environment

Before discussing the results of the study we feel it is advisable to describe the environment in which they were obtained. Thus included herein are brief descriptions of the M44/44X system and the methods employed to obtain and measure the test load programs. (More complete information is available in References 1, 5 and 7.) It is assumed throughout this discussion that the reader is generally familiar with the concepts of virtual machines, paging, time-sharing and related topics; however, a short general discussion on paging characteristics of programs is included in order to establish an appropriate reference frame for the presentation of the experimental data.

The experimental M44/44X system

To the user a virtual computer appears to be a real computer having a precise, fixed description and an operating system which provides various user facilities and links him to the virtual machine in the same way as the operating system of a conventional system links him to the real machine (Figure 1). Supporting the virtual machine definition is a transformation (control) program

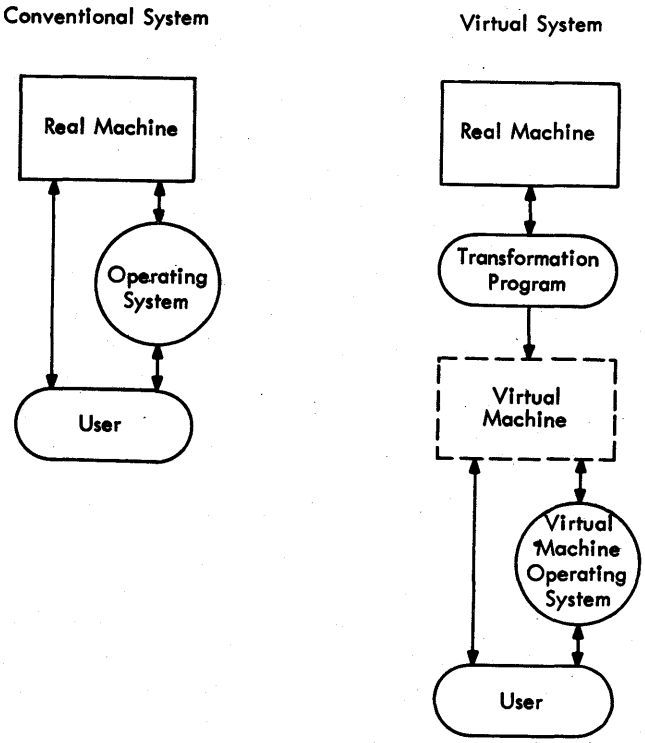


FIGURE 1—Conventional and virtual systems

which runs on the real machine. This program, together with special mapping hardware, "creates" the virtual machine as it appears to the user. Implementation of multi-programming within the framework of the virtual machine concept permits the transformation program to define the simultaneous existence of several separate and distinct virtual machines.

The virtual machine programmer may write programs without knowledge of the transformation program or the configuration of the real machine—his concern being the virtual machine description, which is unaffected by changes in the real hardware configuration or the transformation program. In the M44/44X system the real machine is called the M44, the transformation program is MOS, the virtual machine is the 44X and the virtual machine operating system is the 44X Operating System (Figure 2).

The real computer

Figure 3 shows the hardware configuration of the M44 computer. It is an IBM 7044 with 32K, 36-bit words of 2 μsec core which has been modified to accommodate an additional 184K words of 8 μsec core and a mapping device. The resident control program together with the mapping device and its associated 16K, 2 μsec mapping memory, implement the 44X virtual machines on a demand paging basis in the 8 μsec store. The backup store of the M44/44X system, which is used for both paging and permanent file storage, consists of two IBM 1301 II disks. The page size (a variable parameter on this system) used for our tests was 1024 words (1 K). The average time required to seek and transmit one page from the disk to core is 0.21 second for that page size (computed from our data). The IBM 7750 serves as a message switching device, connecting a number of IBM 1050 terminals and teletype 33's to the system. To facilitate measurement our tests were not run from terminals (foreground) but as background jobs from tape. (The system makes no distinction between the two for the single programmed case—nor for the multi-programmed case as long as *all* jobs on the system are of the same type, i.e., all background or all foreground.)

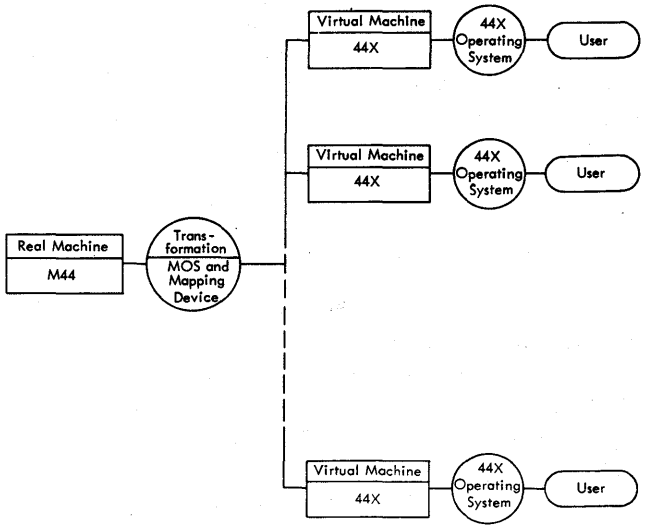


FIGURE 2—M44/44X Multi-programming system

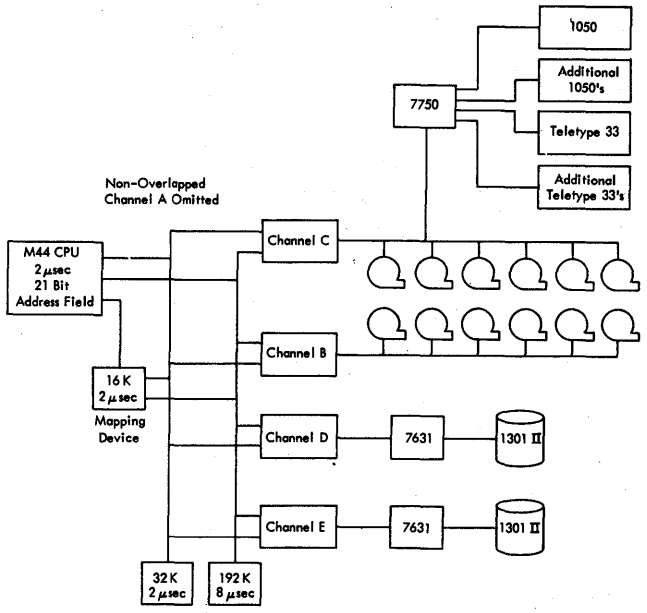


FIGURE 3—M44/44X hardware configuration

The control program

MOS, the control program, resides in the non-paged 2 μsec store. This M44 program "creates" and maintains each virtual 44X machine and enables several 44X's to run simultaneously, allocating the M44 resources among them. All 44X I/O is monitored by MOS, and all error checking and error recovery is performed by MOS. Some of the design parameters of MOS are easily changed to facilitate experimentation. The variable parameters include the page size, the size of execution store (real core) made available to the system, the page replacement algorithm, the time slice and the

scheduling discipline (via a load leveling facility). The last two parameters mentioned are applicable only in the multi-programmed case. As previously stated, the page size used throughout the study was 1024 words. The size of real core was, of course, one of the most important parameters and was varied to investigate paging properties of the programs (in both the single and multi-programmed environments).

For the single programmed part of the study the page replacement algorithm employed was FIFO (First In-First Out). If a page in real core must be overwritten, the page selected by FIFO is the one which has been in core for the longest period of time. Data were also obtained for single programmed paging behavior under a minimum page replacement algorithm developed by L. Belady.[6] A non-viable algorithm, MIN computes the minimum number of page pulls required by examining the entire sequence of program address references.

For the multi-programming part of the study, a time slice of 0.1 second was used. Runs were made using three different page replacement algorithms to determine the effect of this design parameter on system performance. (Available real store is competed for freely by all the 44X's.) The three algorithms were FIFO, BIFO, a biased version of FIFO which favors (on a round robin basis) one 44X by choosing not to overwrite the pages associated with it for a preselected interval of time, and AR, a hardware supported algorithm which chooses a candidate for replacement from the set of pages which have not been recently referenced. (These algorithms are described more fully in Refs. 1 and 6.)

The virtual machine

Each virtual 44X machine is defined to have 2^{21} words of addressable store. The virtual memory speed of a 44X is 10 μsec (44X programs are executed in 8 μsec store and a 2 μsec mapping cycle is added to a memory cycle); the CPU speed is 2 μsec. The user communicates with the 44X virtual machine through the 44X Operating System, a 44X program which permits continuous processing of a stack of 44X jobs; it contains a command language, debugging facilities, a FORTRAN IV compiler, an assembly program, a relocatable and absolute loader facility, routines for handling a user's permanent disk files and a subroutine library.

Test load problems

Test problems were chosen from the scientific, commercial, list processing and systems areas of computer applications. The problems chosen involved large data bases which required the programmer of a conventional machine to concern himself with memory management. The problems discussed in this paper include matrix inversion and data correlation from the scientific area and sorting from the commercial area. (A complete report on the entire study can be found in Ref. 7.)

Programs were initially coded for each problem in two ways:

i) a conventional manner where the burden of memory management is assumed by the programmer (conventional code), and
ii) a straightforward manner utilizing the large virtual memory ("casual" virtual code).

Simple modifications were then made to the "casual" virtual codes to produce programs better tailored to the paged environment. Our interest lay in comparing the performance of the different versions of the virtual codes under variable paging constraints in both single and multi-programming environments. We were also interested in comparing the conventionally coded program performance with that of the virtual (i.e., automatic memory management) codes given the same real memory constraints.

It should perhaps be noted here that for our purposes a program's performance is directly related to its elapsed run time. Thus in a paging environment, where this elapsed time includes the time necessary to accomplish the required paging activity, poor paging characteristics are reflected by increased run time and thus degraded performance.

Measurement techniques

A non-disruptive hardware monitoring device capable of measuring time spent in up to ten phases of program execution was used for all 7044 runs and relevant single-programmed 44X runs. In addition, for 44X runs (both single and multi-programmed), a software measurement routine in MOS was utilized. This routine collects data while the system is running (using the clock and a special high-speed hardware counter) and on system termination produces a summary of the

data including; total time, idle time, time spent in MOS (including idle time), number of page exceptions, page pulls, page pushes and other pertinent run data.

All programs were run in binary object form as background jobs residing on a system input tape; all output was written on tape. For the multi-programmed runs, a facility of MOS was used which permits several background jobs to be started simultaneously. For the single programmed study the 44X programs were first run and measured on the system with sufficient real core available to eliminate the need for paging; these same programs were then run (and measured) in a "squeezed core" environment, i.e., with insufficient real memory available, thus necessitating paging.

Program behavior under paging

Program performance on any paging system is directly related to its page demand characteristics. A program which behaves poorly accomplishes little on the CPU before making a reference to a page of its virtual address space that is not in real core and thus spends a good deal of time in page wait. A program which behaves well references storage in a more acceptable fashion, utilizing the CPU more effectively before referencing a page which must be brought in from back-up store. This characteristic of storage referencing is often referred to as a program's "locality of reference." [6] A program having "good" locality of reference is one whose storage reference pattern in time is more local than global in nature. For example, although a program in the course of its execution may reference a large number of different pages, if in any reasonable interval of (virtual) time, references are confined to only a small set of pages (not necessarily contiguous in the virtual address space), then it exhibits a desirable locality of reference. If, on the other hand, the size of the set is large, then the locality of reference is poor and paging behavior is correspondingly poor. (The "set" of pages referred to in the above example corresponds roughly to Denning's[8] notion of a "working set."

All programs typical of real problems exhibit badly deteriorated paging characteristics when run in some limited real space environment. What is of interest is the extent to which the space can be limited without *seriously* degrading performance. Clearly, the size of this space is related to the program's locality and provides some indication of the size of what might be called the program's critical or characteristic working set. As the single programmed results presented below show, the effects of programming style on the relative size of this space can be enormous.

Single programmed measurement results

We first measured the behavior of the 44X programs in a controlled single programmed environment. The results obtained are discussed in terms of the relative effects of programming style on performance for three problems: T1—Matrix Inversion, T2—Data Correlation, and T4—Sorting. In each case we are concerned with showing how even simple differences in programming technique can make a substantial difference in performance. Unquestionably there are further improvements which could be made in the algorithms employed; however, we feel that our point is best illustrated by the very simplicity of the changes made.

Timing and paging overhead data are given for actual runs made on the system employing a FIFO page replacement algorithm. Also, in order to establish that these results were not unduly influenced by that page replacement algorithm, corresponding computed minimum paging overhead data are given (obtained through interpretive program execution and application of L. Belady's[6] MIN algorithm).

The data collected for the comparison of the automatic and manual methods of memory management is also discussed in this section.

Problem T1 . . . Matrix inversion

The virtual machine codes for this program were written in FORTRAN IV and are intended to handle matrices of large order. They all employ an "in-core" technique since the large addressable virtual store permits the accommodation of large arrays (the burden of real memory management being assumed by the system through the automatic facility of paging). The curves in Figure 4 give the respective program run times as a function of real core size for the three different versions which were written for the virtual machine. These times are for inverting a matrix of order 100 (which is admittedly not an unusually large array, but sufficiently large to illustrate our point without requiring an impractical amount of CPU time).

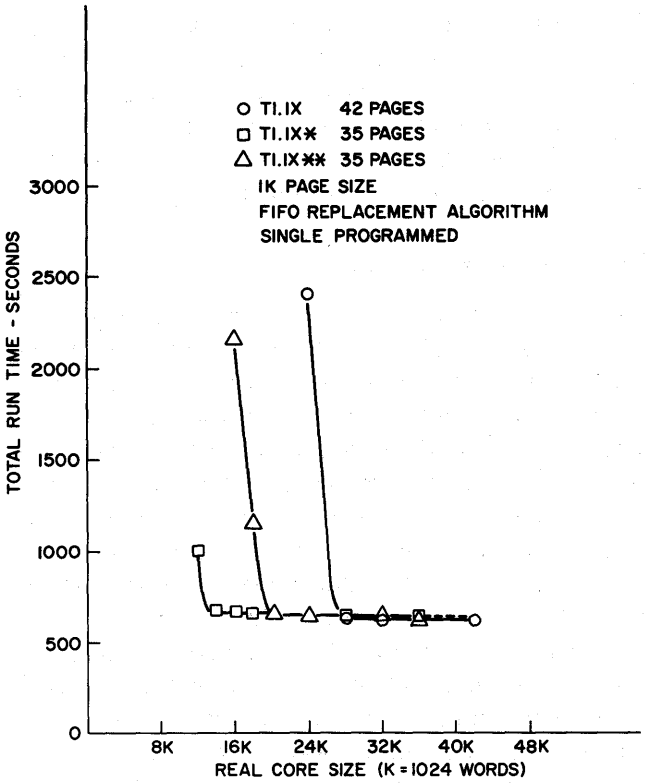


FIGURE 4—Effects of real core size
T1—Matrix inversion (100x100)

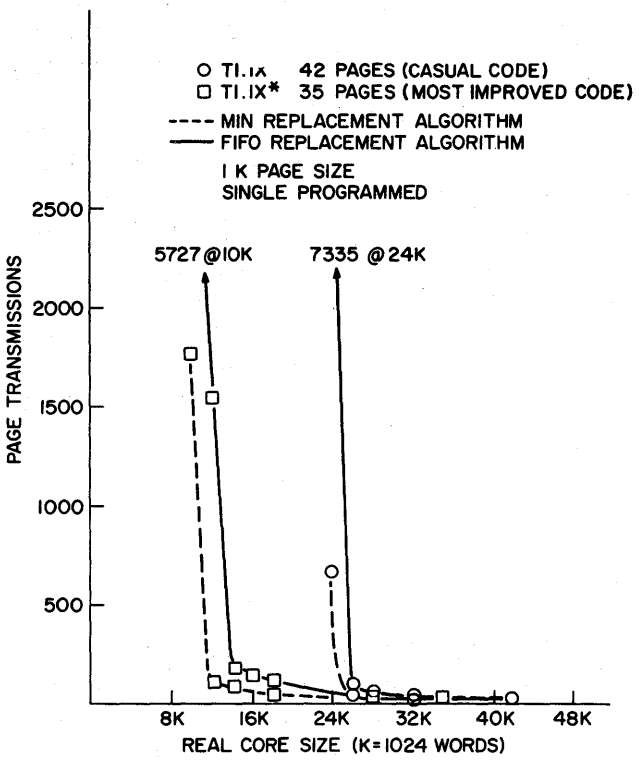


FIGURE 5—Effects of page replacement algorithm
T1—Matrix inversion (100x100)

All three programs employ the same algorithm, a Gaussian procedure utilizing a maximum pivotal condensation technique to order successive transformations. The differences in the three versions are extremely simple. The "casual" version, T1.1X, stores the matrix in a FORTRAN double subscripted array of fixed dimensions (storage allocated columnwise to accommodate a matrix of up to order 150), reads the input array by rows and prints out the inverted array by rows. The innermost computation loop traverses elements within a column. Version T1.1X** is the same as T1.1X except that variable dimension capability was employed (thus insuring the most compacted allocation of storage for any given input array). Version T1.1X* is the same as T1.1X** except that the input and ouput is columnwise instead of rowwise. Obviously neither of these changes is complicated or of any consequence in a conventional environment; however, as clearly shown in Figure 4, they make a considerable difference in a paging environment.

The paging overhead data is shown in Figure 5 for the casual (T1.1X) and the most improved (T1.1X*) versions for both the FIFO algorithm (corresponding to the time curves of Figure 4) and the MIN algorithm. This paging overhead is given in terms of the number of page transmissions required during execution of the respective program when run with a given amount of real core available under the discipline of the particular page replacement algorithm. (Each reference to a page not currently residing in real core requires a page to be transmitted from backup store into real core [a "pull"] and often also requires a page to be copied from real core onto backup store [a "push"]. The total number of pulls and pushes is the number of page transmissions. Given a particular real core size, the MIN algorithm employed gives the theoretical minimum number of pulls required. Belady has shown that the number of page transmissions obtained by this algorithm differs insignificantly from the number obtainable by minimizing both pulls and pushes.)

As can be seen in Figure 5, there is no great disparity between the paging overhead sustained under FIFO and the theoretical minimum possible (under MIN) for either of the programs. In particular it should be noted that the paging behavior of the well coded program is considerably better

under FIFO than that exhibited by the casual program under the most optimum of page replacement schemes. Certainly these data support the argument that improvement in programming style is advantageous to performance, irrespective of what page replacement scheme is used.

Clearly there are modifications which could be made to the algorithm itself which would further improve performance through improved locality of reference. McKellar and Coffman[9] have indeed shown that for very large arrays, storing (and subsequently referencing) the array in sub-matrix form (one sub-matrix to a page) is superior to the more conventional storage/reference procedure employed in our programs. (For the 100×100 array, however, the difference is not significant.)

Problem T2 . . . . Data correlation

For the other problem in the scientific area an existing conventional FORTRAN program, which required intermediate tape I/O facilities because of memory capacity limitations, was modified to be an "in-core" procedure for the virtual machine. The problem, essentially a data correlation procedure, involves reconstructing the most probable tracks of several ships participating in a joint exercise, given a large input data set consisting of reported relative and absolute position measurements. The solution implemented is a maximum likelihood technique; the likelihood functions relating the independent parameters are Taylor expanded to yield a set of simultaneous equations with approximate coefficients. The equations are solved (using the inversion procedure of problem T1), the solutions are used to recompute new approximate coefficients, and the process is reiterated until a convergent solution is reached. (Each iteration involves a single pass of the large data set.) The measured position data, together with the accepted solution are used to compute the reconstructed ships' tracks. (This final step requires one pass of the data set for each ship.)

For the first (or "casual") version, T2.1X, the conventional code was modified for the large virtual store in the most apparent way. The large data set, a mixture of fixed and floating point variables stored on tape for the conventional version, was stored in core in several single-subscripted fixed dimension arrays, one for each variable in the record format. As the curve for this program in Figure 6 shows, the performance is rather poor. This is accounted for in part by the fact that the

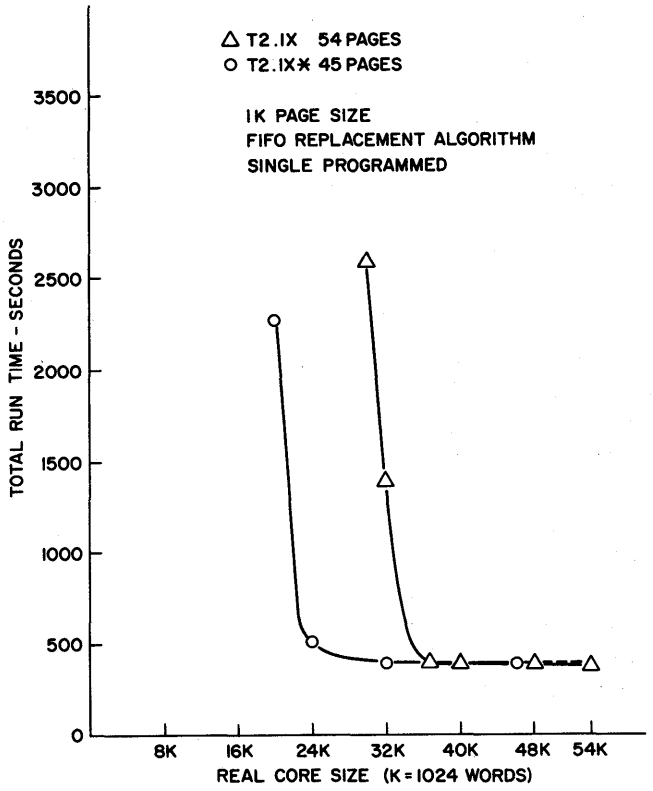


FIGURE 6—Effects of real core size
T2—Data correlation

manner in which the data are stored causes a global reference pattern to occur due to the program's logical use of those data. Version T2.IX* attempts to improve the locality by storing the data compactly in one single-subscripted floating point array, such that all of the parameters comprising a single logical tape record in the conventional code are in sequential locations. (The conversions necessitated by assigning both fixed and floating point variables to the same array name increased the CPU time slightly.) The curves in Figure 6 clearly show that this modification resulted in a significant improvement.

The same ordered relationship exhibited under FIFO holds for the casual and improved versions under the MIN algorithm (Figure 7). Although in the case of the poorly behaving code, the MIN algorithm does appreciably better than FIFO given a core size of 32K where FIFO performance has already deteriorated badly. The improvement is short lived, however, since deterioration under MIN occurs with any further decrease in real core size.

It should be noted that the actual data set used for these runs was not exceptionally large (as the total number of pages referenced indicates). Again, practicality demands that we settle for a data case of reasonable size. The case at hand involved six ships (resulting in 26 equations) and a rather small data base of only 240 reports. The data base storage requirements in the case of the well coded program, T2.1X*, were satisfied by four pages. In the case of T2.1X, however, the several large *fixed dimension arrays* used to store the data in that program required 13 pages; thus not only was the data ordering poor but a great deal of space was wasted as well.

Once again, there are probably other improvements that could be made. For example, because the program is divided into several subroutines (17) of reasonable length, a change in the order of loading the routines could improve (or degrade) performance. We have illustrated here only the effects of a change in the manner of storing the data base.

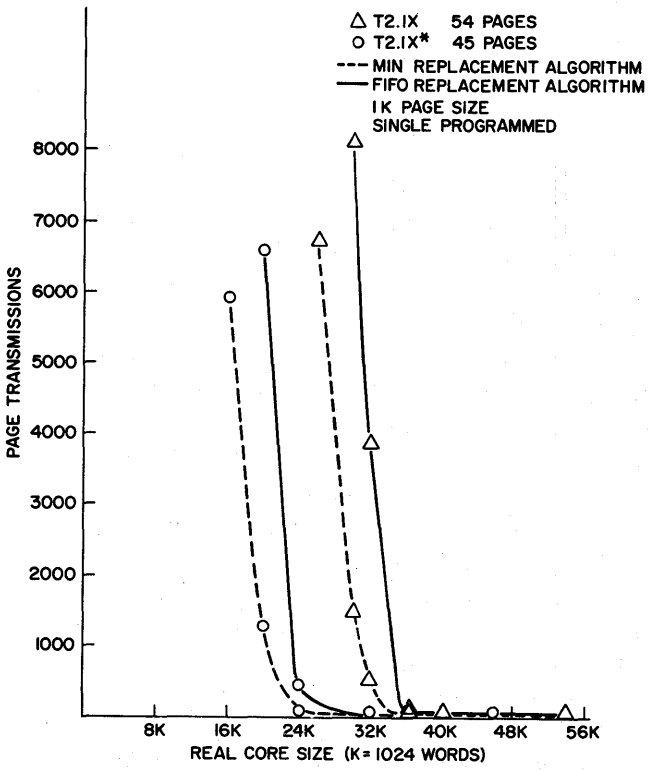


FIGURE 7—Effects of page replacement algorithm T2—Data correlation

Problem T4 . . . . Sorting

Sorting, a classical example of the necessity for introducing complicated programming techniques to accommodate a problem on a conventional memory bound computer, also affords an excellent example of how drastically programming style can effect performance in a paging environment. Ideally, if memory capacity were sufficient for the entire file to be in core, the sort programmer would only need to concern himself with the internal sorting algorithm and never be bothered with the other plaguing procedures involved with doing the job piecemeal. This was the approach taken, programming the virtual machine codes assuming that the file could be accommodated in virtual store.

Initially, two different algorithms were coded—the Binary Replacement algorithm (basically a binary search/insertion technique employed in a generalized sorting program in the Basic Programming Support for IBM System 360) and the Quicksort[10] algorithm (a partitioning exchange procedure). When the completed programs were run with a reasonably long data set, it became immediately apparent that the Binary Replacement algorithm was exceptionally bad for *large* lists because of the amount of CPU time required. (Note that this characteristic presents little problem for the internal sort phase of a conventional code which never deals with a very large list.) We will, of course, acknowledge that someone more knowledgeable in the field of sorting than we would have recognized this characteristic of the algorithm beforehand. Our experience nonetheless pointed out rather dramatically that an accepted technique for a conventional machine need not be acceptable when translated to a virtual machine environment, *irrespective of its paging behavior!* Because of its unacceptable CPU characteristics, the algorithm was discarded and our efforts were concentrated on Quicksort since that algorithm is efficient for either small or large lists.

Four versions were ultimately coded for the virtual machine, each of which is described below. All of the changes made to get from one version to another were simple and required little programmer time. None of these changes altered the total number of pages referenced; they simply improved the locality of reference. The time curves in Figure 8 and the paging curves in Figure 9

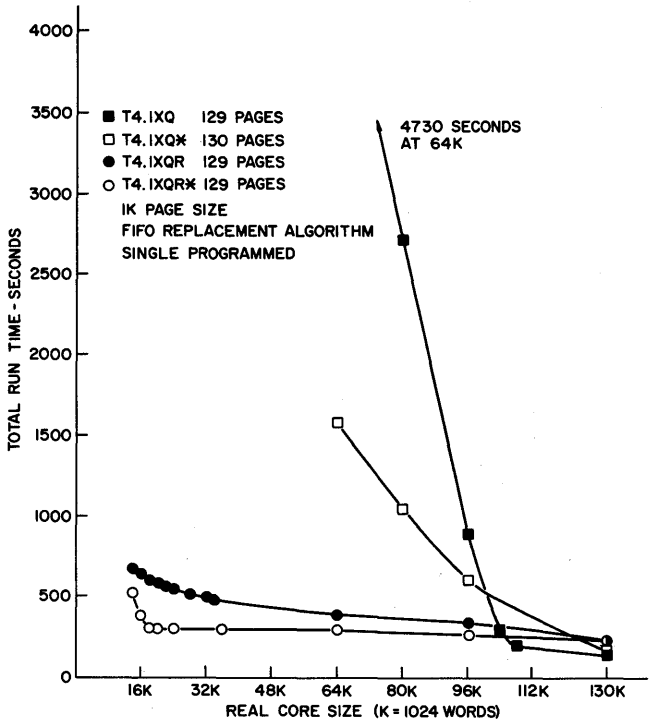


FIGURE 8—Effects of real core size
T4—Sorting (10,000 10-word items)

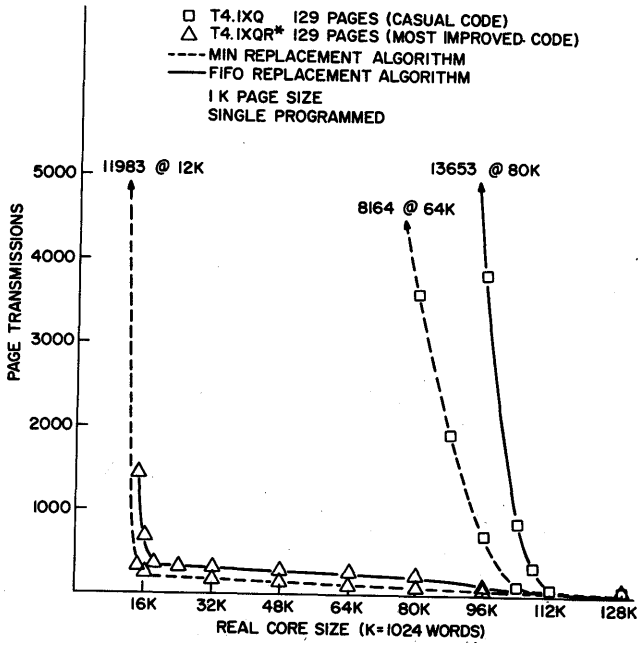


FIGURE 9—Effects of page replacement algorithm
T4—Sorting (10,000 10-word items)

show dramatically how important these relatively minor modifications were to performance. (The size of the file in the case shown is 100,000 words, occupying 100 pages in virtual memory.)

T4.1XQ, the "casually" coded version, reads in the entire file, performs a *non-detached keysort* utilizing the Quicksort algorithm and a table of key address pointers, then retrieves the records for output by using the rearranged table of pointers. The records themselves are *not* reordered during the sort thus storage references are random and global during both sort and retrieval, making locality of reference poor. Deprived of only a small amount of its required store, this program behaves very badly. Note that although the MIN curve in Figure 9 does show some improvement in paging behavior over FIFO, the improvement is of no consequence since performance is still quite unacceptable.

*T4.1XQ** treats the file as N sublists; each is read in, then sorted using the non-detached keysort routine of T4.1XQ. (N is 10 for the case shown; thus the 100 page file is logically divided into sublists of 10 pages each.) When all the sublists have been sorted an N-way internal merge, using the table of pointers, retrieves the records for output. This modification improves the locality of reference for the sort phase (for the case shown, the size of the characteristic working set during the sort is approximately 12; 10 for the current sublist, one instruction page and one page for pointers) but the storage reference pattern remains random during merge-retrieval since the records are not reordered within the ten sublists (during this phase the characteristic working set therefore includes *all* the file pages).

T4.1XQR is the same as T4.1XQ except that a record sort is performed instead of a keysort, i.e., *the records are reorderd* while being sorted, leaving an ordered list to be retrieved for output. The now sequential reference pattern substantially improves paging behavior for the retrieval phase (each page of the file is referenced only once for that phase). Moreover, because of the record reordering, locality during the sort phase benefits substantially from the partitioning characteristic of Quicksort. Performing a record sort, of course, results in a penalty in CPU time (especially for large records) since the transfers involve the entire record instead of the key only. (For this reason, it would not be wise to choose a sorting algorithm which requires an exceedingly large number of transfers.) However, the penalty paid is

relatively insignificant in view of the improved paging behavior.

*T4.1XQR** is the same as T4.1XQ* except that it performs a record sort instead of a keysort. The comments made on improved locality during the sort phase for both T4.1XQ* and T4.1XQR also apply to this version. In addition, because the records are now in sequential order within each sort area (due to record reordering) the merge/retrieval phase also exhibits desirable paging characteristics (as long as there are enough pages available to accommodate *at least* one page from each sort area, i.e. N pages plus instruction and control pages—approximately 13 for the case shown).

Clearly, the behavior of T4.1XQR and T4.IXQR* demonstrates that the large virtual store can be used effectively and in a simple manner if thought is given to the environment. The curves for T4.1XQ and T4.1XQ* demonstrate equally clearly that it can be disastrous not to do so.

Automatic vs programmer-controlled memory management

The objective of this part of the study was to compare the effectiveness of automatic and manual (programmer-controlled) memory management. To meet this objective, our test problems were programmed to run on a conventional machine, using accepted manual methods to accommodate them on the available memory. The efficiency of any program written for a conventional machine, of course, depends on how skillful the programmer is in utilizing available system resources. We felt that, although in no way optimum, the efficiency of the programs coded was characteristic of what is normally achieved under practical constraints of programmer time. (It should be noted that the programmer time involved in writing and debugging these conventional codes far exceeded that required for the corresponding virtual codes.)

The data presented in the previous sections clearly show that the effectiveness of paging as an automatic memory management facility depends not only on internal characteristics of the particular system but also on user programming style. We thus felt that an effective comparison of the two memory management methods should include the effects of virtual machine programming style. We also felt that our comparison should in some way include the effect which overlap capability can have on conventional code efficiency since, in a multi-programming environment, that capability does not exist for the individual user. (We were aware that almost any proposed comparison would be subject to question on one count or another because of the lack of adequate control; we nonetheless feel that the comparison is reasonably unbaised and has sufficient validity to be of interest.)

To make the comparison we proposed to run both the conventional programs and the corresponding virtual programs (i.e., those which utilized the large virtual store), in their respective environments, which were constrained to be equivalent with respect to real machine resources. All of the conventional program I/O was tape I/O and the CPU and memory speed were the same for both the conventional and virtual machines. In each case the virtual programs were run with the size of available real core equal (to the nearest page) to that actually referenced by the corresponding conventional program. The numbers in Table 1 are computed ratios of the respective virtual code run times to corresponding conventional code run times; therefore numbers less than 1 are favorable to the automatic method.

TABLE 1—Comparison of automatic and programmer controlled memory management

	Casual Virtual Code vs Overlapped Conventional Code	Best Virtual Code vs Overlapped Conventional Code	Casual Virtual Code vs Non-Overlapped Conventional Code	Best Virtual Code vs Non-Overlapped Conventional Code
T1 - Matrix Inversion 200 x 200	2.33	1.25	1.47	0.79
T2 - Data Correlation	0.81	0.81	0.71	0.71
T4 - Quicksort 100,000 word file [4-way Merge] [Conv. Code]	>41	1.26	>32	0.99
T4 - Quicksort 100,000 word file [2-way Merge] [Conv. Code]	>27	0.84	>19	0.59
T4 - Quicksort 1,000,000 word file [4-way Merge] [Conv. Code]				1.15

The data indicate that, if reasonable programming techniques are employed, the automatic paging facility compares reasonably well (even favorably in some instances) with programmer controlled methods. While not spectacular, these results nonetheless look good in view of the substantial savings in programmer time and de-

bugging time that can still be realized even when constrained to employing reasonable virtual machine programming methods.

Multi-programming measurements

The importance of programming style to paging behavior was clearly demonstrated in the single programmed part of this study. We were interested in learning if it would have similarly dramatic effects on performance in the domain more common to paging systems, i.e., multi-programming. Because the most notable changes in behavior were observed in the sorting area, we decided to plan our measurement efforts around these programs. An extensive measurement program was undertaken which was designed to give us insight into the relative effects on performance of the following: programming style, page replacement algorithm, size of real core, number of users and scheduling. It should be noted that the question of performance in a multi-programmed environment involves both the individual user response and total system thruput capability. Although the study addressed both of these aspects, the results discussed here pertain only to the latter. (A complete in-depth report on the entire multi-programmed measurement study is given in Ref. 7, Part III.)

The effects of programming style

The two versions of the sort program used for this study were the "casually coded" version, T4.1XQ, and the "most improved" version, T4.1XQR*. Multiple copies of a given program were run simultaneously (as background jobs) on the system with the full real core (184K) available. (No more than 5 background jobs can be run simultaneously because of tape drive limitations.) The curves in Figure 10 compare the multi-programming efficiency obtained with the two different programming styles. These curves are plots of Time/Job vs the number of (identical) jobs run simultaneously on the system (Multi-programming Level).

Clearly the efficiency of the system is nearly identical whether multi-programmed at the two level or the five-level in the case of the well-coded program, T4.1XQR*, but is substantially degraded for each additional job in the case of the casually coded version, T4.1XQ. In fact, multi-programming at even the two level for that program is

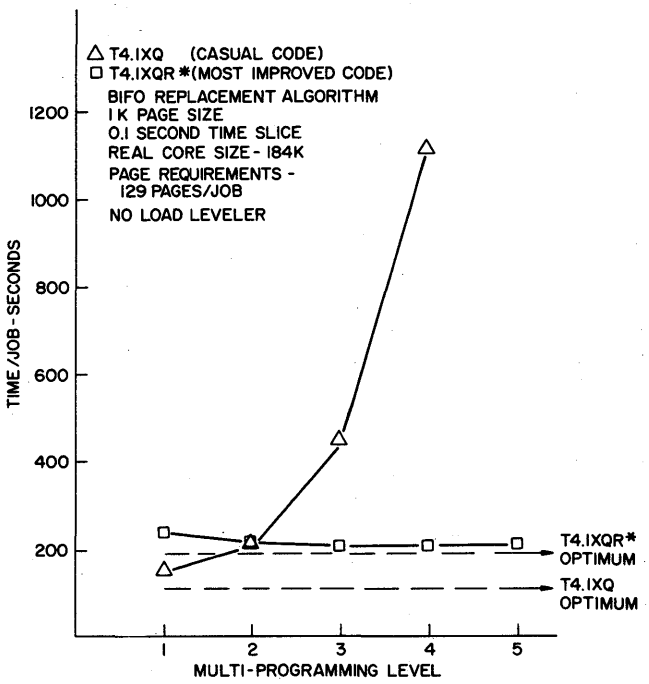


FIGURE 10—Effects of programming style
T4—Sorting (10,000 10-word items)

worse than running sequentially. (For T4.1XQR* multi-programming is consistently more advantageous than running sequentially up through the five-level.)

The effects of load leveling

One of the capabilities available on the M44/44X system aimed at improving efficiency is that of dynamically adjusting the load on the system in order to attempt to avoid the overload condition which is characterized by excessive paging coupled with low CPU utilization. When this load leveling function is activated, the system periodically samples paging rate and CPU utilization, compares them with pre-set criteria to determine if a condition of overload or underload exists, and then takes action appropriately to adjust the system load by either setting aside a user, i.e., removing him temporarily from the CPU queue, or restoring to the queue a user who was previously set aside. The function of the load leveler is thus essentially one which affects scheduling.

The extremely poor behavior exhibited by the casual code when multi-programmed made this case a likely candidate for studying the effects of load leveling. Figure 11 shows the remarkable improvement which the load leveler achieved

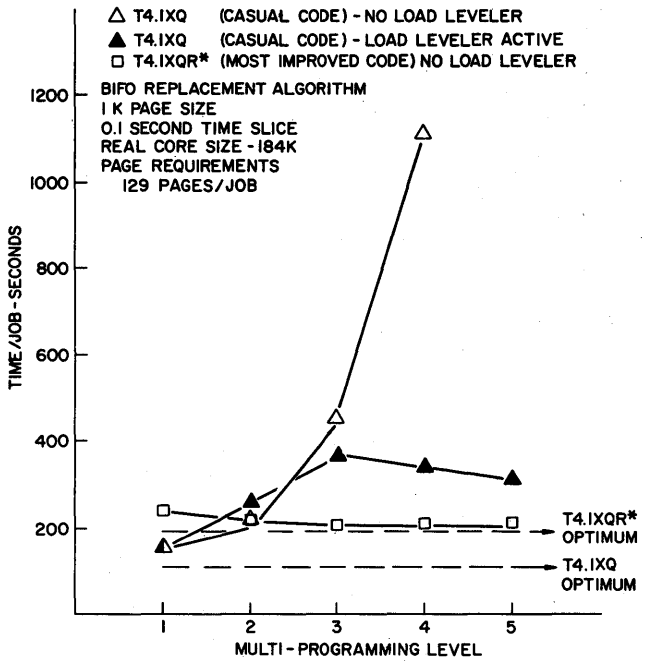


FIGURE 11—Effects of load leveling
T4—Sorting (10,000 10-word items)

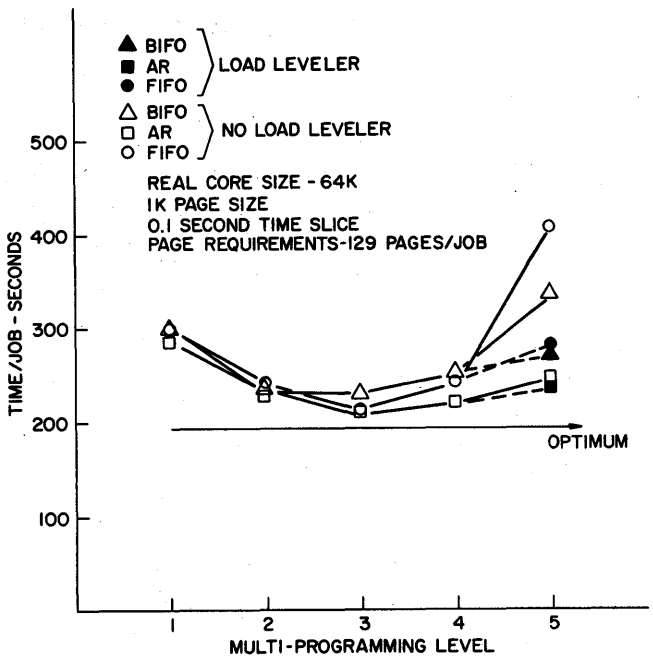


FIGURE 12—Effects of page replacement algorithm
T4.1XQR*

when there were three or more jobs involved. Unfortunately, the efficiency is still substantially worse than in the sequential case. We nonetheless feel that the potential for improved performance achieved through the use of an automatic dynamic facility such as this is promising and indicative that it would be well worth implementing—in particular if it can be kept simple and efficient as is the case with the M44/44X load leveler.

The effects of page replacement algorithm

As might have been suspected from the single-programmed MIN study, the role of the page replacement algorithm appears to be of relatively little significance. In the case of T4.1XQ, runs were made using the more sophisticated AR algorithm but the data collected differed little from that obtained for the BIFO algorithm. Similarly, in the case of T4.1XQR* the difference in the results is inconsequential for those runs made where all of real core (184K) was available. (Figures 10 and 11 show the BIFO data.) However, when the same T4.1XQR* runs were made with the real core size restricted to 64K there was some change in performance for the different replacement algorithms. The curves in Figure 12 compare the effects of using the different algorithms for T4.1XQR* multi-programmed (up to the five-level) with only 64K of real memory available to the entire system, i.e., shared by all the users. The level of multi-programming for which the efficiency is optimum is in all cases three; however, in the case of the AR algorithm, multi-programming at the five-level with only 64K of real memory is still more advantageous than running the five jobs sequentially (with the same 64K of real memory). Note that this is also true when running under the other algorithms with load leveling.

The effects of real size

Performance is so poor for the T4.1XQ program given the full 184K of real memory, that it is obviously unnecesary to show how bad things would be given an even smaller memory! In the case of T4.1XQR* however, performance for the system is so close to optimum that we were curious to learn just how small the real core size could be before performance would be worse than in the single-programmed case (for the same size of real memory). The curves in Figure 13 compare Time/Job for the single-programmed case, with multi-programming at the three and five levels for different real core sizes. Runs were also made multi-programmed at the five level with the load

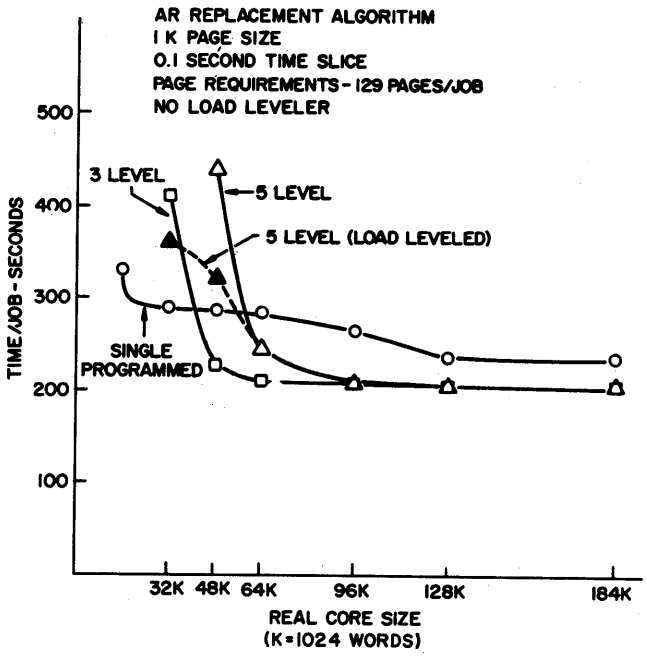


FIGURE 13—Effects of real core size
T4.1XQR*

leveler activated and real core sizes of 48K and 32K. As can be seen in Figure 13, while improving performance, the load leveler was not able to improve it sufficiently to compare favorably with the single-programmed sequential case.

When viewed in the perspective of page requirements per job, the performance of the system is remarkable for the well coded program. Five jobs, each requiring 129 pages, shared a 32K memory and still behaved reasonably well! (The time per job is even a few seconds less than that required for the overlapped 2-way merge conventional code.) On the other hand, the performance for the casual code given the full memory capability of 184K is at best (load leveled) quite a lot worse than sequential and at worst (not load leveled) a minor disaster.

The data which we have presented here on multi-programming represent only part of that collected for the study. The cases chosen are obviously the extreme ends of the spectrum. One would not (hopefully) encounter all "bad" programs running at the same time on a system under real time-sharing conditions, nor (regretfully) is one likely to encounter all "good" programs. The real situation lies somewhere inbetween—and, most likely, so does the characteristic performance of the system. We have not directly addressed the question of individual thruput (or response) time in the data shown here; however; we have shown that total system thruput is most certainly affected by the programming style employed by the users on the system. We have shown in our other work (Ref. 7) that this is also true for individual response time (often even if system thruput is unaffected).

SUMMARY

The single programmed data presented in this paper give strong support to the conclusion that the effects of programming style are of significant consequence to the question of good performance in a paging system. Indeed, as the MIN results indicate, the basically external consideration of programming style can be considerably more important than the internal systems design consideration of replacement algorithm. We feel that data obtained for the multi-programmed case, some of which were presented in the previous section, further support our conclusions. In view of these results, we feel that this aspect of performance must not be disregarded in future endeavors to implement paging systems. Programming techniques should be developed at both the user and system levels which are aimed at achieving acceptable performance on such systems. For example, higher level language processors such as FORTRAN should be designed for paging systems to produce good code for the environment as well as to perform well themselves in that environment.

While we support the stand that paging and virtual machines are inherently desirable concepts with much potential, we strongly feel that in order to fully realize that potential in terms of practical performance characteristics, the notion of programming with complete unconcern for the environment must be discarded. Our data have shown, however, that one can often realize acceptable performance by employing even simple techniques which acknowledge the paging environment. Their simplicity leads us to feel that the programming advantages inherent to the concept of virtual systems can, to a great extent, still be preserved.

ACKNOWLEDGMENT

We would like to acknowledge E. S. Mankin for his extensive contribution in preparing the test load programs for the sort area.

REFERENCES

1 R W O'NEILL
Experience using a time-shared multiprogramming system with dynamic address relocation hardware
SJCC Proceedings Vol 30 1967 pp 611–621

2 P WEGNER
Machine organization for multiprogramming
Proceedings of 22nd ACM National Conference Washington DC 1967 ACM Publication P-67 pp 135-150

3 G H FINE C W JACKSON P V MCISAAC
Dynamic program behavior under paging
Proceedings of 21st ACM National Conference Washington DC 1966 ACM Publication P–66 pp 223–228

4 *Adding computers—Virtually*
Computing Report for the Scientist and Engineer Vol III No 2 March 1967 pp 12–15

5 *The M44/44X user's guide* and the *44X reference manual*
IBM Corp T J Watson Research Center Yorktown Heights New York September 1967

6 L A BELADY
A study of replacement algorithms for a virtual storage computer
IBM System's Journal Vol 5 2 1966 pp 78–101

7 B S BRAWN F G GUSTAVSON
An evaluation of program performance on the M44/44X system Parts I II III
R C 2083 IBM T J Watson Research Center Yorktown Heights May 1968

8 P J DENNING
Working set model for program behavior
CACM Vol 11 5 May 1968 pp 323–333

9 A C McKELLAR E G COFFMAN
The organization of matrices and matrix operations in a paged multiprogramming environment
Princeton University Technical Report No 59 February 1968

10 C A R HOARE
Quicksort
Computer Journal Vol 5 April 1962 to January 1963 pp 10–15

JANUS: A flexible approach to realtime timesharing*

by J. O. KOPF and P. J. PLAUGER

Michigan State University
East Lansing, Michigan

INTRODUCTION

Motivation

A third generation computer seems to cause as many problems as it solves; not because it is difficult to program or too problem directed—quite the contrary. The problems arise because such a computer lends itself so willingly to *all* applications—realtime data acquisition, process control, scientific calculations, bookkeeping and conversational time-sharing. In a nuclear physics laboratory, there are enough imaginative people interested in each of these subjects that eventually all are implemented with some success. The central problem, then, is to develop an operating environment compatible with open-ended development of any or all types of computer usage. Ideally, one seeks a standard operating system providing the framework and resources to aid all such development.

In our case, the desired priority of computer usage was to be: 1. realtime data acquisition and control; 2. interactive on-line operations, especially data analysis; 3. background operation. In a nuclear physics experiment, realtime operation is normally characterized by immense buffers—which are updated at each input event, rather than transmitted as sequential data—indicating the desirability of a small resident monitor, and non-permanently dedicated interrupt routines and buffers. Event rates as high as 50,000 events/second may be expected, implying the need for a powerful computer to perform quickly the operations necessary to each event.[1,2]

The Michigan State University Cyclotron Laboratory installed a Scientific Data Systems Sigma-7 computer in January 1967. We have constructed operating system JANUS for the Sigma-7 to meet the goals outlined above. JANUS has proved to be far more powerful than we originally expected.

The SDS Sigma-7

The SDS Sigma-7 is a high-speed, integrated circuit machine with sophisticated timesharing hardware.[3] It features a 32-bit word, with displacement indexing by 8-bit bytes, half-words, words and doublewords, and direct addressing to 128k words. Timesharing hardware includes master/slave modes, rapid context switching (Exchange- and Load-Program-Status-Doubleword instructions), a powerful interrupt structure with certain functions inhibitable under program control, program traps which are independent of the interrupt structure, and mapping hardware.

The Sigma makes extensive use of scratch-pad memories; integrated flip-flop registers whose access time is insignificant compared with core memory. Thus there are 16 distinct registers, effectively accumulators. Instructions normally reference one or more of these registers. In addition, the computer treats these registers as the first 16 locations of memory: all instructions are valid for register-register operations. The computer is thus effectively a two-address machine, where one address space is a subset of the other. Furthermore, four registers may be used in a block as a decimal accumulator (31 digits plus sign), seven others may be used as index registers (post indexing), and any even-odd register pair may be used for double precision work.

The hardware also makes use of hard-wired table look-up and translation for certain functions. An example is the map. Memory is naturally divided into 512-word pages. The map consists of a scratch-pad memory of 256 bytes, one for each page of virtual memory (virtual memory is the full address space of the machine, independent of

*Supported by the National Science Foundation.

the actual core memory available). When the map is in operation, the first byte of the effective virtual address is used as an index to look up a translation byte from the map, which replaces the original byte to form the actual address used to make the memory reference. As a result, contiguous virtual pages need not be in contiguous actual memory; under a properly initialized map they act as though they are. Associated with each page is a two-bit *access process* code which can inhibit slavemode from writing, executing, or even reading words in the page. In conjunction with a rapid access disc (RAD), this hardware provides the swapping control needed for efficient timesharing.

In addition, the computer has two major means of communicating with the external world. The Input-Output Processor (IOP, of which there may be up to 8) is designed for sequential transmission of data asyncronously with the operation of the computer. The Direct I/O (DIO) provides for the transmission of one word at a time to or from the registers, under program control. JANUS normally uses the IOP for conventional I/O operations; the DIO for acquisition and control.

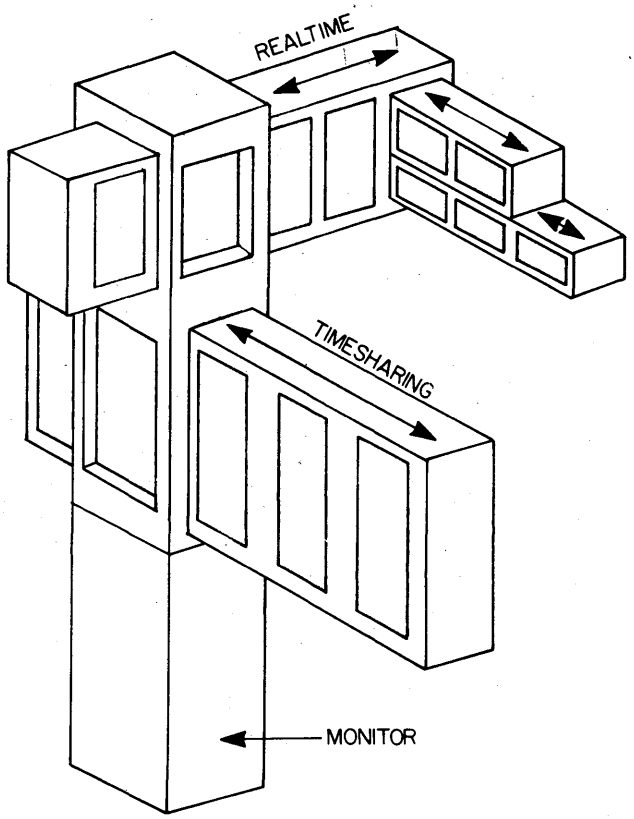


FIGURE 1—Hardware resources of the MSU Sigma–7 system. Items labelled "J" are handled by JANUS. Those labelled "S" are shared by all users on a cyclic basis. All others are loaned out on a first come, first served basis for exclusive use.

Figure 1 details the resources available on the MSU Sigma-7.

Other approaches

Before going into the details of JANUS, we should perhaps explain why we felt existing approaches were inadequate for our needs.

Conventional realtime systems are usually geared for one application, or one set of applications. One cannot randomly start and stop arbitrary functions, even though the particular resources needed may be standing idle. In particular, one cannot "batch" process (i.e. compile, load and run a series of purely computational programs) to take advantage of the usually large CPU time available between interrupts.

By dividing memory into *foreground* and *background* areas, it is possible to operate a batch system in conjunction with one or more realtime operations. Aside from the fact that either of these areas is frequently; a) unused for long periods of time or, b) inadequate for many jobs that *could* be run in full memory, there is a more sophisticated drawback. Since realtime operations must often use the same resources as the batch, a large resident monitor is needed to handle common operations and to prevent conflicts. Furthermore, since realtime operations occur on an interrupt basis, the monitor must either be reentrant to several levels or must inhibit interrupts while it is active (or a little of both). The former solution makes the resident even larger and slower—the latter interferes with fast response to realtime events.

Conventional time-sharing systems[4] can be geared to provide the random stop/start of realtime which we desire, and are better geared to adapt efficiently to dynamically changing memory availability. But the usual approach has been to take an already large foreground/background type monitor and to add a swapper, job scheduler and elaborate I/O queuing routines to the resident. The dedicable memory left over can be vanishingly small.

Figures 2 and 3 caricaturize the distinction we made between what we saw in conventional approaches to realtime timesharing and what we envisioned for JANUS.

There still remains the problem, not yet mentioned, of the hybrid job. It is often desirable to

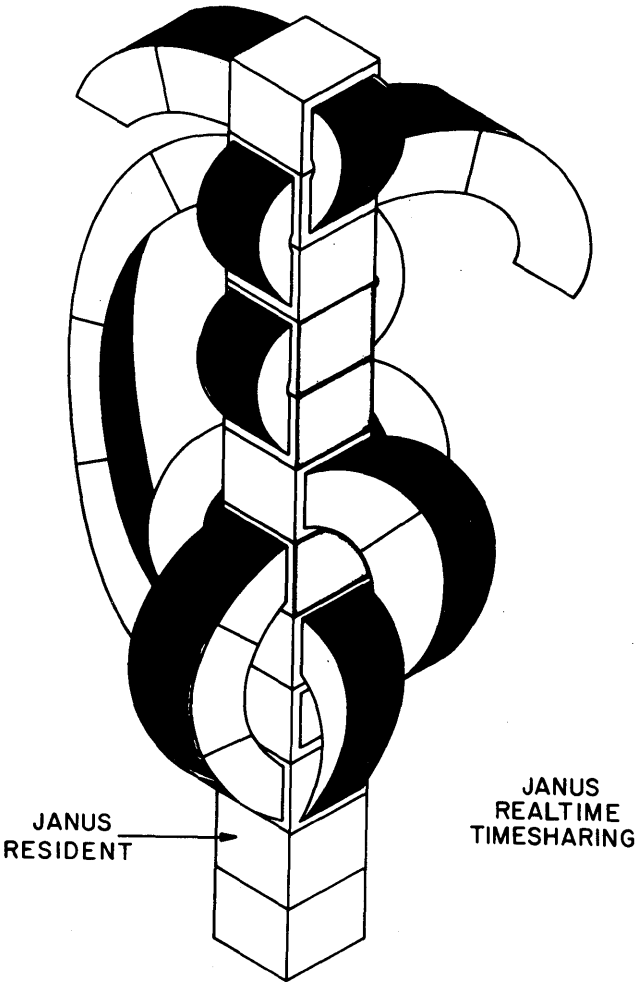


FIGURE 2—Caricature of a "conventional" realtime timesharing system. It is characterized by a large resident monitor and rather rigid frame-at-a-time use of remaining memory.

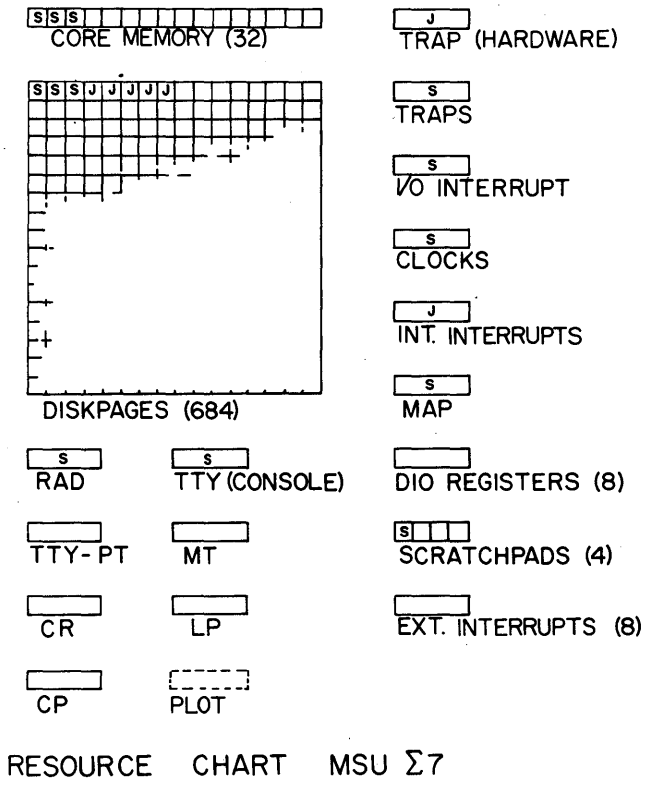


FIGURE 3—Caricature of JANUS system, with small resident and intertwined tasks having both dedicated realtime and timeshared parts.

construct programs having a small realtime part and a large problem-solving part that could happily be timeshared. The question is raised as to specifying such an animal. How is communication between the two parts effected?

We found the answers to these questions, and the inspiration for answering many more, in the definition of PL/I.[5]

Design of JANUS

Terms and philosophy

The PL/I language definition provides a vocabulary and a philosophy (we have probably corrupted both). In PL/I a piece of code containing all routines needed to perform a job, or distinct part of a job, is a *task*. A task can start one or more *subtasks* to perform asynchronous operations; it can go into a wait state until certain *events, signalled* by other tasks, occur, On can specify *on-units* to be activated when *conditions* are raised (interrupts or traps). These are the terms needed to describe what JANUS does.

Equally important is the philosophy. PL/I is a modular language with the built in attitude, "If you don't know about this option, it isn't there. I will do what you most likely want done." With only 16k words of core memory, one must necessarily begrudge the presence of excess code. This is the philosophy by which JANUS works.

It should be emphasized that JANUS does not require the PL/I compiler to operate, nor do we plan to write one. The concepts are quite useful without the compiler.

Timeshared monitors

A monitor is used to provide certain functions, such as control and I/O, which a user either does not want to implement himself, or cannot be trusted with. However, these functions may be made modular in form, and can thus be loaded from a library.

The amount of code that truly must be resident in a timesharing system is really quite small. A scheduler (JOB CHANGER) and RAD handler (SWAPPER) are, of course, required. A console teletype handler is rather important to initiate system actions and to provide a common voice-ear for all users. Other I/O handlers are not required.

If one extends the concept of a task to include all the mastermode routines required for its execution, many interesting by-products result. First, the resident requirements are drastically reduced. If no one is reading cards, no card reader handler is in memory. Second, each task can have a "tailor-made monitor." If task A never does realtime operations, it has no such handlers in its mastermode end. Third, and perhaps most important, monitor routines need not be re-entrant. Each talks only to one user.

All the resident must do is act as referee. Peripherals, indeed any nonshareable system resources (interrupts, extra register blocks, disc space) are handed out on a first come, first served basis, and passed on to the next requestor when released by the current user.

Naturally, such a scheme requires that all mastermode routines be "honest," and completely debugged, but this is a rather ordinary requirement. A malicious or naive mastermode routine can cause damage any number of ways, so JANUS assumes that no such routine exists and performs virtually no checking. While it may appear that spreading the responsibility for system integrity over many routines increases the programming load, the fact that each routine is a module with clearly defined rules of construction actually makes the coding job much easier.

Thus, JANUS is a system that timeshares monitors, in the ordinary sense, and is itself only a referee. Each monitor essentially patches together the small computer it needs to perform a specific operation, leaving all unused system resources available for other tasks. Since, in general, no one job requires more than a few private resources, another subsystem can be constructed from the remainder, and then another, until the entire system is active.

In the MSU implementation, the JANUS resident occupies 3.5k words (7 pages).

Resident and system tasks

The RAD is arbitrarily divided into approximately 680 diskpages, each holding 512 words of useable information. Each distinct page of a task or file has a unique *diskname* (16-bit) by which JANUS can locate it. The first page of every task begins at location X′ C00′ under the map and is called the *task control page* (TCP). All information required by JANUS to bring in a task and start it up is stored in fixed locations in the first 34 to 283 words of the TCP, depending on the size of the task. Thus, JANUS needs only know the diskname of the TCP, called the *taskname,* to bring in the whole task. With 16-bits of status information, the taskname becomes a one-word entry on the resident ring of active tasks.

Note that a task sees only 3.0k of JANUS. (Under the map, each task is executing in its own unique address space of up to 128k-words, generally independent of the available core memory. Task address spaces diverge after 6 pages: the lower 3k are common to all tasks and contain the major portion of JANUS.) The remaining resident page is the TCP of the HOUSEKEEPER task, the nonresident supervisor. Rather than maintain complete lists of resources and requests in resident, JANUS maintains only token lists, to meet short-term needs, and invokes the HOUSEKEEPER as needed to tidy up lower core.

JANUS also serves as intermediary between interrupt routines, which run unmapped in dedicated memory, and their controlling tasks, which are mapped and not necessarily in memory at any given time. The most important service provided is the MESSAGE CENTER, the *only* multiply re-entrant resident routine in the system. It will accept a one word *signal* (consisting a 16-bit taskname and an 8-bit identifier) from any source and pass it on to the specified task, pulling it out of a wait state if necessary. Great pains have been taken to ensure that no signals are lost, duplicated or rejected by JANUS.

Communication in the other direction—alerting interrupts—is generally possible through the hardware. Interrupts can be triggered under program control. But since the I/O interrupt has a software fan out to individual device handlers, JANUS provides a software device-directed trigger, or "kick," to aid communication. A pseudo-acknowledgment status word is added to a queue and the interrupt is triggered. The resident interrupt routine, after acknowledging all real interrupts, empties the queue, passing the contents to the individual routines. This feature simplifies coding of I/O handlers, since all con-

trol operations can be confined to the interrupt end of the handler.

The last major component of the resident is the console teletype handler, used by all tasks to communicate with the operator. Output is sequential in order of request and may interrupt an input. All console teletype I/O is prefix directed to the appropriate task. A prefix pool is available to provide unique prefices upon request. Prefices consist of a byte count, plus up to 3 characters.

One prefix, the ampersand (&) is reserved for directing messages to JANUS. A special task, the AMPERSCANNER, decodes these messages and takes appropriate action. The AMPERSCANNER knows where to find, or how to generate, all tasks built into the system, and thus is responsible for initiating most processes.

A third system task is the MORTICIAN, which dissects dead tasks and files, and returns their component pages to the system diskpage pool.

Other system tasks are an open-ended set of symbionts, used to perform an I/O operation with a disk file. Tasks may either do their own I/O or generate a file, and let the appropriate symbiont perform the necessary I/O operations. Symbionts do not take up any memory space unless they are actively working on a file. Only the system tasks are immortal under JANUS. All other tasks are brought into existence for a specific application and are interred when no longer needed.

Operation

Job changer

Figure 4 is a flow chart of the JOB CHANGER, which is responsible for entering and leaving tasks between time slices, and for delivering messages (signals) to tasks. The lowest priority interrupt is the job-changing interrupt. It fires when its corresponding count pulse interrupt counts to zero (or when triggered by a CPU instruction), and transfers control directly to the task monitor to perform register saving and any other slice-end functions needed before branching to the JOB CHANGER.

Similarly at slice-start, the JOB CHANGER gives the task monitor an opportunity to process signals and restore registers before continuing. A special purpose task thus has the option of streamlining both ends of the process. Note that a task also has the option of putting itself on high priority, a status such that it must be the next task executed, although this practice is discouraged.

No attempt is made to compute an "optimal" slicetime. This interval is fixed by an empirical tuning process and is currently 0.1 second.

Swapper

A flow chart of the SWAPPER algorithm is shown in Figure 5. Once the TCP of the next task is in memory, the SWAPPER works from a variable length table in the TCP to determine what pages *must* be brought in (or located in memory) to permit the task to proceed.

Since it takes an average of 28 milliseconds to read or write one page of memory to the RAD, whenever possible great effort is made to avoid actually swapping. Some of the techniques used are:

1) Storage areas are grouped together, beginning on a page boundary, so that all other pages can be flagged "read only." Thus, the SWAPPER knows not to write these pages back to the RAD.

2) Slavemode storage areas are at first write-protected, so that the task monitor can inform the SWAPPER (via the TCP table) whether or not a page has been modified.

3) Only those pages of a task which are known to be needed (or for which usage cannot be monitored) during the next timeslice are flagged "must be in next time" or "must be in everytime."

4) A four-level "usage priority" is maintained for each page of real memory (see Figures 4 and 5). After each timeslice it is set to a high value if used, reduced toward zero if not. As a result, pages are "turned" according to a weighted LRU (Least-Recently Used) algorithm. A page which must be written back has a higher weight than one which does not. Other weight criteria may also be established. This simple trick helps the JOB CHANGER and SWAPPER "learn" the best way to use memory as the system load changes.

Such great pains were taken to improve swap efficiency, in fact, that a wholly unexpected by-product was created. Slavemode memory usage could be monitored so closely that *the slave portion of a task need never be all in memory at once,* but instead pages would be brought into core upon demand. Thus a problem-solving program could be written using the full 128k address space

FIGURE 4—Flow chart for JOB CHANGER (scheduler). It operates at lowest priority interrupt level.

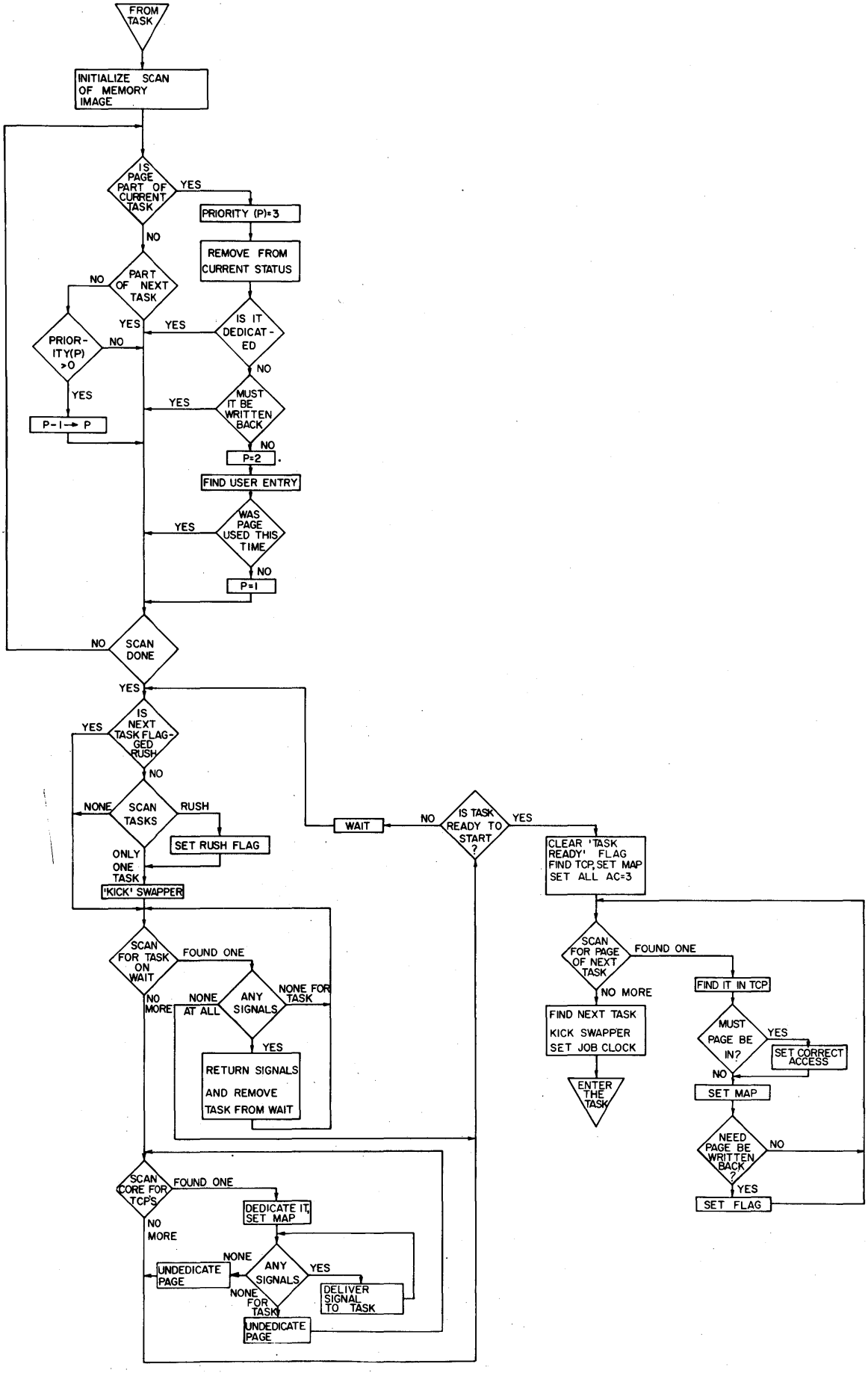
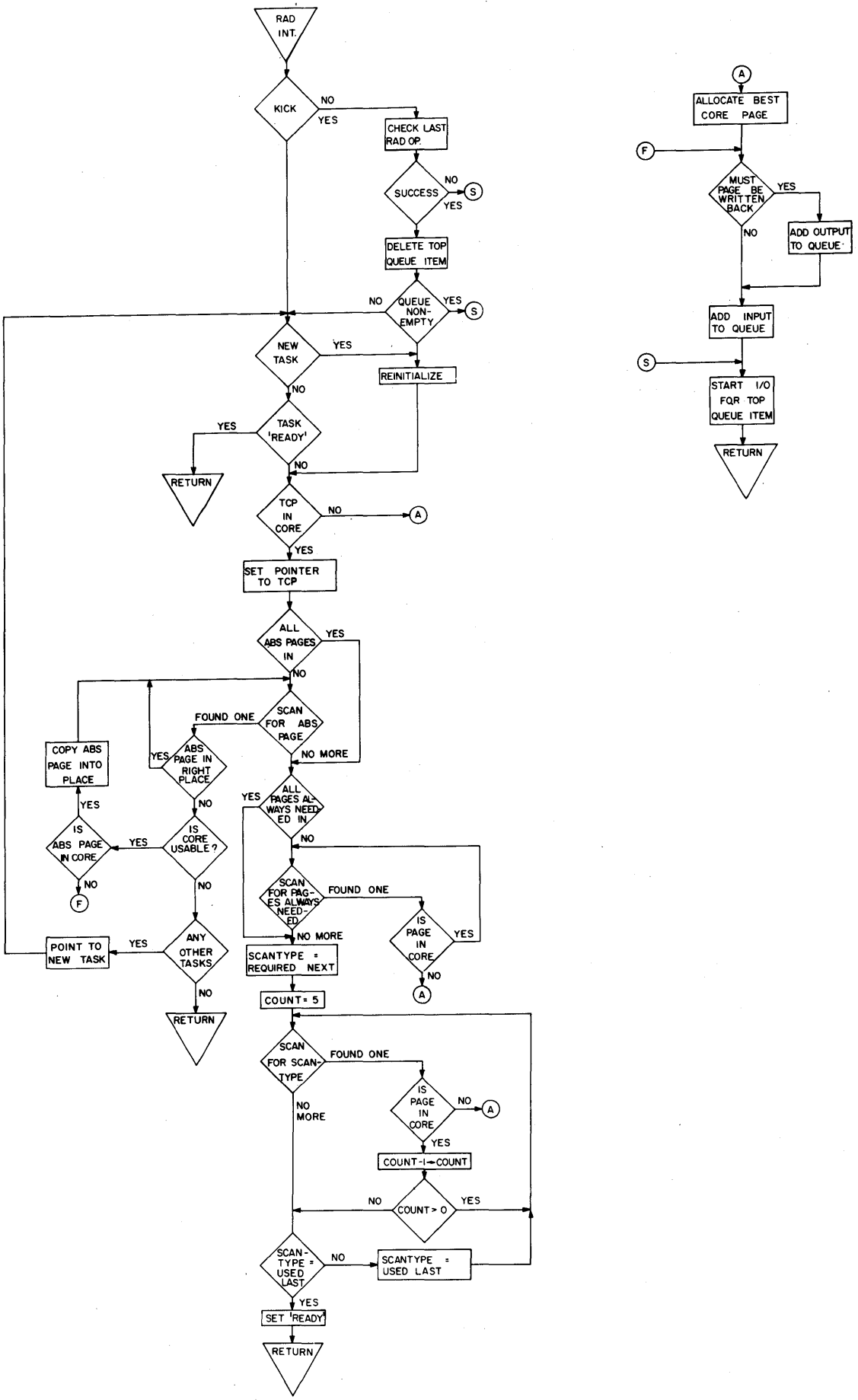


FIGURE 5—Flow chart for SWAPPER. It operates at I/O interrupt level whenever RAD operation completes or software "kick" is administered by JOB CHANGER.



of the machine and, *with no alteration,* run on a Sigma-7 with as little as 8k of actual memory, although less efficiently than in the full machine.

Demand paging

Figure 6 is a flow chart of the demand paging algorithm. It is implemented as an optional task monitor module (a pure process control task would have no use for this feature), and occupies 250-words. Whenever an access protect violation is signalled via the Nonallowed Operation trap, the routine is entered to determine a) what page reference caused the trap and b) whether it was indeed an error or a valid reference. If the latter, the page is made available and usage is noted.

The routine is time consuming because the access protects were not originally designed for this purpose, and hardware deficiencies must be made up in software. But each page traps at most twice during each 0.1 second timeslice, and the possible gain in information weighs heavily against 28 milliseconds per page swaptime.

Except for the Execute instruction, which is infinite level, no Sigma-7 instruction can access more than five pages before reaching an exit point. With special handling for execute, this five page limit applies to all cases. Consequently, timesharing can be maintained as long as there are sufficient non-dedicated pages of memory to hold the largest task monitor, plus five slave-mode pages. The MSU Sigma-7 thus has over half its 16k memory dedicable for I/O and other realtime processes.

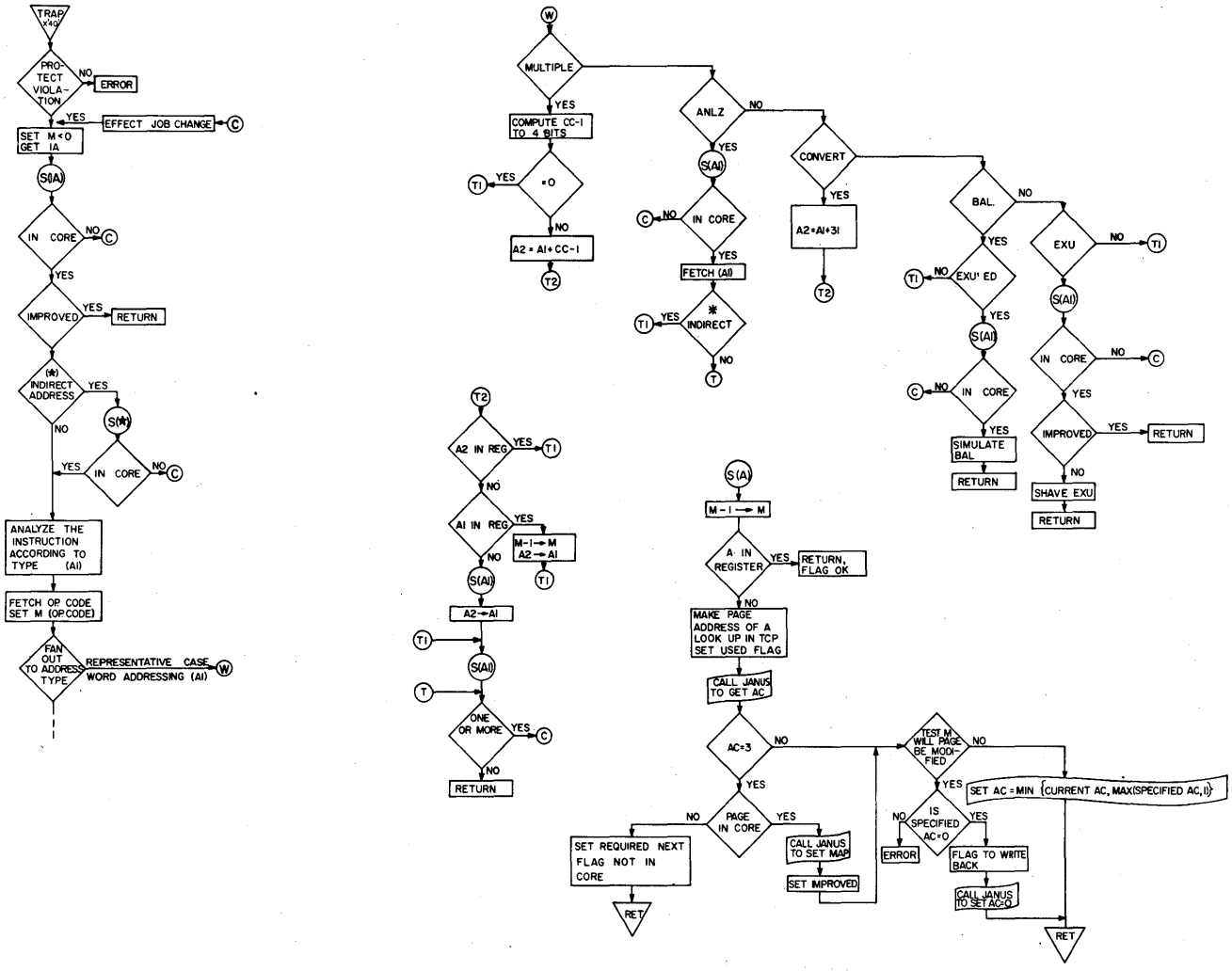


FIGURE 6—Flow chart of Demand Paging algorithm, a 250-word routine which may be included in a task monitor to permit task to occupy full 128k virtual memory. It operates whenever Nonallowed Operation trap occurs in the task.

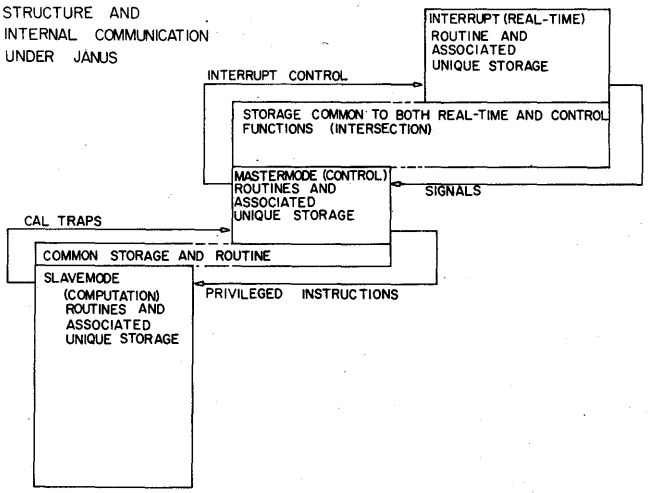


FIGURE 7—Typical realtime (I/O) driver. This illustrates optional three-part division of labor and modes of communication between parts.

Efficiency

It is difficult to measure the overall efficiency of the JANUS system, because loading changes so rapidly with even the simplest uses. Several extreme cases can be cited, however, to provide a feel for overall parameters:

1) In the case where all active tasks fit in memory at once, the SWAPPER "learns" in about 1 second that all codes should be in memory. Afterward, the time spent in the JOB CHANGER and SWAPPER (overhead) drops to less than 5% total CPU time.

2) The FORTRAN optical model code GIBELOMP, which requires approximately 35k words of slave code in a Sigma-7, registers a swap overhead (the fraction of time not spent in slave problem-solving mode) of 20% when running alone in the machine.

3) While it is possible to create pathological cases where swap overhead would exceed 99%, system processors and compiled code seldom burst over 50% overhead.

These figures are, of course, degraded as available memory decreases or external interrupt load increases.

CONCLUSION

JANUS has proved to be a viable solution to the needs outlined in the introduction. By extending the concept of realtime operations to include *all* I/O operations, a small resident is possible, permitting the use of most of memory for production. Whether that production is realtime data acquisition, computation, or a mix is irrelevant—it can proceed if the necessary resources are available. Any realtime process can interrupt the system or background processes at all times, permitting high data rates. The ability to run large computation codes with reasonable efficiency, and in a timesharing environment, is an unexpected boon.

While the concept of timesharing monitors does permit a minimal resident supervisor, it would appear to suffer from the need of writing specialized monitors for the library. However, certain facts should be kept in mind. Each monitor is no more complex than a corresponding stand-alone system would be—only the rules are different. Each monitor need contain only those functions necessary and sufficient for the task in question. The various functions are modular; once a specific function has been coded, it may be placed in a library for future use, rather than having to be recoded again and again. Given such a library of monitor functions, it is possible to define a system loader (actually a task maker) which can generate a tailor-made monitor. This sort of operation is not uncommon in single-user monitors, where, for example, a magnetic tape handler is included only if the programmer references it implicitly or explicitly. It is not an overly restrictive requirement that the possible set of operations to be performed be known before execution is started.

Demand paging requires some form of map with protection, as well as an external storage medium. Although there is some controversy as to the value of demand paging in a timesharing environment, JANUS demonstrates the value of the concept in situations of single user, limited memory machines. While overhead for problems which fit into core is minimal, larger problems can also be run, although at lower efficiency, at less monetary cost than would be incurred by an increase in core memory. Given demand paging, additional memory increases efficiency. Comparing an open-shop, small demand-paged computer with a closed-shop computation center, turn around time is significantly reduced, even if the actual computer running time is increased by orders of magnitude.

The Sigma-7 is an admirable machine for timesharing, but was not designed with demand paging in mind. Certain features necessary for demand paging, while capable of being implemented

by software, would be much more efficient if implemented in hardware. These include: 1. Scratchpad registers, readable and clearable under computer control, which would automatically keep track of all pages referenced and modified. 2. A more powerful hierachy of operation modes and a different sequence of protection. A better sequence of protection might be: write permitted, write permitted from master-mode, no write permitted, and no access permitted. Master-slave modes of operation should be expanded to the point where the protection could optionally apply to master-mode thus permitting demand paging operation of such code. 3. A privileged instruction which would interpretively execute any instruction, and return the reference which would cause a trap. Other features, such as a readable map, would be of great use.

In conclusion, we feel that JANUS may well serve as a model for future computer systems, due to its flexibility and lack of constraints. As much as possible, limitations on computer usage are set by hardware, rather than by system conventions.

REFERENCES

1 J A JONES
On-line computers for research
Nucleonics p 34 Jan 1967 A survey of the field containing many references which will not be duplicated here

2 J BIRNBAUM M M SACHS
Computer and nuclear physics
Physics Today p 43 July 1968

3 *SDS Sigma-7 computer reference manual (#90009-506)*
Scientific Data Systems Santa Monica California May 1967

4 B W LAMPSON
A scheduling philosophy for multiprocessing systems
Communications of the ACM p 347 May 1968

5 *IBM System/360 operating system—PL/I language specifications (#C28-6571-4)*
IBM Corporation New York New York December 1966

A parallel process definition and control system

by DAN COHEN

Harvard University
Cambridge, Massachusetts

INTRODUCTION

(1) General

The purpose of this work is to supply a simple method for definition and efficient control of a network for asynchronous parallel processing.

The system is able to compile a definition of a network of processes which run partially in parallel and partially sequentially, into a set of control instructions for some monitoring process, to be executed at run time.

The defined network is composed of a set of interrelated processes and a monitoring process which initiates processes according to some dynamic conditions. Typical resources for processes are processors, I/O devices, memory banks, and bulk storage.

The system discussed below is concerned with two tasks, the handling of raw input statements (the network definition language and its parsing algorithms), and the network control process, which initiates and inhibits processes as they become needed or unnecessary.

(2) *The states of the processes*

Each process at any time can be in one of the following states:

(a) reset (not initiated)
(b) initiated
(c) successfully completed
(d) failed
(e) not needed

However there is no need to distinguish between states (d) and (e). Each process has three binary variables f, g and h to indicate its state.

f = 1 : the process was initiated.
g = 1 : the process was successfully completed.
h = 1 : the process failed or was found not needed.

The f variable is set externally to initate a process.

The g variable is set internally by the process to indicate its completion. The h variable may be set internally to indicate failure, or externally to indicate that this process is not needed.

The combinations of f, g, h may be interpreted as:

f	g	h	
0	0	0	reset
0	0	1	the process is not needed
0	1	0	illegal
0	1	1	illegal
1	0	0	initiated
1	1	0	sucessfully completed
1	0	1	failed or not needed
1	1	1	successfully completed

(3) *The control language*

Each process, p, has two propositions associated with it:

b(p) which is 'TRUE' if p has already started, and c(p) which is 'TRUE' if p was successfully completed.

A network is defined by a set of statements $\{S_i\}$ of the form:

$$S_i : f_i(c(p_1), c(p_2), c(p_3)...) \Rightarrow b(p_{ji}) \text{ [1]}$$

where the $\{f_i\}$ are propositional functions, which contain *AND* and *OR*.

The meaning of S_i above is that when f_i is 'TRUE' then $b(p_{ji})$ is set to 'TRUE' (which initiates the process p, if it was not already initiated). For a more convenient notation, we write p_k for $c(p_k)$ on the left hand side, and write p_k for $b(p_k)$ on the right hand side.

Statements with identical propositional functions are combined by writing their propositional functions on the left, and writing the set of all their right hand sides, on the right. Example:

---

[1] $\Rightarrow$ means logical implication. e.g. $x \Rightarrow y$ means that $x = 1$ implies $y = 1$

$$c(A) + c(B) \,\&\, c(C) \Rightarrow b(D)^{2}$$

$$c(A) + c(B) \,\&\, c(C) \Rightarrow b(E)$$

is written as:

$$A + B \,\&\, C => D, E^{3}$$

An example for definition of a network

$s \Rightarrow A, B$	read: start with processes A and B (s for *start*)
$B \,\&\, C \Rightarrow D, E$	read: When B *and* C are successfully completed initiate D and E
$A + B \Rightarrow C, D$	read: When A *or* B is successfully completed initiate C and D
$C \,\&\, D + E \Rightarrow e$	read: When (C *and* D) *or* E is successfully completed, end (e for *end*).

Let L be the left hand side of a statement (i.e, the propositional function) and let R be the set of the processes which are initiated when L is 'TRUE.' Hence each statement has the form:

$S_i : L_i \Rightarrow R_i$ (read: L_i *implies* R_i)

We say that p is *used* in $L \Rightarrow R$, if L contains p.[4]

Any process which is used by the network, but is not initiated by it, like s, is called a *source-process*, and its beginning is a *source* of the network.

Any process which is initiated by the network, but not used by it, like e, is called a *sink-process*, and its end is a *sink* of the network.

Each network must have at least one source. A circuit-free network has at least one source and one sink.

(3.1) The control language processor

The input statements are compiled into a set of control instructions to be executed at run time. This compilation has two stages. The first stage checks validity (and outputs diagnostics) and reduces the input statements to some "graphable" form.[5] The second stage produces control instructions according to the reduced statements.

(a) check validity (3.2.1)
(b) simplification (I), if possible (3.2.2)
(c) transform the network so that each process is initiated only once (3.2.3)
(d) separate *AND* from *OR* (3.2.4)
(e) transform the network so that each process is used only once (3.2.5)
(f) simplification (II), if possible (3.2.6)

(3.2) The reduction procedure

Each simplification (I) step is applied as soon as it applicable.

This ordering of the steps insures that no reduction steps except simplifications are needed as a result of a later step.

After this procedure the input statements are reduced to a "graphable" form, where the definition statements correspond to vertices, and the processes correspond to edges. Because of the *AND/OR* separation (3.2.4) each node can be represented by an *AND* or *OR* gate.

During the reduction *dummy processes* are created, which can be recognized as such, thereafter. The role of these dummy processes is similar to the role of temporaries in program compilation. All dummy processes are created and named only by the compiler. We will name all dummy processes hereafter by q_i. Dummy processes have no duration, and do not need any time or resources to be executed. They serve to eliminate ambiguities in the network definition. The nature of these processes will be made clear in (3.2.3), (3.2.4) and (3.2.5) below.

Note that this reduction is not a complete optimization.

(3.2.1): validity checking.

There must be at least one source to any network.

(3.2.2): simplification (I)

The simplification is not essential, however it may save time and space and therefore is desirable.

(a) is s is the only source of the network then replace s & p by p, and s + p by s, in any L. (Note that we drop subscripts for processes, for simplicity.)
(b) If e is the only sink of the network, then if $\{e,p\} \in R$, delete p from R. If p is initiated only in this statement, replace c(p) by 'false' wherever p is used.
(c) Apply any available method of propositional calculus simplification, to L. (i.e., replace A +

---

[2] + and & mean logical *OR* and *AND*. The *AND* takes precedence over the *OR*.

[3] D, E means the set {D, E}.

[4] i.e., c(p) appears in the propositional function, L.

[5] The meaning of a "graphable" form is explained later, in (3.2)

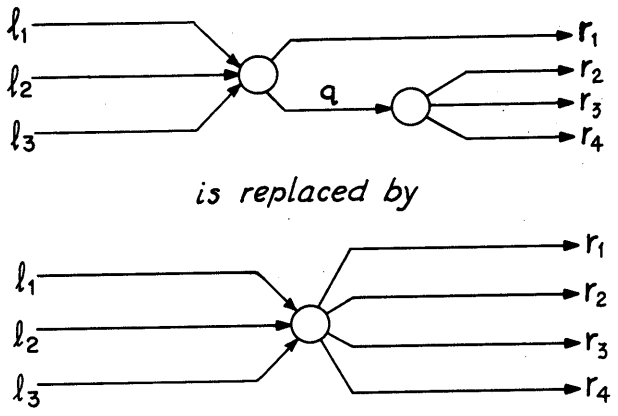


FIGURE 1—An example for rule (3.2.2.f)

A by A, and A + A & B by A.)

(d) Delete $L \Rightarrow$ [6]

(e) Replace $L \Rightarrow R_1$ and $L \Rightarrow R_2$ by $L \Rightarrow R_1, R_2$

(f) Delete $q \Rightarrow R$ (where q is a dummy-process), and replace q by R, where q was initiated. See Figure 1.

(g) Delete $p \Rightarrow q$ (where q is a dummy-process, and p a process) and replace q by p wherever q is used. See Figure 2.

(h) Delete statements of the form 'FALSE' $\Rightarrow$ R. If some process $p \in R$ is initiated only in this statement, replace c(p) by 'false' wherever p is used.

(3.2.3): Uniqueness of initiation

If any process is initiated more than once then there are $L_1 \Rightarrow R_1$ and $L_2 \Rightarrow R_2$ such that $R_1 \wedge R_2 \neq \emptyset$, i.e. there are common terms in the sets R_1 and R_2. In this case replace $L_1 \Rightarrow R_1$ and $L_2 \Rightarrow R_2$ by:

$$L_1 \Rightarrow q_1, R_1 - (R_1 \wedge R_2)$$

$$L_2 \Rightarrow q_2, R - (R_1 \wedge R_2)$$

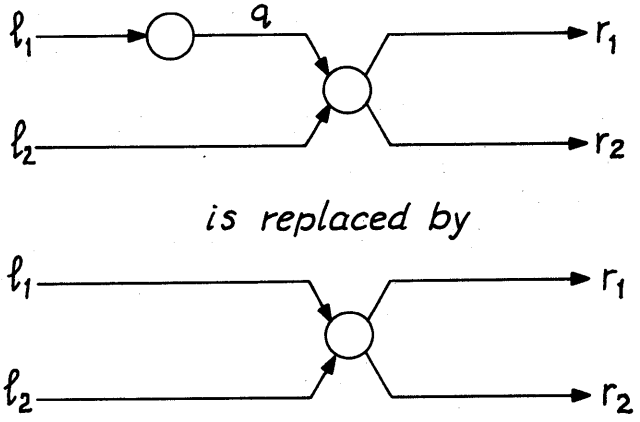


FIGURE 2—An example for rule (3.2.2.g)

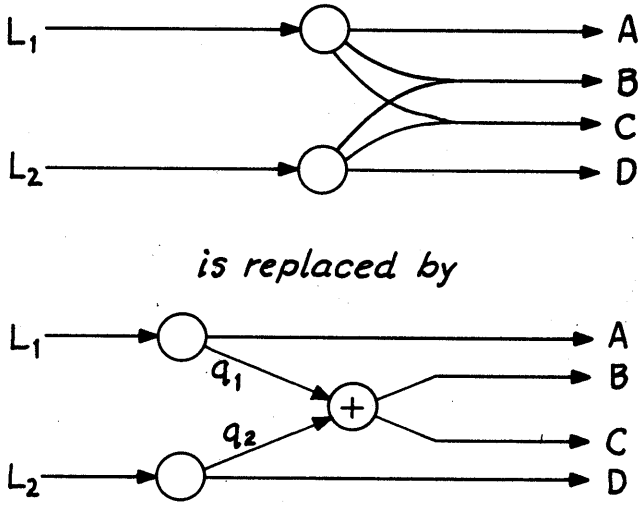


FIGURE 3—An example for rule (3.2.3)

$$q_1 + q_2 \Rightarrow R_1 \Lambda R_2$$

where q_1 and q_2 are new dummy processes.

Example: $L_1 \Rightarrow$ A, B, C and $L_2 \Rightarrow$ B, C, D are replaced by:

$$L_1 \Rightarrow q_1, A$$

$$L \Rightarrow q_2, D$$

$$q_1 + q_2 \Rightarrow B, C$$

This rule is illustrated in Figure 3.

(3.2.4): *OR/AND* separation

Any statement which contains both *AND* and *OR* is converted to several statements which use only *AND* or *OR*. Some dummy processes are used to represent sub-combinations of processes (the same way compilers use temporaries). A simple precedence reduction analysis suffices. For simplicity here, let us assume that L is converted to, or is given as a sum of products: $L = \sum \pi_i$. Sum of products may be simplified by using the idempotant and absorption rules.[7] In this case replace $L \Rightarrow R$ by:

$$\pi_1 \Rightarrow q_1$$

$$\pi_2 \Rightarrow q_2$$

$$\vdots$$

$$\pi_n \Rightarrow q_n$$

$$\sum q_n \Rightarrow R$$

[6] $\emptyset$ is the empty set.

[7] The absorption rule: $x + x \& y = x$ The idempotent rule: $x + x = x$ and $x \& x = x$

where the $\{q_i\}$ are new dummy processes. Repeat this step as necessary, until all the statements are *AND*-statements or *OR* statements.

The similarity between dummy processes and temporaries in program compilation may be illustrated by the following example. Consider the statement:

$$A + B * C * (D + E) \Rightarrow F$$

As an arithmetic statement it may be compiled as:

$$\left.\begin{array}{l} D + E \Rightarrow T_1 \\ B * C * T_1 \Rightarrow T_2 \\ A + T_2 \Rightarrow F \end{array}\right\} \begin{array}{l} D + E \Rightarrow T_1 \\ T_1 \text{ and } T_2 \text{ are temporaries} \end{array}$$

As a process statement it may be compiled as:

$$\left.\begin{array}{l} D + E \Rightarrow q_1 \\ B \,\&\, C \,\&\, q_1 \Rightarrow q_2 \\ A + q_2 \Rightarrow F \end{array}\right\} \quad q_1 \text{ and } q_2 \text{ are dummy processes}$$

See Figure 4.

(3.2.5): uniqueness of usage

If some process, p, is used $m > 1$ times, then add $p \Rightarrow \{q_1, q_2 \ldots q_m\}$, and replace p in its nth usage by q_n (these $\{q_n\}$ are new dummy processes). See Figure 5.

(3.2.6): simplification (II)

Simplification (II) like simplification (I) is not essential, but may save time and space. It may be done only after (3.2.3) *OR/AND* separation, and (3.2.4) uniqueness of initiation, are completed.

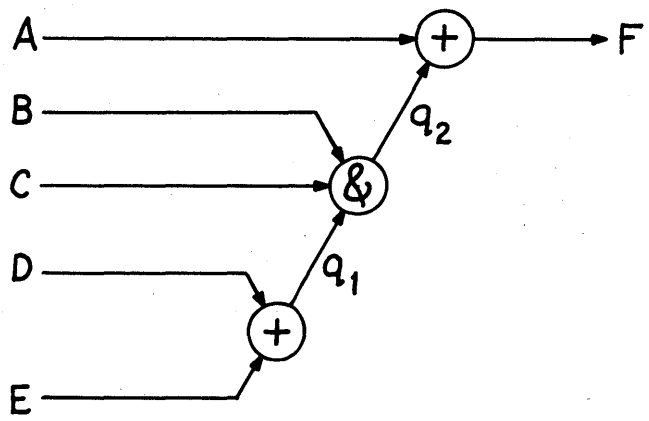
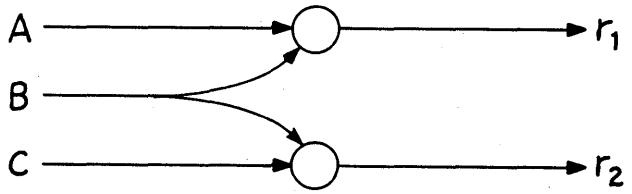


FIGURE 4—An example for rule (3.2.4)



is replaced by

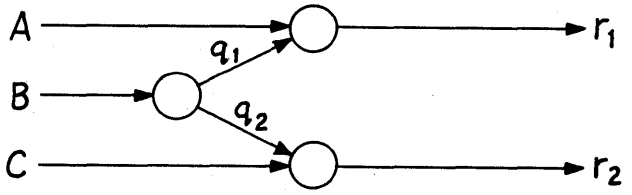


FIGURE 5—An example for rule (3.2.5)

(a) $p + E_1 \Rightarrow q$ and $q + E_2 \Rightarrow R$, where q is a dummy process is replaced by $p + E_1 + E_2 \Rightarrow R$. See Figure 6.

(b) $p \,\&\, E_1 \Rightarrow q$ and $q \,\&\, E_2 \Rightarrow R$, where q is a dummy process is replaced by $p \,\&\, E_1 \,\&\, E_2 \Rightarrow R$. See Figure 7.

(3.3): Example

(a) The input statements:

(S_1) $s \Rightarrow A, B$
(S_2) $A + B \Rightarrow C, D$
(S_3) $B \Rightarrow D, E$
(S_4) $C \,\&\, D + E \Rightarrow e$

(b) D is initiated twice, in S_2 and in S_3. Apply (3.2.3):

(S_1) $s \Rightarrow A, B$
(S_2) $A + B \Rightarrow C, q_1$
(S_3) $B \Rightarrow q_2, E$
(S_4) $C \,\&\, D + E \Rightarrow e$
(S_4) $q_1 + q_2 \Rightarrow D$

(c) S_4 contains both *AND* and *OR*. Apply (3.2.4):

(S_1) $s \Rightarrow A, B$
(S_2) $A + B \Rightarrow C, q_1$
(S_3) $B \Rightarrow q_2, E$
(S_4) $q_3 + q_4 \Rightarrow e$
(S_5) $q_1 + q_2 \Rightarrow D$
(S_6) $C \,\&\, D \Rightarrow q_3$
(S_7) $E \Rightarrow q_4$

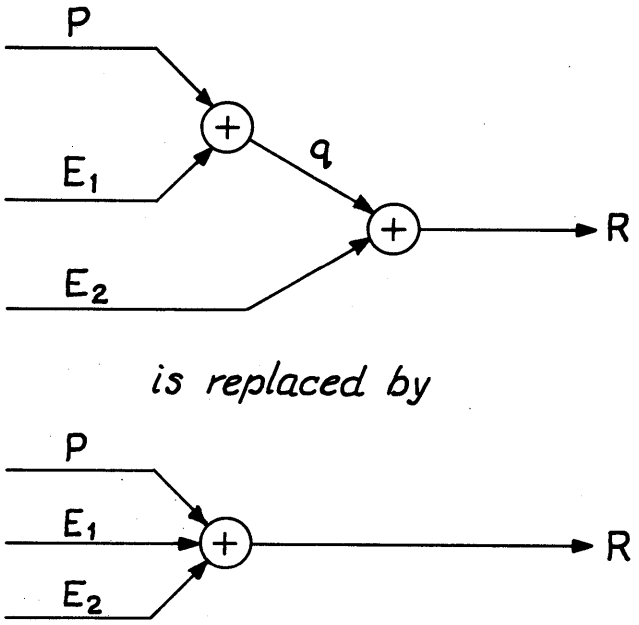


FIGURE 6—An example for rule (3.2.6.a)

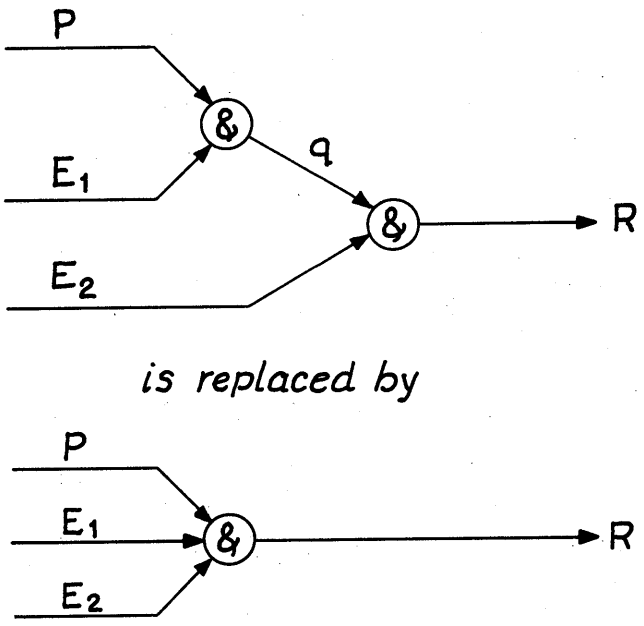


FIGURE 7—An exmple for rule (3.2.6.b)

(d) step (g) of simplification I, can be applied to S_7, and S_4:

(S_1) $s \Rightarrow A, B$
(S_2) $A + B \Rightarrow C, q_1$
(S_3) $B \Rightarrow q_2, E$
(S_4) $q_3 + E \Rightarrow e$
(S_5) $q_1 + q_2 \Rightarrow D$
(S_6) $C \,\&\, D \Rightarrow q_3$

(e) B is used twice, in S_2 and in S_3. Apply (3.2.5)

(S_1) $s \Rightarrow A, B$
(S_2) $A + q_5 \Rightarrow C, q_1$
(S_3) $q_6 \Rightarrow q_2, E$
(S_4) $q_3 + E \Rightarrow e$
(S_5) $q_1 + q_2 \Rightarrow D$
(S_6) $C \,\&\, D \Rightarrow q_3$
(S_7) $B \Rightarrow q_5, p_6$

(f) step (f) of simplification I, can be applied to S_3, and S_7:

(S_1) $s \Rightarrow A, B$
(S_2) $A + q_5 \Rightarrow C, q_1$
(S_3) deleted
(S_4) $q_3 + E \Rightarrow e$
(S_5) $q_1 + q_2 \Rightarrow D$
(S_6) $C \,\&\, D \Rightarrow q_3$
(S_7) $B \Rightarrow q_5, q_2, E$

No more reduction rules can be applied. The network is now in a "graphable" form, as shown in Figure 8.

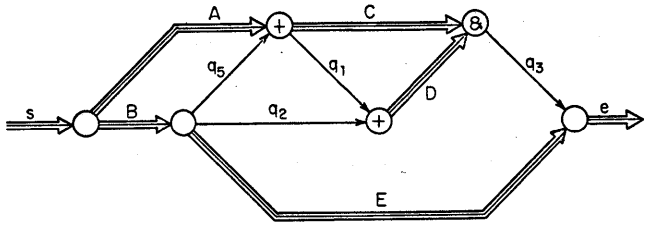


FIGURE 8—The network of example (3.3)

(4) *Compiling the input statements to control instructions*

Now we assume that the input statements have been reduced to their "graphable" form, as described before. The next goal is getting instructions for updating the process variables f,g and h. These instructions are executed at run time by some monitoring process. Let $\{\ell_i\}$ be the processes used in L, and $\{r_j\}$ the processes in R.

(4.1) A set of rules for compiling $\sum \ell_i \Rightarrow \{r_j\}$ can be written:

(4.1.1) If *any* ℓ_i in L succeeds, initiate R (success forward):

$$\sum g(\ell_i) \rightarrow \{f(r_j)\}^8$$

[8] $\rightarrow$ means replacing (setting) e.g. $x \rightarrow y$ means replace y by $x + y$.

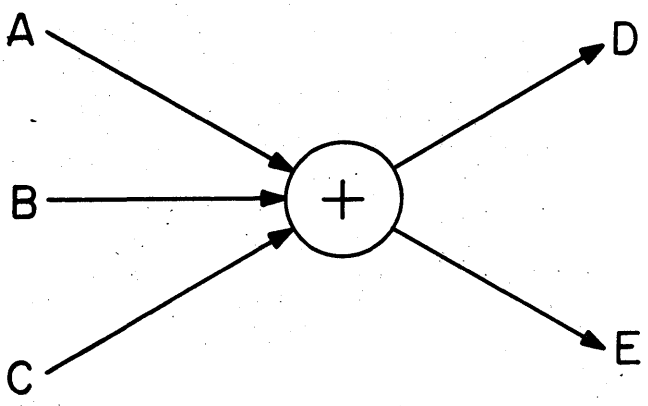


FIGURE 9—An example for (4.1)

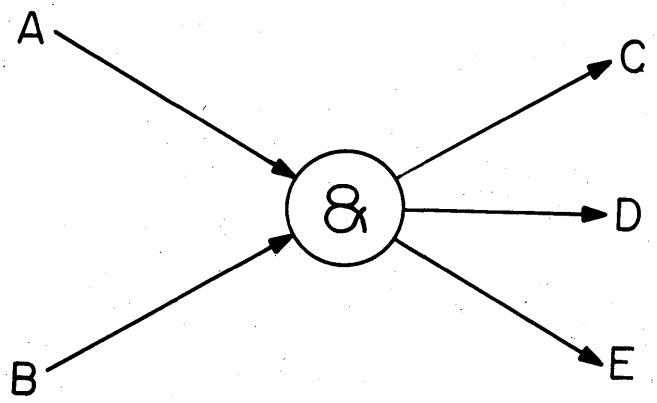


FIGURE 10—An example for (4.2)

Example (Figure 9): if A, *or* B, *or* C succeeds, initiate D and E.

(4.1.2) If *all* $\{\ell_i\}$ in L fail, then R is not needed (failure forward):

$$\prod \bar{g}(\ell_i) \cdot h(\ell_i) \rightarrow \{h(r_j)\}^9$$

Example (Figure 9): is A, *and* B, *and* C fail, so do D and E.

(4.1.3) If R is not needed, then L is not needed either (failure backward):

$$\prod h(r_j) \rightarrow \{h(\ell_i)\}$$

Example (Figure 9): if D *and* E are not needed, neither are A, B and C.

(4.1.4) If any ℓ_i in L succeeds, then its brothers are not needed (inhibit brothers)[10]

$$\sum g(\ell_i) \rightarrow \{h(\ell_i)\}$$

Example (Figure 9): if A succeeds, B and C are not needed. If B succeeds A and C are not needed, etc.

(4.2) A set of rules for compiling $\prod \ell_i \Rightarrow \{r_j\}$ can written:

(4.2.1) If *all* ℓ_i in L succeed, initiate R (success forward):

$$\prod g(\ell_i) \rightarrow \{f(r_j)\}$$

Example (Figure 10) if A *and* B succeed then initiete C, D, and E.

(4.2.2) If *any* ℓ_i in L fails, then R is not needed (failure forward)·

$$\sum \bar{g}(\ell_i) \cdot h(\ell_i) \rightarrow \{h(r_j)\}$$

Example (Figure 10): *if* A *or* B fails, so do C, D and E.

(4.2.3) If R is not needed, then L is not needed either (failure backward):

$$\prod h(r_j) \rightarrow \{h(\ell_i)\}$$

Example (Figure 10): if C *and* D *and* E are not needed neither are A and B

(4.2.4) If any ℓ_i in L fails, then its brothers are not needed (inhibit brothers):

$$\sum \bar{g}(\ell_i) \cdot h(\ell_i) \rightarrow \{h(\ell_i)\}$$

Example (Figure 10): if A fails B is not needed. If B fails A is not needed.

The statement $p \Rightarrow R$ can be compiled according to either rules. However we consider one input node as an AND node. Note that (4.2.4) is not necessary as it is implied by (4.2.2) and (4.2.3).

For each dummy process, q, the setting instruction of f(q) is replaced by the setting instruction of g(q).

The purpose of the rules set forth in 4.1 and 4.2 is to inhibit the execution of processes which are not needed, and to initiate the execution of processes which are needed. A network with one sink only, is always inhibited upon arrival at it.

[9] The overbar means logical complementation, e.g. $\bar{0} = 1$ and $\bar{1} = 0$.

[10] Processes which terminate at a common vertex are called "brothers" here.

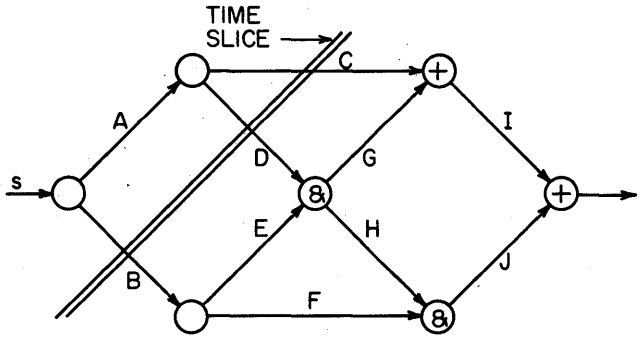


FIGURE 11—The network for example (4.3)

(4.3) Example:

I. Consider the time-slice B–D–C, as marked on the graph in Figure 11. Say that D fails there at that time. Then (4.2.2) inhibits G and H, (4.2.3) or (4.2.4) inhibits E, (4.2.2) inhibits J. and (4.2.3) inhibits F and B. This means that process B, already initiated, is told to quit, since its success is not necessary to produce "e". This leaves only A, C and I not inhibited.

II. Consider process I completed. Process J is now deemed useless, and is turned off, as are its "parents", F and H.

A computer system for automation of a laboratory

by P. J. FRIEDL, C. H. SEDERHOLM, T. R. LUSEBRINK and C. J. JENNY

IBM Scientific Center
Palo Alto, California

INTRODUCTION

In the past, scientists have applied the digital computer in an off-line fashion to the problems of analyzing the voluminous data produced by their laboratory instruments. The use of computer techniques for digital filtering, peak finding, and spectral decomposition has greatly increased the rate at which experimental data can be analyzed. More recently, many instrument manufacturers and users have also begun to apply computers on-line to their instruments in an effort to increase the rate at which useful experimental data can be obtained. This paper describes a laboratory automation computer system which simultaneously supports multiple closed-loop experiments and data analysis programs.

Let us first distinguish between instrument automation and laboratory automation. In the former case a single computer, usually small, is devoted to a single instrument, or at least to one instrument at a time; whereas in laboratory automation, a group of instruments in a laboratory is automated using one central computer system. A discussion of the relative advantages of these two modes of automation follows.

There are several advantages to a dedicated computer. The most important of these is that of isolation. Each individual user prefers to concern himself with his own problems. He does not want malfunctions of other instruments or their interfaces to jeopardize his experiment. Programming considerations associated with his own instrument are sufficiently complex that he doesn't want to worry about other users' programming problems too. He is rightfully afraid of being forced to factor parts of his programming requirements into general purpose programs which serve the entire laboratory (programming by committee).

Immediate accessibility to the computer system can be guaranteed in a devoted computer configuration. This fact also encourages one to favor multiple computers, one per instrument; however, if a time-shared computer system could almost always be made available in a period of a few seconds, or at most one or two minutes, this would not be intolerable. The prospects, however, of having to schedule one's experiments and schedule the use of the computer facilities long in advance is very unpalatable to most users. In general, users prefer fewer facilities which are routinely available to larger facilities available by appointment only. Hence, the desire for immediate computer access tends to favor multiple instrument automation over laboratory automation.

Relative costs should also be considered. A computer system which is capable of expanding to handle the requirements of an entire laboratory will, of necessity, be more expensive, even in a minimal configuration, than a computer system capable of automating a single laboratory instrument. Therefore, when one is taking the first step toward laboratory automation, that is, the automation of a single instrument, there is a very strong tendency to automate that instrument using one small computer since the initial cost is considerably lower. Quite understandably, an organization is reluctant to commit large capital and manpower resources to a project in which it has little or no experience. However, if the ultimate goal of completely automating the laboratory is considered, many of the considerations discussed below imply that a single, shared laboratory computer would provide more performance per dollar.

To realize all of the potential of automation, one must not only acquire data but also control the instrument during the data acquisition step,

process the data which has been acquired, standardize it, compare it against known parameters (e.g., compare an unknown spectrum against a file of spectra of known compounds for identification), and finally present the results in a form which is usable to the experimenter. Data acquisition and control steps can very often be adequately performed by a small computer. However, the data reduction steps, the comparison with data files, and the presentation of results in usable form often require much larger computer capabilities. One solution to this is to record the raw data, which has been acquired with a small stand-alone computer, on a recording medium such as magnetic tape or punched paper tape. This data may then be processed on a large computer when time is available. However, if the raw data is at all voluminous, and magnetic tape must be used, the cost of the magnetic tape drive relative to the cost of the small computer can become very high. Hence, this approach tends to encourage one to minimize the quantity of data taken, often resulting in less precise results. Furthermore, the turnaround time on very large computer facilities still is much longer than the individual investigator would like to wait between epochs of his experiment. That is, if he could have the data from the last epoch back quickly, these results could be used to determine conditions for his next experiment.

A larger shared computer has several advantages for the automation of a laboratory. It can have sufficient core and processing capabilities to do a large portion of the data reduction required in most laboratories without resorting to a large central computer. By dynamic allocation of system facilities one may take advantage of the fact that most analytical and spectrocopic instruments have low duty cycles (i.e., much of the time is spent in sample preparation or with the instrument completely idle). This dynamic allocation of system facilities will allow one to reduce the total size of the required system below the sum total of each experiment's requirements. In addition, under such a system, background data reduction tasks may be carried out by utilizing excess capacity that exists at any moment.

Another advantage of a shared computer is that more sophisticated input/output devices are available to all users. The advantages of having access to large disk files, line printer, card reader/punch, and magnetic tape units are obvious. Only the largest devoted computer/instrument combination could justify the most modest of such devices.

As a result of the above considerations, we have designed and implemented a monitor system. Our goal was to provide a system which could serve a laboratory containing a group of analytical and/or spectrographic instruments operating in a dynamic mode, i.e., a research or development environment. This system, the Palo Alto Laboratory System (PALS), features complete program independence and complete system independence of each instrument from all others. An application program for one instrument, either at the time it is written or executed, need in no way take into consideration other programs running in the system simultaneously. Furthermore, application programs need not be modified as a result of a change in the total instrument configuration attached to the computer system. Each instrument may have its own individual data path to the core of the computer via data channels, so that there need be no sharing of the interfaces between various instruments. Data acquisition associated with the various instruments is completely asynchronuos. Closed-loop control capabilities are provided with a response time of the order of 50 ms. The system dynamically allocates core, disk space, and I/O devices to provide maximum usage of these facilities.

This system uses an IBM 1800 computer. It is possible to operate this system on a computer with 16K words of core; however, it is more useful if the laboratory to be automated is sufficiently large to support a 24K or 32K word machine.

A new device, a digital multiplexer, has been designed and built, specifically to support this laboratory automation system. This digital multiplexer channel provides up to 32 discrete data paths between the laboratory and the core of the computer. Via cycle stealing, it allows data to be acquired from, or presented to, the individual instruments in a demand/response mode with a minimum of computer overhead. The reduction in computer overhead increases the allowable total data acquisition rate from all instruments by more than an order of magnitude. Data acquisition is in a demand/response mode which is of great value in that it allows the instrument to indicate when data is available rather than letting the computer determine when data should be presented. An example of the usefulness of demand/response data acquisition is in acquiring data from an infrared spectrometer which has automatic scan suppression. Since the scan rate is a

function of the first derivative of the absorption, data acquired at equal intervals of time would not be at equal increments in wavelength or wave number. However, with a demand/response interface, the instrument could be run with scan suppression, and demands to take data could be made by the instrument at equal wavelength or wave number increments.

The monitor system

The PALS monitor system is made up of a group of relocatable modules which are initially stored on the disk. Modules serve either input/output devices or are responsible for initiating program loading, linking to the Job Control Language (JCL), controlling multi-task communications, and monitoring time slicing operations.

The only portion of the system which is not relocatable is a 200-word area in low core contains information related to hardware wiring, such as an interrupt transfer vector and the word count and address registers for the various subchannels of the multiplexer.

The various modules making up the system communicate with each other by means of task control blocks and task-complete control blocks. When one module has a task to be performed by a second module, a task control block is generated by the originating module. The address of that block is passed to the second module which performs the task as soon as it is able. When the task has been completed, the second module generates a task-complete control block which is returned to the originating module. The task-complete control block indicates whether the task has been completed successfully or unsuccessfully, and the reason for any unsuccessful completion if appropriate.

Two monitor modules, a real-time module (foreground) and a non-real-time module (middleground) are responsible for the execution of all users' programs and for communication between users' programs and the rest of the system. A second non-real-time module (background) is not part of the system's modules, but is read from disk upon request.

Input/output device modules service disk files, all the subchannels of the digital multiplexer, printer, card reader, the typewriter terminals, etc. There may be several modules stored on the disk servicing the same input/output device at various levels of complexity. That is, one may be a very sophisticated package, having many operations included in it, while another services the device at a minimal level having only one or two very simple operations implemented in it.

The system is configured by reading in a pack of control cards which indicate which modules should be loaded and link-edited together. The reconfiguration of the system, depending on the various users' needs, is very easy and requires only a few seconds. At this time the operator may choose, for example, whether he wishes to have full printer support requiring over a thousand words of core, or minimal printer support requiring only a couple of hundred words of core. It is also possible to eliminate devices which are not to be used. After this deck of cards is read in, the system is cold-started with a one-card cold-start routine which loads and link-edits the appropriate system modules in low core.

Dynamic core allocation

All of core not taken up by the system is divided into pages of 512 words and all of these

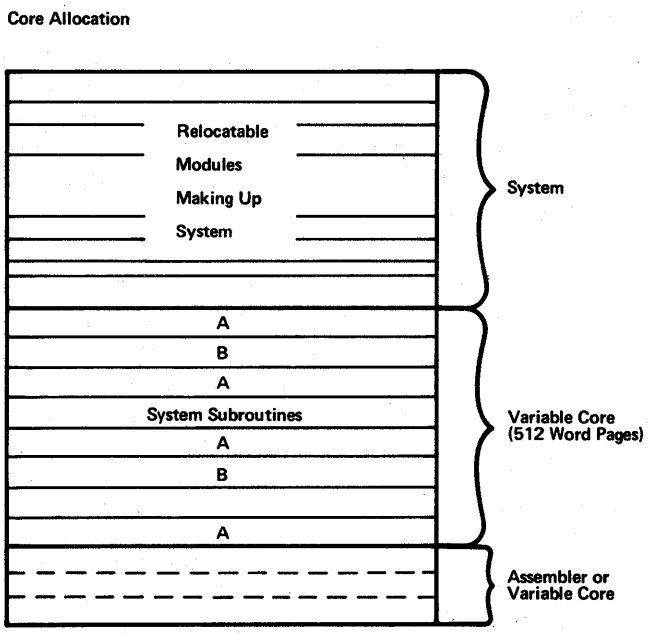


FIGURE 1—Core allocation map

pages are put into a pool of free pages (Figure 1). The loader module is responsible for allocation of the pages of variable core. When a task is given to the loader to load and set a user's program into execution, the loader places that user's program into any pages of core which are presently available. These pages need not be contig-

uous, since the loader takes care of altering the relocatable addresses within the user's program at load time, so that it may execute out of non-contiguous pages of core. In addition, the first and last words on each page of a user's program and all words which are not modified during execution of that program are storage-protected at load time. This provides a high degree of protection of one user's program from another.

Systems subroutines, such as floating addition, multiplication, division, sine, etc., are shared among individual user's programs. These subroutines are stored on the disk, in groups which are assembled into a block that will occupy one page of variable core. At load time, when the loader encounters a call to a systems subroutine from a user's program, the loader checks whether the page on which that system subroutine exists has previously been loaded into core. If it has, the loader links the program being loaded to that systems subroutine. If the systems subroutine has not previously been loaded, the loader loads the page which contains the called systems subroutine from the disk and links it with the program being loaded.

When a program has completed its execution and has called EXIT, a task is given to the loader to return all the pages associated with that user's program to the pool of free variable core. The loader clears all the storage protection bits from these pages and interrogates to see if the terminating program was the only program using any of the systems subroutines. All systems subroutine pages which are not being used by other programs are also returned to variable core. However, those subroutine pages which contain subroutines being used by other application programs still in execution are not affected.

Disk allocation

Many of the problems encountered in a laboratory require random access to individual data or groups of data within large files, hence the file structure on a peripheral disk is of considerable importance. It is usually impossible to define the ultimate size of a data file before it is generated so that a dynamic file allocation is highly desirable. Disk files in the PAL system are organized as logical tapes which are automatically expanded or contracted according to the present length of the data table written thereon. These logical tapes are defined by name and allocated by the system one cylinder of the disk file (2560 words) at a time. Such files may be used in much the same way as a physical magnetic tape, that is, one can read, write, backspace any number of words, rewind, etc. In addition, the inherent advantage of the disk file is preserved, i.e., reading, writing or altering single words anywhere on the logical tape in the direct access mode is still possible. The system allows this by keeping an internal word counter which points to the location where the last access was made. Alteration of this pointer to any value is easily done by giving an instruction to the system. If the data table exceeds the length of the first cylinder, the system automatically adds cylinders to the logical tape, restricted only by the maximum value the word count pointer may attain, 32767 words. The tape may be closed at any value to retain the file for future use. If the length of the tape upon closure is less than a previous length, the excess cylinders are returned to the system, providing automatic contraction of the file. For example, if a previous data file required six cylinders and the present one only requires three and a half cylinders, the fifth and sixth cylinders will be returned to the pool of empty cylinders for reallocation by the system.

Multitasking

Allocation of the CPU is implemented using the multilevel interrupt structure of the 1800. Therefore, if a task is being processed and a task of higher priority is initiated, the low priority task is suspended and the higher priority task is processed immediately.

The entire PALS system is oriented toward performing a variety of tasks, many being associated with input/output. In general, it takes 12 words of a user's program to specify a task for the system. These tasks are accepted by the system and performed as soon as possible. Multiple tasks may be queued for a single input/output device; e.g., the line printer may have a current task in execution while five other tasks are waiting for the line printer to be free. A given application program may have several tasks outstanding simultaneously. For instance, a given application program may instruct the system to (1) acquire a block of data from a given instrument, (2) read a card in the card reader, (3) print a line on the line printer, (4) light a light at the user interface, and (5) write a block of data on a logical tape. These tasks would be given to the system sequentially, but all five tasks would be set into execution before the first one was completed.

CYCLE STEALING

MACHINE CHECK

ANSWER INTERRUPTS

START INPUT/OUTPUT

REAL TIME USERS MONITOR

LOADER

NON-REAL TIME USERS MONITOR

HOUSEKEEPING

JOB CONTROL LANGUAGE

ASSEMBLER

WAIT LOOP

FIGURE 2—Priority list for CPU cycle allocation

If the user's program wishes to initiate an I/O operation, it does so by giving specifications of the task to the user's monitor, which in turn generates a task control block and gives it to the system module responsible for executing that task. As soon as the task has been started or queued, and before the task is completed, control is returned to the user's program. From this point the user's program may send out additional tasks to the system, each time control being returned to the user's program. Two of the arguments in a specified task are entry points to the user's program. One is associated with a normal return address, and one is associated with an abnormal return address. If the task is completed successfully, the user will regain control at his normal return address, whereas if the task is completed unsuccessfully, control will be returned to the abnormal return address.

A priority list for allocation of CPU cycles is given in Figure 2.

. . . The highest priority is cycle stealing for the transfer of data between the various I/O devices, including the instrument interfaces and memory.

. . . The next highest priority is for servicing machine check conditions.

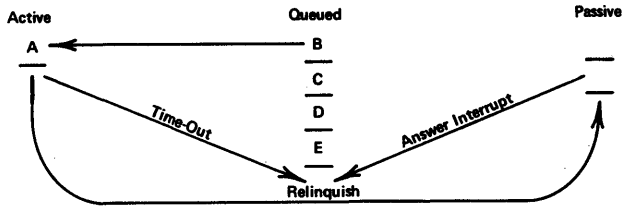


FIGURE 3—Flow diagram of monitor algorithm

. . . Below that, hardware interrupts are answered and I/O devices are started. Each of the system modules which is responsible for an I/O device may queue, to an indefinite length, tasks to be performed in conjunction with its particular I/O device. When a hardware interrupt occurs, indicating the completion of a task or a subtask, the next task or subtask is initiated as a portion of the interrupt handling routine. In addition, when a task has been completed as a portion of the interrupt handling routine, a task-complete control block is returned to the system module which originally issued the task.

. . . Next is the real-time user's monitor under which all real-time programs are executed. A user's program in execution under the auspices of the real-time user's monitor may be in one of three states: passive, queued, or active (Figure 3). When a program originally goes into execution it is placed at the bottom of the queue. As programs are taken from the top of the queue and placed in execution, the program at the bottom of the queue works its way up through the queue. Finally, the program is taken from the top of the queue into active status and the user's monitor transfers control to the user's program. The user's program maintains control for doing processing.

After giving out a number of tasks and having done a certain amount of processing, one of two things can happen which will take a user out of the active status. The first possibility is that he has completed all processing and has given out all the tasks he desires at that time and can do nothing more until one of his I/O tasks has been completed. At this point the user executes a relinquish operation which then causes the user's monitor to remove him from the active status and place him in the passive status. Control then passes to the next

program in the active queue. The user in the passive status does not relinquish core, only his position in the active queue.

The other possibility is that of a "time-out." Each time a user's program is put into execution, the user's monitor sets an interval timer for a nominal five milliseconds, and at the expiration of this time the user's monitor takes control away from the user's program, saves all status and registers, and puts the program from the active status to the bottom of the queue (time slicing). The user then must work his way up through the queue to the top again to resume processing. Using five millisecond time slices, it has been our observation that a real-time user usually relinquishes before he is timed out.

When a user is put into the passive status, the only way he can return to active queue is as a result of a task-complete control block being returned to the user's monitor, indicating that one the user's I/O operations has been completed, either successfully or unsuccessfully. At this point the user's monitor takes the user's program out of the passive status and places it at the bottom of the active queue. A guaranteed response time, i.e., the time from completion of an I/O operation until the time that a user's program may act upon that completion, may be implemented by limiting the number of users allowable in the real-time execution and by implementing five millisecond time slices. If, for instance, the real-time user's monitor limits the number of users to five, a worst case condition would be four users in the queue. When an I/O operation has been completed or a hardware interrupt occurs, the maximum length of time necessary to answer would be five milliseconds for each of the people in the active queue, plus not more than 20 milliseconds overhead associated with the higher priority operations (if overhead were 100 percent). This would mean a total of 40 milliseconds maximum from the time that the interrupt condition occurred until the time that the user's program gained control of the CPU for a five millisecond time slice. Note that interrupt servicing and queue manipulating do not interfere with on-line data acquisition which proceeds via CPU cycle-stealing.

. . . Immediately below the real-time user's monitor in priority is the module controlling loading of programs and dynamic allocation of core. It was given a lower priority than the real-time monitor in order to avoid interference with operational real-time programs. Because the loader module can accept tasks from other modules, operations such as having a real-time program load a non-real-time program and vice versa are possible. A typical example of load time is 1.7 seconds for a seven page program, during which time all of the operations previously discussed under core allocation are performed. This overhead occurs just once at the time each program is initially loaded, and it is negligible compared to the manual set-up times needed to ready an instrument or experiment.

. . . Next is the non-real-time user's monitor. Programs executing in a non-real-time status are executed under the control of this monitor which has the same algorithm for scheduling time to the various user's programs as does the real-time monitor, except the time out period in the non-real-time monitor is a nominal one hundred milliseconds.

. . . Below the non-real-time monitor in priority are various housekeeping modules. These modules are in general responsible for code conversion and spooling operations between various I/O devices. For instance, a user's program may give a task to the system to print an entire logical tape. This task would be executed by a housekeeping module, which in turn would give out subtasks to read sectors of the logical tape and to print the individual lines. This module would also be responsible for the code conversion from EBCDIC to printer code.

. . . Below the housekeeping modules in priority is the module which deals with job control language. This module offers a fairly high level of conversational interaction between the operator of the system and the system. From the console typewriter the operator may load a program or set a program into execution, may cancel a program, may get a dynamic dump of a program while it is in execution, or a dynamic dump of any area core, may get a dump of a logical tape, may define or scratch a logical tape, or may get a status of the entire system.

. . . The lowest priorities are devoted to the language assembler, and a wait loop.

The PALS language

Language requirements for a laboratory automation system include the ability to easily program multiple on-line data acquisition and control tasks as well as off-line data reduction or analysis tasks. Data acquisition and control tasks demand programming of a number of input/output interactions with sensor-based devices. Logical operations are required for the various control functions. Past experience indicates that data reduction and analysis tasks are best served by the FORTRAN or PL/I type of language.

The approach chosen for the PALS system was a macro language with statements natural to the laboratory environment. There are statements for easy handling of sensor-based input/output (e.g., multiplexer channel commands, analog inputs, and a series of special logical statements used to set up bit patterns for control of instrument interfaces, etc.). FORTRAN-like statements are available for data analysis, and they require very little re-learning for users familiar with FORTRAN.

Some examples of the various types of PALS statements may serve to indicate the salient features of the language.

I/O OPERATIONS

SUBCHANNEL OPERATE

SCOP	SC1, 3, DTNAM1, NRET, ARET	Write the contents of table DTNAM1 out over subchannel 1 in demand/response mode (operation code 3). After successful completion of the operation, return control to normal entry point NRET, if not successful to abnormal entry point ARET.

READ LOGICAL TAPE

DRTP	WC, NAME, BUF	Read the number of words contained in location WC from logical tape NAME and place them in core starting at location BUF.

READ CARD INTO CARD BUFFER

CRDR

CONVERSION

CARD BUFFER TO INTEGER VECTOR

CBIV	N, M, VEC, AI	The contents of address N is the number of card columns to be used for each element; location M contains the number of elements per card; VEC is the starting address of the vector; and location AI contains the subscript of the first vector element to be filled by the present card buffer.

INTEGER TO EXTENDED PRECISION FLOATING POINT

FLOT	I, E	Convert the integer at location I to extended precision floating point number at location E.

INTEGER TO EBCDIC

IEBC	I, CH, EBCDIC	Convert the integer at location I to CH number of characters starting at location EBCDIC.

MATHEMATICS

STANDARD FLOATING ADD

FADD	A, B, C, ERR	Add the standard precision floating point numbers located at A and B and put the result in location C. ERR is the entry point of a user error correction routine.

MULTIPLY VECTOR ELEMENTS

IMPX	A, I, B, J, C, K, ERR	Perform a floating point multiply of the Ith element of vector located at A by the Jth element of vector located at B and place result in the location of the Kth element of vector located at C. Branch to ERR if any error.

SQUARE ROOT FUNCTION

FSQT	A, B, ERR	Take the square root in floating point of the number at A and place the result in location B. Branch to ERR if any error.

The macro language permits programmers to mix assembler language with macro statements. It is at the user's discretion to define new statements to meet the needs and level of programming experience of the scientists writing applications.

The PALS macro processor is treated somewhat as if it were an application program, except that it is executed in background mode. It is not paged but is loaded into a partition at the high end of variable core. When the loader is asked to load the assembler, the request is queued until the 12 pages at the high end of core are all free. At that point, the limits of variable core are lowered about 6000 words from the top end of core and the assembler is loaded into this partition. When the assembler has completed its operation, its core block is returned to variable core. This means that several minutes may be required from the time it is requested to load the assembler until the assembler is actually loaded. However, since this system is built on the basis that assembly should be a background operation, this allocation scheme appears to be satisfactory.

Application programs

A joint study was carried out with Varian Associates in order to investigate and demonstrate the usefulness of a time-shared, laboratory automation computer system. A simulated research laboratory, containing an M-66 medium resolution mass spectrometer, an A-60 NMR spectrometer, two Aerograph gas chromatography columns, and a Statos recorder, was linked to an on-line IBM 1800 computer (Figure 4). Each of the instruments was interfaced with the computer via prototype Varian interfaces, an example of which is diagrammed in Figure 5.

Each instrument interface provided facilities which allowed the spectroscopist to have complete control over his use of the computer from his remote spectrometer. These facilities consisted of backlighted pushbuttons, lights, and thumb switches. Prompting lights, operated under program control, served to indicate the present status of the operating program. Functions executable from the remote console included loading and aborting application programs, controlling branch points within the application programs, and entering parameters during the execution of programs. In general, the recorder associated with a particular instrument was used as the graphical output device; the Varian Statos recorder being used for the two chromatographs. Tabular reports were printed on the shared-line printer.

A disk resident master program was associated with each instrument. When an experiment was to be performed, the master program was loaded into the core memory by depressing a

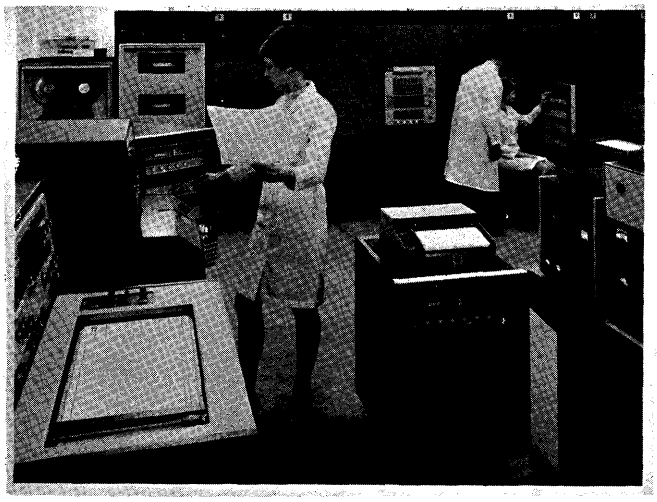


FIGURE 4—An automated analytical laboratory

pushbutton on the interface. The function of the master program was to initialize the spectrometer and to provide a choice of the available application programs to the user. After the master program was loaded, it armed appropriate pushbuttons for program selection. This was indicated by lights behind the several pushbuttons. When

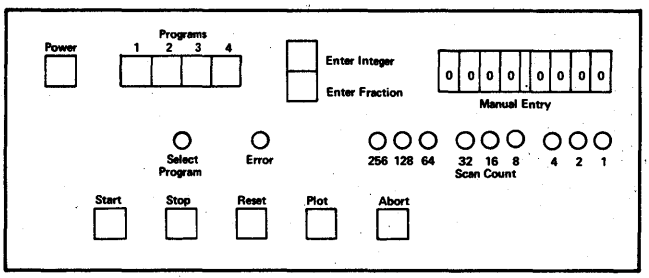


FIGURE 5—Typical control panel of instrument interface

DATA MANIPULATION

TRANSFER VECTOR ELEMENT
TVEI — A, J, B, K
Transfer integer element A_j into integer element B_k.

INCREMENT VARIABLE
INC — I, J
Increment integer at I by the integer J, where $-128 \leq J \leq 127$.

FIND MAXIMUM ELEMENT OF INTEGER VECTOR
IMAX N, YDATA, I, YMAX, IMAX
Search N elements of the vector starting with the Ith element of vector YDATA and place the maximum values of YDATA and its index I into locations YMAX and IMAX respectively.

PROGRAM SWITCHING

IF
IF — I, J, A, B, C
Branch to A if integer quantity (I–J) < 0, to B if $= 0$, and to C if > 0.

COMPUTED GO TO
GOTO — A, B, C, D, POINT
Branch to A, B, C, D if location POINT contains 0, 1, 2, 3 respectively.

REPEAT LOOP
REPT — LOC, N, I, K
Execute all statements starting at location LOC through the REPT statement the number of times contained in N. Each time through the loop, increment integer at I by the number K.

one of these programs was selected, it was loaded and the master program would exit (release its core). After the subprogram completed its function, and before it exited, it requested that the master program be reloaded. This mode of operation minimized the core requirements of a single user.

Several application subprograms, which performed some of the more elementary instrument functions, were jointly specified and written by Varian and IBM. With the A-60 NMR it was possible to: acquire data in a demand/response mode by sweeping the magnetic field; time average the data by repeating the scans any desired number of times; digitally smooth the acquired data; and control the magnetic field homogeneity. The mass spectrometer programs acquired data in a demand/response mode; replotted any desired portion of the data on the instrument recorder; found all peaks in the spectrum and normalized their intensities; and found the five highest peaks and identified the compound by comparison with a table of spectra of known compounds on the disk file. The gas chromatography programs acquired data, detected and resolved peaks, calculated their areas and wrote a report giving retention times with peak areas.

Actual experience with the above system was quite good. Programs were easily and quickly written using the macro language. No noticeable interference between users occurred, even when all instruments and data processing I/O devices were running simultaneously.

In conclusion, we believe that we have been able to produce a system for use in laboratory automation which provides each individual user the isolation, the availability, the real-time control responsiveness, and the price per instrument associated with multiple computers, one per instrument. In addition, this system provides powerful input/output devices, disk files, and a large amount of support for the I/O devices and the files, which is not normally found on a small dedicated computer. Each user, then, has the impression that he has a large computer attached to his instrument and completely at his disposal. The 1800 PALS program, excluding the instrument application programs, is available from the IBM Type III library (PID #5778).

INSTRUMENTATION CONTROL

READ DATA AND WRITE LOGICAL TAPE

RDLT

SCPNT, WCPNT, LTPNT, NPRA, APRA

From subchannel number contained in SCPNT, read the number of words contained in WCPNT ($1 \leq WC \leq 32767$), and store on a logical tape whose name resides in LTPNT. Return control to normal entry point NPRA for successful completion of the read, and to APRA for abnormal conditions.

ALTER SUBCHANNEL BIT

ASCB SC, BIT, N

Alter one bit (number of bit contained in location BIT) of a subchannel register (number of subchannel contained in location SC) to a new value $N = 0$ or $N = 1$.

CONVERT THUMB SWITCH TO FLOATING POINT NUMBER

CTTF F

Read a set of manual switches on instrument interface and convert the setting to a floating point number at location F.

ASSIGN PROCESS INTERRUPTS

ASPI PINO, ENT

Assign a program entry point ENT to the process interrupt whose number is contained in location PINO.

Real time time sharing, the desirability and economics

by B. E. F. MACEFIELD

University of Oxford
Oxford, England

INTRODUCTION

The thoughts expressed in the following contribution have arisen largely as a result of work at the University of Oxford, Nuclear Physics Laboratory. In Oxford, we have had a demand for two real time users and one background user. The achievement of this goal is described in the first reference.[1] It has been clear, however, that many institutions have a similar problem and it was the fact that the solution was not impossibly difficult that prompted this plea for more efficient use of large capital investments represented by on-line computers.

I think most of us would agree that the battle of the stored programme computer versus the fixed wired kicksorter is over in the field of nuclear data collection. The victory has certainly gone to the stored programme device. But once the euphoria had died down many people, the author included, have found that a certain amount of fast front end hardware has been necessary to supplement the activities of the central processor.[2] This requirement will probably be less prominent in the future with the current increase in number of sub-one micro second cycle time computers. I might point out that front end hardware is usually used to make decisions on the admissibility of the incoming data faster than the computer.

The proposition

It is the utilization of the computer used in such a data collection system that I wish to examine. The increasing speed of even the smallest computers has given rise to some interesting consequences. In very general terms the speed of processing the incoming data in a real time environment is inversely proportional to the memory cycle time. The add one to memory or for very sophisticated data the programme manipulation time are both very obviously dependent on cycle time. As a result we can get to a stage where the C.P.U. is infrequently used because the data is serviced in a time considerably shorter than the time between events. Let us examine this further. Assume we have a 1 μsec cycle time machine.

The add one facility from an external source should not take longer than 2 μsec giving a mean rate of 500,000 events/sec. Now there are a few detectors and no analogue to digital converters that will perform at this rate. The best the latter can do is probably 50,000 events/sec at the present time. Thus for a physically realisable configuration we can see 10% utilization in the simple case. For the complex data system we may require programmes which take say a millisecond to sort an event. Now complicated events usually occur at very slow rates, typically 100-200/sec, so our multiparameter data analysis takes at most 20% of the time. Even allowing for a factor of 2 upwards in these estimates we see a sizeable fraction of the total processor time is unused.

But, you will be asking, what about the display and other user requirements. These user requests, often for data manipulation, are very infrequent on a computer time scale and in consequence need not affect the overall argument even though they may take a substantial time to complete. The display, however, is another matter.

Most people working in real time with computers require some form of immediate access to the data. The most usual form for this to take is a visual display. I would distinguish, very obviously, two types. The C.P.U. controlled variety and the direct access to memory type. The former obviously ties up the C.P.U. continuously for display and in my opinion is so bad an investment (regardless of its price) as to be completely disregarded from the current discussion. The latter comes in forms ranging from simple analogue scopes to sophisticated vector and character generators. The major problem about C.P.U. controlled scopes is that once installed, economic arguments will be advanced

forcing them to be used and in consequence be a large inhibiting factor on the application of the computer. (This comment may be toned down somewhat by the advent of large diameter storage tubes and the fact that it is a comparatively trivial modification to convert such a device to a direct access system). But, given we have a scope that will display a frame without C.P.U. intervention, save for start up, then it is unlikely that the total increase in used time would be another 5% giving at most 40-45% total time used.

Having dealt in general terms let me give an example from our work in Oxford. We have several A.D.C.'s in use, each connected to front end hardware.[2] Figure 1 shows the time left over when they are in operation at the event rates indicated. Two sets of curves are shown, a software (API) storage system and hardware storage (DES). We see for the software case less than 10% time left over if 6 A.D.C.'s are working at rates of 2000/sec each. This, however, is not a physically sensible rate for such a system since Figure 2 shows that at this rate A.D.C.'s 1 and 6 have vastly different storage efficiencies as seen by the outside world, making usage very difficult. The solution is thus either to reduce the input rate to get the efficiences more nearly equal, say to 1000/sec, when the time left free is a usuable 30% or build front end hardware and get similar efficiences and 30% free time at 20,000/sec. The derivation of these curves and explanation of the different shapes of the DES and API efficiency curves is set out in reference.[3] If either of the above approaches were adopted we obtain some free C.P.U. time and we must therefore ask why is this time not used?

Nevertheless the question needs some justification even though once put it seems obvious enough.

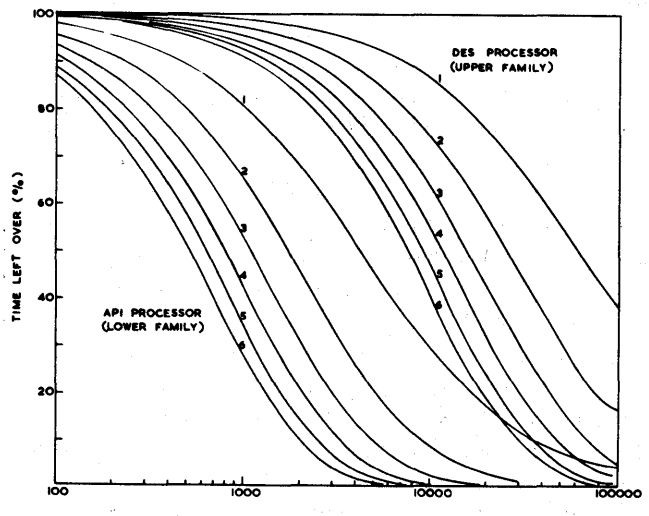


Figure 1—Free C.P.U. time for various input rates on each A.D.C

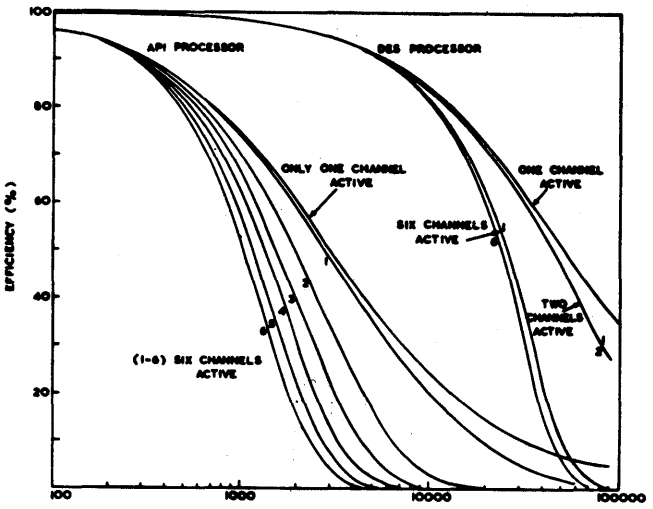


Figure 2—Efficiency of each A.D.C. for various input rates

Modus operandi

When buying a computer the expenditure must be justified to some funding body, as in the purchase of an accelerator facility. In the latter case, however, one of the earliest questions is what is the percentage utilization? If we asked for an accelerator to be used only 50% of the time, it would be completely unacceptable to the financial experts and physicists alike. So why do we not get such questions from the computer users who have most to gain from efficient machine use?

Another point that arises is that a given institution be it a small laboratory or a government enterprise, buys the computer it can afford and this, I propose, results in the computing expenditure being a constant fraction of the total expenditure on the project. In other words all users have the same percentage to gain be they large or small. So on this premise we should be hearing a clamour for time sharing even the smallest machines, but we do not. Why not? Because we have to some extent been brain washed into thinking that only with the largest installations is this at all possible.

Where do we start in attempting to achieve this utopia of 100% computer utilization? We start by delimiting exactly our goals. We are not aksing for 10-20 users but in terms of 1–3 or maybe 4. In Oxford, we have already 3 simultaneous users on a 24k PDP-7; why would others not do the same? The one single problem with most on-line nuclear structure computer systems is the inability to get at the computer after taking the experimental data. Obviously, therefore, the extension of the users from 1 to 2 produces an infinite improvement.

To attain this end we have to split the machine in some way, either by swapping programmes in core or memory allocation. In Oxford we have a mixture of these approaches;[1] memory allocation to separate back-

ground from foreground and swapping for the two on-line users. At this stage we ran into the first problem. The system software as then supplied for our machine was not designed for multiple users and moreover was not organized on a modular concept using bulk storage peripherals. (I would point out that in the latter case this has since been changed with the PDP-9).

In a small system it is not essential to get two background users but the background system must function in the presence of unusual device flag combinations. This implies a flag mask to allow only specified flags to cause interrupts and an efficient automatic priority interrupt system (API). This should be an interrupt chain with successive channels having non-consecutive memory locations. In this way the low priority channels can be assigned to the background system and the higher channels to the foreground system in a completely different and possible protected memory area. (To my knowledge no company has such an API system.)

The concept of memory protection necessarily entails some mode of trapping the incorrect memory references; obviously to be a location different from the API channels and programme interrupt. Since we are in a small system we will need to share peripherals, on the grounds of expense, and thus IO instructions need to be trapped also. Along with IO trapping we will need a supervisor programme to oversee this concurrent peripheral use. This programme need not be extensive, we have an 800_{10} word programme to share Dectape, punch and plotter.

We have now got round to the question of IO. In a real time environment it is evident that all synchronous devices must have a higher priority than random data from a nuclear experiment. By synchronous I include magnetic tapes of all types together with discs but not most paper tape readers which stop between characters. To guarantee this priority sequence we must have all such synchronous IO by data channel.

Necessity hardware configuration

Perhaps it might now be possible to see what we are asking of the computer manufacturers:

(i) A versatile API system
(ii) An effective IO trap system
(iii) Synchronous IO by data channel
(iv) Flag mask.

Hopefully some may conclude that many machines seem to have most of these facilities already. I think this is true, so why are there not more foreground-background systems in existence, for given the above four requirements, time sharing in that context is straightforward.

There seem to be three possible reasons for the present state of the art:

(i) the expense of memory for a memory allocation system
(ii) the cost of fast swapping discs
and (iii) we are persuaded that we need to buy a bigger machine to achieve these results.

The costs of the first two items are continuously falling and from this point of view we may expect an increasing number of multi-user applications. In the third, we, as computer users, must be more critical of what is provided by the manufacturers and not succumb to their advertising.

Economics

Of the three major hardware requirements the ones most likely to be missing are the slit API system, IO Trap and flag mask. Given that the machine runs asynchronously the cost of putting these into an existing machine is unlikely to exceed \$2400. The duplication of the smallest system is unlikely to be less than \$15,000. The latter cost obviously depends on the system while the former is almost independent of the system context.

CONCLUSION

In conclusion then, it is evident that to achieve more than 50% real time utilization in nuclear structure work we require more than one real time user. It is also evident that this can be done with a minimum of effort given the right hardware configuration, which is not so very different from that existing on most small to medium processors. It is thus much more economic to satisfy a demand by a small extension to an existing machine than to duplicate the system.

REFERENCES

1 G L MURRAY B E F MACEFIELD
Nucl Inst & Meth 51 1967 229
2 G L MURRAY B E F MACEFIELD
Nucl Inst & Meth 57 1967 37
3 G L MURRAY B E F MACEFIELD
Nucl Inst & Meth 62 1968 122

A modular analog digital input output system (ADIOS) for on-line computers

by R. W. KERR, H. P. LIE, G. L. MILLER, and D. A. H. ROBINSON

Bell Telephone Laboratories, Incorporated
Murray Hill, New Jersey

INTRODUCTION

The most important single feature that allows a computer to be employed in a broad range of calculations is the fact that one can, by programming, in effect restructure the machine to perform the desired computational task.

It is not possible to retain the same degree of flexibility in on-line systems because of their need to be connected to specialized external hardware. Primarily for this reason the majority of on-line computer systems that have been constructed in the past have been designed to perform a pre-defined class of specialized operations. This situation is analogous to that which existed before the invention of the stored program machine when a computing device would be constructed to perform each new special task.

The system described here is the result of an effort to obtain a reasonable degree of flexibility in on-line computer controlled environments and is based on careful considerations of the factors that tend to limit such flexibility. The consequences of such problems in previous systems has been evidenced by difficulties of adding or reconfiguring hardware and interface equipment. Such difficulties have reduced the potential versatility of many existing systems in which the effort required to implement useful changes is uneconomic and such changes are therefore only rarely made.

It is possible, however, by the use of appropriately designed modular units interconnected by a common two-way analog and digital data-bus to obtain the desired degree of flexibility and power. The next section of this paper outlines the general consideration involved in the design of such data-bus systems, while the remainder of the paper describes the implementation of these ideas for a specific small computer, namely a Digital Equipment Corporation PDP–8.

Design considerations

Of the many schemes whereby equipment may be connected to a computer perhaps the simplest division is between "radial" and "bus" systems. In the former, the interconnecting cables can be thought of as radiating like the spokes of a wheel to connect the computer to each external unit, while in the latter each external unit is connected to a common "highway," "party line," or "bus" cable system. In a certain sense this distinction is artificial since in the last resort even a radial connection is handled on a bus basis once the signals enter the computer hardware proper. However, the distinction is a valid one for the domain of equipment external to the computer itself and can serve as a starting point for comparing different systems.

The greatest single advantage in a radial system, from the user's point of view, is the fact that external equipment need only be plugged into a suitable connector to be on-line with the computer. The outstanding disadvantage, however, is that such systems are relatively inflexible and can become expensive if large numbers of external units are required. Furthermore, the user may become overly dependent on the computer vendor and his instrument division since only their equipment is automatically interfaced with the machine. In bus systems, on the other hand, the organization is different in that each external unit is connected to a common set of cables which

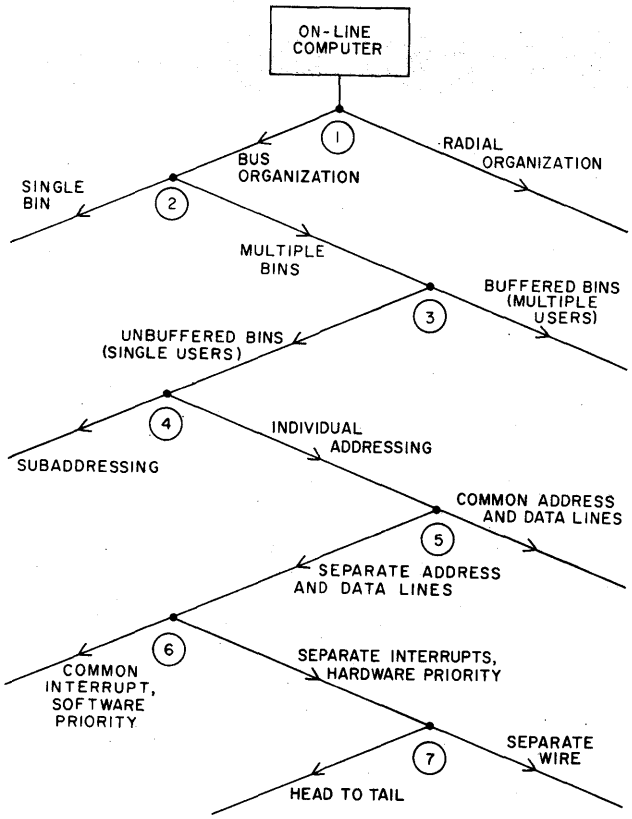


FIGURE 1—Logic tree indicating the major decisions involved in the design of a data bus system

carry address, data, and status information to and from the computer.

A feature of such bus systems that is worth noting is that the interfacing operation between the computer and the external world occurs only once, namely between the computer and the data bus. It is therefore possible to design such systems so that the same, or different, collections of on-line equipment can be connected to many different computers by changing only the bus-to-computer interface.

The differentiation into radial and bus systems is indicated by the node labeled 1 in Figure 1. Other important decisions follow at other nodes in this diagram and it is the purpose of this section of the paper briefly to indicate the major considerations involved at each branch. In order to forestall any misunderstanding it may be well to point out that though a particular design path was followed in the PDP–8 system described in the remainder of this paper, it is by no means claimed that the resulting system is ideal for every application. As will become clear the design of any system is dependent on numbers of factors, major ones being, for example, the total size of the system envisaged (i.e., number of input and output units together with information on their spatial separation), and whether the system is to be employed by a single user or time shared by several non-interacting users simultaneously. Another important consideration is of course input/output speed and data-rate. Interestingly enough, however, in a number of actual on-line experimental environments that we have considered, it turns out that the bus approach imposes only a small time burden on the system. In the last resort this arises from two causes, first because operations proceed sequentially inside the computer, requiring a certain time to service each external unit, and second because external units are themselves often quite slow (e.g., $\sim$50 μs conversion time for a typical 12 bit nuclear physics ADC). The result of this is that if reasonable care is exercised in the design of the bus system the access and I/O time for external units can become quite small compared with the sum of the device operation and computer servicing time. Obviously it is always possible to envisage situations where this is not the case, but we believe them to be only a small subset of most on-line situations. In those cases where I/O speed becomes an unavoidable limitation, e.g., CRT displays, it is usually worthwhile to consider the use of a separate dedicated piece of equipment (such as a disc with suitable DAC's, etc., for display) to perform the critical function at all times.

Returning to the general considerations indicated in Figure 1, the next question is one of system size. It is taken for granted that the user's hardware will consist of some form of modules which plug into bins (see for instance the European standard IANUS system or the ADIOS system described here), and the question is whether there will be one bin or several. This decision involves questions of how multiple bins are to be addressed, and what will be their physical separation and distance from the computer. This latter point is highly important though easily overlooked. Its importance can be seen in the following way. If the cable length is long then the cables must be terminated to avoid reflection. The logic levels employed for modern microcircuits are typically 0 and +4 volts. For 50 ohm terminated cable this means 80 ma/bit. Since on-line systems usually employ common grounds between analog and digital hardware it follows that the transfer of a 16 bit word can involve a current pulse at over 1.2

amperes in the ground return. The noise and crosstalk implications of this are obvious. (A new data bus system presently being designed at Bell Telephone Laboratories for a multi-user environment using SDS Sigma computers circumvents this problem by using balanced current-driven twisted pair. No such extreme steps were necessary for the relatively small PDP–8 system described in this paper.)

If multiple simultaneous users are envisaged it is advantageous to employ a buffer unit between each bin and the data bus. This has a number of advantages for large systems, not least being the ability to provide logical buffering between bins to prevent one user from wiping out another by, for instance, unplugging a module. This decision is shown at node 3 in Figure 1.

Again in large multi-bin environments it can be advantageous to provide a bin address with unit sub-addresses within each bin, (this is the route followed in both the European IANUS and BTL Sigma system designs).

At node 5 a difficult choice must be made regarding the extent to which the bus cables are shared by time, or other multiplexing arrangements. The advantage of multiplexing lies in its ability to reduce the number of cables in the system. The disadvantages are reduced I/O speed and added complications to unit hardware and system programming.

The way in which external units signal the computer via the interrupt system is also one of central importance. In this connection the major choices lie, as indicated at node 6, between using a single common interrupt line, or of employing a hierarchical or priority system. The latter can be organized two ways, either by using a separate physical interrupt wire from each external unit to the computer, or by connecting the external unit interrupts in head-to-tail fashion whereby priority is defined by position in the chain. Neither of the latter system is well suited to a flexible data bus system designed to accommodate a wide variety and number of external units since each time a change is made in the configuration of modules, numbers of separate physical wires must also be re-routed.

It will be appreciated that the foregoing discussion of general questions is of necessity superficial, though we believe it to indicate most of the major hardware considerations involved. Without going too deeply into details of software and logical design one other question regarding module addressing must be raised. This is the issue of what we term "generic" addressing and its importance can be seen with a simple example. Suppose the on-line system involves a number of external devices which must be turned on and off in exact time synchronism. Since any bus system is by definition sequential, in that only one set of address lines is used, it is not at first clear how this can be achieved. A solution to the problem can be provided by allowing units to recognize more than one address. Each unit or module recognizes its own unique address and having been so addressed one of the commands to which it can then respond is to *enable recognition of another address.* Such other addresses are termed generic addresses and they can be common to many different units. In this way the computer can issue generic commands which apply simultaneously to any subset of external modules, allowing them to operate in exact time synchronism.

By way of concluding this section on general design considerations it may be illuminating to consider a number of questions that can be asked regarding the logical organization of any on-line computer system. *Does it, for example, require special timing, logic and drive circuits to be added to an external unit before it can talk to the computer?* If the answer is yes then the chances are that considerably less experimental innovation will be carried out with the on-line hardware than would otherwise be the case.

Another important point to bear in mind is the ability of the system to check itself. *Can the computer tell what units are connected and whether they are operating?* This feature can be very important in systems employing many modules.

An area that is outside the scope of this account is that of programming, but it is obvious that the hardware and software of any on-line system must be harmoniously designed. Less frequently considered from the outset, however, is the question of the ease of debugging the operation of the entire on-line system. Our experience with the present system has shown the extreme desirability of being able to "force" external equipment to well defined conditions, by hand, as a check in debugging programs and hardware. This also is, therefore, a point to consider in comparing system configurations, *how difficult is it to debug the hardware-software interaction in preparing programs for the on-line system?*

A corollary to this point is the related one of investigating the degree to which the computer

is able to exercise external equipment. In this connection it has been found extremely useful to prepare programs which operate all the computer-accessible features of a module sequentially. This approach allows convenient debugging of modules as they are produced since an operator can examine repetitive waveforms at his leisure, proceeding sequentially through a series of test conditions under programmatic control.

A final point involves the provision of an analog measurement capability within the data bus system. This has been found to be most useful in the PDP–8 system described here, and comprises a shielded twisted pair in the data bus cable connected to a central 12 bit ADC at the computer. In conjunction with suitable external modules this furnishes the ability to both measure and provide analog levels. Together with the digital capabilities of the system this provides a combined digital and analog capability which encompasses a broad range of applications.

The data bus

A diagram of the data bus system chosen for a PDP–8 and a single-user environment is shown in Figure 2. The logic of the operation of the data bus closely parallels that of the PDP–8 computer. Commands are sent to a particular module by placing the module address on the address lines and activating the instruction lines. The three instruction pulses in this system occur at 1 μs intervals and are .4 μs in width. Because one of the system rules is that the modules operate with instruction pulses of any length greater than ~.2 μs, there is no restriction on the maximum length of the cycle. (Thus the system may also be used with other computers of lower speed than the PDP–8 provided a suitable computer to data-bus interface is constructed.)

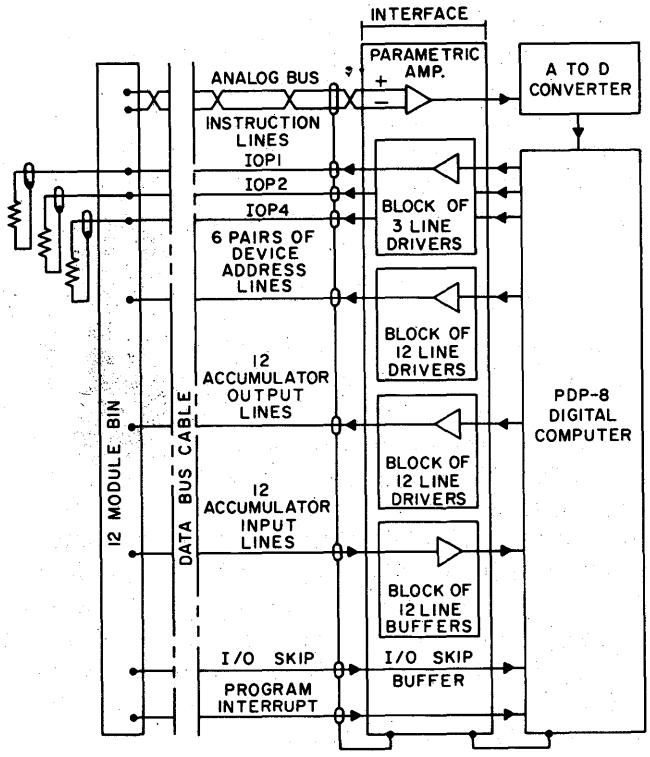


FIGURE 2—Simplified block diagram showing the computer interface and data bus cable

The type of operation performed with each instruction pulse has been standardized as follows.

Instruction Pulse	*Operation*
IOP1	Augmented Instructions
IOP2	and I/O Skip
IOP4	Input to computer
	Output to modules

Since it was useful to have many more than three instructions, a system for deriving a set of augmented instructions during IOP1 is used wherein the six low order bits of the computer *output* lines are each interpreted as a separate instruction. The six high order bits have been reserved to be used in coincidence to obtain 64 additional augmented instructions, should such a need arise in the future. IOP2 is used to generate all inputs so as to relax the requirement on the fall time of the input pulses which must be clear before the next instruction is executed.

A module requests attention from the computer by energizing a common interrupt line until it is serviced by the computer. The computer then interrogates the modules to determine which one is requesting service. (Since a priority interrupt system might be desirable when the system is interfaced to a different computer, four additional lines in the data bus have been provided, which may be used in this manner.)

The I/O skip line provides a means for a module to respond to interrogation by the computer. An affirmative response is signalled by energizing this line, which causes the PDP–8 to skip an instruction. If the system were connected to a different computer, the I/O skip line could set a status bit in the machine.

The distribution of the analog input lines to the module bins is performed with shielded twisted pair. The interface contains a parametric amplifier in a configuration which converts the

single ended 0 to —10V range of the analog to digital converter in the PDP–8 to a +10V to —10V differential system with good dynamic common mode rejection.

Since the logic of the bus system is compatible with the PDP–8, the interface is used simply to provide the level shifting and buffering that is necessary to communicate directly with the integrated circuits in the modules.

The interface unit converts the negative logic levels of the PDP–8 to standard microcircuit levels as used in the data bus system. In addition the interface input buffers provide noise filtering and an input threshold which can be varied in order to investigate noise margins. Tests have shown the data bus system capable of operating with cable lengths of more than 100 feet.

The data bus consists physically of 48 miniature coaxial cables, together with one shielded twisted pair, which interconnect the required number of module bins. Within the bins, the data bus loops through twelve 50 pin connectors into which the modules connect upon insertion into the bins.

The plug-in modules

Four general purpose modules were designed to operate in conjunction with the computer to assist in the operation and control of experiments and in the acquisition of the resulting data. Figure 3 shows one of each type of module installed in a module bin. A modified NIM power supply located at the rear of the bin provides local power for the modules.

The construction of all the modules is similar to

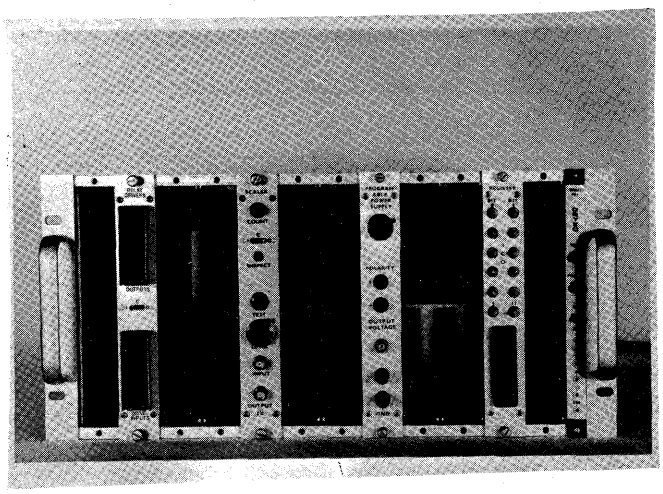


FIGURE 3—Photograph of one modified NIM bin containing one of each of the four modules

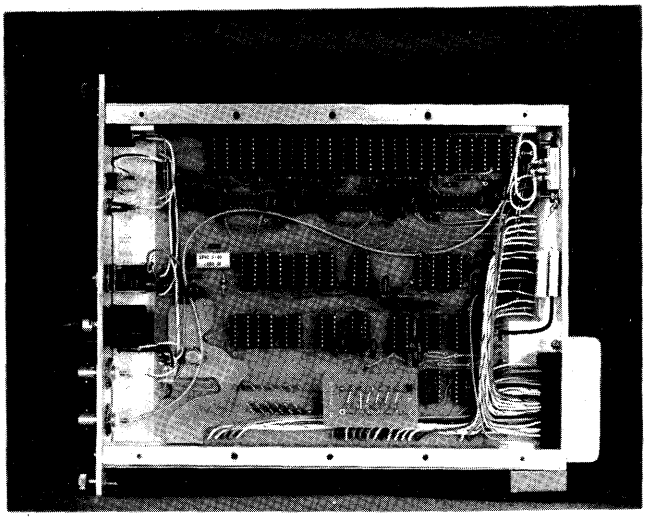


FIGURE 4—Photograph of a scaler module, showing the location of the plug-in address cards at the lower center of the printed circuit

that of the scaler, which is shown in Figure 4. The use of integrated circuits on a single special purpose board results in considerable reduction of cost and size over the more usual technique of using general purpose logic boards. The cost of the more complex modules is approximately $300 each.

In order to simplify programming and debugging, the module addresses are defined by small plug-in cards, visible in Figure 4, which may be changed at will. Removal of a unit for repair may thus be performed by switching its address card to a new module. A brief discussion of the structure and operation of each of the modules is given in the following sections.

Register

The register provides a general purpose interface for digital devices. It is capable of accepting a 12 bit word from an external device and inputting the word to the computer. It can also accept a word from the computer and present it, with buffering, to the outside world. The block diagram of this unit is shown in Figure 5. Table I lists the commands for the unit, most of which require no explanation.

The voltage levels for input and output are standard NIM and integrated circuit levels. The use of jam transfer makes resetting unnecessary, and allows alteration of selected bits of the register without even a momentary change in the other bits.

The use of master-slave flip-flops as buffers

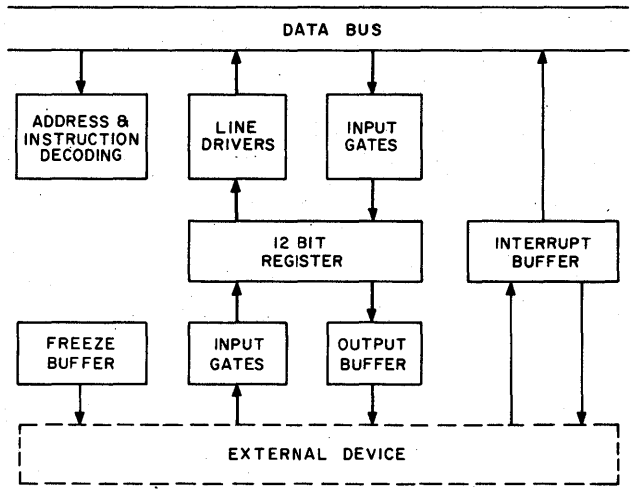


FIGURE 5—Simplified block diagram of register module

permits the register to be used as a hardware bit-reformatting device by connecting the outputs of the register to the inputs in the desired sequence. The word to be reformatted is sent out to the register and then read in as external data. This feature also permits use of some bits of the register as input and some as output, since by connecting a bit output line to the corresponding bit input line one makes the value of the bit independent of the Load Register from External Unit command.

The most serious stumbling block in interfacing an external device to a computer is not the compatibility of the input/output levels, but the necessity of establishing logical communication between the devices and the computer in a simple way. In the register the "freeze" circuitry allows the computer to command a device to remain stable while being read. The interrupt circuitry allows the device to request service from the computer. A busy line informs the unit of the status of its request.

Relay module

The logic of this module is shown in block diagram in Figure 6. It contains twelve single pole double throw high speed relays each capable of switching 3 amps. It is used in conjunction with a register module to handle signal and power levels which are inconvenient to handle electronically.

The relay driver commands are shown in Table II. The Test Unit Ready command allows the computer to test for the presence of the module.

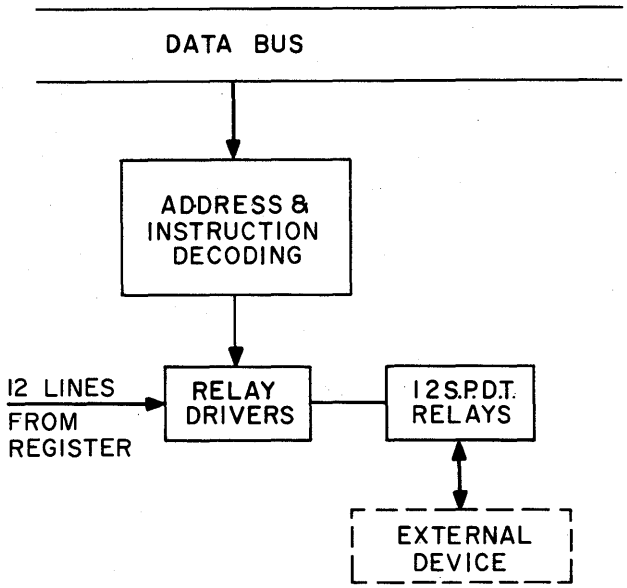


FIGURE 6—Simplified block diagram of relay driver module

Scaler

This module comprises a 12 bit binary ripple counter and the necessary logic to permit the module to function on the PDP–8 data bus system. In operation the module sends an interrupt to the computer for every 4096 input pulses. The system records the number of these interrupts and therefore functions as a scaler modulo 4096. At the end of the counting period the fractional count remaining in the scaler is added to the previously recorded total. Figure 7 is a logic block diagram of this unit. A unique feature in this design is the use of a two address command structure. The first of these is the generic address and the second the unit address. By use of

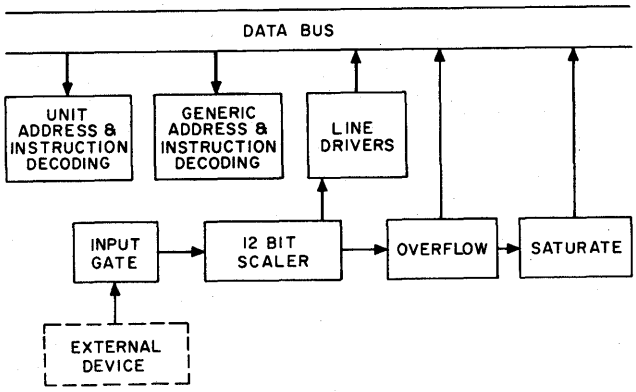


FIGURE 7—Simplified block diagram of scaler module

the generic address the computer can execute any one of three commands and have all scalers sharing that address respond simultaneously. The command structure for the unit is shown in Table III. Most of the commands are self-explanatory. While Increment operates in front of the input gate, Preset, which also increments, operates at all times. Enable Generic and Disable Generic permit the generic commands to be obeyed or ignored.

The overflow and saturate logic allow the computer to serve as the high order portion of the scaler in the following manner. When the high order bit of the scaler overflows, its overflow flip-flop is set and a program interrupt is sent to the computer. The computer then initiates a search using the Test Overflow instruction and thereby ascertains which module interrupted. Should a second overflow occur in a scaler before the previous one has been recorded by the computer then the saturate flip-flop is set. The computer can test this flip-flop, and thus either ascertain that the scaling has been performed without error or take appropriate action to insure correct scaling.

A rear panel switch connects the output of the most significant bit to either the overflow detecting circuitry or to a front panel connector, thus allowing the use of a second scaler module to form a 24 bit scaler if desired.

A discriminator is located at the input to the module and its level is adjustable from —5 volts to +5 volts. A front panel lamp indicates the status of the input gate.

A three position switch allows manual setting of the unit in either the start or stop condition, or returns this control to the computer.

Programmable power supply

The primary purpose of this module is to allow the computer to supply adjustable voltages to external devices. As shown in Figure 8, the computer controls the power supply voltage by causing rotation of a motor driven ten turn potentiometer which serves as an inexpensive analog memory. The series regulators, which operate by re-regulating the bin power, are built either as positive or negative supplies, and furnish 0 to 10 volts with overcurrent protection from 10 to 250 ma.

The control commands for this unit are shown in Table IV. The computer controls the unit by operating the potentiometer while simultaneously monitoring its output voltage, thus becoming part of a servo loop.

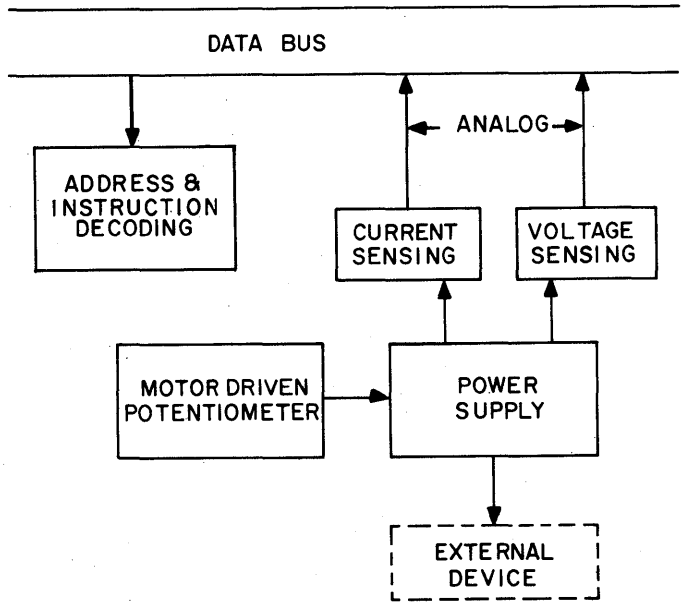


FIGURE 8—Simplified block diagram of power supply module

The front panel dial attached to the potentiometer indicates the supply voltage directly while also permitting manual setting of the voltage. In addition, the module may be used as an analog input from the operator, since the computer can, in effect, read the dial setting. Connection of the potentiometer shaft to other rotating equipment could also permit the computer to cause controlled motion in the external equipment should this be desired.

Sample applications

The system has been used to control, and process data from space experiments; to run nuclear analysis displays using DAC's; and as the data acquisition and control center for an automatic Hall-effect measuring system.

Figure 9 is a block diagram showing how experiments are connected into the system. Note that the computer output can control the experiment via the data bus. Computer input can store digital outputs from the experiment via the data bus, and make analog measurements via the analog bus and computer ADC.

Testing of a satellite charged particle spectrometer

A simplified block diagram of a satellite par-

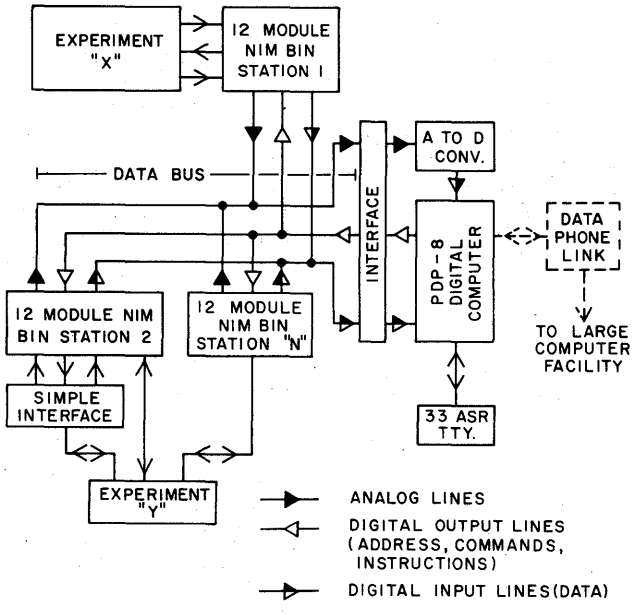


FIGURE 9—Simplified block diagram showing three bins connected to the data bus

ticle detector experiment is shown in Figure 10. Particles incident on the semiconductor detector assembly deposit their charge in one or more detectors. Coincidence logic applied to detector outputs determines the particle type, while the linear system sorts the energy of each particle type into one of five consecutive energy ranges.

Sixteen different particle identifying modes and the five channel energy ranges are controlled by the digital outputs from the spacecraft sequence clock in such a way that each mode lasts for approximately 10 seconds. For in-flight calibration the experiment contains a test pulse generator and two internal sources, each activated by certain states of the sequence clock once every six hours.

When tested in thermal vacuum in the laboratory by the computer system, outputs from a register unit simulated the sequence clock and thereby controlled the experiment modes. Calibration modes were arranged to alternate between the test pulser and internal source modes for every complete sequence of experiment modes, interspersed with complete sequences of no excitation. In this way a calibration cycle was repeated about once every 25 minutes, so large amounts of calibration data were processed in a relatively short time. The sequences of no excitation were useful for the observation of noise counts.

Scalers were used to accumulate the five channel outputs and to transfer the counts into the computer for printout.

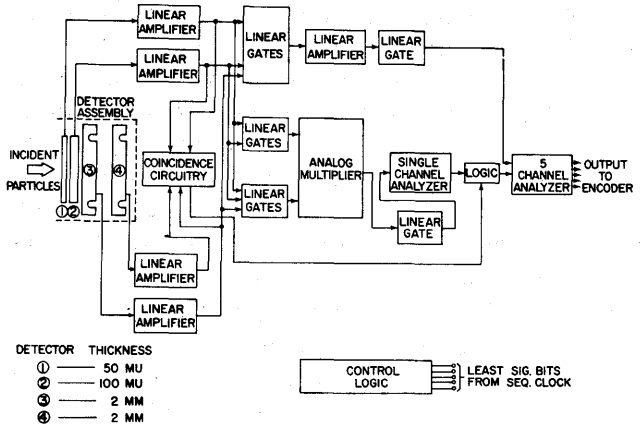


FIGURE 10—Simplified block diagram of a satellite experiment

Temperatures were also recorded. The outputs of temperature sensors were switched onto the analog bus using a relay driver and register combination, and were measured by the computer ADC.

Although not used in this particular test, it would be appropriate to use programmable power supply units in a test of this kind to investigate the effect of varying power supply voltages on circuit performance.

Automated hall-effect measurements

An ion implantation laboratory is in operation and many implanted diode samples will require extensive electrical testing. Each sample is expected to go through several stages of annealing, and following each stage measurements will be made to evaluate Hall coefficients, specific conductivity, carrier concentration and carrier mobility, over the temperature range 2°K to 300°K. Figure 11 shows a simplified block diagram of the electrical system.

A large number of voltage measurements from contact to contact are required at each temperature of interest, and the temperature stability at each measurement point must be carefully controlled.

Figure 12 is a flow chart showing the main steps in the computer controlled system. After the sample is mounted and ready to be lowered into the cryostat, the program starts with a comprehensive check of the hardware and a "FAULT" printout is generated indicating the nature of any malfunction. An "OK" printout in-

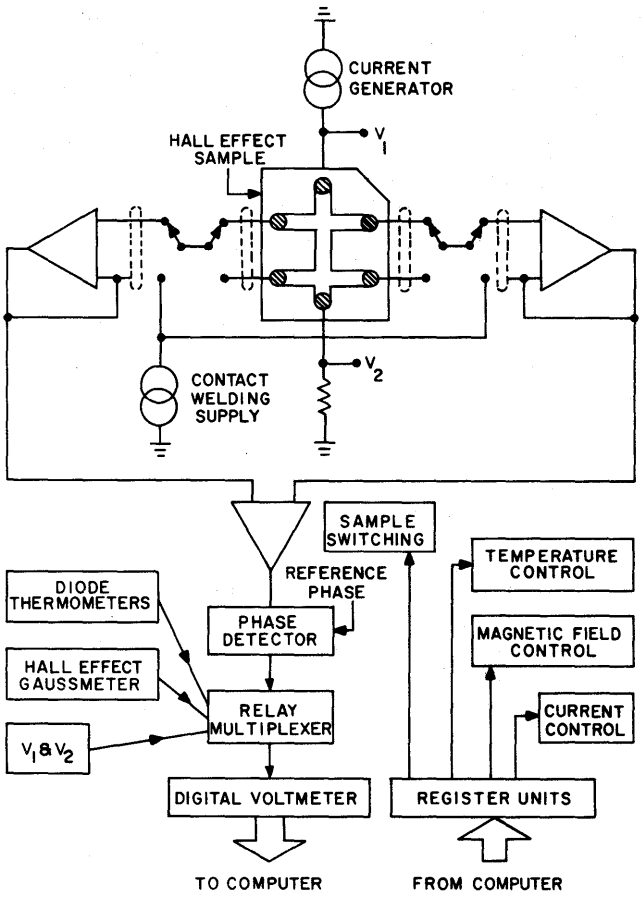


FIGURE 11—Electrical block diagram of Hall effect measurement system

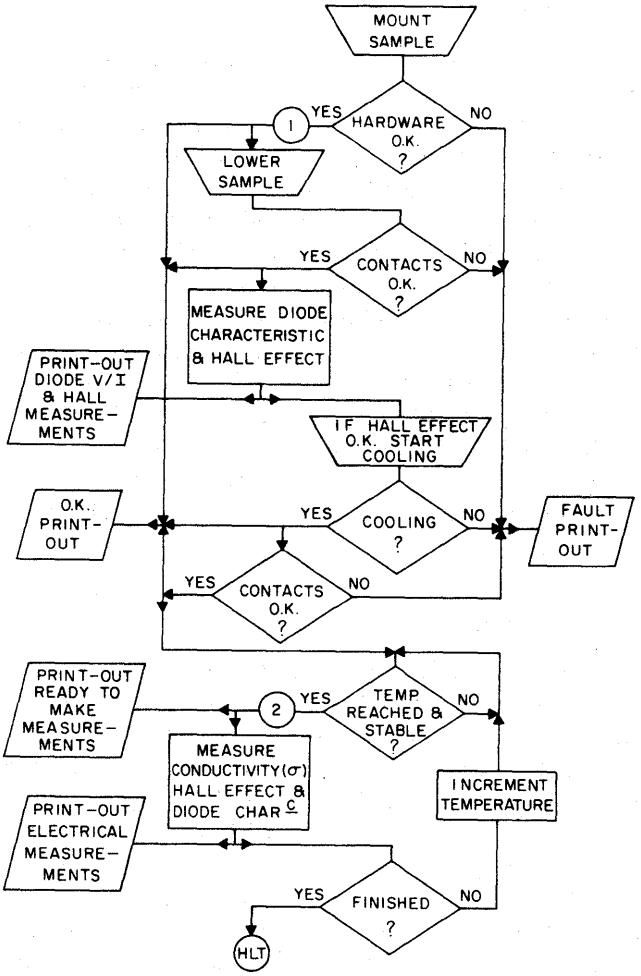


FIGURE 12—Simplified flow-graph of Hall effect measurement

dicates the satisfactory completion of each test.

When the hardware test is completed the sample is manually lowered into the cryostat. The program continues by welding the sample contacts to ensure good connections and then checking that the voltage drop across each is within acceptable limits. The weld current and the contact test current paths are selected, by using relay units, so that they flow in the appropriate direction (depending on the diode junction type) and through any desired contact.

The next step is the measurement of a complete set of diode characteristics. These are made at several values of current, taken from a table in memory. A first set of Hall measurements is then made. Coefficients are processed and printed out. A manual decision is then made whether the Hall properties exhibited by the sample make continuation of the measurements worthwhile.

In continuing the test, the operator types in the upper and lower limits of temperature range and the increments at which measurements are to be made, and opens the liquid helium valve on the cryostat. The computer selects the temperature values by interpolation from a table in memory, starting with the lowest specified temperature. The required temperature control is provided by the setting of a programmable power supply whose output controls the power applied to the sample heaters.

The computer proceeds in a similar way to control and check the remaining experimental conditions, as indicated in Figure 12.

The measurement of each of the many voltages is accomplished by using relay units as multiplexers at the input of a digital voltmeter. Two register modules are used to receive the DVM digital data and transfer it to the computer via the data bus. One of these registers is used to trigger, and also to recognize the end of each DVM measurement, signaling to the computer that data is ready for input.

Other applications

Many uses other than those already described have been considered. An example is that of automated sample liquid scintillation counting in which programmable power supplies can provide levels defining pulse-height windows. In conjunction with scalers this provides pulse height analysis, while relay/register combinations can exercise electromechanical control.

The system may also serve as a versatile, economical alternative to large multiparameter pulse height analyzers when used in conjunction with pulse analog-to-digital converters, and fast digital-to-analog converters for CRT displays.

When connected to an engineering breadboard, the system has been used as a versatile programmed pulser and circuit tester.

DISCUSSION

The point can be made, and with justification, that computer manufacturers realized years ago that peripheral hardware was best handled on a bus basis, which is all that is being achieved with the system described here. This is quite true. The differences that arise with on-line systems are primarily those of degree (with the exception of analog bus facilities) rather than those of kind. One example will suffice to make the point. The present system might be required to handle 60 scalers all counting at ~1 MHz (e.g., a data rate of 6×10^7 bits/second) with the subsidiary requirement that various subsets of them be gated on and off in exact time synchronism. Such requirements are not encountered with standard computer peripherals for which supervisory control and timing can always be exercised in a logical sequential manner.

The major point being made here is really a different one, namely how to design a modular system with a small number of different types of modules to encompass a large number of on-line tasks. While examples of such tasks are endless it is hoped the outline of the rationale of the design, together with the sample applications given, demonstrates the flexibility that such an on-line modular analog-digital system can provide.

ACKNOWLEDGMENTS

It is a pleasure to acknowledge the many contributions of others to the work presented here. Notable among these, have been E. H. Cooke-Yarborough and his associates at AERE Harwell in discussions of design philosophy and in providing information on their IANUS system, R. Stensgaard of the University of Aarhus in all phases of the work on Hall effect measurements, and W. L. Brown of Bell Telephone Laboratories for constant encouragement and support.

IOP	DATA BIT	COMMAND
1	6	Test Interrupt
1	7	Generate Interrupt from Computer
1	8	Disable Interrupt
1	9	Enable Interrupt
1	10	Set Freeze Output
1	11	Load Register from External Unit
2	—	Load Computer from Register
4	—	Load Register from Computer

TABLE I—Register module commands

IOP	COMMAND
1	Test Unit Ready
2	Enable Relay Drivers
4	Disable Relay Drivers

TABLE II—Relay module commands

A. UNIT ADDRESS COMMANDS

IOP	DATA BIT	COMMAND
1	6	Test Overflow
1	7	Test Saturate
1	8	Disable Generic
1	9	Enable Generic
1	10	Increment
1	11	Clear
2	—	Load Computer from Scaler
4	—	Preset

B. GENERIC ADDRESS COMMANDS

IOP	COMMAND
1	Start Scaling
2	Clear
4	Stop Scaling

TABLE III—Scaler module commands

IOP	DATA BIT	COMMAND
1	11	Motor Off
1	10	Measurement Off
1	9	Measure Current
1	8	Measure Voltage
2		Motor Counterclockwise (Lower Voltage)
4		Motor Clockwise (Raise Voltage)

TABLE IV—Power supply module commands

A standardized data highway for on-line computer applications

by I. N. HOOTON and R. C. M. BARNES

Atomic Energy Research Establishment
Harwell, England

INTRODUCTION

In nuclear experiments the quantity and complexity of the data have led to the widespread adoption of automatic processing equipment. This equipment may be divided generally into two categories. The first consists of data gathering and converting devices intimately involved with the experiment, e.g., radiation detectors, scalers and analogue-to-digital converters. The second category consists of devices for storing and processing the data, and for controlling the experiment. Increasing use is now being made of small general purpose computers to perform these latter functions, in place of the special purpose devices previously employed.

Each experimental situation requires a unique system of equipment, although in principle the functions to be performed are very similar. This has led, historically, to the creation of modular instrumentation systems to satisfy the experimental requirements in the first category above. For example, the Harwell 2000 Series,[1] the ESONE System[2] and the U.S.A.E.C. N.I.M. System[3] provide standard hardware which allows up to 5, 8 and 12 modular units, respectively, to be held in a 19 inch rack-mounted crate. Within each system mechanical and electrical compatibility is achieved by specifying standards and codes of practice. None of these systems however incorporates a means of communicating with computers, and individual laboratories have adopted *ad hoc* arrangements. As a consequence the representatives of major European nuclear laboratories (see Appendix) have collaborated under the auspices of the ESONE (European Standard of Nucleonic Equipment) Committee to draw up recommendations for a modular system incorporating a standardised data highway. The basic features of this system, known provisionally as IANUS, may be summarised as follows:

a) It is a modular system so that any combination of functional units may be assembled easily by the experimenter.
b) Each module makes direct connection to a highway which conveys digital data, control signals and power. The highway standards are independent of the type of module or computer used.
c) The mechanical structure is designed to exploit the high component packing density possible with integrated circuit packages and similar devices.
d) The data transfer highway and modules are kept as simple as possible. Any system complexity is introduced in the interface between the computer and the highway.
e) Although the recommendations for the IANUS system were drawn up by representatives of nuclear laboratories it was designed as a generalised data handling system for use in any field of instrumentation.
f) IANUS is a non-proprietary specification which is freely available to all.
g) The IANUS system incorporates the experience gained with previous modular systems and aims to augment rather than replace these systems.

The system has provisionally been given the name IANUS after the Graeco-Roman god Ianus (or Janus) Gemini who had two faces. The system looks to the experimental environment and to the computer. It also looks back to previous modular instrumentation and forward to the fully automated laboratory.

This paper gives an informal description of the IANUS system and then outlines the way it is used at the Atomic Energy Research Establishment, Harwell, as a standard interface between digital computers and peripheral devices.

The IANUS system

IANUS is a modular instrumentation system which links transducers or other devices with digital controllers or computers. Figure 1 illustrates the use of the system in a generalised experiment such as is commonly

TABLE I—Standard dataway usage

Title	Designation	Pins	Use at a Module
Command			
Station Number	N	1	Selects the module (Individual line from control station).
Sub-Address	A1, 2, 4, 8.	4	Selects a section of the module.
Function	F1, 2, 4, 8, 16.	5	Defines the function to be performed in the module.
Timing			
Strobe 1.	S1	1	Controls first phase of operation (Dataway signals must not change).
Strobe 2.	S2	1	Controls second phase (Dataway signals may change).
Data			
Write	W1 - W24	24	Bring information to the module.
Read	R1 - R24	24	Take information from the module.
Status			
Look-at-Me	L	1	Indicates request for service (Individual line to control station).
Response	Q	1	Indicates status of feature selected by command.
Busy	B	1	Inhibits change in Dataway signals (except as permitted at S2).
Non-Addressed			Operate on all features connected to them, no command required.
Initialise	Z	1	Sets module to a defined state. (Accompanied by S2)
Inhibit	I	1	Disables features for duration of signal.
Clear	C	1	Clears registers. (Accompanied by S2)
Reserved			
Reserved Bus	X	1	Reserved for future allocation. May be used as a patch bus.
Private Wiring			
Patch Points	P1 - P5	5	Free for unspecified interconnections. No Dataway lines.
Mandatory Power Lines			The IANUS Crate is Wired for Mandatory and Additional Lines
+24V D.C.	+24	1	
+6V D.C.	+6	1	
-6V D.C.	-6	1	
-24V D.C.	-24	1	
OV	0	2	Main power return.
Additional Power Lines			Lines are Reserved for the Following Power Supplies
+200V D.C.	+200	1	Low current for indicators etc.
+12V D.C.	+12	1	
-12V D.C.	-12	1	
117V A.C. (Live)	ACL	1	
117V A.C. (Neutral)	ACN	1	
Clean Earth	E	1	Reference for circuits requiring clean earth.
Reserved	Y1, Y2	2	Reserved for future allocation.
TOTAL		86	

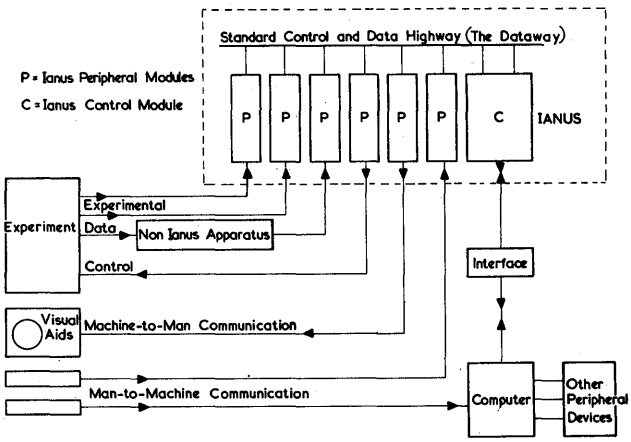


Figure 1—A generalised experimental system

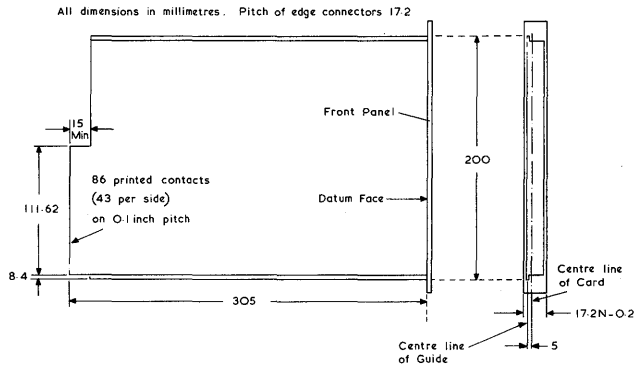


Figure 2—Critical dimensions of the IANUS module

encountered in R. and D. establishments. Experimental data produced by transducers may be taken directly to peripheral modules (P) where, for example, it is digitised or buffered before being presented in standard form to the control and data highway or 'Dataway.' Alternatively, use may be made of equipment in other standards with an adapter module to produce compatible signals. Other peripheral modules may be used to provide communication between the experimenter and the computer or to provide control signals to the experiment. Previous experiences suggest that the experimental parameters should be set up via the computer so that they may be checked and recorded. A controller (C) supervises the operation of the Dataway and provides connection between the IANUS system and the computer. It will be noted that the peripheral modules are isolated from the computer and are hence independent of its input/output standards.

Mechanical standards

A standard IANUS crate mounts in a 19 inch rack and is fitted with 25 edge-connector sockets. Each socket, together with upper and lower guides, constitutes a 'station' into which a module may be inserted. A module consists basically of a printed card and a front panel. Runners on the module engage with the guides in the crate, and 86 printed plug contacts on the card mate with the edge-connector socket. Figure 2 shows the critical dimensions of the module which define the position of the printed contacts in relation to the runners and the front panel. Any method of construction which conforms to these dimensions will be compatible with a standard crate. A module may occupy as many crate positions as are required. Units in the U.S.A.E.C. N.I.M. format will fit into the guidance system, with each NIM single width occuping two basic IANUS widths and a simple adapter completing the connection from the AMP connector on the NIM unit to the edge connector socket.

The Dataway

The Dataway is a standard highway which conveys digital data, control signals and power. It consists mainly of bus-lines joining corresponding pins on the 86-way edge-connector sockets within a standard crate. One station, known as the 'control station,' has individual lines to each other station. The module occupying the control station acts as a 'controller' for its crate. During each Dataway operation the controller generates a command on the control lines of the Dataway. It is implicit in all data transfer operations that one participant is the controller and the other is the module or modules specified by the command. In practice the various features which constitute the controller may be distributed among several physical units. It is convenient to distinguish between the controlling and controlled parts of a system by the generic terms 'controller' and 'modules.' The Dataway includes lines by which modules can demand attention and a line by which the controller can test the status of a module.

The Dataway lines (see Table I) may be divided into seven categories as follows:

1. *Commands*: A command consists of signals which select a specified module or modules within a crate (by station number lines), a particular section of the module (by sub-address lines) and the function to be performed (by function lines). Each normal station is addressed individually by a signal on one of the station number lines (N) which link the control station separately to each other station (see Tables II and III). There is no restriction of the number of modules that may be addressed simultaneously, so that the same command may be given to any desired selection of modules in the same operation. The duration of the signal on the N line defines the Dataway operation period.

TABLE II—Pin allocation at normal station viewed from front of crate

Individual Patch Point	P1	B	Busy - Bus Line
" " "	P2	F16	Function " "
" " "	P3	F8	" " "
" " "	P4	F4	" " "
" " "	P5	F2	" " "
Bus Line with Patch Point - Reserved	X	F1	" " "
" " " " " - Inhibit	I	A8	Sub-Address " "
" " " " " - Clear	C	A4	" " "
Individual Lines - Station Number	N	A2	" " "
with Patch Points - Look-at-Me	L	A1	" " "
Bus Line - Strobe 1	S1	Z	Initialise " "
Bus Line - Strobe 2	S2	Q	Response " "
24 Write Bus Lines	W24	W23	
	W22	W21	
	W20	W19	
	W18	W17	
	W16	W15	
	W14	W13	
	W12	W11	
W1 = least significant bit	W10	W9	
W24 = most significant bit	W8	W7	
	W6	W5	
	W4	W3	
	W2	W1	
24 Read Bus Lines	R24	R23	
	R22	R21	
	R20	R19	
	R18	R17	
	R16	R15	
	R14	R13	
	R12	R11	
R1 = least significant bit	R10	R9	
R24 = most significant bit	R8	R7	
	R6	R5	
	R4	R3	
	R2	R1	
Reserved for -12 volts D.C.	-12	-24	-24 volts D.C.
Reserved for +200 volts D.C.	+200	-6	-6 volts D.C.
Reserved 117 volts A.C. Live	ACL	ACN	Reserved for 117 volts A.C. Neutral
Reserved	Y1	E	Reserved for Clean Earth
Reserved for + 12 volts D.C.	+12	+24	+24 volts D.C.
Reserved	Y2	+6	+6 volts D.C.
0 volts (Power return)	0	0	0 volts (Power Return)

Signals on the sub-address bus lines (A8, A4, A2 and A1) specify one of the 16 sub-addresses in the module. This sub-address may be used to select a specific register, to define which of sixteen different flags controls the Response signal, or to direct functions such as 'enable,' 'disable' or 'clear' to the required section of the module.

Signals on the five function bus lines (F16, F8, F4, F2 and F1) specify one of 32 functions. Sixteen of these functions are fully defined within the IANUS recommendations. This permits the same command to be interpreted correctly in modules from different designers and assists in the design of table-driven software. The remaining sixteen codes are available for special functions at the discretion of individual designers. The standardisation of codes is discussed more fully below in the section on 'Function Codes.'

2. *Timing*: Each Dataway operation occurs asynchronously, and is timed by two strobes S1 and S2 which are generated in sequence on separate bus lines. The times, T1 to T5, shown in Figure 3 may each have any value greater than a prescribed minimum. This minimum is currently defined as 200 nS. The signals conveying the command are maintained throughout each Dataway operation and the other signals are set

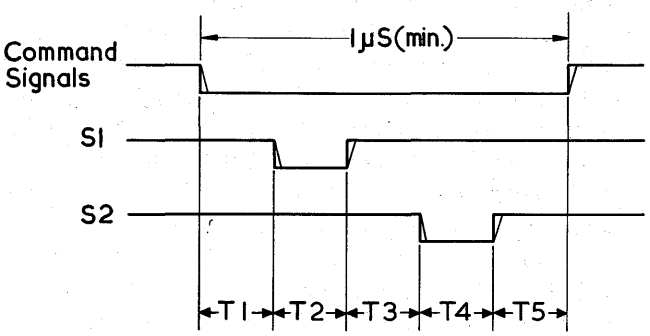


Figure 3—Dataway timing

TABLE III—Pin allocation at control station viewed from front of crate

Individual Patch Point	P1	B	Busy – Bus Line
" " "	P2	F16	Function – " "
" " "	P3	F8	" – " "
" " "	P4	F4	" – " "
" " "	P5	F2	" – " "
Bus Line with Patch Point – Reserved	X	F1	" – " "
" " " " " – Inhibit	I	A8	Sub-address – " "
" " " " " – Clear	C	A4	" – " "
Individual Patch Point	P6	A2	" – " "
" " "	P7	A1	" – " "
Bus Line – Strobe 1	S1	Z	Initialise – " "
Bus Line – Strobe 2	S2	Q	Response – " "
	L24	N24	
	L23	N23	
	L22	N22	
	L21	N21	
	L20	N20	
	L19	N19	
	L18	N18	
	L17	N17	
	L16	N16	
	L15	N15	
	L14	N14	
24 Individual Look-at-Me Lines	L13	N13	24 Individual Station Lines
	L12	N12	
	L11	N11	
	L10	N10	
	L9	N9	
	L8	N8	
	L7	N7	
	L6	N6	
	L5	N5	
	L4	N4	
	L3	N3	
	L2	N2	
	L1	N1	
Reserved for -12 volts D.C.	-12	-24	-24 volts D.C.
Reserved for +200 volts D.C.	+200	-6	-6 volts D.C.
Reserved for 117 volts A.C. Live	ACL	ACN	Reserved for 117 volts A.C. Neutral
Reserved	Y1	E	Reserved for Clean Earth
Reserved for +12 volts D.C.	+12	+24	+24 volts D.C.
Reserved	Y2	+6	+6 volts D.C.
0 volts (Power Return)	0	0	0 volts (Power Return)

up as soon as the command has been interpreted. For example, a module instructed to transmit will establish its data on the 'Read lines' in response to the command. Sufficient time is allowed for the data to settle before the first strobe (S1) admits this data to the controller. S1 is used for actions which do not change the state of signals on the Dataway. The second strobe S2 may be used to initiate actions which change the state of Dataway signals, for example, clearing a register which has just been read and whose output is connected to the Dataway.

3. *Data*: Up to 24 bits may be transferred in parallel to or from the selected module. Independent lines ('Read' and 'Write') are provided for the two directions of transfer.

All information carried on these data lines is regarded as 'data' although it may in specific instances be concerned with the status or control features of the modules. The 24 parallel lines in each direction set an upper limit to the word length but shorter words may be transferred.

As shown in Tables II and III, the data lines are not taken to the control station, where the corresponding pins are used for N and L lines. A controller therefore requires connection to the control station and one normal station.

4. *Status*: Individual Dataway lines from each station to the control station are used for 'Look at Me' signals (L) by which modules can demand attention. The action taken in response to such a signal is a property of the controller and/or computer. The L signals may be cleared, enabled and disabled by appropriate commands.

The status of the Dataway itself is indicated by a Busy signal (B). While the N signal specifies the duration of a Dataway operation to the modules involved, the B signal indicates to all modules that an operation is in progress. It is generally used to staticise the con-

ditions on the Dataway so as to reduce crosstalk. While B is present all signals, including for example L signals, must remain constant unless their state is modified at S2.

The status of any specific feature of a module may be tested by a command to transmit a signal on a common Response (Q) bus line. The appropriate signal is set up on the Q line by the module as soon as the command is recognised, it is strobed into the controller at S1 and may be changed at S2 if the feature being tested is set or cleared then. One bit of status information may thus be sent to the controller with every Dataway operation and may, for example, be held in a one-bit register for testing by the computer. The module designer is free to decide which feature, if any, is tested.

An obvious application of the Response signal is to test for the source of a 'Look at Me' demand in a system which has only a single-level interrupt. In more complex installations it may identify which of several possible demands has originated from the same module.

5. *Common Controls*: Common control signals operate on all modules connected to them without requiring individual addressing signals.

The Initialise signal (Z) is generally connected to all modules and forces them to a basic state by resetting all data and control registers. It also clears all L signals and where possible disables them. It provides a quick and sure method of dealing with the situation at switch-on when registers and control bistables may, in principle, have assumed unpredicted states.

A common time control is provided by an Inhibit signal (I). This may be connected to any modules which require accurate timing of activities, such as data taking, independent of computer access times. The Inhibit signal may be controlled by any appropriate signal from the computer, the IANUS system or elsewhere.

A common Clear signal (C) may be connected to any selection of modules in which data registers require to be cleared.

As a protection against spurious signals both the Initialise (Z) and Clear (C) signals are accompained by the S2 strobe.

6. *Private Wiring*: Five ways on the 86-way socket are not connected to bus lines but may be brought out to individual local pins on the Dataway wiring. They are available for unspecified patch connections subject only to the restriction that the signals must not interfere with standard Dataway signals and must be able to tolerate some crosstalk from the Dataway lines. Highly sensitive signals or those that require coaxial connection are more appropriately located in the space provided above the edge connector socket.
7. *Power*: The mandatory power supplies, which may be used by any module, are +24V, +6V, −6V and −24V d.c. The maximum current loading for a crate is 6A for each 24 volt line and 25A for each 6 volt line. The recommended total power dissipation is 200W per crate. Two pins are provided in parallel on the edge-connector as a heavy current 0 volt return for digital circuits.

Lines are provided for +12V and −12V d.c. optional supplies. Low current lines are allocated for a +200V d.c. supply (primarily for neon indicators) and for 117V a.c. (two lines.) There are also two 'reserved' power lines which may be allocated in the future. A return, independent of and isolated from the digital 0 volt line, is provided for low current operations that require a 'Clean Earth.'

Signal standards

The signal standards are derived from Compatible-Current-Sinking-Logic (CCSL), that is, Diode-Transistor-Logic (DTL) and Transistor-Transistor-Logic (TTL). In all situations where more than one module may feed onto a line intrinsic (or wired) OR operation is specified. This requirement makes it appropriate that the 'Low' state (short to ground) be interpreted as the significant or '1' state while the 'High' state (open circuit) is the non-significant or '0' state.

Function codes

While it is desirable for table-driven software, for autonomous operation, and for multiple addressing to have standardised functions, it is impossible to legislate for all possible modules. Half of the 32 functions (see Table IV) are therefore fully specified and the other half are available for special applications. For software convenience a single bit (F4) distinguishes between standard (i.e., universal) codes and non-standard (i.e., local) codes. Similarly, bit F16 specifies the direction of data transfer. This is an important feature with some computers when the controller must interpret the function in order to set up the required data path.

Registers within a module may be divided into two groups which have independent commands. It is therefore simple to distinguish between control and data registers. This is particularly useful in a system which makes large scale use of autonomous transfers or in which it is desired to 'step through' registers on the sub-address codes.

The incrementing functions are provided mainly as a test facility; code 27 in particular permitting a group of preselected registers within a module to be incremented simultaneously.

TABLE IV—*IANUS function codes*

No.	Function	State After S1	State After S2	F 16	F 8	F 4	F 2	F 1	No.
0	Read Group 1 Register	$Q = F_A$; $C_i = {}^1A_i$	$L_A = 0$; ${}^1A_i = {}^1A_i$	0	0	0	0	0	0
1	Read Group 2 Register	$Q = F_A$; $C_i = {}^2A_i$	$L_A = 0$; ${}^2A_i = {}^2A_i$	0	0	0	0	1	1
2	Read and Clear Group 1 Register	$Q = F_A$; $C_i = {}^1A_i$	$L_A = 0$; ${}^1A_i = 0$	0	0	0	1	0	2
3	Read Complement of Group 1 Register	$Q = F_A$; $C_i = \overline{{}^1A_i}$	$L_A = 0$; ${}^1A_i = \overline{{}^1A_i}$	0	0	0	1	1	3
4	Non-standard			0	0	1	0	0	4
5	Reserved			0	0	1	0	1	5
6	Non-standard			0	0	1	1	0	6
7	Reserved			0	0	1	1	1	7
8	Test Flag	$Q = F_A$	$F_A = F_A$	0	1	0	0	0	8
9	Clear Group 1 Register	$Q = F_A$	${}^1A_i = 0$	0	1	0	0	1	9
10	Clear Flag	$Q = F_A$	$F_A = 0$	0	1	0	1	0	10
11	Clear Group 2 Register	$Q = F_A$	${}^2A_i = 0$	0	1	0	1	1	11
12	Non-standard			0	1	1	0	0	12
13	Reserved			0	1	1	0	1	13
14	Non-standard			0	1	1	1	0	14
15	Reserved			0	1	1	1	1	15
16	Overwrite Group 1 Register	$Q = F_A$; ${}^1A_i = C_i$	$L_A = 0$ *	1	0	0	0	0	16
17	Overwrite Group 2 Register	$Q = F_A$; ${}^2A_i = C_i$	$L_A = 0$ *	1	0	0	0	1	17
18	Selective Overwrite Group 1 Register	$Q = F_A$; If $K_i = 1$, ${}^1A_i = C_i$; If $K_i = 0$, ${}^1A_i = {}^1A_i$	$L_A = 0$ *	1	0	0	1	0	18
19	Selective Overwrite Group 2 Register	$Q = F_A$; If $K_i = 1$, ${}^2A_i = C_i$; If $K_i = 0$, ${}^2A_i = {}^2A_i$	$L_A = 0$ *	1	0	0	1	1	19
20	Non-standard			1	0	1	0	0	20
21	Reserved			1	0	1	0	1	21
22	Non-standard			1	0	1	1	0	22
23	Reserved			1	0	1	1	1	23
24	Disable	$Q = F_A$		1	1	0	0	0	24
25	Increment Group 1 Register	$Q = F_A$; $R_A = R_A + 1$		1	1	0	0	1	25
26	Enable	$Q = F_A$		1	1	0	1	0	26
27	Increment Preselected Register	$Q = F_A$; $R_p = R_p + 1$		1	1	0	1	1	27
28	Non-standard			1	1	1	0	0	28
29	Reserved			1	1	1	0	1	29
30	Non-standard			1	1	1	1	0	30
31	Reserved			1	1	1	1	1	31

Q is the state of the Q line.
F_A is the state of the Flag associated with the selected subaddress.
L_A is the state of the Look at Me associated with the selected subaddress.
C_i is the state of the i^{th} bit of the register in the controller.
${}^1A_i$ is the state of the i^{th} bit of the register, selected by subaddress, from a first group of registers in the module.
${}^2A_i$ is the state of the i^{th} bit of the register, selected by subaddress, from a second group of registers in the module.
K_i is the state of the i^{th} bit in a mask register.
R_A is the content of a register selected by subaddress in the module.
R_p is the content of a register preselected in the module.
$Q = F_A$ indicates that the state of the Flag at the selected subaddress is tested before the Flag is cleared.
* Data transfers ${}^1A_i = C_i$ and ${}^2A_i = C_i$ may, alternatively, occur at Strobe S2.

The term Overwrite (codes 16-19) is synonymous with 'Jam Transfer'—a term which has led to some confusion, particularly in translation.

Application of the system

The IANUS specification sets out in detail the logical and electrical features of a data and control highway within a single crate system. It also defines the mechanical dimensions necessary to ensure that modules from different sources are mutually interchangeable. These recommendations have been accepted by major nuclear laboratories throughout Europe.

In practical applications of computer-based data processing it is also necessary to specify the computer-to-crate and crate-to-crate interconnections. The following sections describe the local standards and techniques adopted at A.E.R.E., Harwell, for computer-based IANUS systems.

External representation of the command

A command is represented within a IANUS crate by a 5-bit function code, a 4-bit sub-address code, and signals on the appropriate station number (N) lines. When the command is transmitted externally, e.g., between a crate and a computer, it is generally perferable to use a 5-bit code for N instead of the internal 24-bit form. The decoded values 1 through to 24 are used to address the corresponding stations directly. The remaining codes (25 through to 31) may have special applications. For example Code 31 may be interpreted as 'address all modules' and Code 30 as 'address preselected modules.' This makes it possible to address the same command to a selection of modules simultaneously.

The command to a single crate requires 14-bits. In multi-crate systems the command is extended to include a binary coded crate address, e.g., a 16-bit command will control a 4-crate system.

Multi-crate operation

In a single-crate system the controller has to perform logical and level conversions between the specific computer and the IANUS standards. This 'Master

Controller' is given the crate address 'O' so that it can operate with 14-bit commands. The number of crates may be extended by providing a 'Line Driver' module in the master crate. This buffers the Dataway bus-lines onto an external highway which feeds a 'Slave Controller' in each added crate. The line driver and slave controller modules are independent of the computer type. The master crate may contain more than one line driver module, so that the connections to slave crates can be arranged as a star rather than as a highway (see Figure 4). Some connections, in addition to the Dataway, are required between the master controller and line driver modules, e.g., for the coded crate and station addresses. These are provided by a multiway socket above the Dataway at the rear of the crate.

Program interrupt demands

The individual Look-at-Me (L) signals within a single crate are brought to the controller via the Dataway. There are three general methods by which the computer may identify the source of the demand. Firstly, the L signals may be taken individually from the controller to a multi-level priority interrupt option on the computer. Secondly, the demands may be combined into a single program interrupt request, leaving the computer to identify the highest priority demand by a search algorithm. This may operate on either the individual sources of L signals by 'Test Flag' commands, or on a 24-bit status word giving the pattern of demands on the L lines. Thirdly, the controller complex may include a

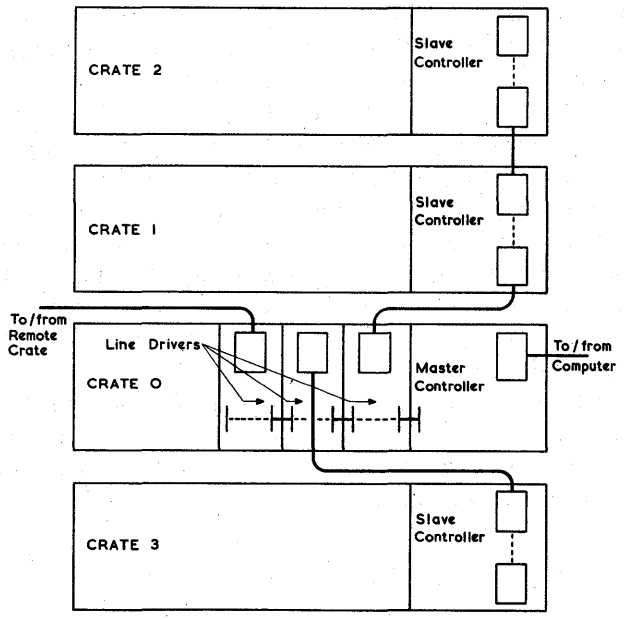


Figure 4—Multi-crate interconnection

'priority sorter' module to select the highest priority demand and identify it to the computer. These three methods may be combined in various degrees.

In a multi-crate configuration a single status flag is generated within each crate to indicate that a demand is present. The intercrate connection provides a common line for these flags, a response line by which the master controller may staticise the crate demands, and lines on which the highest priority crate may indicate its binary coded address. In such an arrangement each additional crate automatically adds a crate priority level to the system. Alternatively the individual crate flags could be assembled into a status word in the master controller. When the crate has been identified by the computer any of the previously described methods may be used to locate the individual module making the request.

Autonomous operations

Autonomous operations typically transfer data to or from the computer store via direct access I/O channels in response to a Look-at-Me signal, without requiring a direct command from the computer program. Each demand may result in one Dataway operation (Simple Autonomous Operation) or a sequence of Dataway operations (Complex Autonomous Operation).

Simple autonomous operations: Demands for simple autonomous operations originate in precisely the same way as demands for program interrupts, i.e., as Look-at-Me signals generated in modules and transmitted along the individual L lines to the crate controller. Here they are not combined, as demands for program interrupt may be, but are routed as individual signals to a priority sorter module in the same crate. This module generates a request for autonomous operation on a common line to the master controller, which then freezes the priority sorters and reads a channel number code corresponding to the highest priority demand present. The channel number specifies the type and direction of transfer required. The controller, in conjuction with a direct-access I/O channel of the computer, initiates either a Dataway Read operation followed by an input to the computer, or an output from the computer followed by a Dataway Write operation. The simplest priority sorter generates a command consisting of one function bit associated with the channel code (giving a choice of function 0 for Read transfers or function 16 for Write transfers), and a station number derived from the identity of the L line. The sorter can, however, be elaborated to generate a full command with choice of station address, sub-address and function codes. On completion of the transfer the priority sorters are released. If a buffer area limit ('End of Record,' 'Word Count Overflow')

in the computer has been reached, a program interrupt is generated.

Multi-word records may be transmitted from a module by maintaining the demand flag after the completion of each transfer until the record is complete. However, since the system operates in multiplexer rather than selector mode, a higher priority demand will interrupt the sequence.

Complex autonomous operations: These consist of a defined sequence of different Dataway operations initiated by a single demand. One complex operation may include, for example, reading some registers, writing into others and generating a program interrupt. One or more such operations are controlled by a 'programmer' module which holds a list of extended commands, each consisting of the normal command together with a channel code (as described above). There is also a tag bit indicating the last extended command in the sequence. A Look-at-Me signal demanding a complex operation starts the appropriate sequence in the programmer, which then competes with any other requests for autonomous operations. When its request is granted the programmer sets up the first extended command, and the master controller initiates the appropriate I/O and Dataway transfers. The programmer repeats its request and generates successive extended commands until the sequence is completed. If a higher-priority demand intervenes the programmer remains locked at its current state until it is able to resume the sequence.

The programmer may have a patchboard on which the list of commands is set up manually, or a scratchpad memory which is loaded by the computer. Alternatively it may have fixed, prewired commands to suit a specific application.

Module-to-module transfers

Information may be transmitted between modules via the computer in two Dataway operations. A direct module-to-module transfer in a single operation may be performed under program control or autonomously. In the latter case it is identified by a specific autonomous channel code. Since two modules are involved, one Reading and the other Writing, it follows that one of them must interpret the Dataway command in a non-standard manner. During this operation the master controller couples the Read and Write lines in order to complete the data path.

Error checking

The IANUS system incorporates facilities for checking data transfers within its own boundaries. In a Write operation a data word is established on the Write lines of the Dataway by the controller and strobed into the module by S1. The data signals are maintained but the function code is then forced to 'Read Complement' (Code 3). The content of the module register is unaffected by this command but the complement of its content is put out on the Read lines. A separate module checks that corresponding bits on the Read and Write lines are in fact complementary. The original function code is then allowed to return and the Dataway operation completed.

With a Read operation the same procedure is followed except that, after the data have been transferred from the module to the controller, the 'Read Complement' function causes the module to put the complement of its register on the Read lines and the controller to put the content of its data register on the Write lines. The comparison then continues as before.

This technique is more rigorous than a simple parity check and requires no additional lines on the Dataway. The hardware to generate complements need only be added in those modules which require error checking. The technique is equally applicable to program controlled and autonomous transfers. Only one comparison module is required in a multi-crate system and may be installed in any crate.

Sub-standard systems

Simple laboratory experiments often make use of small computers which may have only a 12-bit word length, program controlled input/output, and a single-level program interrupt. Cost is a major consideration is these systems.

Many modules transfer 12-bits or less per operation. When more bits are essential they may be accessed in 12-bit byte mode by separate transfers at different sub-addresses. The computer word length makes it desirable that the command is also limited to 12-bits. This may be achieved with 4-bits for the coded station number (N), 4-bits for the sub-address (A) and 4-bits for the function (F). The restricted address code permits only 16 stations to be addressed. However, the controller itself occupies a number of stations, and certain modules such as fixed gain amplifiers do not require addressing. The reduced function code permits the use of a 16 function subset containing all the standard codes shown in Table IV, or an eight function subset of standard codes (such as Codes 0, 2, 8, 10, 16, 18, 24 and 26) with an additional eight non-standard codes.

The simple interrupt facilities of the computer are used by combining all the L signals as a common interrupt request. With a short connection between the crate and the computer it may also be possible to join the Response line (Q) directly to the computer 'Skip' or 'Branch' facility.

IANUS is particularly suitable for simple applications since the compatibility problems are restricted to the controller. Any peripherals developed specifically for simple systems are fully compatible with larger systems.

Such simple systems may be expanded by the addition of a data buffer register to give the controller 24-bit capacity. Data transfers will now take two I/O operations for modules utilising more than 12-bits. However, virtually the entire range of standard modules becomes available to the experimenter.

System expansion

The expansion of a system to provide any of the facilities outlined above is achieved by additions to the controller and associated units. This capability therefore neither requires modifications to standard modules and the Dataway nor raises their basic cost.

The growth of a minimum program-controlled system into a complex system with autonomous and programmed transfers will be given as an example. Conversion from a sub-standard system to one with the full range of function codes and station addresses involves the replacement of the controller. All further developments are achieved by the addition of appropriate units.

Extension from a single-crate to a multi-crate system requires a line driver module in the master crate and a slave controller in each additional crate. Systems with autonomous transfers need priority sorting modules for simple operations and programmer modules for complex operations, the number of each depending on the system configuration. The same peripheral modules may be used in both programmed and autonomous modes. An error-checking module may be added to any standard system. Since a standard command structure is used, system expansion does not destroy program compatibility. At any stage a change to a different type of computer involves only the replacement of that portion of the master controller which adapts the logical and electrical signal standards. A highly simplified outline of a multi-crate system is shown in Figure 5.

CONCLUSIONS

The pooled resources of major European nuclear laboratories have established the specification for a new range of modular instrumentation as an international standard. This includes the connection of modules to a data and control highway within a single crate. At A.E.R.E., Harwell, this work has been extended to

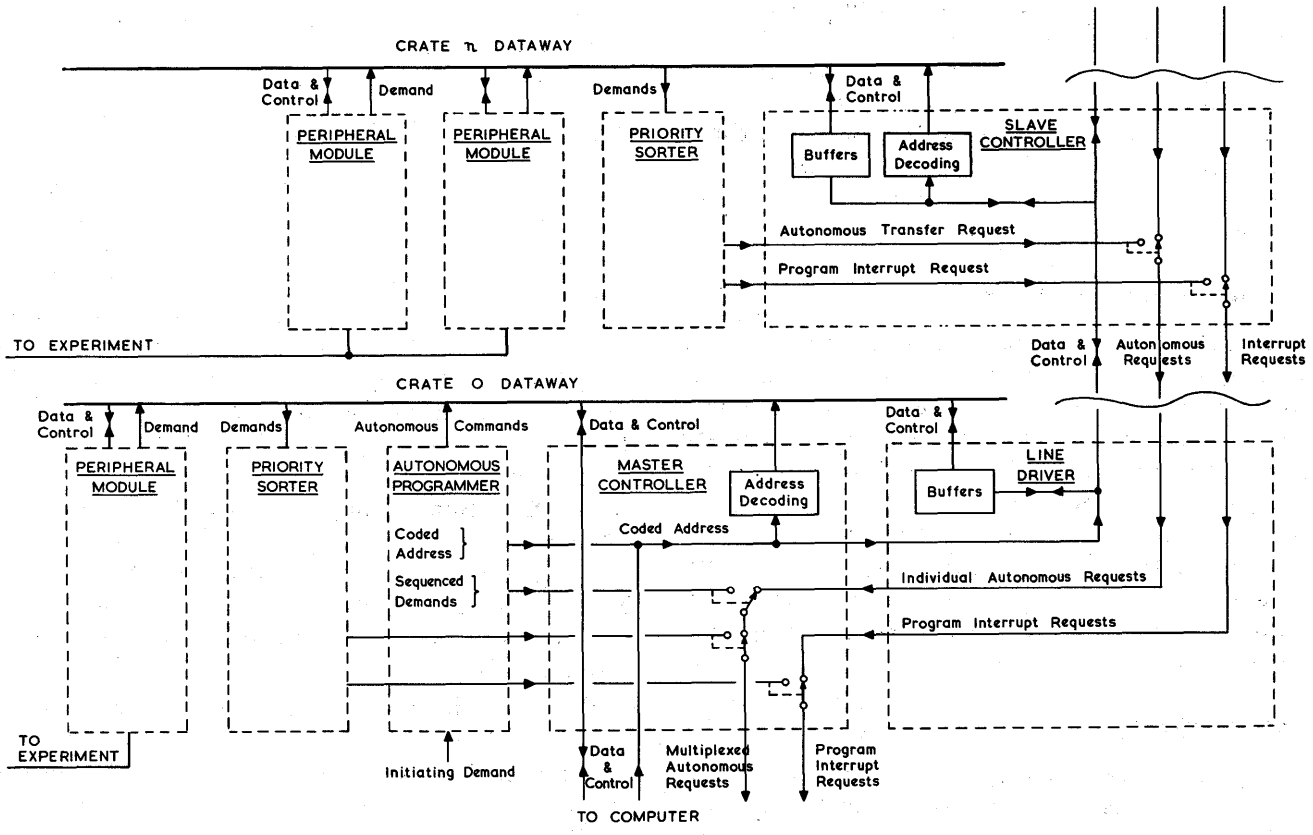


Figure 5—Schematic representation of a complex system

include multi-crate systems with programmed and autonomous transfers to and from on-line computers.

ACKNOWLEDGMENTS

The development of the IANUS system would have been impossible without the active collaboration and good will of representatives from laboratories throughout Europe. The authors hope that this attempt to present the results of that collaboration will indicate their gratitude.

The work at the Atomic Energy Research Establishment, Harwell, has been conducted in the Electronics and Applied Physics Division and the authors must express their thanks to their many colleagues on this project.

REFERENCES

1 *Harwell 2000 series specification and guide SG2000*
AERE UKAEA
2 *ESONE standards EUR 1831e*
Euratom Ispra
3 *USAEC nuclear instrument module series*
TID 20893 (Revision 2) National Bureau of Standards
Washington, DC

APPENDIX

Organisations which took part in the specification of the IANUS system

International	CERN, European Organisation for Nuclear Research, Geneva	Switzerland.
	Centro Comune di Ricerca (Euratom), Ispra	Italy.
	Bureau Central de Mesures Nucleaires (Euratom), Geel	Belgium.
Austria	Studiengesellschaft für Atomenergie	Vienna.
Belgium	Centre d'Etude de l'Energie Nucleaire (CEN)	Mol.
Britain	Atomic Energy Research Establishment	Harwell.
	Rutherford High Energy Laboratory (SRC)	Chilton.
	Daresbury Nuclear Physics Laboratory (SRC)	Daresbury.
France	Centre d'Etudes Nucléaires	Saclay.
	Centre d'Etudes Nucléaires	Grenoble.
Germany	Physikalisches Institut der Universität	Marburg.
	Deutsches Elektronen Synchrotron	Hamburg.
	Hahn Meitner Institut	West Berlin.
	Kernforschungsanlage	Julich.
	Kernforschungszentrum	Karlsruhe.
	Physikalisches Institut der Universität	Frankfurt.
Holland	Reactor Centrum Nederland	Petten.
Italy	Laboratori Nazionali (CNEN)	Frascati.
	Centro Studi Nucleari (CNEN)	Cassaccia.
	Centro Studi Nucleari Enrico Fermi (CESNEF)	Milan.
	Centro Informazioni Studi Esperienze	Milan.
	Instituto di Fisica	Bari.
Yugoslavia	Boris Kidric Institute	Belgrade.
Switzerland	Institut für Angewandte Physik der Universitat	Basle.
Secretariat	Dr. W. Becker, CCR, Euratom, Ispra	Italy.

Use of computers in a molecular biology laboratory

by T. H. GOSSLING and J. F. W. MALLETT

M.R.C. Laboratory of Molecular Biology
Cambridge, England

INTRODUCTION

On-line computers have been making considerable progress in recent years, to the point where they represent a sizeable proportion of the computing field. Only lately however, have they started to make their mark in the laboratory as a research tool. In this paper, we shall illustrate this application from our experience in molecular biology.

In our laboratory at Cambridge, we have been using a time-sharing on-line computer (a Ferranti Argus 300) for a little over three years. We hope that the techniques that we have developed will be of interest to others, and also that our experience in running this kind of computer in a specialized environment may be illuminating. We shall also say a little about possible future developments.

Molecular biology is a science in which computers have a central role. This is particularly true in the study of protein structure, simply because of the enormous quantity of data to be processed; one of the final processes, a three dimensional Fourier transformation, may typically contain some 10^5 terms, and have required as many measurements. Until a few years ago, this was entirely off-line computing, i.e., data in numerical form had to be prepared by hand, and the output, also numerical, required considerable further hand processing to get it into a useable state. Certain parts of the process, particularly data collection, have gradually become automated to the extent that punched tape or cards can be directly generated. From now on, however, we shall see more computers being used for on-line data collection and reduction, for control of experiments, and presentation of comprehensible results.

Since we shall be concentrating mainly on protein structure, it might be as well to summarize briefly what is already known on this subject. A protein is simply a very large organic molecule—of the order of thousands of atoms. Fortunately, this apparent complexity is reduced by the fact that the molecule is a folded chain of about 200 amino-acids, and there are only twenty amino-acids to choose from. To understand how a protein works, we need to know both the *sequence* of amino-acids in the chain and the *physical structure*—the way in which the chain is folded. There are thus two parts to the study, chemical and physical, and these are complementary.

In Figure 1, we show these two approaches, with an indication of the places where computers come into the picture, or might do so in the future. The computers have been classified as "small," "medium" and "large"; this classification is deliberately vague, but roughly the dividing lines may be taken in this context as about 20,000 and 100,000 bytes (of 8 bits) of core store, with appropriate backing store in each case. The many computers shown do not have to be separate, of course. In our case, we make do with two: the on-line computer already mentioned in the laboratory, and an IBM 360 a few miles away. The two machines communicate with each other by magnetic tape, on a daily courier basis.

Use in crystallography

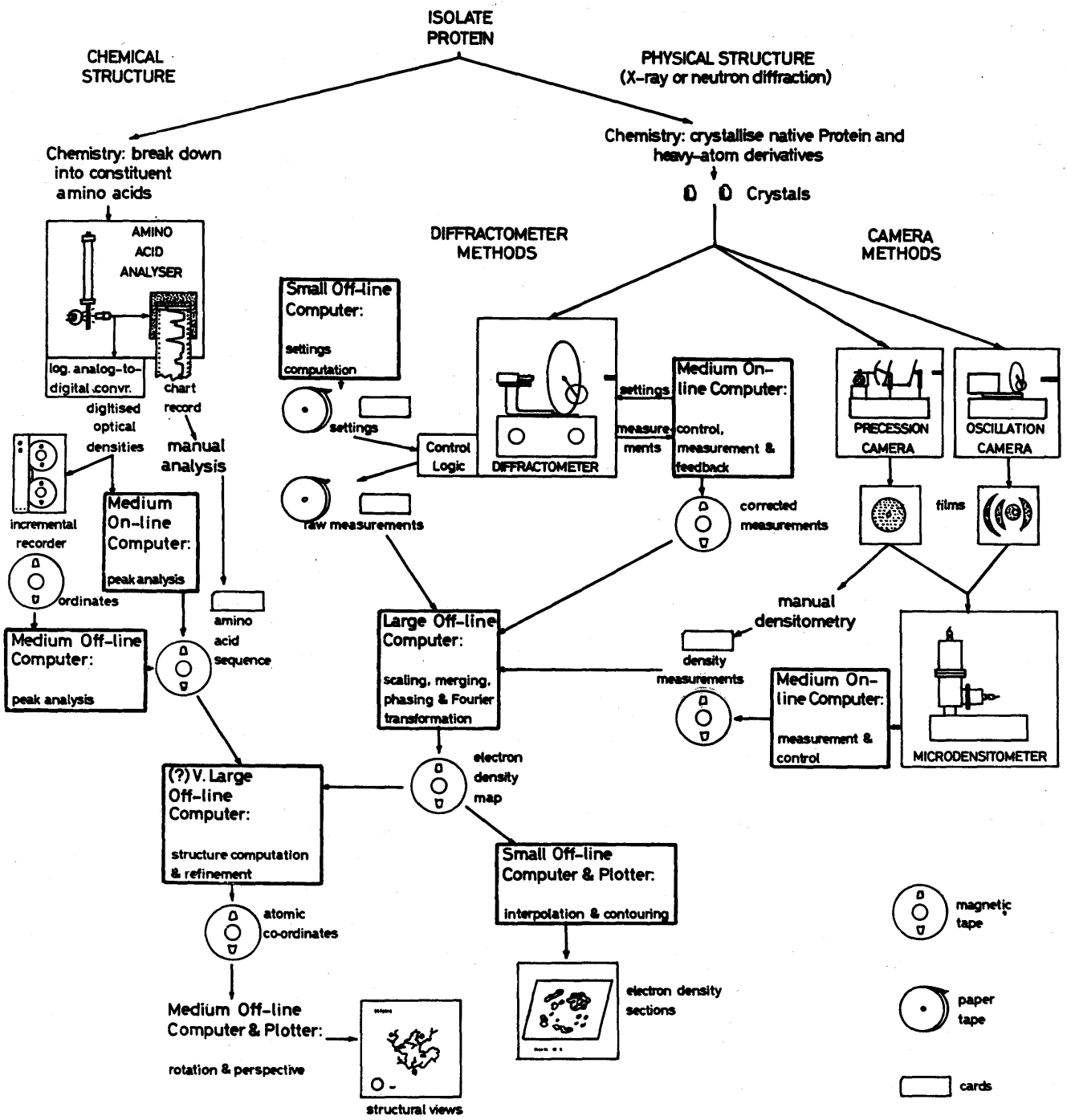
Let us begin with the physical approach—the study of structure by means of X-ray crystallography. To use this technique, the protein must be grown as a crystal, which is a regular lattice of identical molecules in the same orientation. If this is placed in a monochromatic X-ray beam, it acts as a diffraction grating, in that, for certain specific orientations, diffracted beams, called "reflections," are produced. From the intensities and phases of the reflections, it is possible to reconstruct the internal form of the molecule, by Fourier synthesis. Unfortunately, phases are not directly measurable; they can, however, be inferred by repeating the measurements for the same protein, with a heavy atom chemically attached at a known point—this process is similar to holography.

Usually two or three such "heavy-atom derivatives" are needed, and this adds to the amount of data that must be collected. Typically, a total of 10^5 reflections would resolve a protein to about 2 Angstrom Units, which is not fine enough to see individual atoms, but al-

lows identification of the amino-acids, given the chemical sequence.

There are two basic methods of collecting the intensities of reflections from a crystal: diffractometers and X-ray film. A diffractometer (Figure 2) is an instrument for orienting the crystal and an X-ray quantum detector accurately—to about one minute of arc—orientations being changed from reflection to reflection. Computation of the setting angles involves a certain amount of trigonometry, and the accuracy required implies that it must be done digitally. X-ray quantum counting is also inherently a digital process, and most diffractometers are now therefore automated to the extent of taking in settings from pre-computed cards or paper tape, and punching out the measurements in a similar form. From here it is a small step to connect the diffractometer directly to an on-line computer. This offers two advantages. First, the data can be compressed as they are collected, so that not only is the convenience increased but also the errors introduced by electro-

FIGURE 1—General diagram of the stages in solving a protein structue



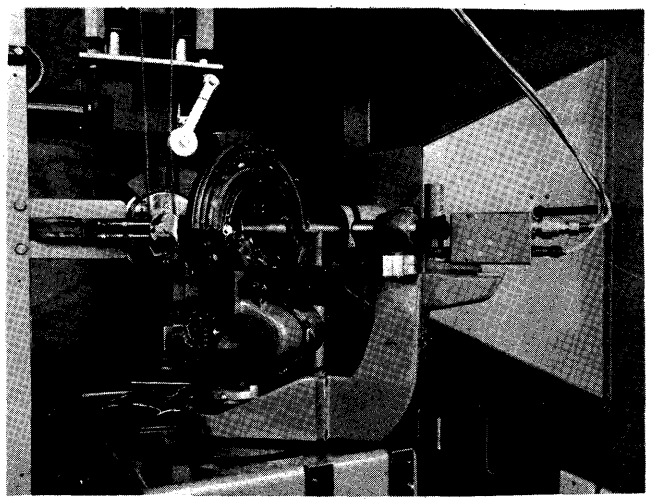


FIGURE 2—A typical diffractometer

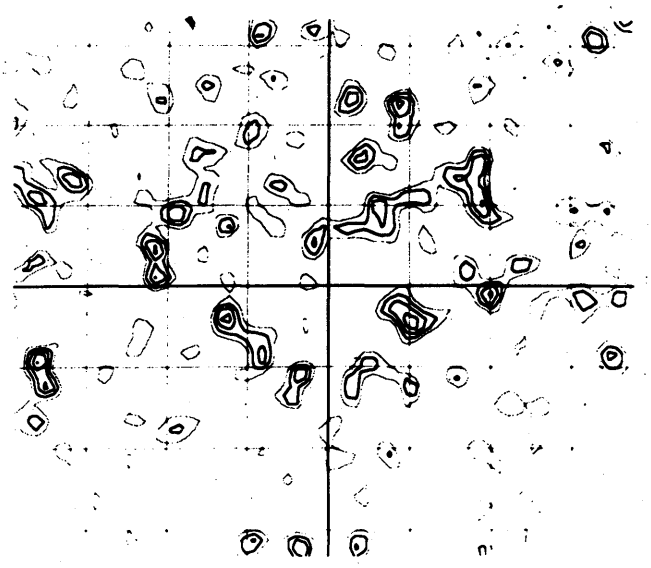


FIGURE 3—Contour map of a section of chymotrypsin (courtesy D.M. Blow)

mechanical devices are reduced. Secondly, it becomes possible to monitor the data collection process in a "closed-loop" mode, and correct for minor errors as they arise.

The control of diffractometer setting and output of basic measurements can be carried out on quite a small computer. If it is larger—somewhere in our "medium" range—then it is possible to do some of the preliminary processing of data, and add the monitoring function. The final assembly of complete data, involving scaling of measurements from a number of crystals and determination of phases, must in any case be carried out on a large off-line machine.

Presentation of results

Before turning to the other method of data collection, X-ray film, we must digress to the far end of our subject, namely the presentation of results. After the measurements have been assembled in a large computer, a Fourier transformation is applied to them, the result being a three-dimensional map of electron density in the molecule. This map is in the form of spot values of density on a regular grid. Traditionally, this was listed on the output printer, section by section, and contoured by hand; if the crystal axes were not orthogonal, then the numbers had first to be transferred to suitably laid out paper, also by hand. In all, the process took about a day and a half per section—two to three months for a complete molecule.

When we purchased our on-line computer, for diffractometer control, it was decided to take advantage of its time-sharing capability by attaching a cathode ray tube (c.r.t.) plotter, and automate the contouring process. This uses a high resolution c.r.t., with a flat face suitable for photography; we have two cameras—35 mm. for sequences of sections, or 4″ × 5″ taking Polaroid film for singles. There is also a large slave c.r.t. for visual inspection. We accept the density map on a regular grid as before, but on magnetic tape, the on-line computer interpolating bilinearly between each set of four spot values. From the resulting continuous function, we either produce conventional contour maps (Figure 3) or generate direct representations of density by a stippling technique based on Monte Carlo methods; both these are described in more detail elsewhere.[1] The latter form has certain attractions for presentation, but most scientists prefer to work from contour maps in view of

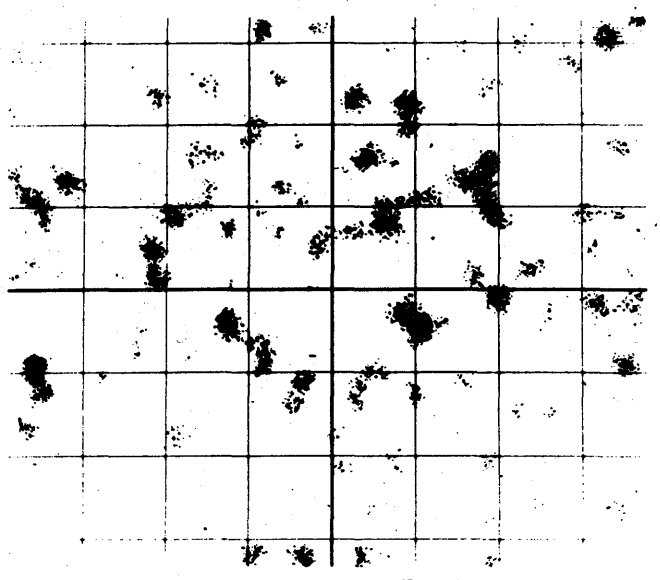


FIGURE 4—Density map corresponding to Figure 3

their more quantitative nature. Each section can be drawn in about five minutes. Enlargements are subsequently made on transparent sheets, which are stacked to give the three dimensional picture.

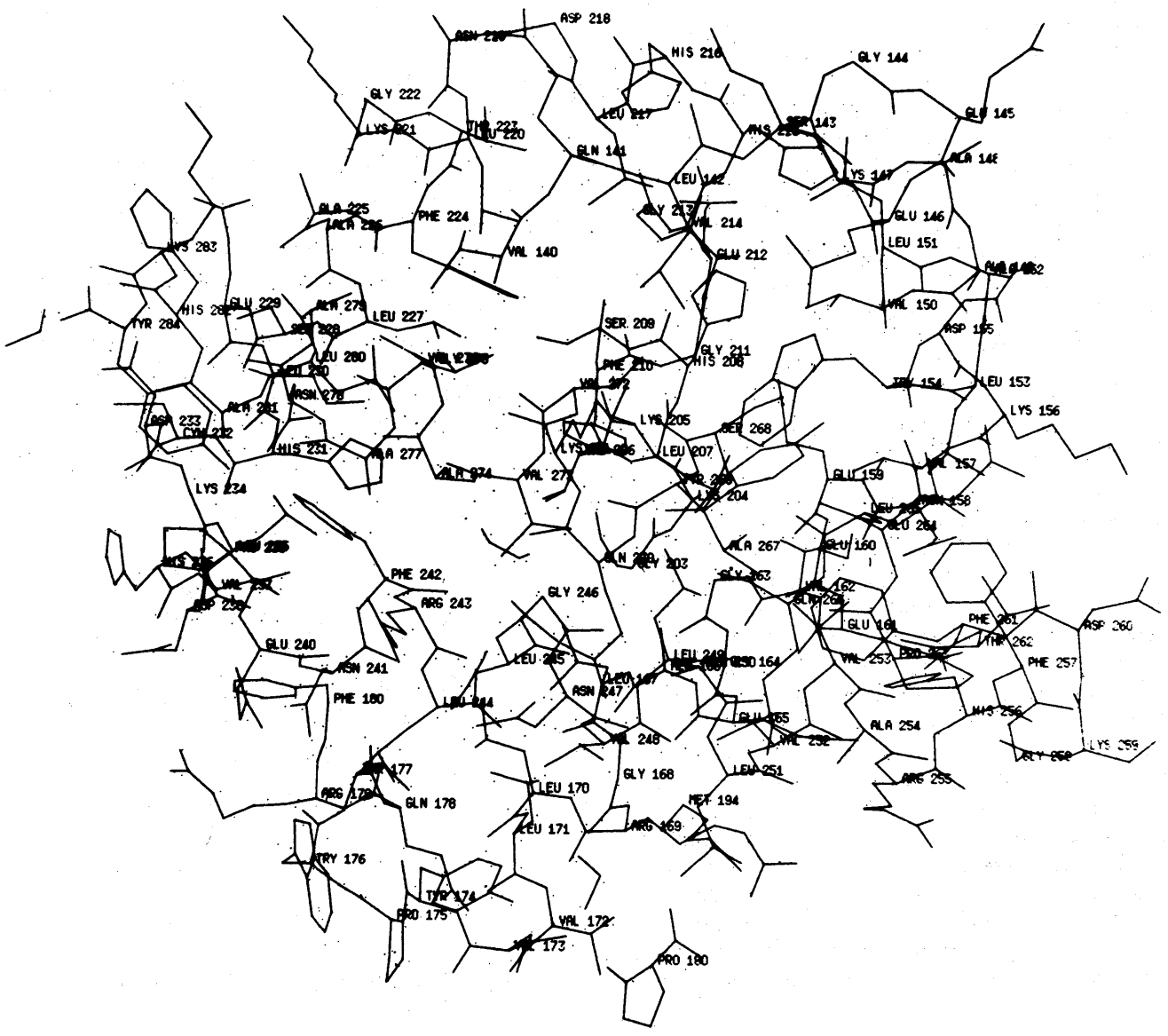
The next stage in the process is to fit the chemical sequence to the density map. It is probably fair to say that the computer for this job has yet to be built (and the programmer yet to be found), so that for the the time being it remains a matter of human intuition. However, a computer can be used in the later stages of structure refinement, and the final structure can be put on to magnetic tape for display as a "wire model of the atomic bonds (Figure 5). We have a set of handswitches on-line to the computer, which are used to rotate the model, expand or contract it, or select limited parts of it for display; this technique is similar to that used by Leventhal,[2] although we cannot achieve his speed and continuity of display. The handswitches are also used by other programs, their interpretation being indicated by a slip-over card.

Microdensitometry

We can now return to the other method of crystal data collection—the measurement of X-ray reflections on film. The advantage of a diffractometer is that it measures X-rays as accurately as possible, i.e., to one quantum. Film is not in practice as efficient as this, but on the other hand many reflections are produced simultaneously for one setting of the crystal, and these can be collected on different parts of the film.

It is customary to use for this purpose a device called a "precession camera." This is essentially a mechanical analog computer, which continuously rotates the crys-

FIGURE 5—Computed model of part of haemoglobin (courtesy M.F. Perutz)



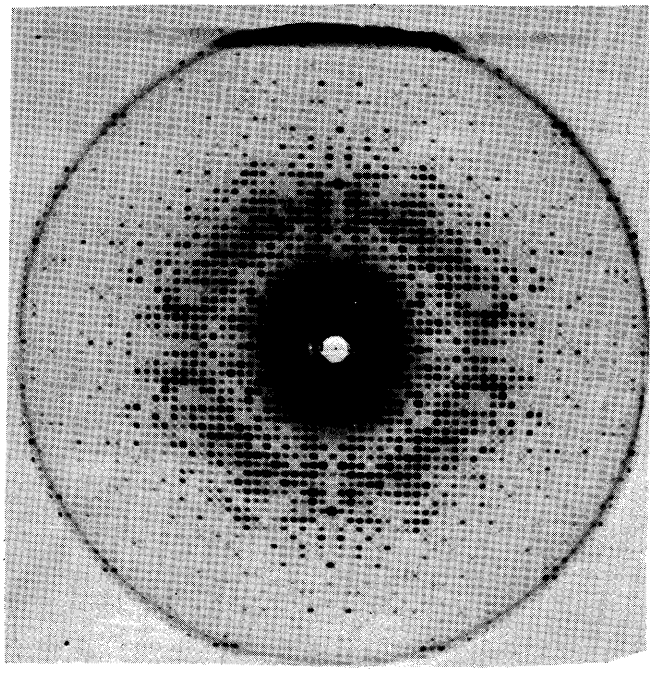


FIGURE 6—A typical precession photograph (courtesy D.M. Blow)

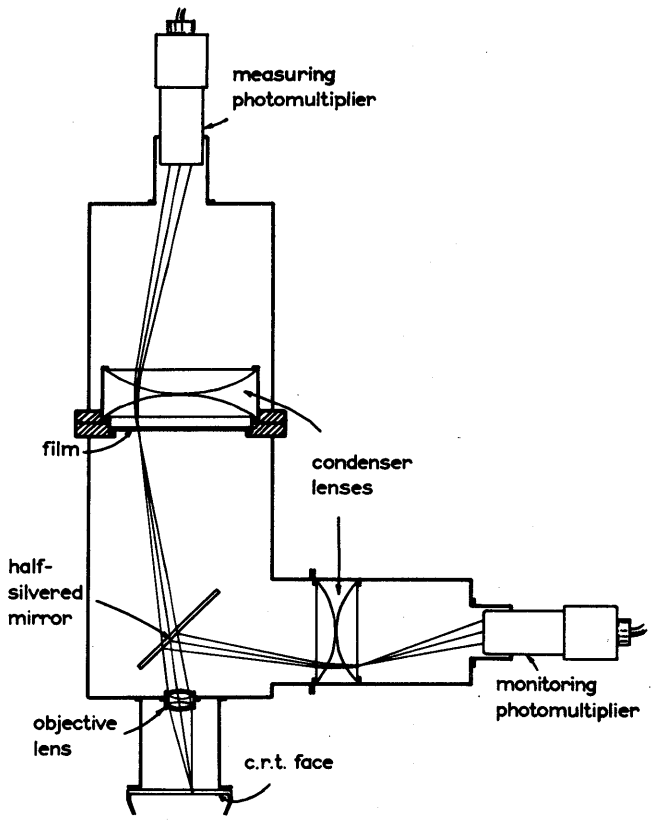


FIGURE 7—Sectional view of the c.r.t. microdensitometer

tal (and also the film) so that most of the possible reflections are produced. Many of these are deliberately discarded by a lead "layer-line" screen, in order that the remainder shall appear as a regular array of spots on the film (Figure 6). The densities of the spots are then measured by a semi-automatic method and punched by hand on to cards. From the film shown in Figure 6, there are about 2,000 spots, and to measure them and punch and verify the data would take about two days.

After we had been using the c.r.t. plotter for some months, it occured to us to try using it as a microdensitometer (Figure 7). The spot of light from the c.r.t. face is imaged on to the film, and the transmitted light, I_t., is passed through a condenser lens system to a photomultiplier tube. To allow for variations in the c.r.t. phosphor, a half-silvered mirror is used to deflect part of the incident light, I_i, to a second monitoring photomultiplier. The optical density of the film. D, which is proportional to the X-rays received, is given by

$$D = \log(I_i/I_t)$$

This equation is simulated by a special analog-to digitial converter,[3] which measures the time taken to discharge a capacitor between voltages proportional to I_i and I_t. The automatic logarithm calculation in the converter saves a considerable amount of computer time.

Since it is only necessary to look at those parts of the film where the spots are know to lie, and the c.r.t. is a random-access device, there is no need to fill the store of the computer with an image of the complete film. This is particularly important in the case of a relatively small computer, where it would be quite impossible to hold an adequate image in the store. The basic precision of the measuring system is 8 bits, i.e., 256 grey levels, but limitations of the film reduce the overall accuracy to about 3% in D, which is still slightly better than can be achieved by conventional methods. We can measure a film such as Figure 6 in five to fifteen minutes.

The output of some ten precession cameras is now regularly processed by this method, which represents a considerable breakthrough in speed and convenience. For example, we can now obtain a contoured Patterson map, which requires a few thousand reflections and no phase information, within hours of the developed films leaving the drying cabinet; formerly, this could be expected to take at least a fortnight, and often several months. In consequence, the value of a protein derivative can be assessed soon enough to modify the chemistry immediately if it is unsatisfactory.

Looking into the future, the next step is to discard the layer-line screen from the camera. If this is done, all possible are collected as they appear, although not necessarily on one film; an example of the resulting "oscilla-

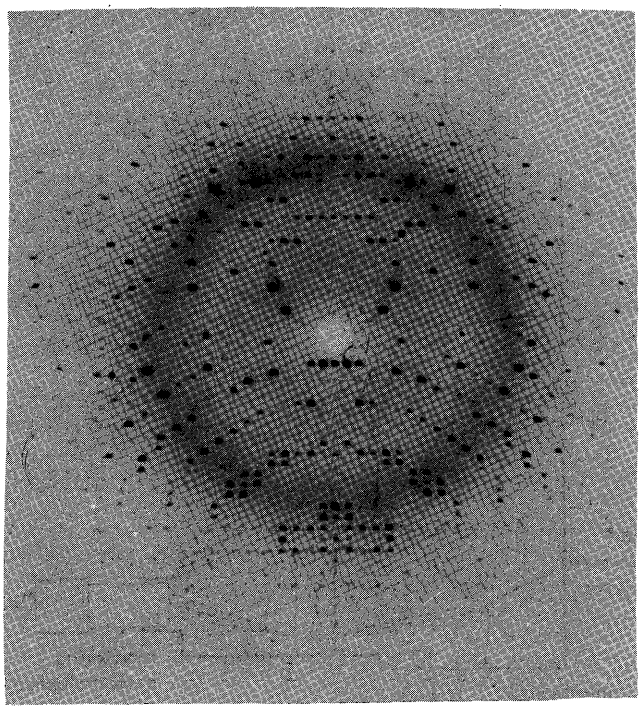


FIGURE 8—An oscillation photograph (courtesy R. P. Phizackerley)

tion photograph" is shown in Figure 8. The positions of the spots are no longer so regular, but they are still computable. The importance of this kind of photograph is that, since no reflections are discarded, the total exposure of a crystal to X-rays for a given number of reflections is reduced. Consequently, the number of crystals is similarly reduced, and the problems of scaling data between different crystals are made less severe.

The interest shown in the microdensitometer in the laboratory has not been restricted to protein crystallography. We have been making measurements of photographs of other kinds—including viruses, of which we shall say more later. Unfortunately, although the c.r.t. is attractive in terms of speed, and its resolution is better than one part in 2,048, it suffers from two drawbacks: its spatial linearity is not good, and the nature of the optical system causes the range of measurable density to be limited by light scattering. Consequently, we have found it necessary to build a second microdensitometer (also computer controlled), in which the film is mechanically translated between a fixed conventional light source and the photomultiplier. This is inevitably a slower machine, but its advantages of accuracy and positional stability are overwhelming in some cases.

Chemistry

We should now turn briefly to the chemical side of protein structure. Having chemically broken down a protein, or part of one, into its constituent amino-acids, the proportions of these can be determined by an amino-acid analyser. The output of the analyser is normally in the form of a strip chart, of which Figure 9 is an example. The amino-acids show up as peaks in the chart. The area under each peak was until recently calculated

FIGURE 9—A strip chart from an amino-acid analyser

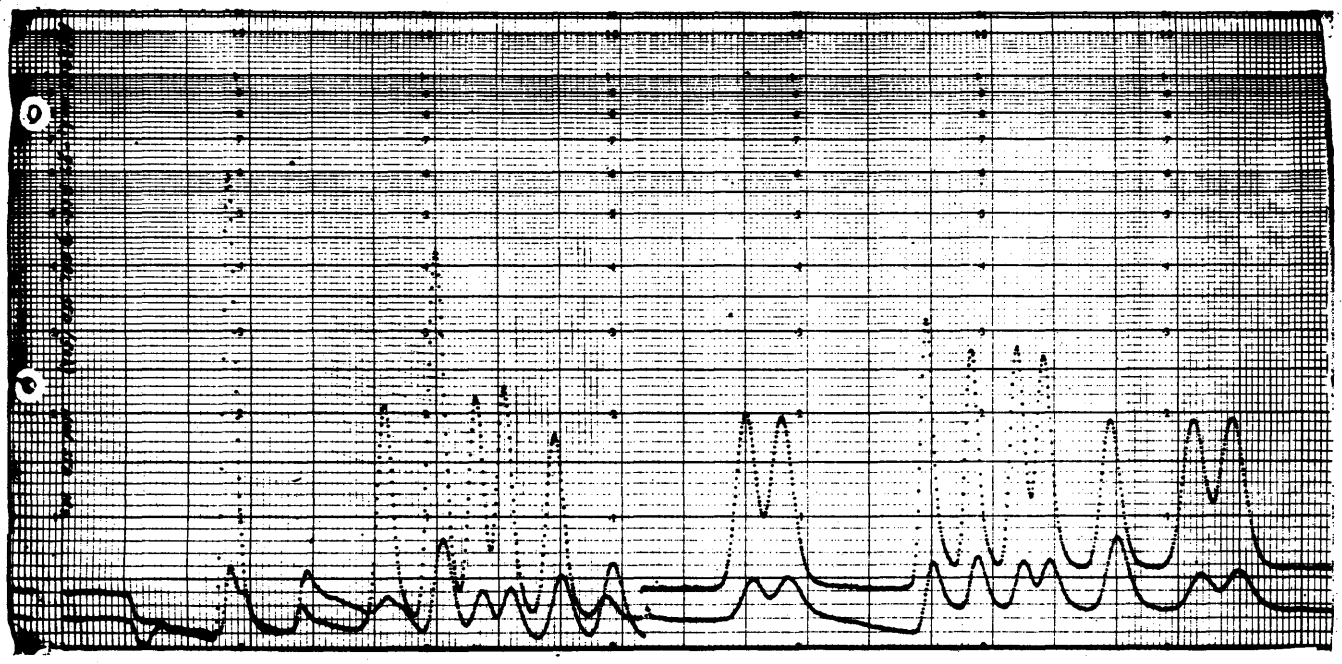


FIGURE 10—An electron micrograph of virus particles (courtesy J. T. Finch)

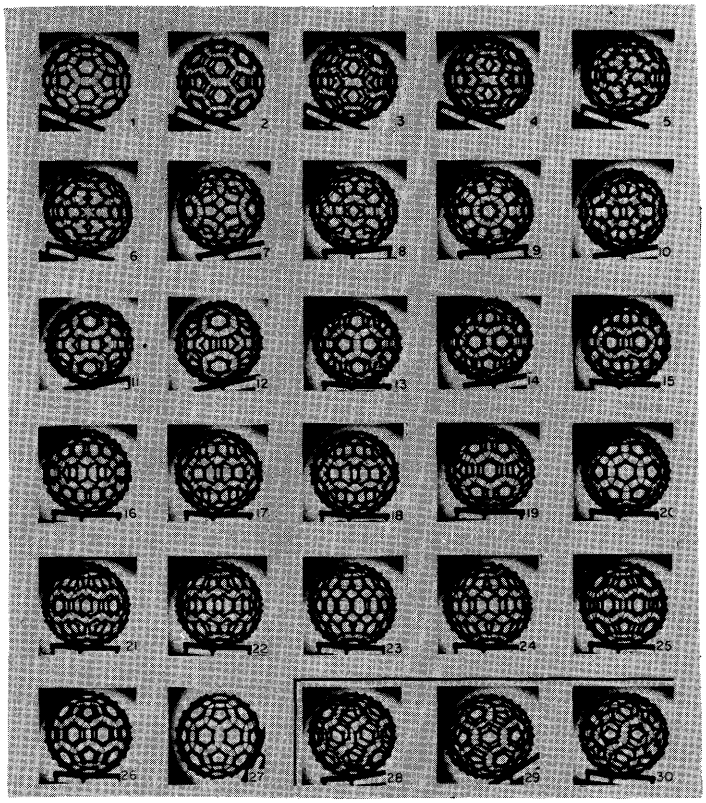


FIGURE 11—Several views of a physical model of a spherical virus (courtesty J. T. Finch & A. Klug)

manually, the most intuitive part of the operation being the determination of the base line.

The readings are in fact taken from an optical cell, for which the same law for density applies as for the microdensitometer. We have therefore attached a similar digitiser to a second slidewire in the chart recorder, and fed the output, together with control signals to show when the servo-mechanism has balanced, to the computer. We have developed a program to evaluate peak areas, and the base lines that it assumes correspond satisfactorily with those assigned by human operators.

Virus structure

When the on-line computer arrived in 1965, the plotter (and subsequently the microdensitometer) aroused considerable interest amongst other workers in the laboratory who were not concerned with proteins. In particular, it soon became clear that it could be helpful in the study of viruses. A virus is a much larger entity than a protein molecule. Whilst some viruses can be persuaded to crystallize, the data collection problem in X-ray crystallography increases by several orders of magnitude, and it may be some time before this method can be used with a reasonable chance of success. More manageable information (though not in such fine detail) can be found in electron micrographs (Figure 10).

In an electron micrograph, individual viruses can be easily seen. However, because of the staining techniques used, their structure is not immediately obvious—the pattern seen is a projection of the complete particle. Given this limited information, there are two approaches to the three-dimensional solution. One can either build a model and compare projections of it in different orientations to the micrographs, or measure the micrographs and attempt, by various mathematical techniques, to deduce the structure directly. We have used both methods, the second being relatively new, since it was only made possible by the construction of the second microdensitometer.

The viruses that are studied in Cambridge come in two basic forms: spherical and helical. In both cases, each particle consists of a number of linked identical protein "subunits." In spherical viruses, there are a multiple of 60 subunits, having overall icosahedral symmetry. The traditional approach was to build a mechanical model and prepare a "gallery" of views in silhouette (Figure 11), for comparison with the micrographs. As can be seen, quite small changes in attitude can make drastic changes to the appearance, so that a gallery of some 150 views is needed. A change in the subunit must be repeated 60 times (or more), followed by completely re-photographing the model.

This process has now been replaced by feeding into the computer a hypothetical model of the subunit,

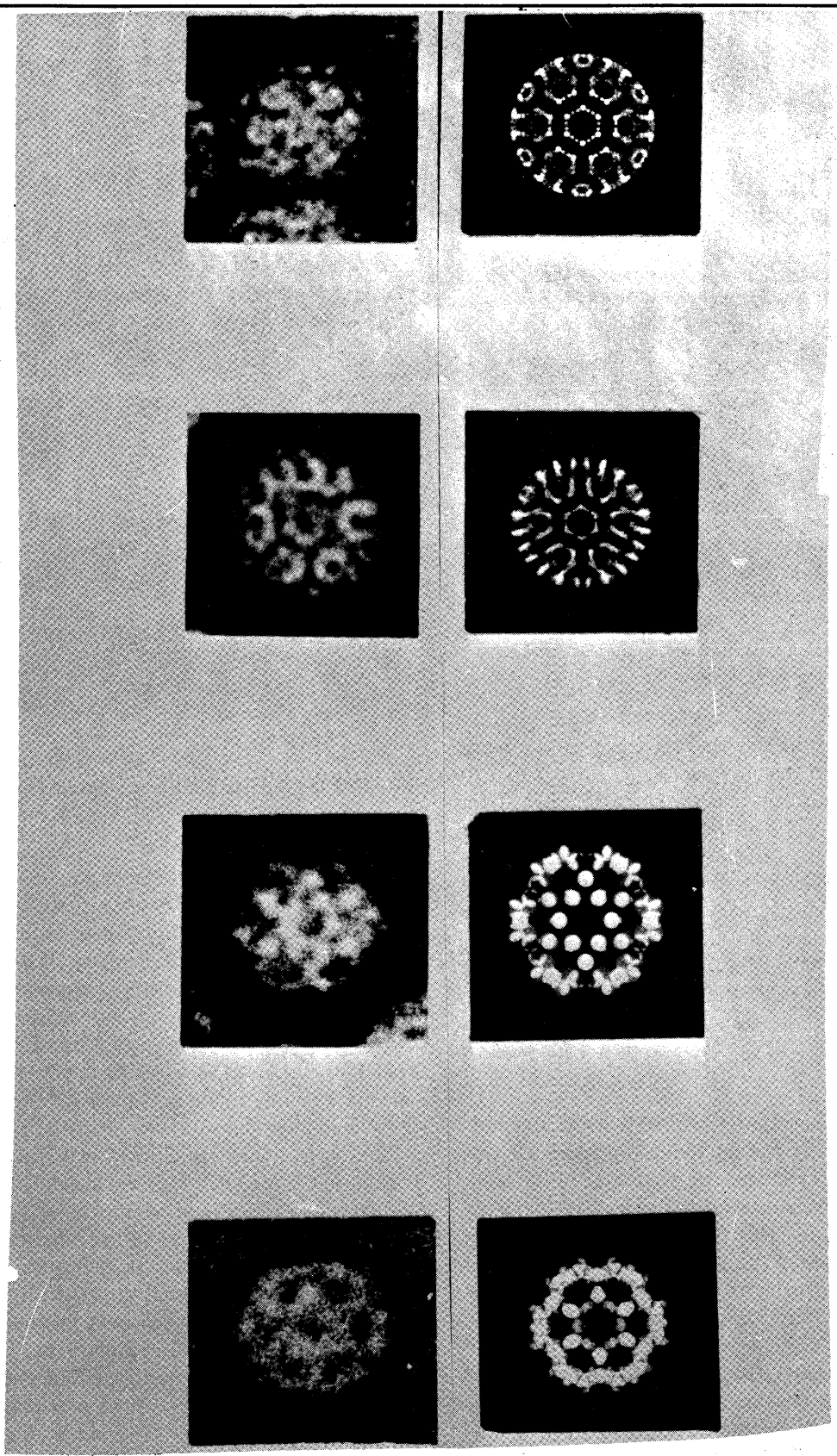


FIGURE 12—Electron micrographs of spherical viruses (courtesy J. T. Finch & A. Klug)

FIGURE 13—Computer-drawn models corresponding to Figure 12

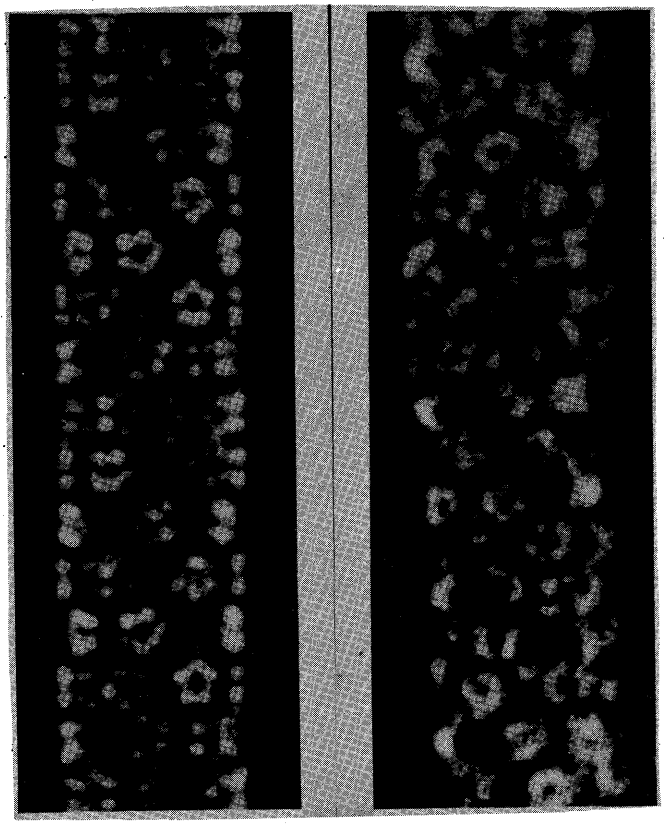


FIGURE 14—Computed model of a helical virus (courtesy N. A. Kiselev)

FIGURE 15—Part of an electron micrograph corresponding to Figure 14

which is repeated by the program according to the symmetry. Projections are then drawn in different orientations, each taking about five minutes. Changes in the hypothetical model are now made by altering the data to the program, and in this way galleries for a wide range of possible subunits have been produced. Figure 13 shows some typical views, which may be compared to the micrographs in Figure 12.

In helical viruses, the subunits are repeated along a double helical lattice to form a cylindrical tube. A similar program to that for spherical viruses produces pictures such as Figure 14, which may again be compared to the micrograph in Figure 15.

Finally, since the microdensitometer came into use, work has been put in hand on measuring micrographs directly. The mathematics for deducing the structure is very complicated, particularly for spherical viruses, and has to be carried out off-line on the IBM 360. The on-line computer is at present restricted to putting the measurements on to magnetic tape; ultimately, as structures are solved, it will again be used for plotting results.

Operating experience

At this point we shall leave the technical side of laboratory automation, and turn to the human aspect. We find that the problems of human interaction with a computer system are just as difficult as the engineering details in a research environment—perhaps more so. Research scientists are very independent individuals, and tend to demand that their own experiments shall at least appear to be independent of those of others. This places the operator of a shared computer in a difficult position.

To start with, one operating console is not enough. We have found it necessary to provide at least a bank of handswitches for each on-line device (such as the plotter), and in most cases an output printer as well.

Secondly, scientists usually like to write their own programs. This is fine so long as they can use a high-level language, such as Fortran or Algol. Unfortunately, compilers for these take up an excessive amount of space in the store (if only temporarily), and the object programs they produce are seldom efficient. Also, they offer only limited facilities for input and output to on-line devices and for dealing with specialized timing requirements. We have had to limit ourselves, therefore, to programs written in the internal language of the computer, which in this case is fortunately fairly easy to learn.

Lastly, there is the problem of developing new equipment. We have been lucky in that there have been very few engineering difficulties in attaching new devices to the computer. However, it still takes time to test out anything new, and to avoid interference to (or from) other users, we usually have to take the computer "off the air" for short periods from time to time. (We sometimes also have to do this for fault servicing.) This makes us very unpopular with anyone whose experiment is running at the time, and who is liable to feel that his research is being needlessly delayed. (As an example, we have found it necessary to detach the amino-acid analysers from the computer, since they require long uninterrupted runs, and they do not need closed-loop control. Their output is now fed to an incremental magnetic tape deck, for off-line processing.)

We have not yet solved this problem to our satisfaction, and we can well understand the attractions of the "one small computer per job" approach. Nevertheless, the cost of a large number of small machines is usually higher than for one equivalent larger one, and larger machines are easier to program. The choice is not easily made, and depends on what is available in the computer market at any time. This market is undergoing rapid changes at present, and it will be interesting to see what future trends emerge.

REFERENCES

1 T H GOSSLING
Acta Crystallographica 22 465 1967
2 C LEVENTHAL
Scientific American 214 42 1966
3 U W ARNDT R A CROWTHER J F W MALLETT
J Scientific Instruments 1 510 1968
4 J T FINCH A J KLUG
Molecular Biology 10 570 1967

A small computer as an on line multi-parameter analyser for a neutron spectrometer

by S. B. WRIGHT and M. G. SILK

A.E.R.E., Harwell
Berkshire, England

INTRODUCTION

The techniques in current use for the measurement of the neutron energy spectrum above a few kilovolts in the core of a reactor rely on the measurement of the energy of the secondary particles produced as a result of a neutron interaction with a nucleus. The most common reactions are scattering by hydrogen in which the energy of the recoil proton is measured, and the reaction between a neutron and Lithium-6 to produce an alpha particle and a triton which are detected in coincidence. In both these reactions the energy of the reaction products is a linear function of the neutron energy, and it is difficult to cover an energy much greater than a factor of ten in a single measurement with these techniques. This means that at least 4 and often 6 or 7 different measurements are required to cover the energy range from a few keV up to 10 MeV.

A neutron spectrometer which could cover this whole energy range in a single, or at least fewer measurements would be very desirable and such a spectrometer could be achieved by using the angle between the reaction products of the Li^6 (n, α)H^3 reaction as the measure of the neutron energy rather than their total energy. The reaction products of this reaction are emitted at 180° in the centre of mass system. If the incident neutron has zero or near zero kinetic energy this corresponds to the laboratory system. However, if the neutron carries significant energy the centre of mass system is moving relative to the laboratory system, and the angle between the reaction products is less than 180° in the laboratory system. As the reaction is isotropic relative to the direction of motion of the incident neutron the angle between the reaction products is not a unique function of the neutron energy, but the relationship is sufficiently simple for the neutron energy spectrum to be unfolded from the distribution of the angles. For energies below a few MeV the angle of the reaction products is a function of the square root of the incident neutron energy and so a spectrometer based on measurement of this angle should cover a wider range of neutron energies than one based on measurement of the total energy of the reaction.

Spectrometers based on this principle have been proposed by Beets[1] using a number of separate counters to detect particles at different angles. In the spectrometer discussed here the reaction products are detected in 3 position sensitive semi-conductor detectors, and a Honeywell DDP 516 used on line analyses the output of these detectors to give the angle between the reaction products. In this way data which are generated as 9 coincident parameters each of 8 bits are analysed and stored as a single parameter of 6 bits, and the data handling problem becomes manageable.

The spectrometer and the interface to the computer are in manufacture at the time of writing, and delivery is expected during September. The programme is basically written and awaiting the delivery of the computer. We therefore expect to have the first results available by the end of 1968.

The design of the spectrometer

In order to define the angle between the reaction products, the point of interaction and points on the tracks of both of the secondary particles must be determined. This can be accomplished using a set of three position sensitive detectors arranged in a stack as shown in Figure 1. The central detector, being coated with the lithium-6 carbonate target, defines the point of interaction while the outer detectors define the X and Y co-ordinates of the points of incidence of the secondaries. The Z co-ordinates are defined by the diode separation.

The position sensitive detectors used are of the type developed by Owen and Awcock.[2]. and are sensitive in two directions. The sensitive region of the detectors is a square of side 1.35 cms and it is anticipated that the spatial resolution will be better than 1/3 mm. Four

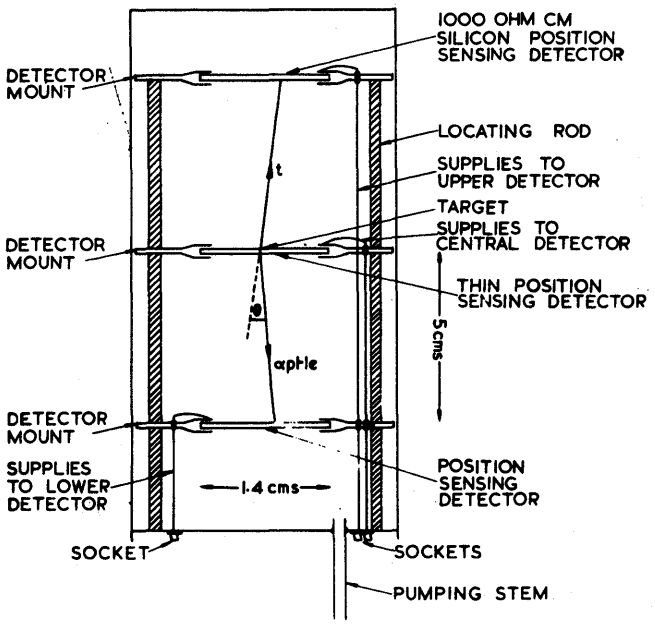


Figure 1—The design of the position sensitive neutron spectrometer.

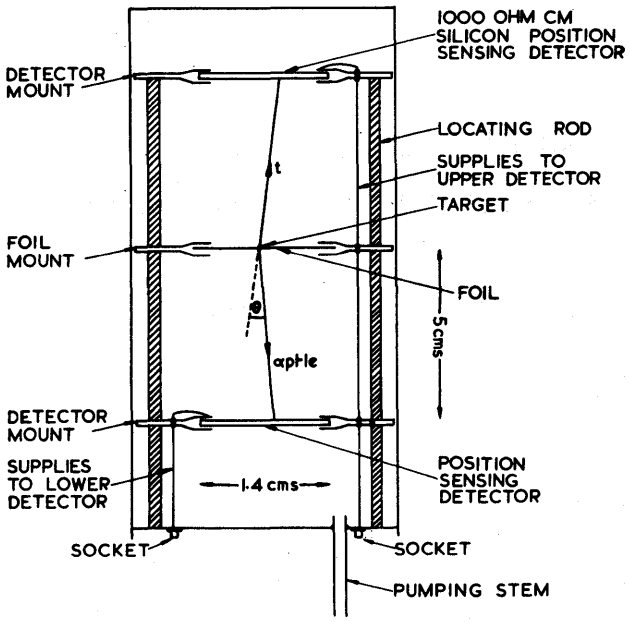


Figure 2—The design of the prototype neutron spectrometer.

pulses may be taken from each detector proportional to EX, EY, E(S-X), E(S-Y) where E is the total energy released in the detector, X and Y are the position co-ordinates and S is the side of the detector (1.35 cms). Adding the pulses corresponding to EX and E(S-X) produces a pulse corresponding to ES. Clearly

$$X = S(V_x/V_s) \quad Y = S(V_x/V_s)$$

Thus to determine the X and Y co-ordinates in each detector, three parameters (V_x, V_{ex} and V_y in this case) need be recorded.

The centre diode of the final spectrometer is a very difficult diode to make satisfactorily. A prototype spectrometer has therefore been designed in which the centre diode is replaced by a small spot of lithium-6 carbonate on a Mylar or similar film[3] (Figure 2). This prototype will require the an alysis of only 6 parameters, but all the electronics and programme design has been done allowing for the 9 parameters of the full spectrometer.

The expected range of angles for this device is 180° to 150° corresponding to a neutron energy range of approximately 10 keV to 10 MeV. This will be covered by 64 equal groups on a logarithmic energy scale.

Data collection

The fast neutron spectrum can be determined to a reasonable accuracy if about 10^5 counts due to fast neutron events are obtained. Thus, to determine the spectrum in a reasonable length of time, (say 25 hours) a count rate of about 1 fast neutron event/second must be obtained.

In thermal reactor systems only about 1 in 1000 of the recorded events will be due to fast neutrons; the remainder being due to the thermal and epithermal flux. These thermal events are indistinguishable from fast neutron events until at least a partial analysis has been carried out so the slowest acceptable storage rate in the recording system is about 1000 events/second. An event is defined by 9 parameters each of which is required to 8 bit accuracy. The storage rate required is thus 10^5 bits/sec. and the total stored data will amount to 7.2 10^9 bits which is well beyond the capacity of conventional multichannel analysers. Clearly the fact that only six parameters are required from the present prototype spectrometer eases the data handling problems but the general argument is not affected. In any case, it would be shortsighted to consider the prototype spectrometer separately from the final design unless the data handling problems were dramatically different.

Tape systems can be used and will handle data at the required rate with little difficulty although some derandomization would be required. However, the capacity of a typical 9 track tape is only about 2.10^8 bits so, even with the closest packing of data (impossible with events of a statistical nature) the data from one run will fill more than 35 full sized tapes. This technique is thus both cumbersome and expensive.

For these reasons, it was considered that the best means of storing the data was to introduce an on line

data processor. This processor will reduce the data storage problems in two ways. Firstly a relatively simple test will eliminate the 'straight line' events due to thermal neutrons and to neutron reactions in the silicon of the detectors. The rapid elimination of thermal neutron events is very important since they are so much more frequent than fast neutron events. Thus the overall count rate is governed by the rate at which thermal events can be processed. Secondly those events not eliminated as straight lines will be analysed and the angle between the reaction products calculated from the recorded parameters. In this way, the quantity of data to be recorded is reduced by a factor of almost 5000. The calculations necessary are briefly described below. It is convenient to define an angle θ as being the supplement of the angle between the reaction products (Figures 1, 2).

The co-ordinates of points on the tracks of the secondaries will be calculated from the appropriate pulse heights. However, as the detector response is non-linear the true values of these co-ordinates must then be calculated from a stored response matrix for each detector.

The elimination of thermal events is then accomplished by a simple test for a straight line for which

$$(X_1 - X_m) + (X_2 - X_m) = 0 \pm \delta$$

and

$$(Y_1 - Y_m) + (Y_2 - Y_m) = 0 \pm \delta$$

where the X and Y co-ordinates observed in the upper, lower and centre detectors are X_1, Y_1; X_2, Y_2; and X_m, Y_m respectively.

The value of δ will take into account the accuracy of the particular diodes used.

The value of θ is then obtained from the equation

$$\sin^2\theta = \frac{(A_1B_2 - B_1A_2)^2 + (A_1C_2 - C_1A_2)^2 + (B_1C_2 - C_1B_2)^2}{(A_1^2 + B_1^2 + C_1^2)(A_2^2 + B_2^2 + C_2^2)}$$

where

$$A_1 = X_1 - X_m \qquad B_2 = Y_2 - Y_m$$
$$A_2 = X_2 - X_m \qquad C_1 = Z_1 - Z_m$$
$$B_1 = Y_1 - Y_m \qquad C_2 = Z_2 - Z_m$$

The values of $\sin^2\theta$ obtained are compared against a table of values designed to correspond to equal intervals on a logarithmic scale of neutron energy.

The choice of computer was dictated by considerations of computing speed and word length. The word length must be great enough to ensure that potential increases in experimental accuracy are not limited by the computer and that rounding errors do not seriously contribute to the experimental errors. A word length of at least 14 bits is thus required and this eliminates 12 bit computers for which relatively slow double length arithmetic is required for part of the calculation. The computer used in the present experiment is the Honeywell DDP–516 which is a 16 bit computer with a core cycle time of 0.96 μsec. The best estimate of the mean total calculation time with this machine is 830 msec. which corresponds to a rate of acceptance of events of 1.25 10^3/sec. It seems likely that the calculation can be streamlined at many points once operational experience with these detectors has been obtained. The more usual and mathematically simpler formula for $\cos^2\theta$ is not used in the calculation since at small values of θ the preservation of the overall accuracy would require parts of the calculation carried out to an accuracy corresponding to a computer word length of 18 bits.

It is estimated that less than 2000 store locations are needed for the program and the storage of the final data, so the size of store is dictated mainly by the response matrices for each detector. Present experience, which is limited to only a few detectors,[2] suggests that detector non-linearity can be allowed for to a high order of accuracy if 16 × 16 matrices are used for each detector. Thus a 4K store should be more than adequate for the present experiment.

The computer system described above has advantages both in cost and convenience over the alternative tape systems. In addition to this, it is more versatile since it can be readily applied to other experimental systems.

Electronics

A block diagram of the electronics is shown in Figure 3. Three signals are taken from each diode giving a total of 9 from the complete spectrometer (6 from the prototype). These signals are amplified and fed to separate gate and integrator circuits which give an output proportional to the integral of the pulse input when a suitable gate pulse is present. This gate pulse is generated by the coincidence circuit which demands a threefold coincidence between pulses from each detector to define a real neutron event. Threefold coincidence is adequate since it is clear that the release of charge in a detector will produce pulses at all three outputs. Cases where one output is not present due to a fault in the electronics are detected by program.

Sample and hold circuits based on a design by Kandiah[4] are used to store the integrated signals and these are then sequentially sampled and fed to a single

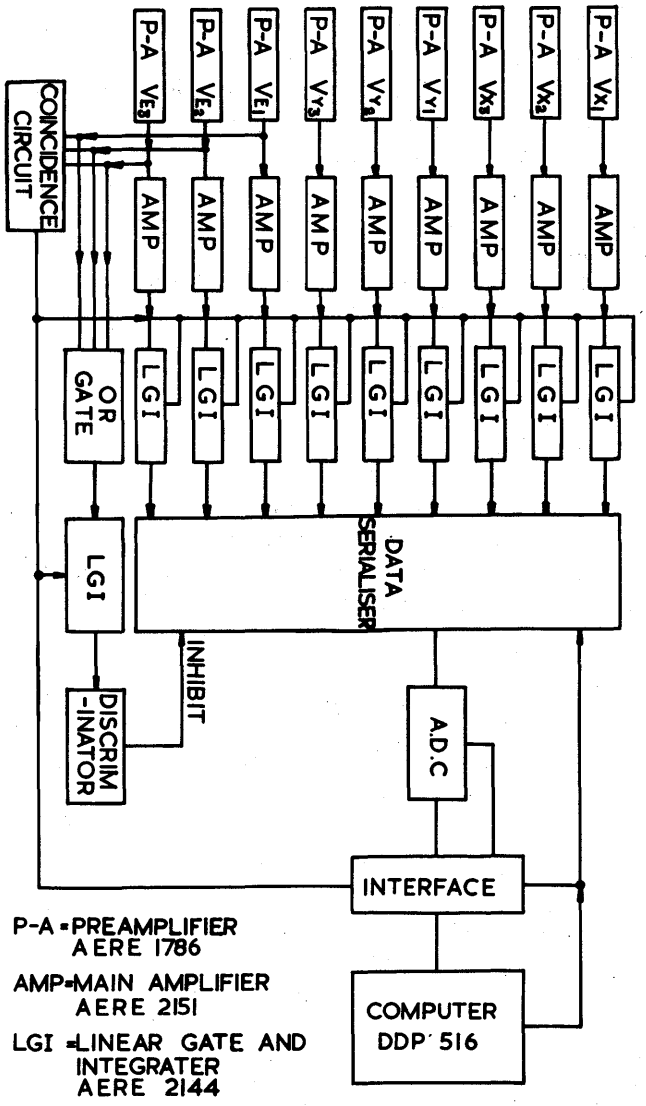


Figure 3—A block diagram of the electronics.

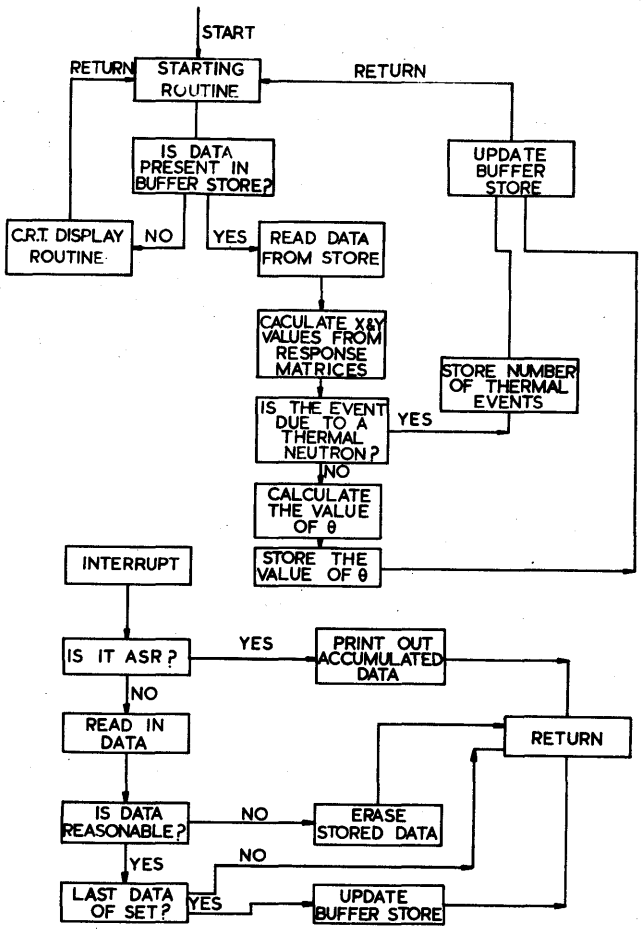


Figure 4—A flow diagram of the program.

ADC. The ADC conversion time in each case will be ~ 70 μsec. maximum giving a total time of 630 μsec. which is well within the estimated computing time.

It is important that the data should be rapidly read into the computer from the interface. This is accomplished by means of the interrupt procedure using the I/O bus. It thus takes about 75 μsec. to read in each parameter which is quite rapid enough for the present experiment and which will not limit any foreseeable improvements in the data handling. The data output is provided by the teletype. A faster output than this is not required in an experiment of this nature in which a comparatively modest output of data is required at intervals of several hours.

Description of program

A flow diagram of the program is shown in Figure 4. The basic steps shown are explained in the earlier sections on data collection and the electronics.

At the start of the program, and at the end of each complete logical sequence, control is returned to the starting routine. A test is made to see if data are present in the buffer store and if so the main sequence of calculations begins. These calculations may be terminated prior to the calculation of the angle if the event is judged to be due to a thermal neutron. In either case the buffer store is updated before control is returned to the starting routine.

A simple display routine is included which provides a display of the data input on an oscilloscope. This may either be a straightforward display of the spectrum of values of θ or may be partially corrected to provide an estimate of the form of the neutron spectrum. The routine comes into operation when no data are present in the buffer store.

Data input is carried out using the interrupt mode of the computer. Because of the sequential operation of the ADC, nine separate interrupts are used to input one complete set of data. The number of interrupts is

counted and at the end of each set the data are transferred into the buffer store. The data are checked as they are read in to remove anomalous sets in which one of the detectors failed to record. If this happens consistently an appropriate message is printed out.

The print out of data is also governed by an interrupt from the teletype. This places the experiment under the control of the operator which is felt to be desirable at this stage. Similar control of the display will be provided.

Future development

The rate of acceptance of data is not as fast as we would like. It is estimated that the total computing time could be reduced, especially if the response of the detectors is predictable enough to enable the elimination of some thermal neutron events before the matrix correction is introduced. This would require a corresponding improvement in the total ADC conversion time, which could be accomplished either by the use of faster ADC's or by the use of two or more ADC's in parallel.

In the present system standard preamplifiers are used but, with nine signal channels, it is clearly impossible to mount the preamplifiers close to the detector inside a practical reactor system. The extra cable runs required may cause a deterioration in the signal to noise ratio and thus in the positional resolution of the detectors. In this case, it may prove desirable to mount the preamplifiers onto the base of the spectrometer. The use of integrated circuits makes this a practical proposition and it is not anticipated that the overall length of the spectrometer will be seriously increased.

It has been shown[2] that the calculation of X and Y for each detector may be carried out using an analogue system. This system reduces the number of signals taken to the computer from each detector from three to two and therefore enables the rate of data collection to be improved. An even more useful consequence, if the reproduceability of the detectors permits, would be the possibility of removing many thermal events before they are loaded into the computer. This would bring about a very significant improvement in the speed and efficiency of the system.

SUMMARY

A neutron spectrometer based on the measurement of the angle between the reaction products of the $Li^6(n,\alpha)H^3$ reaction offers prospects of significant advantages over methods of neutron spectrometry in current use. However, the data handling problem is such that using conventional multichannel techniques the spectrometer would be completely uneconomic. The use of an on-line computer to reduce the data from 9 coincident parameters to a single parameter while the experiment is in progress reduces this data handling to such an extent that the experiment becomes straightforward. In addition to reducing the data to a single parameter so that multichannel analysis is possible the computer allows a crude analysis to be undertaken during the course of the experiment so that a display of the data in the form of a neutron spectrum rather than a spectrum of angles is possible. Thus the data are available to the experimentalist during his experiment in a form which allows him to judge the progress of the experiment.

The use of an on line computer has therefore made economic an experiment which would probably never be attempted using other forms of data handling equipment, and in addition has made it possible to provide adequate data to the experimentalist for him to have full control of the experiment at all times.

ACKNOWLEDGMENTS

The authors would like to acknowledge the assistance of Mr. M. Awcock and other members of the counters group of the Electronics Division who are undertaking the construction of the spectrometer. We would also like to thank Mr. K. Kandiah, Mr. F. D. Pryce and other members of the Electronics and Applied Physics Division for their assistance in the design and construction of the eletronic system.

REFERENCES

1 C BEETS S de LEEUW G deLEEUW-GIERTS
Proc of a Conference on Radiation Measurements in Nuclear Power Berkeley 1966
2 R B OWEN M L AWCOCK
AERE R 5393 1967
3 M G SILK
AERE–M 1850 1967
4 K KANDIAH
Private communication

Applications of digital computers to the long term measurement of blood pressure and the management of patients in intensive care situations

by JOHN L. CORBETT*

University of Oxford
Oxford, England

INTRODUCTION

Blood pressure is one of the most frequently measured variables in clinical practice[1] and is one of the most important in influencing treatment.[2] Measurements are normally made with an indirect method and since the first such techniques were described[3-5] progressive development has been made towards their automation.[6-18] Even in patients with normal blood pressures, however, these methods have been found to be inaccurate[19-23] and they are most liable to error with extremes of pressure—when high in severe hypertension or low in shock—and when peripheral vasoconstriction is present. It is in precisely these situations when it is often of critical importance to know the blood pressure, and at such times the direct measurement of intra-arterial pressure is the only reliable method. This approach is now used in many intensive care units and is likely to be implemented progressively more as the techniques are further improved and the usefulness of the results is more widely appreciated. The volume of data is potentially very large and there is the possibility of many derived results. Digital computers have been previously applied to the analyses involved,[24-28] the general approach being to compress the results of intermittent measurements made in real time on an arterial recording to a form essentially similar to standard nursing charts although considerably more comprehensive, and to leave interpretation to the attending physician. The present paper describes a method based on a cumulative sum technique for demonstrating trends in the mean level and variability of the recorded parameter and its derivatives and for assessing the significance of trends detected. It has been applied to subsequent analysis at high speed of recordings made on a multi-channel F.M. tape recorder, but the basic programme could also be used for on-line analysis and the general approach is suitable for the analysis of heart rate, central venous or other pressures, some respiratory variables, and temperature.

Outline of system

Acquisition of data (Figure 1)

Studies have been based on continuous intra-arterial recordings made on over 50 patients for periods ranging from 2 hrs to 13 days. In these recordings a teflon cannula has been inserted percutaneously into either the axillary, brachial, or femoral artery or a teflon catheter has been introduced into the artery with the Seldinger[29] guidewire technique, and connected to a strain gauge pressure manometer via a device for continuous perfusion and serial damping. The system for perfusion and damping has been developed in Oxford in the Electro-Medical Research Unit of the Medical Research Council for short and medium term measurements and has been fully evaluated in this use.[30-32] Subsequent modifications have been made to enable long-term measurements in intensive care situations to be made more effectively.[33]

*British Medical Association Research Fellow, and Medical Research Fellow of St. Peter's College, Oxford.

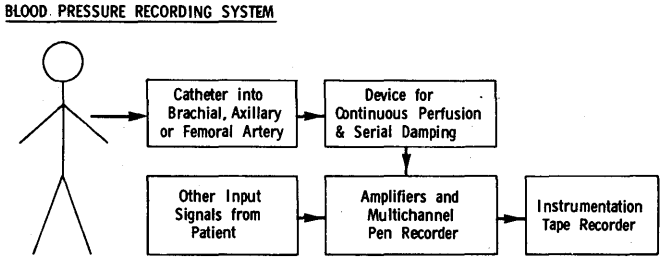


FIGURE 1—Blood pressure recording system

The overall catheter system is critically damped and has a static and dynamic accuracy of about 1% and a frequency response of 30 - 40 Hz, these being considered adequate.[32,34-38] Analogue recordings are made simultaneously on a standard multichannel penwriter and on a high class 14-track multi-channel F.M. instrumentation tape recorder (Sangamo) with excellent stability and linearity and long playing times (up to 16 inch reels of 1 inch tape can be accommodated). On-line computing facilities are not available and the tape recorder is therefore taken to a hybrid computing laboratory for subsequent processing of the records. This is done on a beat-to-beat basis at 16 to 32 times the original recording speed It should be pointed out that this is in effect an on-line system, since the signal from the tape recorder is a close reproduction of that from the patient, although the time base has been compressed. Although it is possible to deliver to the tape recorder an analogue of the blood pressure signal with a 1% accuracy, this is a very demanding accuracy level, and in long term recordings drift checks are necessary. In practice, an accuracy of about 1.5 to 2% would be more common, although attention to detail is required to achieve even this figure. A high class tape recorder can be expected to worsen linearity by about 0.5% and increase noise slightly (by about 1% at low speeds of operation).

Computing hardware (Figure 2)

The hybrid computing facility employed in this study consists of a standard hybrid computer, type TR48, made by Electronic Associates Ltd., in the United Kingdom, together with a sampling unit, analogue-to-digital converter, an interface and digital clock, and an I.B.M. 1130 digital computer with an eight thousand word memory and disc store of four hundred and eighty thousand words, and a punched card and paper tape output. An English Electric KDF9 computer was also available and some of the results derived with the I.B.M. 1130 were transmitted to the KDF9 on paper tape, since the programme used for cumulative sum analysis was available only in KDF9 autocode. The hardware is illustrated in Figure 2 with input indicated from a tape recorder or patient and a display oscilloscope incorporated, and the overall system from patient to computer is shown in Figure 3. The maximum sampling rate attainable with the interface used was approximately one thousand per second and the analogue signal could be quantized to one part in 255 or one part in 2055.

Method of analysis

Normal blood pressure waveforms: General considerations for programming

In this approach analysis of systolic, diastolic, mean and pulse pressures, and heart rate (Figure 4) has been based entirely on the recording of intra-arterial pressure and it is therefore neces-

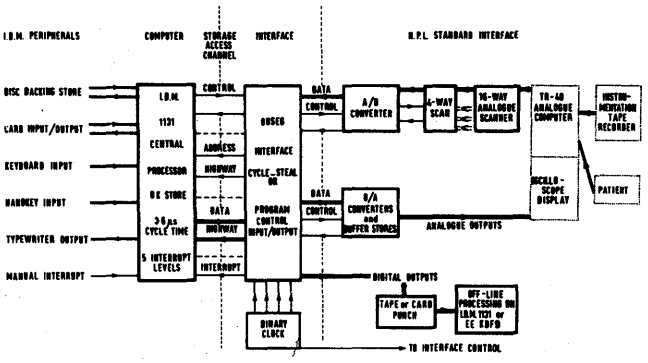


FIGURE 2—Hybrid computing facility

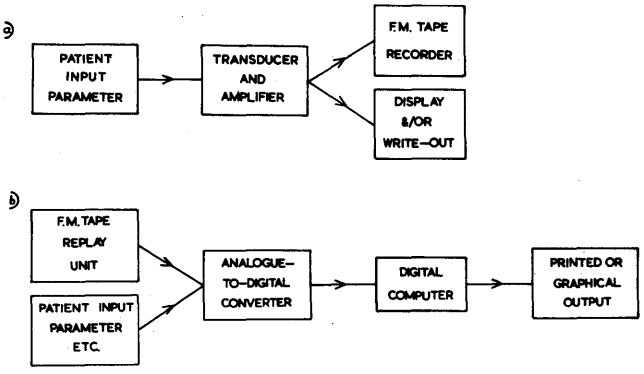


FIGURE 3—Overall system for computing of blood pressure recordings

sary to appraise the physiological and physical characteristics of the signal. As illustrated, the peak of the wave represents the highest pressure in the arterial tree as the heart contracts and is called the systolic blood pressure. The heart beats rhythmically, and during its relaxation phase, pressure is maintained by the elastic recoil of vessels, with back flow of blood being prevented by valves. The lowest pressure reached at this stage is called the diastolic pressure. Pulse pressure is the difference between these two values, and mean pressure is very commonly taken as the diastolic pressure plus one-third of the pulse pressure, on the assumption that the waveform is approximately triangular, although as the figure indicates, the average of the area under the curve is a more correct measure of it. The mean square value or the mean square about the mean may also be an important derivative. Heart rate is simply the reciprocal of the period, and when computed from beat-to-beat is usually referred to "instantaneous."

An aspect of the waveform which is important in digital computing is its amplitude/frequency spectrum (Figure 5) the illustration being taken from work by Patel.[38] Ninety-five per cent of the shape and amplitude of the wave comes from harmonics up to nine times the basic frequency, which varies roughly from 1 to 3 Hz, corresponding to a heart beat of 60 to 180 per minute. For accurate analysis, sampling should therefore be based ideally on a frequency of approximately 30 Hz, requiring a sampling rate of 60 per second or above to avoid aliasing.[39,40] In practice it has been found possible to use a sampling rate of 40 per second with little loss of accuracy since the contribution of frequencies above 20 Hz is usually small. This is equivalent to a sampling rate of approximately 600 per second when the time base of the record is compressed by a factor of 16. At this sampling rate an 8-bit word is adequate to keep quantization errors insignificant.[40]

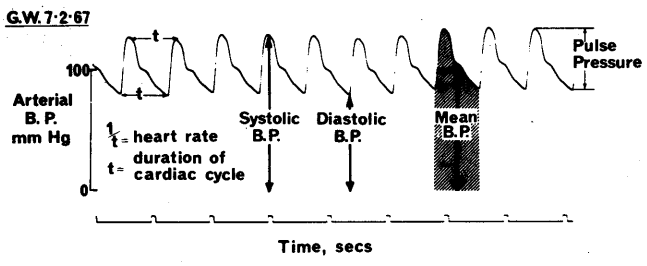


FIGURE 4—Blood pressure record—description of terms

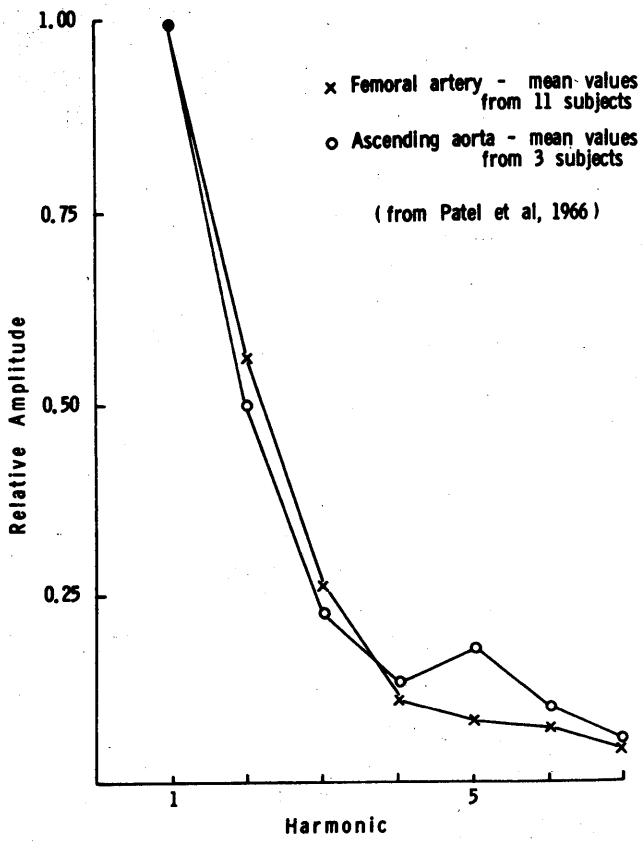


FIGURE 5—Amplitude-frequency spectrum of blood pressure waveform (From Patel, 1965)

"Problem" waveforms

While the classical shape of the blood pressure waveform is that illustrated in Figure 4, other shapes are frequently encountered. A sample of such shapes is shown in Figure 6, the records being taken from 6 different patients. All of the patterns shown are accurate recordings made with the same critically damp system as described above and the differences are therefore not due to artefacts, although their explanation lies outside the purpose of this paper. Their importance lies in the fact that short periods of the overall wave during which a sign reversal occurs must not be recognized by the programme as new heart beats. Difficulties arise particularly when detailed predictions based on the exact shape of the waveform are to be made, especially since the pattern may not be constant even in one patient. This is demonstrated in the records shown in Figure 7 where the shape, DC level, pulse amplitude and rate of the blood pressure wave all change rapidly in response to a voluntary temporary increase in the pressure inside the subject's chest.[41,42] During this manoeu-

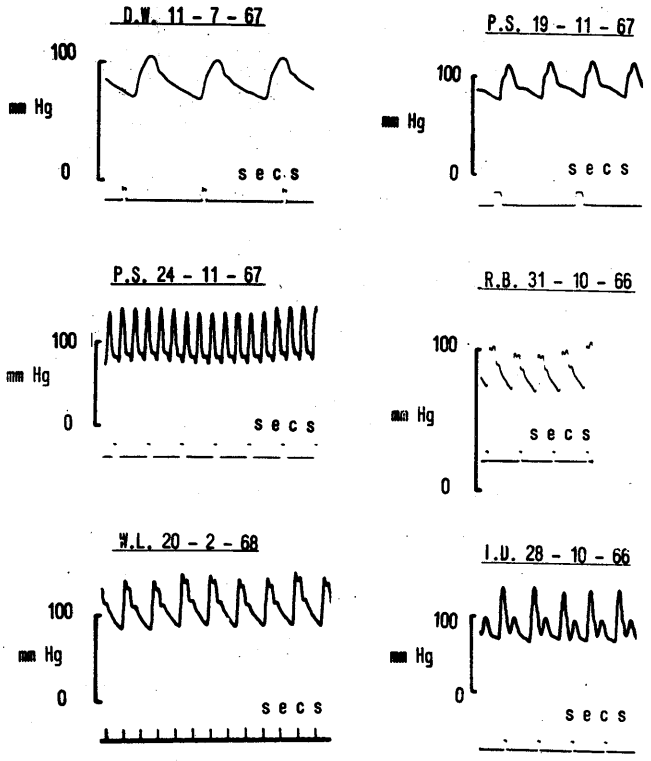


FIGURE 6—Varying blood pressure waveforms

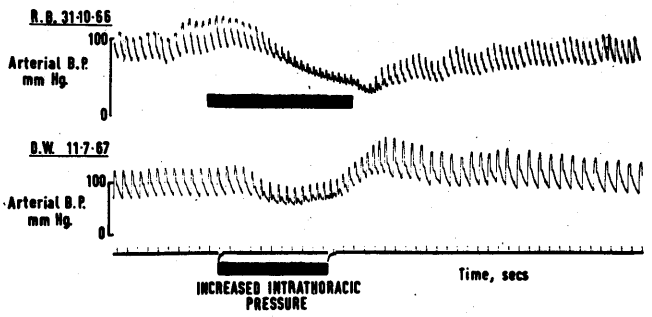


FIGURE 7—The effect on blood pressure and heart rate of a voluntary temporary increase in the pressure within the chest

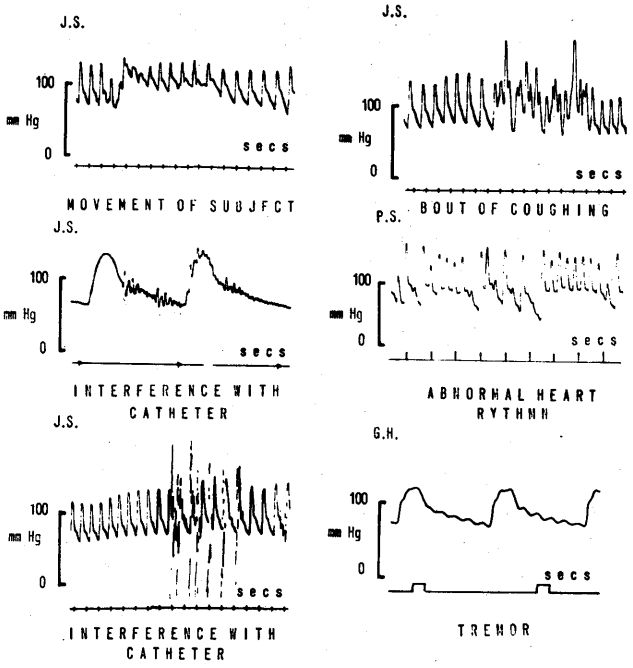


FIGURE 8—Noise in blood pressure recordings

vre, even in normal subjects, there may occur pulse pressure variations between successive heart beats of up to 20% and heart rate variations of up to 45%, while the mean pressure may vary by up to approximately 80% of its original value depending on how hard the subject "blows." Similar but less marked changes in the wave shape also occur with changes of posture and may be compounded by the presence of disease. A further difficulty is the frequent presence of noise (Figure 8) which has been interpreted here in a very wide sense. An example of a tremor in the patient, coughing, movement, interference with the catheter and an abnormal heart rhythm are shown, all of these events being common in clinical situations. It will be apparent that the frequency and amplitude of these events will usually distinguish them from a normal recording.

Programme for analysis of individual waveforms

Detection of the start of contraction of the heart (systole) has been found possible with a method based on the speed and duration of the rise of pressure during this period—probably the most constant feature of the blood pressure waveform. The computer is programmed to recognize this event by searching for a number of monotonically increasing sample values, the number chosen depending on the sampling rate being used. The diastolic pressure is then taken as the immediately preceding minimum and the systolic pressure as the highest subsequent maximum occurring within a specified time. If this time is properly chosen the dicrotic notch, the "notch" seen during the downstroke of the wave in a number of the illustrations in Figures 6 and 7, will be ignored. Once systolic and diastolic pressures have been determined it is a simple matter to calculate the mean and pulse pressures and heart rate. Constraints are built into the programme to detect

high frequency noise and off-scale values, and limits are set for the maximum acceptable percentage variation between successive amplitudes and rates. When an unacceptable value is found the programme will either stop or register a fault count depending on which choice the user has made in a programme "option." Other features of the programme are an automatic calibration search, an initial scan to check that the signal conforms to the user parameters inserted, simple 5 point smoothing to remove high frequency noise, and options for the type of output and the duration of analysis.

The set of user parameters in the present version of this programme is indicated in Table I. The programme has been written in Fortran and also in machine code to increase its speed of operation. It will reliably accept samples at a rate of 600 per second to produce an analysis of the derivatives indicated in Figure 4 with an accuracy within 2%, enabling the speed of replay of the tape recorded signals to be increased by a factor of 16. When the tape speed is increased to 30 times the original and sampling rate is increased to a thousand per second, which is the highest of which the available hardware is capable, the accuracy of analysis decreases by a further 1 to 2.5%, depending on heart rate. It will be apparent that the amount of data will exceed available core store during analysis of long recordings since one track of a 16 inch spool can accept a continuous 20 hr recording at a speed of 1⅞ inches per second. The programme is therefore designed to automatically dump data on the disc store after every 320 heart beats and then return to the analysis. Incoming data is lost during this period.

TABLE I—User parameters for blood pressure analysis programme

1. Sampling rate of Analogue - to - Digital Converter.
2. Values of calibration signals (in mm Hg.).
3. Number of cycles, or period of time, to be analysed.
4. Permissible variation in a) amplitude b) heart rate between successive cycles.
5. Number of monotonically increasing points required to indicate systole.
6. Fall in amplitude after systolic point before beginning search for diastolic point.

At the end of the entire analysis further programmes are instituted for averaging set periods of the data and determining auto- and cross-covariances. Depending on the latter findings and on the nature and duration of the initial recording one or more selected derivatives are output on punched tape and occasionally on cards for cumulative sum or other subsequent analysis. For certain tests an example of which is given later, a "marker" signal is also recorded on the tape during the initial recording. Two types of marker have been employed, viz., a DC signal on an adjacent (unused) track which is then sampled alternately with the blood pressure signal during processing, and an AC signal of fixed amplitude and duration, superimposed on the data track itself. The latter method has been found more efficient for most purposes and the marker signal is detected by a separate loop in the programme. A similar method is used to indicate the end of a record.

Cumulative sum analysis

The beat-to-beat values derived from the foregoing blood pressure analysis constitutes in statistical terms a time series of nonstationary data in which the serial values are highly dependent and in which both the mean and root mean square vary with time.[40] If the degree of auto-covariance is known the initial derived series can be sampled at sufficiently infrequent intervals to convert it into a time series of independent samples. Present experience indicates that the degree of auto-correlation in the data varies considerably at different times in the one patient, and between patients. For the present illustration (Figure 10), serial half-hour-average values of heart rate, derived from a fifteen-day continuous blood pressure recording have been used. The method of time series analysis applied has been developed for another application by Woodward and Goldsmith,[43] and is illustrated in block form in Figure 9. Its purpose is to detect changes in the average level between groups of data in a time series and to determine the point of onset of such changes. The programme causes the computer to read in a time series, to calculate the cumulative sums of the series using the grand mean as a reference value, and to point out the occurrence of significant changes of slope in the cumulative sum chart. These changes can be determined at different probability levels, and the standard deviations of significantly different stages in the series are calculated. The output includes a graph of the

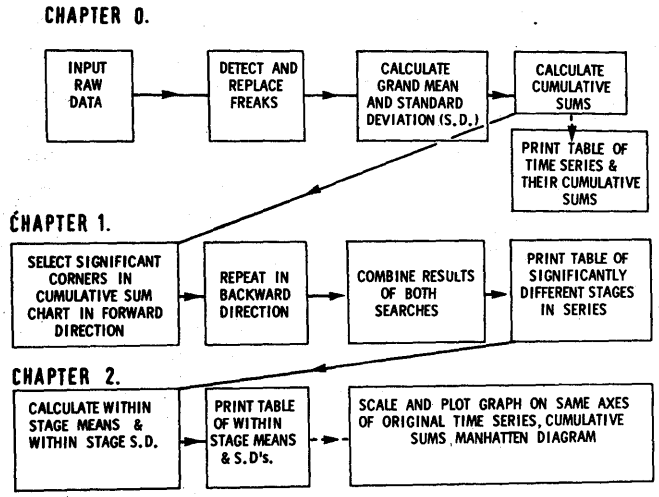


FIGURE 9—Outline of programme for cumulative sum analysis

original data, its cumulative sum or "cusum," and a "Manhatten" diagram, a term used to describe the graph of significantly different stages in the series.

It is obvious from Figure 10 that the initial series (a) is highly irregular and that trends of variation in the mean level are not very obvious. The cusum chart (b) however, gives a very clear indication of the overall trend, and visual inspection confirms the presence of a pattern in the original data which could easily have been overlooked. The Manhatten diagram (c) has condensed the scattered original data into a relatively small number of groups whose difference from their neighbors is significant, in this case at the 1% level. The within-group standard deviations, although available from the analysis, have not been included in this illustration. Differences produced by treatment of the patient with two anaesthetic agents, nitrous oxide and halothane, are clearly shown in both the cusum and Manhattan plots, the mean having decreased significantly. This method has also been used to test objectively the effects of various other treatments (to be published).

A further important application is shown in Figure 11, where an assessment has been made of the variability of the heart rate in the same patient over the same period. The difference between the highest and lowest values of instantaneous heart rate (excluding "freaks" produced by an abnormal rhythm) has been measured for each successive half hour period, and used as a primary measure of variability. These figures have then been proc-

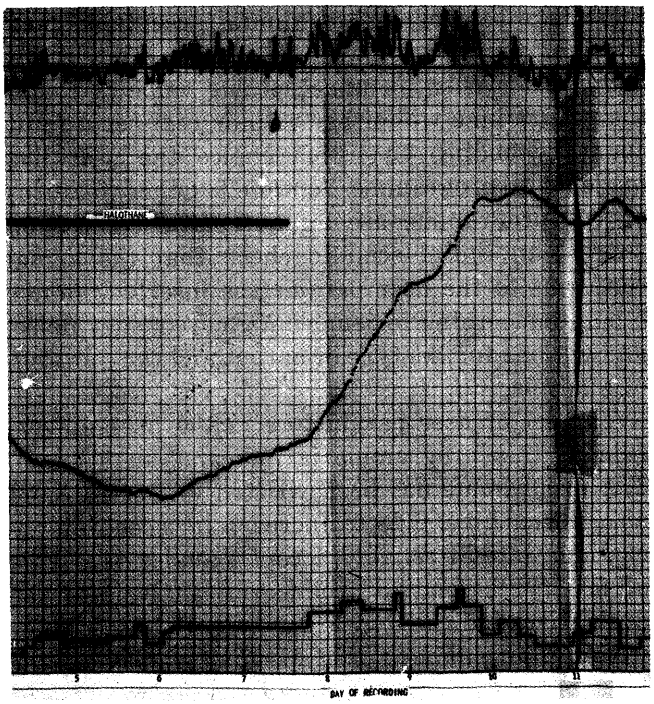


FIGURE 10—Cumulative sum analysis of half-hourly-mean heart rates during a 10-day period in a patient with tetanus (see text)

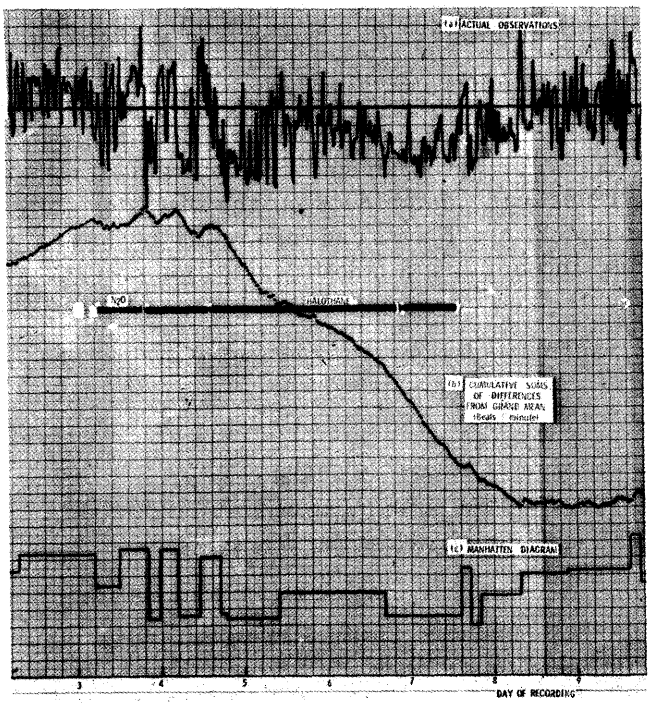


FIGURE 11—Cumulative sum analysis of the variability of heart rates during a 10-day period in a patient with tetanus (see text)

essed in a fashion identical to that described above. The general comments already made on the type of output resulting from this analysis again hold true, and it is also clear that the changes in variability detected by this method do not closely parallel the changes in the average heart rate, although treatment with an anaesthetic agent has produced a clear decrease in the variability.

DISCUSSION

Blood pressure and heart rate are usually measured and recorded by a nurse. She is relatively cheap, easily understood, replaceable, reliable and compact. An automatic monitoring system must offer significant advantages over her to justify the increased capital cost and increased complexity. There is little doubt that most automated systems are indeed superior but there is equally little doubt that most currently available systems are much more effective in producing large volumes of unprocessed data than in efficiently compressing results—they tend to act rather as a team of nurses taking measurements more frequently. The presence of large volumes of potentially useful but unprocessed data in intensive care and research units is a growing problem, and blood pressure records of this type are in fact largely unprocessable, due to the time required for manual analysis. For instance, this may require up to 5 - 6 weeks for analysis of a continuous record lasting 7 - 10 days, and even after this time the analysis is limited. The need for computing is very real and will grow with time.

Reference has already been made to previous applications of digital computers to these problems. The present tendency is to make automatic comprehensive measurements and thereafter to compress the data simply by averaging over varying periods of time. Unfortunately, this is not always a particularly sensitive method of compressing the data without losing its information content. The point is illustrated in Figure 12, where five sections of a blood pressure record taken from the same patient at different stages of a disease process are shown. The duration of each of the traces is about 25 minutes, and a calibration signal is shown for each record. On the right hand side an approximate mean level is written, and the highest and lowest pressure during the period is indicated by a mark. It is immediately apparent to the eye that in the upper record, there is great lability of blood pressure, while this decreases in the lower records. Neither the mean pressure however, nor the blood pressure range adequately reflect the various stages between a very labile and a very stable pressure, although these differences are very important in describing and interpreting changes brought about by the disease process.[33,44] The method of cumulative sum analysis described appears to be a more sensitive method for automatically and objectively describing and assessing such results and also approximates to what one normally attempts to estimate by eye. The in-built application of statistical methods of testing the significance of changes pro-

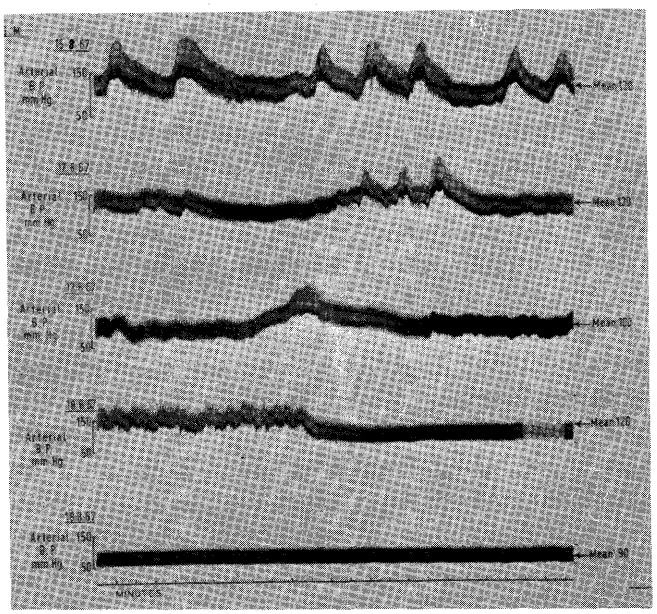


FIGURE 12—Blood pressure recordings taken on a pen recorder with a slow time base (see text)

vides one means of assessing treatments which is free from observer bias and this may well prove to be one of the most important applications, as well as being a critical test of the method. Such assessments must at present be made with caution since a comprehensive analysis of the auto-correlation in such biological parameters as heart rate and blood pressure is not yet available.

It is to some extent a disadvantage of this method that the analysis is necessarily retrospective, since prior knowledge of the grand mean and overall standard deviation is required to determine the cusums. The analysis is however, rapid, and may prove convenient for many units which lack on-line computing facilities. It may also prove possible to develop a method based on the use of provisional estimates of the mean and variance or, in time, to define normal limits of these derivatives.

The methods described for blood pressure analysis should have important applications to the study of other physiological phenomena, the principal differences for many applications being that a lower sampling rate is required (Table II). The variables shown in the table are all very commonly monitored in clinical situations and in research and the important frequencies and derivatives are shown. Comments made above for the near impossibility of measuring the variability and quantitating trends of blood pressure with standard methods may be made again for some of these functions. Harmonics of the basic frequencies do not at present appear to require analysis for these signals, so that the frequencies to be analyzed are much lower than with blood pressure recordings. In addition, it is only the rate or the mean level which needs to be derived in most cases. The similarity lies in the fact that, each function being a time-series in which the degree of autocorrelation is high, they are statistically similar, and are all probably suitable for analysis and presentation by similar techniques, with the addition of cross-correlation analysis. The table is by no means exhaustive, and measurements could easily be extended to include the study of bladder and alimentary tract pressures and motility, some aspects of locomotor activity, and probably other body functions as well.

Research is also required to determine which primary derivatives of the blood pressure wave form have physiological significance and therefore need processing. The mean pressure, mean square pressure, and heart rate may provide as much information alone as they do in combination with systolic, diastolic and pulse pressures. The mean square pressure in particular is likely to be a powerful derivative, though its derivation has not previously been proposed. For certain applications it could also prove more suitable to derive such primary derivates with analogue techniques. Both in these research applications and in clinical practice cumulative sum techniques appear to have important applications. Above all the necessity for arithmetic processing of records must be affirmed. It is only in the critical analysis of a record that its worth or otherwise becomes apparent, and a great deal of information is likely to be lost if complete reliance is placed on simple visual inspection.

TABLE II—Frequencies and sampling rates for physiological variables suitable for study by techniques similar to those for blood pressure analysis

APPROXIMATE SAMPLING RATES REQUIRED FOR DIGITAL COMPUTER ANALYSIS OF COMMON PHYSIOLOGICAL SIGNALS

Physiological Signal	Frequence range/ min. in adults.	Derivatives usually required	Approximate sampling rate/ sec required for computer analysis in real time
Arterial Blood Pressure	45 - 200	Systolic Pressure Diastolic Pressure (Mean Pressure) (Pulse Pressure) (Heart Rate)	120
Central Venous Pressure	45 - 200	Mean Pressure	10
Heart Rate (from instantaneous ratemeter)	45 - 200	Heart Rate	10 for instantaneous rate 1 for average rate
Respiration	8 - 60	Respiration Rate	10
Temperature	0 (slowly varying D C level)	Temperature	1

ACKNOWLEDGMENTS

I am grateful to Dr. D. Clarke and Mrs. H. Somner for substantial help with programming, Dr. A. Barr for advice on statistics, Dr. J. M. K. Spalding and Miss R. Williams for other support, and the Oxford University Systems Engineering Group for the use of their facilities. The investigations were supported by grants from the National Fund for Research into Crippling Diseases, the Nuffield Trust and the Wellcome Trust.

The programme for cumulative sum analysis has been kindly made available by Dr. P. L. Goldsmith, M.A., D.I.C. of Imperial Chemical Industries Limited.

REFERENCES

1 W H LEWIS JR
Procedures in measurement of blood pressure: a historical note
Practitioner 184 243 1960

2 W A SPENCER C VALLBONA
Application of computers in clinical practice
JAMA 191 917 1965

3 S RIVA-ROCCI
Un nuovo sfigmomanometro
Gazz Med di Torino 47 981 1896

4 L HILL H BARNARD
Simple and accurate form of sphygmomanometer or arterial pressure guage contrived for clinical use
Brit med J 2 904 1897

5 N S KOROTKOFF
Concerning methods of study of blood pressure
Tr Imp Mil Med Akad St Petersburg 11 365 1905

6 W E GILSON H GOLDBERG H C SLOCUM
Automatic device for periodically determining and recording both systolic and diastolic blood pressure in man
Science N Y 94 194 1941

7 M B RAPPAPORT A A LUISADA
Indirect sphygmomanometry physical and physiologic analysis and new procedure for estimation of blood pressure
J Lab Clin Med 29 638 1944

8 J C ROSE S R GILFORD H P BROIDA A SOLER E A PARTENOPE E D FREIS
Clinical and investigative application of a new instrument for continuous recording of blood pressure and heart rate
New Eng J Med 249 615 1953

9 J H GREEN
Blood pressure follower for continuous blood pressure recording in man
J Physiol London 130 37P 1955

10 J H CURRENS G L BROWNELL S ARONOW
An automatic blood pressure recording machine
New Eng. T. Med. 256, 780 1957

11 R A JOHNSON
Model 16 atuomatic blood pressure measuring instrument
USAF Wright Air Dev Ctr Dayton Ohio Tech Rept 59–429 1 1959

12 T von VEXKÜLL F KILLING
Ein apparat zur fortlaufenden unblutigen registrierung von puls and blutdruck
Münch med Wschr 101 380 1959

13 R W WARE A R KAHN
Automatic indirect blood pressure determination in flight
J Appl Physiol 18 210 1963

14 A KAHN R W WARE O SIAHAYA
A digital readout technique for aerospace biomedical monitoring
Am J Med Electron 2 152 1963

15 R JONNARD chairman
Symposium on patient monitoring. 15th annual conference on engineering in medicine and biology
The Instrument Publishing Company Inc Pittsburgh Pennsylvania 1963

16 L A GEDDES H E HOFF C VALLBONA G HARRISON W A SPENCER J CARRZONERI
Numerical indication of indirect systolic blood pressure heart rate and respiratory rate
Anesthesiology 25 861 1964

17 L A GEDDES H E HOFF W A SPENCER C VALLBONA
Acquisition of physiological data at the bedside: a progress report
Ann NY Acad Sci 115 1091 1964

18 B L STEINBERG S B LONDON
Automated blood pressure monitoring during surgical anaesthesia
Anaesthesiology 27 6 861 1966

19 W W HOLLAND S HUMERFELT
Measurement of blood pressure: comparison of intra-arterial and cuff values
Brit med J 2 1241–1964

20 F H VAN BERGEN D S WEATHERHEAD A E TRELOAR A B DOBKIN J J BUCKLEY
Comparison of indirect and direct methods of measuring artierial blood pressure
Circulation 10 481 1954

21 L N ROBERTS J R SMILEY G W MANNING
A comparison of direct and indirect blood pressure determination
Circulation 8 232 1953

22 J M STEELE
Measurements of arterial pressure in man
J Mt Sinai Hosp 8 1049 1941–2

23 W F HAMILTON R A WOODBURY H J HARPER JR
Physiologic relationships between intra-thoracic, intraspinal and artierial pressure readings
J Am Med Ass 106 853 1936

24 R E JENSEN H SHUBIN P F MEAGHER M H WEIL
On-line computer monitoring of the seriously ill patient
Med Biol Engin 4 265 1966

25 H SHUBIN M H WEIL
Efficient monitoring with a digital computer of cardiovascular function in seriously ill patients
Ann Intern Med 65 453 1966

26 M H WEIL H SHUBIN W RAND
Experience with a digital computer for study and improved management of the critically ill
JAMA 198 1011 1966

27 S H TAYLOR H R MACDONALD M C ROBINSON R P SAPRU
Computers in cardiovascular investigation
Brit Heart J 29 352 1967

28 H SHUBIN M H WEIL M A ROCKWELL JR
Automated measurement of arterial pressure in patients by use of a digital computer
Med Biol Engin 5 361 1967

29 S I SELDINGER
Catheter replacement of the needle in percutaneous arteriography, a new technique
Acta Radiol 39 368 1953

30 F D STOTT
Medium term direct blood pressure measurement
Bio-medical Engineering 1 457 1966a

31 F D STOTT
Methods of assessment of variations of blood pressure
Bio-medical Engineering 1 544 1966b

32 A L MACMILLAN F D STOTT
Continuous intra-arterial blood pressure measurement
Bio-medical Engineering 1 20 1968

33 J L CORBETT
Long-term measurements of intra-arterial pressure in man
In preparation

34 A T HANSEN E WARBURG
Acta Physiol Scand 19 306 1949

35 A T HANSEN
Pressure measurement in the human organism
Teknisk Forlag Copenhagen 1949

36 D L FRY F W NOBLE A J MALLOS
An evaluation of modern pressure recording systems

Circulat Res 5 40 1957
37 H W SHIRER
Blood pressure measuring methods
IRE Trans BME 116 1962
38 D J PATEL J C GREENFIELD W G AUSTEN A C MORROW D L FRY
J Appl Physiol 20 459 1965
39 R B BLACKMAN J W TURKEY
The measurement of power spectra
Dover Publications Inc New York 1959
40 J S BENDAT A G PIERSOL
Measurement and analysis of rancom data
John Wiley & Sons Inc New York 1966
41 A M VALSALVA
De aure humana tractatus
G vande Water Utrecht 1707
42 G DE J LEE M B MATTHEWS
E P SHARPEY-SCHAFER
Brit Heart J 16 311 1954
43 R H WOODWARD P L GOLDSMITH
Cumulative sum techniques
Oliver and Boyd Ltd Edinburgh 1964
44 J L CORBETT C PRYS-ROBERTS J H KERR
Cardiovascular disturbances in seven tetanus due to over-activity of the sympathetic nervous system.
Submitted for publicatiou

Some conclusions on the use of adaptive linear decision functions

by E. R. IDE, C. E. KIESSLING and C. J. TUNIS

International Business Machines Corporation
Endicott, N.Y.

INTRODUCTION

Any pattern recognition system can be considered to have three sections:

1) a transducer section,
2) a measurement section, and
3) a decision section.

The first two sections transform each pattern to be recognized into one point in a multidimensional space. The axes of this space are the measurements or characteristics of the pattern. The decision section must assign regions of the measurement space to particular classes of pattern. One common and convenient decision surface is the linear boundary or hyperplane; much work has been done with "adaptively derived" hyperplanes.[1,2] Algorithmic procedures have been developed for positioning the linear decision boundaries in the measurements space on the basis of statistically meaningful samples of each pattern class.[3]

Theoretical studies of the classification capability of linear decision boundaries can be performed by assuming a particular statistical distribution of the measurements in the measurement space for each class. For certain assumed distributions in the two-class case, it has been shown that a linear decision boundary is the optimum one.[4] Of course, this boundary would not be the optimum one for many real recognition problems. However, the linear boundary has been the subject of many experimental investigations because it is:

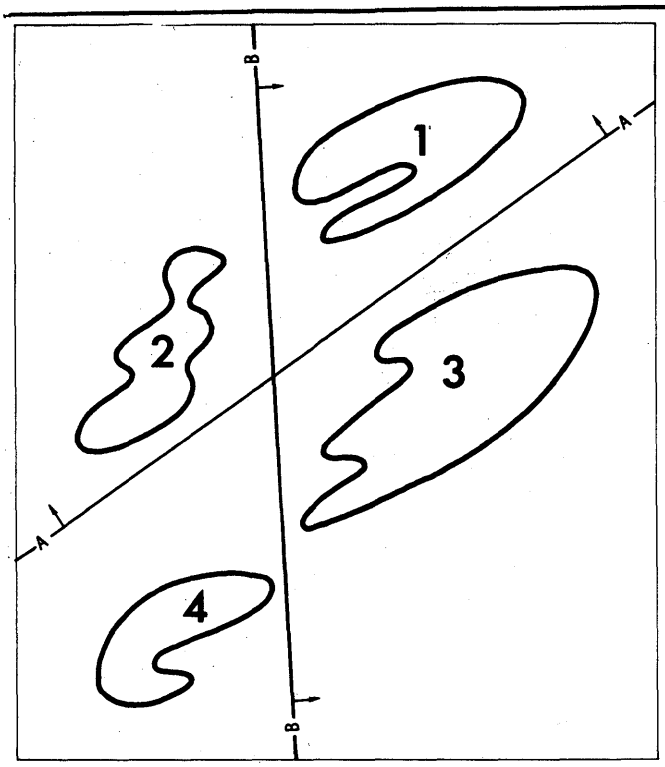
1) optimum in certain idealized cases,
2) a convenient boundary to implement in hardware, and
3) convergent adaptive algorithms[5] exist for this boundary.

The coding problem

In order to have a pattern classifier, we must position a sufficient number of linear boundaries in the measurement space; this separates the measurements arising from one pattern class from the measurements arising from all other pattern classes.

There are a variety of ways of positioning linear boundaries. Figure 1 shows four pattern classes (in a two-dimensional measurement space) separated by only two planes. (The contour in the figure can be taken to mean that 99% of all patterns of this class will give rise to measurements that fall within this boundary.) Note that these two planes, given the dichotomies they are assigned (i.e., plane "A" must separate classes 1 and 2 from 3 and 4), can indeed be positioned in such a way as to perform the separations. If, on the other hand, we had intended to use a single plane to separate classes 1 and 3 from 2 and 4, the separation would not have been possible. This situation, i.e., nonlinear separability, is referred to as the coding assignment problem and is described in a previous paper by the authors.[6]

Figure 2 shows another way of separating each class in the measurement space. This particular method uses a significantly greater number of planes. Here, one plane separates each class from one other class. This will be referred to as the class-pair plane coding assignment. This should be the best linear-boundary classifier since each plane only separates one pair of patterns. Note

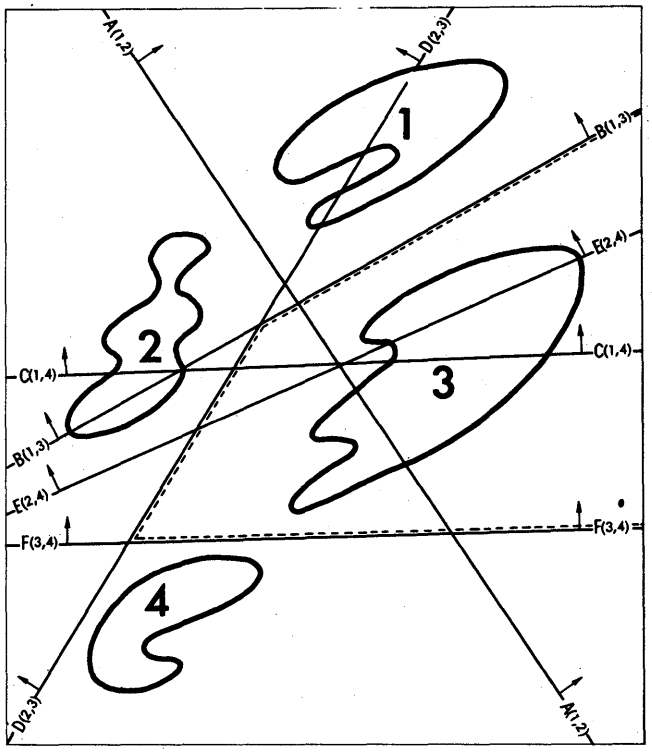


Notes: • Planes are designated by A and B
• Arrows designate "on" side of the planes

CODING ASSIGNMENT

Class	Plane A B
1	1 1
2	1 0
3	0 1
4	0 0

FIGURE 1—Coded boundaries



Notes: • Planes are designated by A, B, C, D, E, and F
• Arrows designate "on" side of the planes
• Numbers in parentheses designate class separation (e.g., A (1,2) means Plane A separates class 1 from class 2)

CODING ASSIGNMENT

Class	Plane A B C D E F
1	1 1 1 x x x
2	0 x x 1 1 x
3	x 0 x 0 x 1
4	x x 0 x 0 0

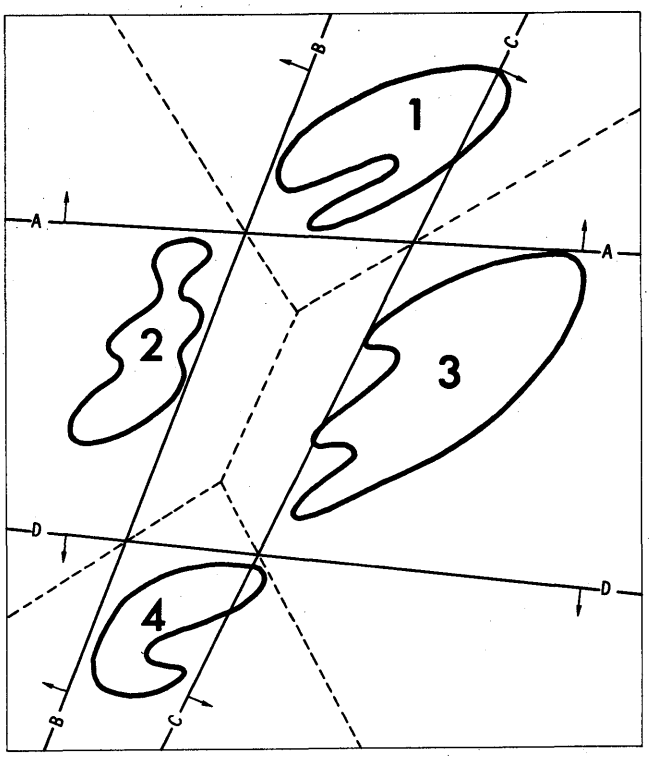
FIGURE 2—Class-pair boundaries

that the regions assigned to each class are determined by different numbers of planes. For example class 3 is determined by three planes while class 4 only needs one plane (planes E and C are redundant with respect to class 4). The disadvantage of this coding assignment is that a large number of planes is required. For example, if there are n classes then n (n– 1)/2 planes are required.

Another popular method of separating classes with linear boundaries is shown in Figure 3. Here we have assigned one plane to separate each class from all others. This is sometimes referred to as the 1-out-of-n code, because there are as many planes as classes (n) and only one plane will be "on" for one pattern presentation. (By "on" we refer to the state of the threshold circuit corresponding to the plane; a classifiable measurement will be in those regions of the measurement space delineated by the positive side of one plane and the negative side of all other planes.) The problem here, of course, is that it may not be possible to separate one class from all others by means of a single plane.

A variant of the method shown in Figure 3, called the matched filter approach, uses as many planes (or linear functionals) as the previous case (i.e., only n) but the actual decision boundaries are the bisectors of the various pairs of planes[2] (see dotted lines in Figure 3). The threshold circuit network corresponding to this classifier is organized to allow only one plane to be "on" at any one time.

The purpose of our experiment was to compare the classification performance of all class-pair planes (in one particular problem) to the classification performance of the matched filter approach. In addition, we tried to develop a classifier that would provide the performance of the class-pair classifier but would have fewer planes. The method started with the class-pair classifier and attempted to eliminate geometrically redundant planes and also to utilize other dichotomies performed by the class-pair boundaries. We show that the performance of the matched filter classifier is surprisingly close to the performance obtained by using all class-pair planes. Our method of eliminating some of the class-pair planes quick-



Notes: • Planes are designated by A, B, C, and D
• Arrows designate "on" side of the planes

CODING ASSIGNMENT

Class	Plane A	B	C	D
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

FIGURE 3—Matched filter boundaries

ly brought performance levels that were inferior to those of the "1-out-of-n" code, even though we were using more decision boundaries.

The classification problem used for this set of experiments was the recognition of a small number of spoken words. In our experiment 15 words or classes were handled. The measurements were obtained by a spectrum analysis of each word; this provided a binary representation of the energy peaks in the frequency domain as a function of time. The speech analyzer and some initial recognition experiments on this data set were reported in an IBM Journal article.[7] The data from Reference 7 were used in this experiment.

Our data base is divided into two parts:

1) one set of sample utterances (i.e., the analysis sample); used for adapting or positioning the hyperplanes in the measurement space and
2) an equal size sample (i.e., the test sample); used to test the performance of the classifying hyperplanes

It is the performance of the hyperplanes on this hitherto unseen sample that is reported as the recognition performance of the system.

The initial step of the experiment was to adaptively position all class-pair hyperplanes on the basis of the training sample. Since the number of classes was 15, we positioned 105 hyperplanes—each hyperplane having the assignment of separating one class of spoken words from one other class. After the class-pair hyperplanes were positioned, we did a recognition run on this same training set. We were not concerned with how well the class-pair plane separated the two classes for which it was designed (it generally does very well, since this is the training set), but rather we were concerned with what other classes of pattern each class-pair plane separated. Recognize that here the word "separator" means *degree* of separation; for example, a particular class-pair hyperplane may correctly separate 90% of the members of class 7 from those of class 9 and only 75% of members of class 3 from those of class 5. Thus, we compiled a number of tables indicating not only which classes each class-pair plane separates but also to what level, i.e., 95% separation, 90% separation, etc.

The second step in our experiment was to choose a subset of all class-pair hyperplanes that still separated all patterns, one from the other. From the table that shows what class each class-pair planes separates to a 95% level, we had to select as many as 43 hyperplanes to partition the space. Each pattern class was described by its position with respect to the 43 planes. This set of 43 binary symbols ("1" denoting that the region is the "on" side of a plane, "0" denoting the "off" side) constituted a code word. Looking at the table that shows what classes each class-pair plane separates to a 90% confidence level, we had to select only 23 planes that would separate all the classes. We then trained the chosen subset of the class-pair planes on the training sample, taking into account the other classes of pattern that we wanted each plane to separate. Once these new planes were chosen and adapted, we determined their performance on the test sample. By using the test sample, we also determined the performance of the classifiers consisting of all class-pair planes, the 1-out-of-n, and the matched filter. This performance data is tabulated in Table 1.

Table 1 shows us that the class-pair classifier was the best linear classifier. The results are sur-

TABLE I—Coding of adaptive linear classifiers

Code	No. of Hyperplanes	Rejects (R) (%)	Substitutions (S) (%)	$\sqrt{RS}$
Class Pair	105	4.8	0.4	1.4
Reduced Class Pair	43	5.1	1.3	2.6
Reduced Class Pair	25	6.2	1.1	2.6
Reduced Class Pair	22	5.8	1.2	2.6
1-out-of-n (Matched Filter)	15	4.1	1.4	2.4
1-out-of-n (Thresh)	15	11.0	0.3	1.8

Note: This table was prepared by sampling 1050 words.

prising because the 1-out-of-n coded classifier, requiring only 15 hyperplanes, is exceedingly close to the performance of the class-pair classifier. Also, we observe the failure of our method of eliminating some of the class-pair planes in arriving at a code that is only nearly as good as the simple 1-out-of-n code. We cannot claim that every recognition problem can be handled so well by this obvious coding assignment, but we have shown an example here of a practical pattern recognition problem where this was indeed the case.

The effect of weight quantization

The implementation of a linear decision function requires that there be a variable weighting of each of the binary inputs or measurements. The question we seek to answer in this section is, "How does the categorization performance of the decision function vary with the number of discrete weight levels available?" This, of course, has tremendous implications to the ease of implementing the linear decision function in hardware. If a resistor array is used, then the tolerance on each resistor is eased. If the linear decision function is implemented by storing the weights in a digital memory and simulating the summation effect of the network, then the amount of storage required is affected significantly. We thus describe an experiment where adaptively derived linear decision boundaries are obtained and then the effect of different available weight quantization levels on the recognition performance of the network is determined.

The application considered here was the recognition of the ten numeric characters and one "special" class (period, comma, dollar sign, minus sign) from approximately forty typewritten fonts. The transducer and pre-processor were similar to those described in Liu and Kamentsky.[8] Characters were scanned by a flying spot CRT optical scanner; a raster image was produced in a shifting register. Position-invariant measurements were made on a character as it passed through the shift register to produce a 100-bit (approximate) measurement vector.

This 100-bit measurement was the input to the network of linear functionals explored here. There are eleven linear functionals, one assigned to each class of character (in the "matched filter" assignment of the previous part). The total sample obtained for this experiment consisted of 55,000 characters. Approximately one-half of these were used for the adaptation of the weights; the other half were used as a test sample.

We were interested in obtaining the weights by first simulating the adaptive process on the training sample on a digital computer and then storing the weights in our recognition machine. The recognition portion of the machine was a special purpose digital computer which "emulates" the set of linear functionals. Thus, the number of bits required to store the set of weights was of interest.

After adaption on the training sample, the weights were all within the range ± 256; thus, we require 9 bits to store each weight. Table 2 shows the recognition performance on a test sample when these finely quantized weights were used. It also shows the deterioration in the substitution and reject rates as the levels of quantization in the weights were successively reduced. Also shown is the square root of the product of the substitution and reject rates (sometimes used as a Figure of Merit* for a recognition system). This is used because there is a "trade-off" between these two rates for any recognition system, but their product is approximately constant. Note that when using even three-bits per weight (which corresponds to only eight distinct weight levels), the recognition rate has not yet degraded by a factor of two, while the substitution rate has remained the same. This behavior is a complex function of the nature of the adaptive procedure, the shape of the decision boundary, and the distribution of characters in the measurement space. With assumed statistical distributions, it would be possible to "compute" this functional relationship, but it is interesting to observe it in this "real-world" application.

*The larger this figure, however, the less the merit of the recognition system.

Thus, if this decision procedure were implemented by storing the individual weights in a digital machine, only 3300 bits** of storage would produce a quite respectable classifier. This storage requirement is about 1/20 of that required by a familiar method[9] (that of storing one, or more, ternary references for each class and computing the "distance" of the unknown from the stored references), but achieves a comparable recognition rate. This advantage of linear decision functions has not, to the authors' knowledge, been noted in the literature.

Unsupervised learning

The literature of pattern recognition, signal detection, adaptive systems, and self-organizing systems has treated the subject of unsupervised learning (learning without a teacher) from both the theoretical and empirical points of view.[10]

We now summarize an experimental investigation of an unsupervised adaptive pattern recognition algorithm previously reported elsewhere.[11] Our adaptive linear classifier undergoes an early training period in which it is presented with a number of labeled samples followed by a later period (which may be indefinitely long) during which it continues to adapt its parameters based on its own decisions.

An unsupervised system can begin with parameters generated from a small labeled sample, and then can use a large sample of unlabeled patterns to design accurate class boundaries. In a typewritten character problem, Nagy[12] used starting parameters generated from nine type fonts. His unsupervised system (quite different from that used here) designed decision criteria for each of twelve other fonts from a five-hundred-character sample of each. Using *no* labeled patterns from these twelve fonts, he achieved "essentially single-font, single-machine performance" on each.

It is intended that, during normal use of a recognition system, unsupervised adaption will take place and will allow the system to follow gradual changes in the class distributions due to data changes or hardware degradation. Such "tracking" was attempted by Koford and Mays[13] with a supervised algorithm which, without data repeating (without using a given input more than once), will track changing statistics and remain close to optimal, if the changes are not too rapid. Cooper and Cooper[14] suggested an unsupervised algorithm for tracking time-variant statistics in a very special case that does not generalize.

The supervised algorithm

We repeat the supervised algorithm: We construct a vector W_i and a constant t_i for each class in the problem, such that each pattern vector X is assigned to a class as follows:

$$\text{If} \quad W_i \cdot X + t_i > W_j \cdot X + t_j + \epsilon \quad \text{for all } j \neq i, \tag{1}$$

the pattern X is assigned to the class i. This type of classifier is sometimes referred to as a trainable matched filter.

If there is no i such that Equation (1) is true, the pattern is assigned to no class; it is rejected. For this reason, ϵ h is often called the reject threshold.

We obtain the set of vectors and constants by presenting iteratively to the network a sequence of sample measurements from each class to be identified, where the class of each sample is known. Our sample must be statistically representative of the individual classes.

The algorithm has advantages noted elsewhere. It is general in application and is relatively easy to implement in hardware. It adjusts the weights (W_i and t_i) after each pattern, eliminating the need to save patterns or sums of patterns in some sort of storage. In previous experimental work with both spoken word and optically sensed typewritten patterns (done by one of the authors), it has displayed somewhat better results than those of several other current methods.[15]

The unsupervised algorithm

To the advantages mentioned above, the unsupervised algorithm adds the benefits of unsupervised adaption: labeling elimination and tracking ability. It differs from the supervised algorithm in that the class of X is unknown. Therefore, the adaption must proceed as if the class assigned by the decision rule, Equation (1), is in fact the class of X.

Specifically, the algorithm proceeds in the following way: If the pattern is rejected or strongly assigned to a class, the vectors are not changed. If the pattern is weakly assigned to a class m, W_m and t_m are incremented as though X be-

**Three bits for each weight; 100 weights/functional; 11 linear functionals, 1 functional assigned to each class.

longed to class m, and the vector of the nearest other class (n) is decremented. The details of the algorithm are presented in (11).

Results

Data

The performance of an adaptive linear classifier designed (trained) using the unsupervised algorithm just described was extensively tested in two distinct pattern recognition problems—spoken word recognition (the same data base as in Part I) and handprinting recognition. The two problems clearly had different statistical properties, although they had roughly the same number of pattern classes. The measurement space of the spoken word problem was 320 dimensional; that of the handprinting recognition problem was 180 dimensional.

We shall repeat the detailed results concerning the word recognition data, but with the fact noted that substantially the same kind of performance was obtained in the other, rather different, application. This fact tends to indicate that the algorithm has general applicability.

Generalization

The classifier was first trained in the supervised mode (i.e., on a labeled set) of thirty alphabets. (An alphabet is defined as a set of single utterances of each word in the vocabulary.) Then the performance of the classifier (weight vectors) was tested on the identification sample after each additional ten alphabets of unsupervised training. The results, shown in Figure 4, show a decrease in the reject rate from 10.5% to 1.6%

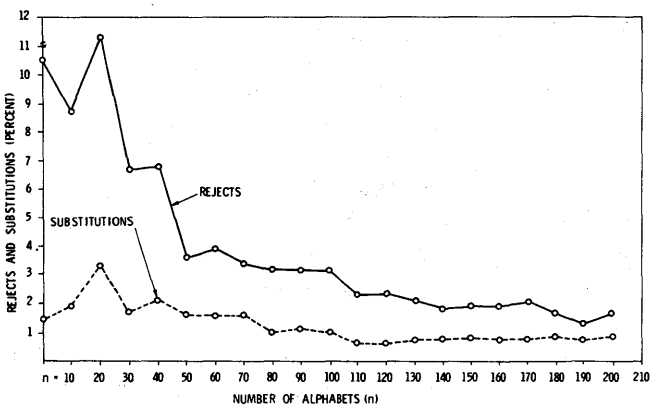


FIGURE 4—Unsupervised generalization on the ID sample after each ten alphabets

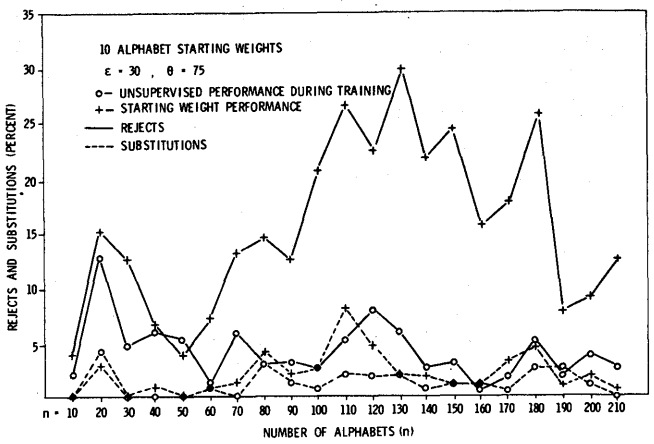


FIGURE 5—Performance of starting weights and of the unsupervised system during each ten alphabets of training

as the classifier is "trained" on additional alphabets. There was also a decrease in the substitution rate.

Performance during training

However, the normal mode of an unsupervised classifier would be described by its "performance during training"; i.e., its recognition rate as it is adapting itself on incoming patterns. To obtain a rate, we show the recognition rate for each successive 10 alphabets (150 words) as the classifier is exposed to these alphabets. Figure 5 shows the reject and substitution rates (for each 10 alphabets) as the system is undergoing unsupervised

TABLE II—Performance of a set of weights quantized into a varying number of levels

Weight Size	Rejects (R) (%)	Substitutions (S) (%)	$\sqrt{RS}$
9-bit weights as originally generated	0.443	0.111	0.219
6-bit weights derived from 1	0.426	0.110	0.215
5-bit weights	0.462	0.111	0.226
4-bit weights	0.523	0.136	0.226
3-bit weights	0.634	0.111	0.265
2-bit weights ("large" reject zone)	4.07	0.154	0.791
2-bit weights ("small" reject zone)	1.06	0.555	0.767

Note: This table was prepared from approximately 55,000 characters obtained from a CRT flying-spot scanner; the characters were then preprocessed to about 100 measurements. Also, 27,930 characters were used for test sample and 27,830 were used for adaptation of the weights (training sample).

training. In addition, we show the comparable recognition performance of the fixed classifier using the starting weights, i.e., the weights obtained after supervised learning on thirty labeled alphabets. The unsupervised weight classifier fluctuates much less than the fixed classifier and generally improves as its experience increases.

Tracking pattern shifts

To test "tracking capability" under more severe conditions, a systematic and severe change in the measurement statistics was artificially created. This change was a right shift by one column of all the bits in the pattern matrix. We felt that performance of the unsupervised algorithm in tracking such a distortion would be generally indicative of its capability of tracking a variety of changes in the statistics due to other malfunctions.

A sample of 100 alphabets was used in this experiment. An initial supervised learning was done on these alphabets in their nominal position. These initial classifier weights were preserved; their recognition performance on the patterns of the subsequently shifted 100 alphabet sample is shown in column One of Table 3. Each pattern of the 100 alphabet sample was then shifted right one column, and a classifier was trained on them (1 pass) in the unsupervised mode, using the previously mentioned supervised weights as the starting weights. For comparison, a supervised (labeled) training was also done on the shifted patterns under the same conditions. Four successive column shifts were performed. For each shift the performance of the unsupervised and supervised classifier was measured (see columns Two and Three of Table 3).

TABLE III—Shifting of patterns—Percent correct recognition (forced decision) of fixed weight, unsupervised, and supervised tracking classifiers

Shift in Bits	Weight Classifier	Unsupervised Classifier	Supervised Tracking Classifier
0	100.0	—	—
16	92.5	99.2	99.3
32	55.0	97.4	99.1
48	28.0	96.2	99.3
64	18.0	93.1	98.9

It can be seen (from column One, Table 3) that after the four shifts, the original weights are indeed useless in a fixed classifier, for their recognition rate is less than 20%. However, the unsupervised classifier has traced the changing data and is still getting 93% correct recognition.

This experiment clearly indicates that unsupervised training allows the recognition system to survive (i.e, continue to perform well), even a relatively rapid change in measurement statistics. Such changes may occur for a variety of reasons such as degradation of hardware or changes in the data input environment.

SUMMARY

Based on the experimental work reported here (and other work reported elsewhere), the authors conclude that the use of adaptively derived linear decision boundaries in practical pattern recognition systems deserves serious consideration. It has been shown to be more than competitive with other classification methods.[15] We have further shown here that the simple, trainable matched filter represents a powerful coding scheme in the practical cases investigated here. Undoubtedly, there are applications where piecewise linear decision boundaries will have to be used.[3]

Once the linear decision boundaries have been derived, surprisingly few number of discrete weight levels have been shown to be usable without significant sacrifice in performance. This allows either the digital storage requirements to be relaxed if a special purpose digital processor simulating the effect of the linear functionals is used, or allows the use of analog implementations with relaxed tolerance on the stored weights. Hitherto, the tolerance requirement has been the main barrier to an economical analog hardware implementation of linear decision functions.

The experiments reported here show that unsupervised learning has potential utility. It has been demonstrated, in two applications, that a classifier in the unsupervised mode can follow changing statistics of the input pattern set. This may indeed be the most useful aspect of adaptively derived linear decision boundaries.

ACKNOWLEDGMENT

The authors wish to acknowledge the assistance of Mitchell P. Marcus.

REFERENCES

1 N J NILSSON
Learning machines: Foundations of trainable pattern classifying systems
New York McGraw-Hill 1965

2 J S GRIFFIN JR J H KING JR C J TUNIS
A pattern-identification device using linear decision functions
In *Computer and Information Sciences*
J T Tou and R H Wilcox Eds Washington D C Spartan pp 169–193 1964

3 R O DUDA H FOSSUM
Pattern classification by iteratively determined linear and piecewise linear discriminant functions
IEEE Transactions on Electronic Computers vol EC–15 pp 220–232 April 1966

4 J S KOFORD G F GRONER
The use of an adaptive threshold element to design a linear optimal pattern classifier
IEEE Transactions on Information Theory vol 12 No 1 pp 42–50 January 1966

5 N J NILSSON
Op cit Chapters 4 and 5

6 C E KIESSLING C J TUNIS
Linearly separable codes for adaptive threshold networks
IEEE Transactions on Electronic Computers
Vol EC–14 No 6 pp 935–936 December 1965

7 J H KING JR C J TUNIS
Some experiments in spoken word recognition
IBM Journal of Research and Development January 1966

8 L A KAMENTSKY C N LIU
Computer-automated design of multifont print recognition logic
IBM Journal of Research and Development
Vol 7 pp 2–13 January 1963

9 C N LIU G L SHELTON JR
An experimental investigation of a mixed-font print recognition system
IEEE Transactions on Electronic Computers vol EC–15 pp 916–925 December 1966

10 J SPRAGINS
Learning without a teacher
IEEE transactions on information theory
Vol IT–12 No 2 pp 223–230 April 1966

11 E R IDE C J TUNIS
An experimental investigation of a nonsupervised adaptive algorithm
IEEE Transactions on Electronic Computers Dec 1967

12 G NAGY G L SHELTON JR
Self-corrective character recognition system
IBM Yorktown Research Report RC 1475 1965

13 J S KOFORD C H MAYS
Adaption of a linear classifier without data repeating
Record of the 1965 International Space Electronics Symposium November 2–4 1965

14 D B COOPER P W COOPER
Nonsupervised adaptive signal detection and pattern recognition
IEEE Transactions on information Theory vol IT–12 No 2 pp 215–222 April 1966

15 R G CASEY Editor
An experimental comparison of several design algorithms used in pattern recognition
IBM Yorktown Research Report RC 1500 1965

Experiments in the recognition of hand-printed text: Part I—Character recognition

by JOHN H. MUNSON

Stanford Research Institute
Menlo Park, California

INTRODUCTION AND BACKGROUND

Among the many subject areas in the field of pattern recognition, the recognition of machine-printed and hand-printed alphanumeric characters has perhaps been the classic example to which people have referred in exemplifying the field. Interest in character recognition has long run high; an extensive literature in hand-printed character recognition alone dates back to at least 1955.[1–36]

In recent years, the recognition of machine printing has become a commercial reality. Following the introduction of the highly controlled E13B magnetic font by the banking industry, several advances in optical character recognition (OCR) capability have been brought to the marketplace. The trend of these advances is toward the acceptance of broader and less controlled classes of input: from single, stylized fonts to multi-font capability; from high-quality copy to ordinary inked-ribbon impressions, and even to multi-part carbons of surprisingly poor quality. Still, in contrast to hand printing, the approaches to OCR have been able to rely on the lack of gross spatial distortions in the character images, and to make considerable use of templates.

Progress in the off-line recognition of hand printing has been slower. The problem is intrinsically harder than that of OCR, as reflected in the fact that the human recognition error rate for isloated, hand-printed characters is many times higher than for machine printing. The great spatial variability of hand-printed characters has led many researchers to explore non-template methods for recognition.

Thus, the major effort of many researchers has been the exploration of unique methods of preprocessing, or feature extraction, applied to the hand-printed character images. Dinneen,[1] in one of the earliest papers, investigated local averaging and smoothing operations to improve the quality of the character image. Similar operations have appeared as a part of many other approaches.[4,7] Lewis,[15] Uyehara,[21] Stern and Shen,[23] and Rabinow Electronics[31] have used schemes in which the sequence of intersections of a slit scan with the character image, or the equivalent, gave rise to features for classification. Lewis[15] was one of the relatively few to emphasize the use of multiple-valued rather than binary-valued features, an ingredient we have found important in our own work.

Singer[12] and Minneman[30] employed a circular raster, which can facilitate size normalization and rotation invariance. Unger,[7] Doyle,[9] and Glucksman[27] have emphasized features derived from shape attributes such as lakes, bays, and profiles. The building up of a character representation from component elements matched to the image, such as short line segments or portions of the boundary, has been attempted by Bomba,[4] Grimsdale et al.,[6] Kuhl,[19] and Spinrad.[26] Correlation techniques have been tried by Highleyman[13] and Minneman.[30] Contour-following with a captive flying-spot scan or its simulated equivalent has appeared in the work of Greanias et al.,[20] Bradshaw,[22] and Clemens.[28] The work of Greanias et al.,[20] is especially significant because it led to the method used in the IBM 1287 character reader.

Other workers have placed greater relative emphasis on classification techniques and on the selection of features from a feature set or pool. Chow[16,29] has long worked with statistical classification methods. Bledsoe and Browning[3] and Roberts[8] applied adaptive procedures to features obtained from more or less random connections with the image raster. Uhr and Vossler[11] performed an important pioneering study of a program that "generates, evaluates, and adjusts" its own parameters. Not surprisingly, however, the automatically generated features were confined to simple, local templates.

The recognition of characters printed subject to

specific constraints (such as guide markers appearing in the printing area) has been studied by Dimond,[2] Kamentsky,[14] and Masterson.[18]

It may be said of most of these investigations that they were in the academic, rather than the practical, realm. In general, the methods were never tested against a body of real-world data large enough to give some estimate of their performance in a practical situation. This probably reflects a common emphasis on checking out a preprocessing scheme rather than attacking a particular application problem; it certainly also reflects the labor and equpiment requirements involved in collecting and controlling a significant body of data. An exception to this general statement is the work of Highleyman and Kamentsky in the early 1960's, in which they used data files numbering in the thousands of characters.[13,14] Also, several files each containing many thousands of characters of graded quality were gathered in conjunction with the development of the IBM 1287 character reader and are currently in use at IBM and in our group. Bakis et al.[35] describe these data, on which they and others at IBM have performed extensive experiments.

The use of context to improve recognition performance, which figures prominently in our own work, was discussed briefly by Bledsoe and Browning,[3] but otherwise has received scant attention in the past. Some studies have been carried out under simplifying assumptions such as Markov dependence in digrams and trigrams.

Chodrow et al.[31] surveyed hand-printed character-recognition techniques in 1965 and discussed at some length the procedures of Clemens,[28] Greanias et al.,[20] and Rabinow Electronics. The book *Pattern Recognition* by Uhr[32] reprints a number of the important source papers[3,6,8,11] and contains a well written survey. An early progress report on the work described herein was given by Munson.[36]

Recently, commercial organizations have announced the capability to read off-line hand printing. At the date of this writing (early 1968), one system (the IBM 1287 optical reader) has achieved pilot production operation. The 1287 reader can read the ten numerals and five letters. Another system is announced to have full alphanumeric capability.

A common characteristic of the announced systems is that they are intended to work with hand printing of very high quality, produced by coders who have undergone training in the skill of printing for machine recognition. If individual characters must be recognized with, say, better than 99.9% accuracy in order to yield usable document acceptance rates, this type of training is clearly required. Some experiments that will be described in the next section show that humans cannot recognize isolated characters printed by an untutored population with any rate approaching the required accuracy.

In our work, we have taken the alternative approach: Given text from an untutored coder, in which the individual characters cannot be recognized (by man or machine) with high accuracy, contextual analysis is used to reduce the error rate. Every form of text has its own contextual structure, which is utilized by humans in a complex, largely unconscious process. We have therefore emphasized the following points in our research: the establishment of large hand-printed data files of known quality; the choice of a well defined character alphabet and textual situation (FORTRAN program texts) as a vehicle for study and the reporting of results; the use of multiple approaches to preprocessing; context analysis to improve recognition; and the preservation of non-binary confidence information between the preprocessor and classifier and between the classifier and the context analyzer.

In a companion paper,[37] Duda and Hart describe the use of programmed contextual analysis in the recognition of FORTRAN program texts. The present paper will therefore concern itself only with the problem of recognizing individual characters.

Problem definition

In a recent paper, the author has argued that there is an infinity of character-recognition problems, and that recognition results are meaningless as they are often reported in the literature, without an adequate description of the problem being treated.[38] Accordingly, we shall try to describe the two recognition problems dealt with in this paper thoroughly enough that the reader can form an intuitive opinion of the difficulty of the problems.

We must first distinguish between off-line character recognition from a printed page, and on-line recognition, in which the characters are generated by a light pen, RAND tablet, or similar device.[24,33,34] On-line recognition is much simpler because the data provide a nearly exact trace of the path of the writing instrument and give accurate stroke-position and time-sequence information. Furthermore, an error rate of as much as 5% may be considered acceptable, because each character can be classified, displayed, and corrected immediately by the writer if it is wrong.

The recognition of hand-printed characters should also be distinguished from that of cursive (connected) script.[25] The separation of the printed characters and the fact that each belongs in a well-specified category obviate the "segmentation problem" that makes cursive-script recognition much more difficult.

Within the framework of off-line block hand printing, the difficulty of a particular problem is still affected by many variables: the size of the alphabet; the "standard" forms of the individual characters and the degree of constraint placed on their formation; the size, spacing, and arrangement of text on the page; the writing instrument(s); the number of writers; their training and motivation; and the (fixed and time-varying) characteristics of each individual writer. To illustrate the variability of hand printing, we may cite several instances of human recognition rates on samples of hand printing. Neisser and Weene reported a 4.1% average error rate on characters printed by visitors at the front gate at Lincoln Laboratory.[10] With all subjects voting together, the error rate was 3.2%. We have reported an error rate of 11% on the well-known quantized character set collected by Highleyman, which suffers from crude quantization of the characters.[39] On the multiple-coder data file used in our experiments and described below, the error rate was 4.5%; on the single-coder file, 0.7%. Finally, present commercial systems are intended to operate with character error and reject rates on the order of 0.1% to 0.01%.

The most significant determinants of hand-printing quality are the training and the motivation of the printing population. Our choice in the work described in this paper was to treat data from an essentially untutored, moderately motivated population, represented by computer users who hand-code program texts for keypunching. Such a coder has typically received no instruction in printing, beyond a few rules about slashing or crossing characters to avoid such confusions as I-1, 0-zero, and 2-Z. He does receive feedback of the results from prior keypunching jobs, which motivates him to maintain (perhaps grudgingly) a certain level of legibility. Thus, while this printing is far sloppier than that allowed by presently announced recognition systems, it is more legible than that produced by the general public while, for example, addressing mail.

Two files of data were used in the experiments reported in this paper, a multiple-coder file and a single-coder file. The characters in both files were hand-printed on standard general-purpose coding sheets obtained from the Stanford Research Institute computer center. The cells on these sheets measured 1/4 inch high by 3/16 inch wide, with no extra spacing between cells. A thin-lead mechanical pencil with an HB (soft) lead was used, after brief experimentation indicated that no other conventional writing instrument gave crisper images when viewed through our input system. (A pencil is the preferred instrument because it facilitates erasure.) The coder was free to use whatever character size he found natural.

The 10 numerals, the 26 uppercase letters, and the symbols [= * / + − ., $] comprised the alphabet of 46 characters. This is the basic FORTRAN alphabet, with brackets substituted for parentheses in accordance with the convention associated with our computer system at the time. The blank was not treated as a character category, the recognition of blanks being more a function of a document-scanning subsystem than a pattern-recognition problem. We instructed the coders to print zero with a diagonal slash and Z with a midline slash, and to put crossbars on the letter I. Numeral 1 was to be without serifs; several coders, however, added serifs. Other choices were left to the individual, such as open versus closed 4, the crossbar on J, and the number of verticals in $.

Multiple-coder file

Printed data from 49 individuals were included in the multiple-coder file. Each person was asked to print several 46-character alphabets on a coding sheet (at one sitting), and the first 3 alphabets from each sheet were taken for the file. The data from the first 32 persons (96 alphabets, 4416 characters) were used as training or design data during the experiments, and the data from the remaining 17 persons (51 alphabets, 2346 characters) for test. The coders of the training data were all personnel of the author's laboratory and the computer center at SRI. The coders of the test data were 8 from SRI and 9 from the US Army Electronics Command, Fort Monmouth, N.J. Any cross-country bias in printing styles is probably small compared with individual differences.

Portions of several of the test alphabets are shown in Figure 1. The coders were asked to print naturally, being neither especially casual nor especially meticulous. However, it is obvious that data gathered this way are not candid; they are probably better than data from actual coding sheets prepared for keypunching. Unfortunately, it was not feasible for us to process candid data from a number of people using a variety of coding forms and languages.

Five human subjects were asked to classify the characters in 17 of the test alphabets—one from each coder—viewing the quantized images (see the section on scanning) in isolation and in random order on a cathode-ray tube display. The error rates ranged from 3.0% to 6.4%, with an average of 4.5%. Taking a plurality vote among the five responses, the error rate was 3.2%.

Single-coder file

Experiments were also performed with a single-

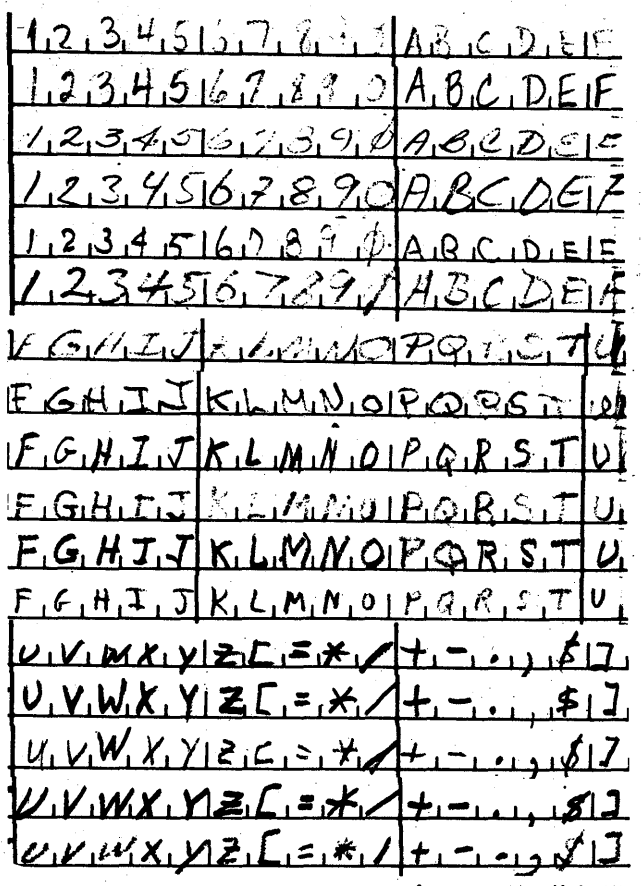


FIGURE 1—Portions of several multiple-coder test alphabets

coder file, in order to investigate the improvement in performance resulting from allowing the recognition system to specialize in the printing of a single individual. This file contained 1727 training characters and 1042 test characters. The training set included 15 alphabets (690 characters) of the type collected for the multiple-coder file. The remaining 1037 training characters were taken from FORTRAN text on coding sheets, as were the 1042 test characters. The 15 alphabets were included in the training set to ensure adequate representation of all the character categories, since their appearance in actual text was haphazard.

The text characters were taken from FORTRAN coding sheets prepared by the author in the course of actual program development, some months before the recognition experiments were performed. The coder corrected major malformations of characters as he noticed them, but avoided printing with unnatural care. Thus, while these data are not candid, it is felt that they closely model a realistic situation that would be obtained if one tried to serve a coder who was making a minimal effort to assist the system.

A sample of the test data is shown in Figure 2. We may describe these characters as being quite legible to

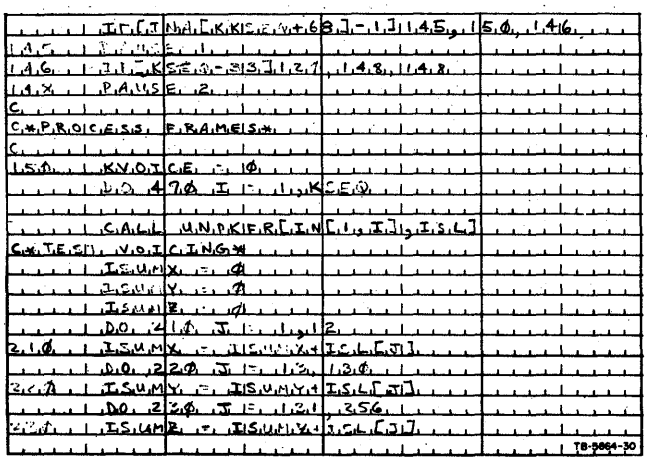


FIGURE 2—A sample of the single-coder test data

humans but not highly regular. Ten human subjects were asked to classify the test characters. The average error rate was 0.7%. Taking a plurality vote among the 10 responses, the error rate was 0.2% (2 errors in 1042 characters).

Scanning

The hand-printed characters were scanned from the source documents (the coding sheets) by a vidicon television camera fitted with a close-up lens and operated under the control of an SDS 910 computer. Each document was mounted in a concave cylindrical holder so that, as the camera panned across the document, the viewing distance and hence the image scale remained constant. The field of view was approximately one inch square. The camera generated a standard closed-circuit television waveform, which was quantized to two levels (black/white) by a Schmidt trigger and sampled in a raster of 120 × 120 points.

The document was illuminated by four floodlights mounted around the TV camera. A colored filter was placed over the camera lens, to suppress the colored coding-sheet guidelines appearing on the document. The guidelines could have been used for locating the characters, but we preferred to strive for a free-field character-locating procedure that could ultimately handle between-the-lines corrections or coding on a blank sheet of paper. Also, without a color-sensitive input system, separating the guidelines from the characters where they crossed or coincided could be a major problem.

The field of view was chosen so that a single character image was usually a little less than 24 points high and about 15 points wide. The computer began the scanning by reading in a 120 × 120 picture containing, in general, several character images. A scanning routine

then proceeded approximately horizontally through the picture, finding and isolating character images. Provisions were included for tracking a line of text, and for accepting multi-part character images such as equals signs and characters with unconnected crossbars. When the scanning routine got to the right of the 120 × 120 picture, it requested the camera to move to the right and input another picture.

As each character was isolated, it was placed in a standard 24 × 24 raster format (Figure 3). No corrections for magnification or rotation were applied. The BCD code of the character was entered manually

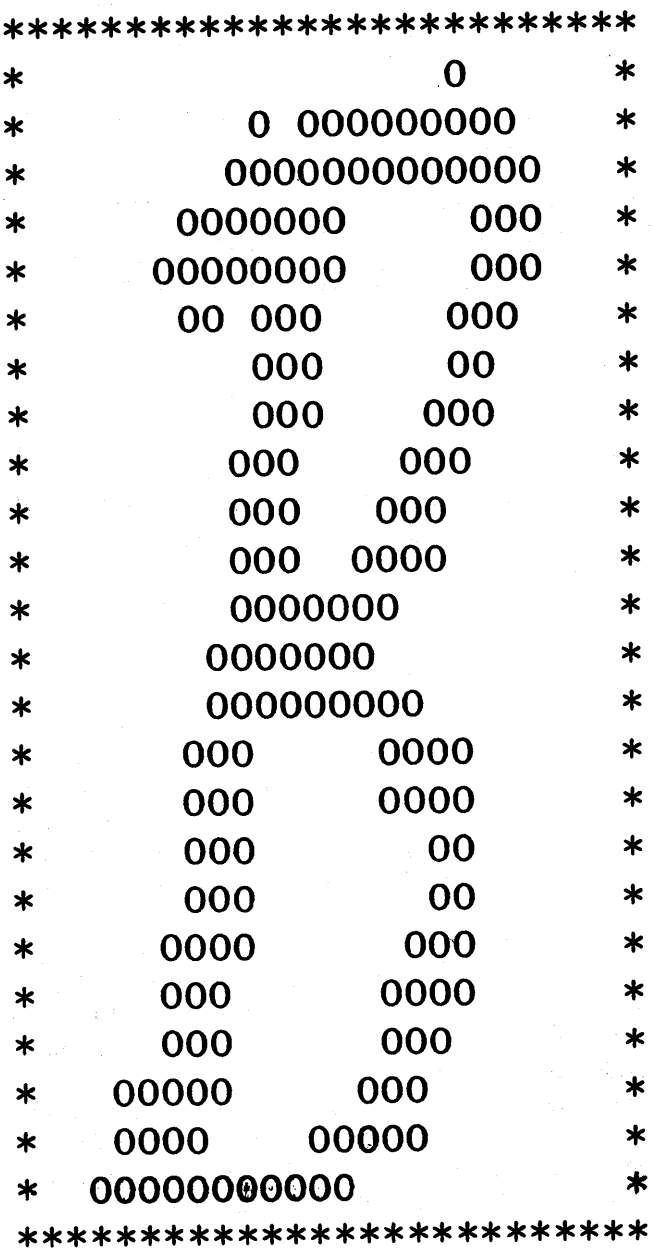


FIGURE 3—A hand-printed character in the standard 24 × 24 format

at the console typewriter and attached to the character record, for subsequent use in the training and testing procedures. The two files (single-coder and multiple-coder) of quantized 24 × 24 black/white character images served as the starting point for all subsequent processing. We hope to make these files available to other researchers through the efforts of the Subcommittee on Reference Data Sets of the Committee on Pattern Recognition of the IEEE Computer Group.

Our scanning setup was "strictly experimental." It was an inexpensive substitute for the sophisticated optical scanner and mechanical transport required for a high-volume production system. Although the scanning routine enabled us to gather the thousands of quantized characters in our data files, it was never capable of running without an attendant to rescue it from its errors. These were due to badly non-uniform sensitivity across the field of view (common in vidicon tubes), which made it impossible to set a single quantization threshold valid throughout the field, and to the lack of precise knowledge of the position of the TV camera. (Incidentally, by solving these problems, it should be possible to create a low-speed, inexpensive automatic scanning system along the lines of the one described above.)

Other files of digitized hand-printed data, supplied through the courtesy of W. Highleyman and researchers at IBM Corporation and Recognition Equipment, Inc., have been processed merely by converting them to our standard 24 × 24 format. In some cases, this has required changing the size of the character raster by copying or deleting rows and columns.

Preprocessing

The term "preprocessing" has acquired a variety of meanings. We use it here to refer to the specific activity of feature extraction: The calculation, from the (quantized) character image, of a set of numerical feature values that form the basis of subsequent pattern classification.

Two preprocessing methods were used in these experiments. The first, embodied in a computer program called PREP, was a simulation of a previously constructed optical preprocessor capable of extracting, in parallel, 1024 optical correlations between a character image and a set of photographic templates, or masks.[40] The second, a program called TOPO, extracted a large number of topological and geometric features of the character image.

The PREP preprocessor

The PREP program performed edge detection on the 24 × 24 quantized images through the use of

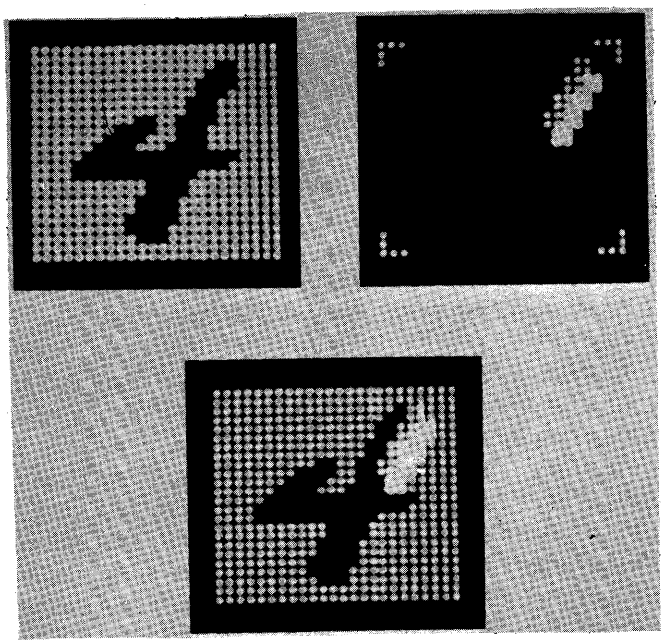


FIGURE 4—Edge-detecting masks in PREP
(a) Quantized character image
(b) An edge mask
(c) Character and mask together

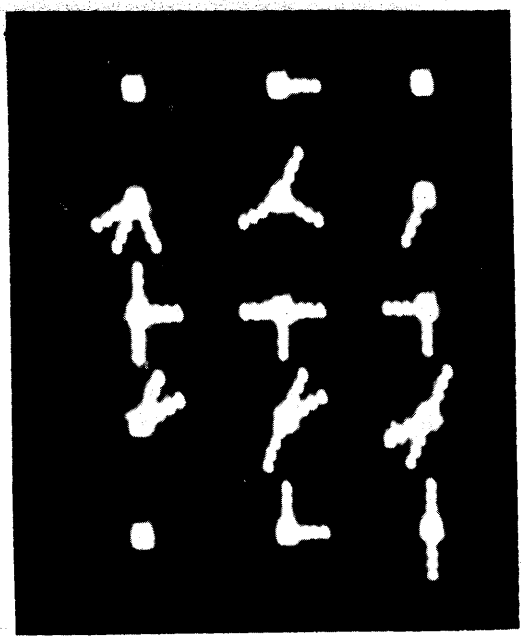


FIGURE 5—Responses of the PREP edge-detecting mask groups to a numeral "2"

edge-detecting mask pairs, or templates. Each mask pair consisted of two 2 × 8 rectangles of points, adjacent to each other along their long edges. One of the masks was given positive weight, the other, negative, and a threshold was set such that if the positive mask encountered six more figure points than the negative one, the binary response of the mask pair was ON (Figure 4).

To provide a limited degree of translation invariance, the responses of five such mask pairs were OR-ed together to give a single binary component of the output feature vector. The five mask pairs in a group had the same orientation and were in the same region of the 24 × 24 field. Nine regions were allotted to each of the four major compass directions, and six regions were allotted to each of the eight secondary directions (at 30° intervals). Thus, the complete feature vector consisted of 84 binary components, and the significance of a typical component was, "An edge oriented north-of-west has been detected in the left central region of the field." Figure 5 shows a computer display in which the lines are normal to edges detected in a sample of the numeral 2. The lines emanate from 15 loci representing the allotted regions.

Each quantized image was presented to the PREP preprocessor nine different times, first in the center of the 24 × 24 field, then in the eight positions formed by translating it vertically and/or horizontally by two units. Thus, for each pattern, a set of nine 84-bit feature vectors was formed. The use of these multiple-view feature vectors to improve classification performance is described below.

The TOPO preprocessor

The TOPO preprocessor was a sizable collection of computer routines assembled to extract topological and geometric features from the character image. In general, these features described the presence, size, location, and orientation of such entities as enclosures and concavities (lakes and bays) and stroke tips in the character.

TOPO began with a single connected character image in the 24 × 24 field. (The equals sign was sought out in advance, and treated as a special case. Other unconnected figures were forcibly joined by growing a bridge between the individual connected regions. If this failed, the lesser region(s) were discarded.) The *perimeter* of the figure was first found (Figure 6). The perimeter was defined as a list of figure points, beginning with the bottommost of the leftmost points of the character figure, found by stepping along the edge of the figure and keeping the figure always at the right hand and the ground (non figure) at the left. The perimeter has the property of including all figure points hand and the ground (non-figure) at the left. The peri-

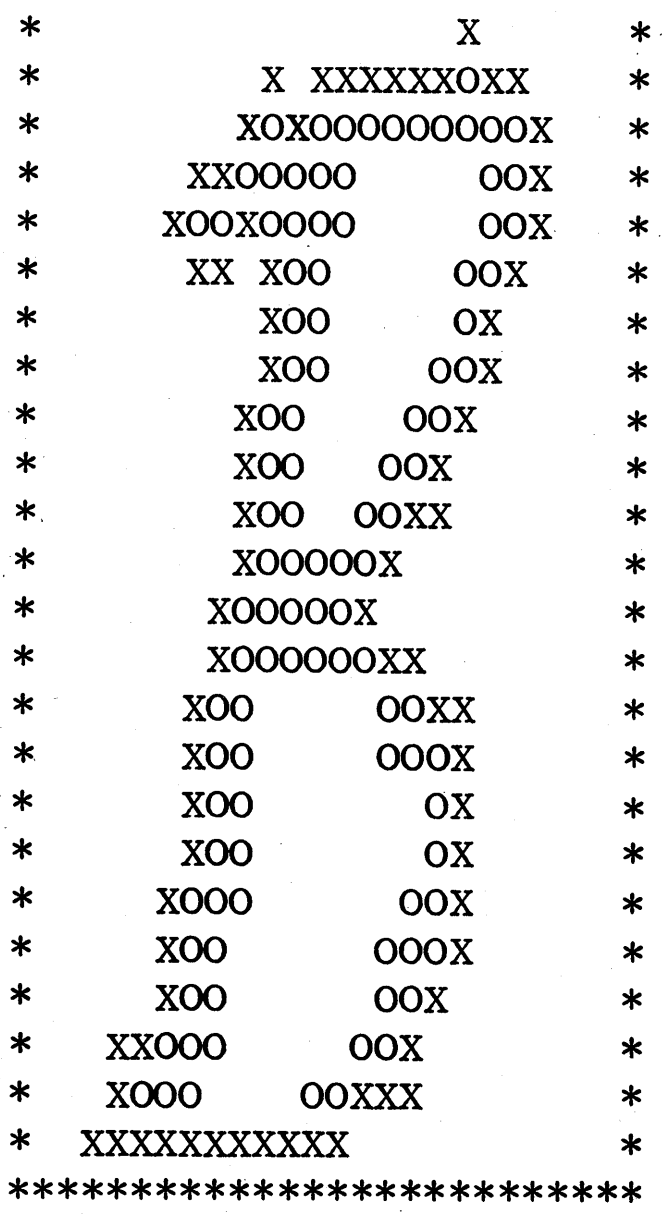


FIGURE 6—Hand-printed character with perimeter points marked X

FIGURE 7—Character with convex hull boundary (CHB)

meter has the property of including all figure points that are 8-adjacent (adjacent horizontally, vertically, or diagonally) to ground points outside.

Next, the *convex hull boundary* (CHB) of the figure was found (Figure 7). The CHB of a two-dimensional figure may be thought of as the outline of a rubber band stretched around the figure. In the case of a quantized figure, some arbitrariness is required in the specification of the CHB, because a straight line between two points on the image grid does not generally fall on exact grid locations. We defined the CHB to include all the extremal points of the character image, represented by letters ot herthan "X" or "O" in Figure 7. In between these extremal points, the CHB was to follow as straight a path as possible, but never falling outside of the theoretical straight line connecting the extremal points. Keeping the CHB to the inside reduced the number of small, insignificant concavities found subsequently. To find the extremal points in the CHB, it was only necessary to search among those perimeter points at which the perimeter turned to the right.

After the CHB was obtained, the concavities and enclosures of the character image could be found quite readily using computer routines that simulated Boolean and connectivity operations performed in parallel over

the entire 24 × 24 field. Let the *border* consist of those ground points in the outermost rows and columns of the 24 × 24 field. Let an image be formed consisting of the ground, minus the CHB. The portion of this image that is not connected to the border lies within the CHB and consists of the concavities and enclosures of the character. Those regions that are connected to the border by a path of ground points (including ground points in the CHB) are concavities; those regions that are not are enclosures within the figure. The character in Figure 7 contains two concavities and two enclosures.

Multiple concavities and/or enclosures were extracted all at once in a single 24 × 24 array by the parallel operations. They were then separated (again using the connectivity operations) and sorted by size for subsequent use.

The *spurs* of a character are those strokes that end in an isolated tip. Ideally, the letter X has four spurs, the letter O, none, and the letter S, one spur with the special property of having a tip at each end. The list of perimeter points was used to find the spurs. Consider two pointers moving down the list of perimeter points, with one pointer ahead of the other by, say, 15 places. As the pointers moved, we calculated the Euclidean distance between the two perimeter points indicated by the pointers. Some of these distances are represented by arrows in Figure 8(a). Most of the time this distance would be approximately 15 units. A sudden decrease of the distance between the two points to a minimum that was less than half its usual value indicated that the perimeter had gone around a sharp bend—i.e., had gone around the tip of a spur. The position of the spur tip, indicated by the perimeter point halfway on the list between the two minimum-separation points, was the primary attribute of the spur used for forming features.

Once a spur was found, it could be traced by the "caliper method" [Figure 8(b)]. Imagine that the legs of a pair of calipers are placed at the two minimum-separation points. The calipers are then "slid" along the spur by stepping the legs of the calipers along the perimeter, away from the tip. The calipers are moved as far as they can go without having to be spread by more than, say, seven units. In some cases, such as the numeral "6," the calipers will be obstructed by the body of the figure and must stop. In other cases, such as the letter "S," the legs of the calipers will travel all the way along the figure and meet at the far end, indicating a "single-stroke" figure. The midpoint of the moving calipers traces out the backbone of the spur, and a list of the midpoint positions can be stored to represent the spur (the heavy line in Figure 8b).

Another set of character attributes found in TOPO

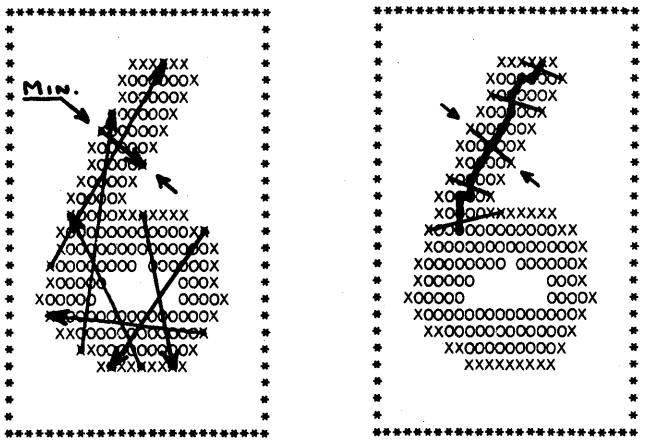


FIGURE 8—Spur-finding
(a) Finding the spur tip
(b) Tracing the spur

and used for feature generation were the *profiles* of the character image. The profiles were four lists, of 24 entries each, specifying the first row (or column) in which a figure point was encountered in each successive column (or row) as seen from the top, bottom, left, and right. The profiles were the basis of a number of specialized feature calculations, designed to discriminate among particular categories, that evaluated such properties as the width of the character at various levels, the number of reversals of direction in a profile, and discontinuities in the profiles.

Numerical feature calculation in TOPO

After the topological and geometric components of the character image—concavities, enclosures, spurs, profiles, etc.—were extracted, it remained to convert them to numerical components of a feature vector suitable for subsequent classification by an adaptive machine. This task was beset with several conceptual and practical difficulties that may not be obvious at first.

In TOPO, the task was carried out in two steps. First, *descriptors* (individual numerical quantities) were derived from the information at hand. Second, *features* in a standard form were calculated from the descriptors.

Each descriptor had to be chosen so that it always represented a unique characteristic of the character image. For example, suppose that one descriptor were to represent the vertical position of the rightmost spur tip. Such a descriptor would help to discriminate, for example, between T and L. But this descriptor would give unpredictable results for characters such as C,E, and [, depending on which spur extended farther to the

right, and would probably be detrimental to the classification of characters in these categories. In addition, there is the problem of vacuous descriptors: What value do we assign to the above descriptor in the case of a letter O?

In TOPO, these problems were countered by a careful choice of the definition of the descriptors. In many cases, it was possible to devise a descriptor that was always well defined. For example, if a spur-descriptor is put in the form, "To what extent is there a spur in the upper right-hand corner," it is defined for any number of spurs and can properly be given its minimum value for a figure with no spurs at all. In addition, this form of definition (unlike the preceding one) has the important property of *continuity*: Deformations of the character image that move the spurs by small amounts always cause small changes in the value of the descriptor. In another paper, the author has argued that the preservation of continuity is important throughout the various stages of the pattern-recognition process.[38]

As the final step in TOPO, the actual features (the numerical components of the feature vector for classification) were calculated from the descriptors. A first requirement on the features was that they be of comparable magnitudes, so that none would dominate the sums formed in the pattern classifier. Thus, the features were all given a standard range of zero to 100. (Note that these features were multiple-valued, whereas those from the PREP preprocessor were binary.)

A second, heuristic requirement on the features was that they emphasize the significant differences among character classes. In a two-category classification problem, it is feasible to analyze the discriminating power of a feature statistically (or even by inspection), and to adjust the transformation from descriptor to feature so as to maximize this power. In our 46-category problem, we could only guess at reasonable transformations. In any case, one should not expect the feature to be a simple linear function of a descriptor.

The derivation of features in TOPO may be indicated by an example. Consider a descriptor, MCONC(up), which is a measure of the presence of an upward-facing concavity in the character. For a flat-topped or round-topped character, such as T or O, MCONC(up) should have the value zero. For a character such as U or V, MCONC(up) should have a value of eight or greater. For a Y or an open-topped 4, however, we should only expect values of five or greater. Owing to the linear nature of the dot-product units used in the pattern classifier, it is impossible for a single feature proportional to MCONC(up) to discriminate between Y and T, for example, without treating U as a "super-Y." We actually require *two* features—one that "switches" in the range 0 to 5 and one that does so at a higher range.

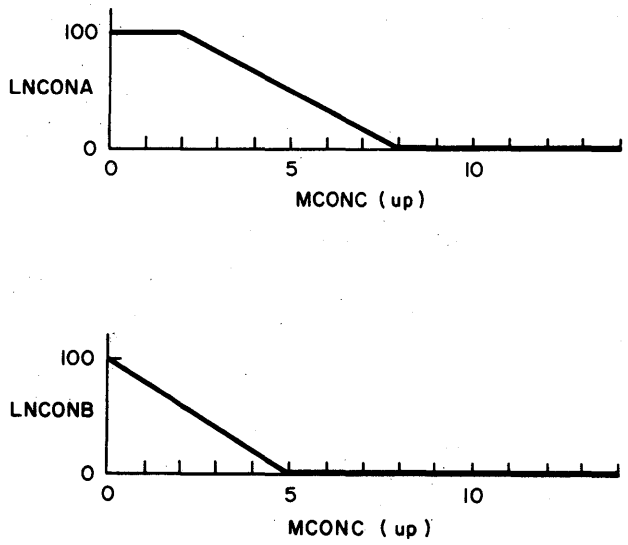


FIGURE 9—Two transformations that derive features from a concavity descriptor

The two transformations that derived features from MCONC(up) in TOPO are shown in Figure 9. There were two such features corresponding to each spur descriptor and concavity descriptor in TOPO. In all, TOPO produced 68 features: 16 for the spurs, 16 for the concavities, 8 for the enclosures, 6 for overall character size and shape, and 22 resulting from special calculations about the width of the character at various levels, discontinuities in the profiles, etc. Each feature was calculated from a numerical descriptor by a transformation arrived at by inspection.

It should be evident from the foregoing description that the development of TOPO was a cut-and-try affair. The extraction of topological entities and the generation of descriptors and features were continued only as far as patience permitted. For example, a feature to look for structure within an enclosure and help discriminate between O and Q was never implemented. It is the author's opinion that the generation and selection of features for pattern classification, especially in the multi-category case, is the greatest problem area in pattern recognition at the present.[38]

Classification

An adaptive pattern classifier, or learning machine, was used to classify the characters on the basis of the feature vectors generated by a preprocessor, either PREP or TOPO. The learning machine was of the piecewise linear (PWL) type, described by Nilsson.[41]

The learning machine for these experiments was implemented by a computer program called CALM (Collected Algorithms for Learning Machines),[42] running on the SDS 910 computer, which simulated the action of the MINOS II hardware learning machine constructed earlier during this project.[40,43,44]

Briefly, a learning machine embodies a set of Dot Product Units (DPU's) that form the *dot product* (also called the inner product or vector product) between the incoming pattern, or feature vector, and a set of stored *weights*. The j^{th} DPU of the machine forms the dot product

$$S_j = X \cdot W_j = \sum x_i w_{ij}$$

between the pattern vector X and the weight vector W_j associated with the j^{th} DPU. In a PWL learning machine, a small number of DPU's are assigned to each of the 46 character categories. The largest dot product formed among the DPU's assigned to a category is taken as the *response* for that category.

The category responses may be utilized in two ways. If it is desired to explicitly categorize a character, the character is assigned to the category with the largest response. A *testing margin* or *dead zone* may be employed, so that any character for which the largest response does not exceed the second largest by the margin is classed as a reject. In the performance results listed below, the reject margin is not used. The performance scores are thus of the simplest possible type: percentage of successful classifications with no rejects allowed (response ties are broken arbitrarily).

Alternatively, if the goal is not to achieve a succinct performance measure but rather to use the character-classification information for contextual analysis, the responses may be used to obtain *confidence information*. The simplest confidence measure is the set of 46 responses from the learning machine, with a higher response indicating a higher confidence that the character belonged to the category in question.

To adapt a learning machine, a *training pattern* is presented, and the responses to that pattern are obtained. If the response in the true category of the training pattern does not exceed the largest response among the other categories by a value called the *training margin*, the DPU yielding the response in the true category is marked to be *incremented*, and that yielding the competing response is marked to be *decremented*. This is done by setting

$$a_{true} = 1 \quad ; \quad a_{competing} = -1$$

in the *adapt vector* A, and setting all the other components of A to zero. Adaptation of the weights is then performed according to the *fixed-increment error correction rule*:

$$W_j \leftarrow W_j + a_j \cdot D \cdot X, \quad \text{for all j.}$$

In other words, the pattern vector is added to or subtracted from the j^{th} weight vector, depending on a_j. D is an overall multiplying factor called the *adapt step size*, usually set to a small integer throughout a block of training. (Other methods of determining the responses and A and D lead to learning machines other than the PWL machine.)[41,42]

The adaptation causes the subsequent dot product between the pattern vector and the weight vector to be changed by an amount

$$\bar{\Delta} S_j = X \cdot (a_j \cdot D \cdot X) = a_j D |X|^2.$$

Since X^2 and D are always positive, the sign of a_j automatically determines whether the response (i.e., the dot product) of the j^{th} DPU with the pattern X is enhanced or reduced. Through this means, appropriate DPU's can be made to respond to certain patterns and ultimately to classes of patterns.

To perform a learning-machine experiment, the adaptive weights w_{ij} are initialized, usually to zero. The training patterns are then presented sequentially. The responses to each training pattern are formed, and if the classification is incorrect the machine is trained. One pass through the training patterns is called an *iteration*. Typically, repeated iterations through the training set are performed until the classification performance on the training patterns ceases to improve. At that time, the test patterns may be presented, and the classification performance on them recorded. This performance is generally taken as the measure of success of the learning machine on the task represented by the training and test patterns.

In dealing with the nine-view sets of feature vectors produced by the PREP preprocessor, the running procedure was modified slightly (Figure 10). During training, one of the nine feature vectors representing a training pattern was selected quasi-randomly at each iteration. Thus, it took nine iterations for the machine to encounter each view of each pattern. The use of multiple views had the effect of "broadening" the training experience of the learning machine. During "nine-view testing," all nine views of each test pattern were presented and the nine responses in each category were added together to form cumulative responses that were used as the basis for classification. It will be seen that the redundancy achieved by accumulating the nine responses led to a significant improvement in performance. This technique has also been used successfully

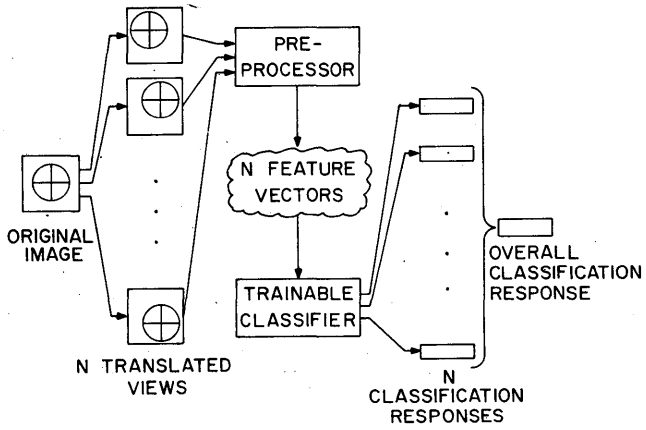


FIGURE 10—Multiple-view testing procedure

by Darling and Joseph in the processing of satellite photographs.[45]

Experimental results

A series of experiments were performed on the single-coder and multiple-coder data files, using the preprocessors and learning machine described above. Experiments were run under four conditions.

In Condition 1, the characters were preprocessed by PREP, but only the one feature vector representing the central view of each pattern was used for training and testing the learning machine. Only the single-coder file was run under Condition 1.

In Condition 2, the characters were preprocessed by PREP in all nine views, and nine-view training and testing were performed as described above. A PWL learning machine with two DPU's per category was used in Conditions 1 and 2.

In Condition 3, the characters were preprocessed by TOPO, and the single feature vectors produced by TOPO were used for training and testing. Owing to computer restrictions, a learning machine with only one DPU per category was used. This is generally called a linear rather than a PWL learning machine, after the form of the discriminant functions in feature space.[41]

In Condition 4, the responses of the learning machines in Conditions 2 and 3 for each test pattern were added together and taken as a new basis for classification. This procedure was a way of harnessing the preprocessor-classifier systems of Conditions 2 and 3 "in tandem" in order to improve classification performance, in a manner analogous to the nine-view testing of the PREP feature vectors.

The results of the experiments are presented in Table I. The results show a significant improvement in performance for the case of nine-view training and testing over single-view training and testing, and further improvement with the combined system of Condition 4. The most important results can be summarized as follows:

Using the combined system, a correct character-classification rate of 97% (with no rejects) was obtained on independent test of relatively unconstrained hand printing in the 46-character FORTRAN alphabet, when the learning machine was allowed to specialize on data from a single coder. When the learning machine was trained on the printing of 32 coders and tested on the printing of 17 others, the correct classification rate was only 85%. These rates are for the isolated characters, without context.

Condition	Preprocessor	Number of Iterations	Final Classification Scores	
			Training Patterns	Test Patterns
Single-Coder File				
1	PREP, 1 view	10	99%	88%
2	PREP, 9 views	27	89%*	96%
3	TOPO	10	94%	91%
4	Combined	--	---	97%
Multiple-Coder File				
2	PREP, 9 views	18	65%*	78%
3	TOPO	4	84%	77%
4	Combined	--	---	85%

*Single-view classification scores

TABLE I—Experimental results on two files of hand-printed alphanumeric characters

A well known set of quantized hand-printed character images (letters and numerals only) collected by Highleyman were also processed under Condition 2, yielding a test classification score of 68%. Previously reported classification methods, not employing preprocessing, had achieved scores of 58% or less. These characters are of very poor quality, being only 86% to 89% classifiable by humans. These results are described in Ref. 39.

A large number of preliminary and auxiliary experiments, not described in this paper, were performed. In particular, during the development of the TOPO preprocessor, an attempt was made to use the features produced by TOPO in a binary decision-tree classifier. The results of this effort were very poor, because it was impossible to find features reliable enough to serve for dichotomization of the character classes. For example,

the presence of an enclosure was a useless feature, because quantization noise introduced some spurious enclosures, and other expected ones were lost because they were filled in or not completely formed. It thus appears to us that, for patterns with the variability of hand printing, an approach that considers all the features in parallel is a necessity.

The development of the TOPO preprocessor, the exploration of variations of the PREP preprocessor, and the running of classification experiments with different learning-machine configurations and different data files were all severely restricted by system limitations of two types. First, it was awkward to handle the necessary large data files on our computer, which had paper tape input/output and one magnetic tape unit. Second, all of the operations of scanning, preprocessing, and classification were performed in serial or at most in 24-bit parallel by simulations in the computer (an SDS 910, with 12K words of 24-bit memory and an 8-μsec cycle time). It sometimes took days to accomplish the experimental runs. However, the flexibility, guaranteed reproducibility, and operational advantages of computer simulation compensated for the drawbacks. With a view toward practical systems, it may be noted that the operations of PREP and the learning machine, and even much of TOPO, are extremely well suited for implementation in parallel hardware.

CONCLUSION

This paper has been concerned solely with the classification of individual hand-printed characters, in isolation. To this end, performance scores based on the positive classification of each character have been presented, as the simplest and most understandable measure of performance. The end goal, however, is text recognition, not character recognition *per se*. The results presented here, including the scores for classification by humans, indicate that context analysis will be a necessary adjunct to character classification for the recognition of fairly unconstrained, untutored hand-printed text. The companion paper by Duda and Hart[37] describes our effort in context analysis. To assist context analysis, the classifier puts out not explicit classifications but lists of alternate categories and their confidences for each character. This may be viewed as a way of retaining additional valuable information generated by the classifier, under a continuous transformation (from feature space, through the learning machine, to the space of category confidence values). This preservation of alternative information is a vital aspect of context-aided text recognition, not to our knowledge previously discussed in the character-recognition literature.

In experimenting with a large body of actual data from an untutored hand-coding population, we are attempting to set a benchmark in an area not covered by the experiments in the former literature, nor by the present commercial developments. At the current levels of available computing power and cost, contextual analysis appears economically infeasible, at least for syntactically rich texts such as FORTRAN. However, if the progress of OCR is a guide, we may expect a pressure for the extension of recognition systems for hand printing to accept input from broader and less highly trained classes of coders. The character-recognition methods described in this paper and the context-analysis procedures described by Duda and Hart may then point toward systems of future importance.

ACKNOWLEDGMENTS

The author wishes to express his gratitude to the many people in the Artificial Intelligence Group at Stanford Research Institute whose work during the past several years has paved the way for this research, especially Dr. Richard O. Duda and Dr. Peter E. Hart.

Support for this work has come from the United States Army Electronics Command, Fort Monmouth, New Jersey, under Contract DA 28-043 AMC-01901(E).

REFERENCES

1 G P DINNEEN
Programming pattern recognition
Proc 1955 WJCC pp 94–100

2 T L DIMOND
Devices for reading handwritten characters
Proc 1957 EJCC pp 232–237

3 W W BLEDSOE I BROWNING
Pattern recognition and reading by machine
Proc 1959 EJCC pp 225–232 Also in 32

4 J S BOMBA
Alpha-numeric character recognition using local operations
Proc 1959 EJCC pp 218–224

5 L A KAMENTSKY
Pattern and character recognition systems—picture processing by nets of neuron-like elements
Proc 1959 WJCC pp 304–309

6 R L GRIMSDALE F H SUMMER C J TUNIS T KILBURN
A system for the atuomatic recognition of patterns
Proc IEE Vol 106B No 26 pp 210–221 March 1959 Also in 32

7 S H UNGER
Pattern detection and recognition
Proc IRE pp 1737–1752 October 1959

8 L G ROBERTS
Pattern recognition with an adaptive network
IRE 1960 International Convention record pp 66–70 Also in 32

9 W DOYLE
Recognition of sloppy hand-printed characters
Proc 1960 WJCC pp 133–142

10 U NEISSER P WEENE
A note on human recognition of hand-printed characters
Information and Control Vol 3 pp 191–196 1960
11 L UHR C VOSSLER
A pattern-recognition program that generates evaluates and adjusts its own operators
Proc 1961 WJCC pp 555–569 Also in *Computers and Thought* Feigenbaum and Feldman Eds pp 251–268 McGraw-Hill Book Co New York N Y 1963 and in 32
12 J R SINGER
A self-organizing recognition system
Proc 1961 WJCC pp 545–554
13 W H HIGHLEYMAN
An analog method for character recognition
IRE Trans on Electronic Computers Vol EC–10 pp 502–512 September 1961
14 L A KAMENTSKY
The simulation of three machines which read rows of handwritten arabic numbers
IRE Trans on Electronic Computers Vol EC–10 No 3 pp 489–501 September 1961
15 P M LEWIS II
The characteristic selection problem in recognition systems
IRE Trans on Information Theory Vol IT–8 pp 171–178 February 1962
16 C K CHOW
A recognition method using neighbor dependence
IRE Trans on Electronic Computers Vol EC–11 No 5 pp 683–690 October 1962
17 W H HIGHLEYMAN
Linear decision functions with application to pattern recognition
Proc IRE Vol 50 No 6 pp 1501–1514 June 1962
18 J L MASTERSON R S HIRSCH
Machine recognition of constrained handwritten arabic numbers
IRE Trans on Human Factors in Electronics Vol HF–3 p 62 September 1962
19 F KUHL
Classification and recognition of hand-printed characters
IEEE 1963 International Convention Record Part 4 pp 75–93 March 1963
20 E C GREANIAS et al
The recognition of handwritten numerals by contour analysis
IBM Journal Vol 7 No 1 pp 14–21 January 1963
21 G U UYEHARA
A stream-following technique for use in character recognition
IEEE 1963 International Convention Record Part 4 pp 64–74 March 1963
22 J A BRADSHAW
Letter recognition using a captive scan
IEEE Trans on Electronic Computers Vol EC–12 No 1 p 26 February 1963
23 D M STERN D W C SHEN
Character recognition by context-dependent transformations
Proc IEE Vol 111 No 11 pp 1923–1931 November 1964
24 W TEITELMAN
Real-time recognition of hand-drawn characters
Proc 1964 FJCC pp 559–575
25 N LINDGREN
Machine recognition of human language part III-cursive script recognition
IEEE Spectrum pp 104–116 May 1965
26 R J SPINRAD
Machine recognition of hand printing
Information and Control Vol 8 pp 124–142 1965
27 H A GLUCKSMAN
A parapropagation pattern classifier
IEEE Trans on Electronic Computers pp 434–443 June 1965
28 J K CLEMENS
Optical character recognition for reading machine applications
Mass Institute of Technology PhD Thesis 1965
29 C K CHOW C N LIU
An approach to structure adaptation in pattern recognition
IEEE Trans on Systems Science and Cybernetics Vol SSC–2 No 2 pp 73–80 December 1966
30 M J MINNEMAN
Handwritten character recognition employing topology cross correlation and decision theory
IEEE Trans on Systems Science and Cybernetics Vol SSC–2 No 2 pp 86–96 (December 1966
31 M M CHODROW W A BIVONA G M WALSH
A study of handprinted character recognition techniques
Technical Report No RADC–TR–65–444 Contract No AF 30 (602)–3721 Information Dynamics Corp Reading Mass February 1966
32 L UHR
Pattern recognition
John Wiley & Sons New York N Y 1966
33 G GRONER
Real-time recognition of hand-printed text
Proc 1966 FJCC pp 591–601
34 J G SIMEK C J TUNIS
Handprinting input device for computer systems
IEEE Spectrum pp 72–81 July 1976
35 R BAKIS et al
An experimental study of machine recognition of hand-printed numerals
Research Report RC–1778 IBM Research Center Yorktown Heights N Y February 1967
36 J H MUNSON
The recognition of hand-printed text
In Pattern Recognition L Kanal Ed Thompson Book Co Washington DC 1968
37 R O DUDA P E HART
Experiments in the recognition of hand-printed text
Part II—context analysis
in this volume
38 J H MUNSON
Some views on pattern-recognition methodology
To be published in the Proc International Conference on Methodologies of Pattern Recognition University of Hawaii Honolulu 1968
39 J H MUNSON et al
Experiments with Highleyman's data
IEEE Trans on Computer Vol C–14 No 4 pp 399–401 April 1968
40 A E BRAIN J H MUNSON
Graphical-data-processing research study and experimental investigation
Final Report Contract DA 36–039 AMC–03247(E) SRI Project ESU 4565 Stanford Research Institute Menlo Park Calif April 1966 Astia Document No AD 632 563
41 N J NILSSON
Learning machines
McGraw-Hill Book Co New York N Y 1965
42 J H MUNSON
User's manual for the CALM learning-machine simulation program
Technical Report Contract AF 30 (602)–4216 SRI Project ESU 6021 Stanford Research Institute Menlo Park California

November 1966 Astia Document No AD 648 959

43 J H MUNSON et al
A pattern-recognition facility with a computer controlled learning machine
Proc IFIP Congress 65 Vol II pp 360–361 May 1965

44 *Graphical-data-processing research study and experimental investigation*
Quarterly Technical Reports ECOM–01901–23 thru ECOM–01901–30 SRI Project ESU 5864 Standard Research Institute Menlo Park California July 1966 thru May 1968

45 E M DARLING JR G D JOSEPH
Pattern recognition from sattellite altitudes
IEEE Trans on Systems Science and Cybernetics Vol SSC–4 No 1 pp 38–47 March 1968

Experiments in the recognition of hand-printed text: Part II—Context analysis

by RICHARD O. DUDA and PETER E. HART

Stanford Research Institute
Menlo Park, California

INTRODUCTION AND BACKGROUND

Problem specification

The work described in this paper is part of a larger effort aimed at the recognition of hand-printed text. In a companion paper, Munson[1] describes the scanning of the text, and the preprocessing and tentative classification of individual characters. In this paper, we describe techniques for using context to detect and correct errors in classification.

The source text used in this experimental study consisted of hand-printed FORTRAN programs. The choice of this common programming language gave us a problem in which contextual relations were both fairly elaborate and well defined. This is to be contrasted with simpler problems, such as might be encountered with business forms, where contextual relations are rudimentary, and with the much more difficult problem of handling natural language, where complex semantic considerations play a large role.

The techniques we developed are embodied in a LISP program called the *context-directed analyzer*. The input to this program, which is obtained from the pattern classifier, consists of a list of possible alternative classifications for each character in the source program. Associated with each alternative is a number that measures the confidence that the alternative is in fact correct. Thus, if presented with a hand-printed A, the classifier might produce the output

Choice	Character	Confidence
1	R	−22
2	A	−28
3	H	−43

indicating an erroneous first choice, but a correct second choice ranking near the first in confidence.*

*The way in which these confidences are actually obtained is described in a later section. Ideally, each confidence is proportional to the logarithm of the probability that the corresponding alternative is correct.

The input to the context-directed analyzer is a list of such lists of alternatives and confidences for all of the characters in a FORTRAN program which we assume to be syntactically legal. The task of the analyzer is to achieve as low an overall error rate as possible by making appropriate choices from among the alternatives.

Past approaches

The utilization of contextual constraints to improve the performance of pattern classifiers has been the subject of a number of investigations.[2–11] One of two basic approaches has generally been followed, the table look-up method or the Markov approach. The table look-up method is based on the assumption that every word in the text is selected from a known finite table. A word of text is classified by comparing it with every table word having the same length and finding the best match. Gold[2] used such a table of legal Morse-code symbols in his system for recognizing hand-sent Morse code, and Bledsoe and Browning[3] used a table of English words in their pioneering experiments in recognizing hand-printed characters.

The Markov approach is rooted in the assumption that the true category of a character is related in a probabilistic manner to the true categories of a small number of surrounding characters. Its use leads to the estimation, from sample text, of the probabilities of all possible pairs, triples, or in general n-tuples of characters. The Markov method can be expected to correct locally improbable character strings, but it ignores global considerations. Harmon[4] used this method to detect errors in the recognition of cursive script. Edwards and Chambers[5] and Carlson[6] also employed this technique to correct errors encountered in conventional optical character recognition. An interesting mixture of the two approaches was used by McElwain and Evens[7] to correct garbled Morse code, and both methods were compared experimentally in a lucid paper by Vossler and Branston.[8]

Both of these approaches rest on the theoretical

foundations of compound decision theory (see Abend [9] for a clear tutorial presentation and for references to the appropriate statistical literature). Both Abend[10] and Raviv[11] point out the importance of considering the alternatives that can be supplied by a classifier. They derive the formal decision-theoretic solution for the optimum use of context and show how it can be simplified by the Markov dependence assumption. However, this assumption, which seems necessary to make the optimum procedure computationally feasible, again limits the ability to exploit global relations.

A formal solution

In this section we outline the general formal solution to the compound decision problem and then point out its drawbacks.

The solution

Suppose for a moment that we have on hand the output of the classifier for a single FORTRAN statement. The basic problem is to select, from among all the alternatives, the correct string of characters. A formal solution to this problem is provided by compound decision theory and requires two ingredients. We must be able to specify, for an arbitrary string of characters (1) the confidence of the string, and (2) the prior probability of the string. The former is provided by the classifier, while the latter must be assumed or estimated ahead of time. A formal solution, derived and made precise in the Appendix, is given by the following intuitively appealing rule:

> Compute the confidence of every string of characters of the given length. Bias each string confidence by adding the logarithm of the prior probability of that string. Set the answer equal to the string having the highest biased confidence.

If we assume that the confidences of the individual characters are the logarithms of their probabilities, then the confidence of a string is just the sum of the confidences of each of the characters in the string. We shall adopt the convention that any character not explicitly listed by the classifier as an alternative is correct with some uniformly low probability, and hence has an accompanying confidence of some low number.

Let us illustrate this rule with the following example. Suppose the classifier returns, for a single FORTRAN statement, the following alternatives, where for readability we list all first choices on the first row, second choice on the second row, and so on.

6	O	T	D	S
G			O	5
			=	

We display the associated confidence of each alternative in a similar array:

−60	−20	−30	−50	−30
−70			−60	−40
			−70	

In order to use the formal compound decision theory solution, we must know the prior probabilities. Let us assume initially that all legal strings of characters are equally likely, and also that all illegal strings have zero probability. The highest confidence string is 6OTD S, with confidence -190, but has prior probability zero. In fact, of the twelve possible strings of characters that can be formed from the alternatives presented, all but four are illegal FORTRAN statements. The four legal ones are

String	Confidence
GOTO S	−210
GOTO 5	−220
GOT = S	−220
GOT = 5	−230

Therefore the final selection is the assigned GO TO statement GOTO S.

In this example we have been casual about the existence of spaces in the text, and throughout the paper we shall ignore questions of spaces and other pragmatics (continuation marks, separating the label field from the rest of the statement and the like) since character position information is assumed to be provided by the original data scanning and input routine.

The disadvantages of the formal solution

The formal solution illustrated above suffers from at least two serious problems: A combinatorial explosion, and an inability to exploit semantic information in a natural way.

Problem 1: Combinatorial explosion

The first objection to the formal solution is that it rapidly gets out of hand combinatorially. Notice that the solution requires explicit enumeration of all possible strings of characters of the given length. We can, of course, adopt the reasonable heuristic that strings of characters will be formed only from among alternatives specifically listed by the classifier. Even so, a statement only ten characters long with, say, four alternatives for each character gives rise to over a million possibilities.

Of course, if we could order strings by confidence then we need not enumerate all possible strings since the decision procedure outlined above selects the highest con-

fidence legal string as the final answer. Thus we could examine the highest confidence string, and if it were legal we would have the answer. If not, we could examine the second most confident string, and so on. In general, this approach requires a method for selecting the k^{th} most confident string of characters given the $(k-1^{st})$ most confident. The problem of ordering strings by confidence is far from trivial. A solution, based on a modification of dynamic programming as suggested to us by R.E. Larson of Stanford Research Institute, is described in Reference 12. While the dynamic programming approach is considerably more efficient than the brute force approach and is used frequently in the analyzer implemented, it also suffers from severe combinatorial problems and can be used only on combinatorially simple data structures. For our computing facilities, the limit of combinatorial complexity for dynamic programming seems to be something on the order of a few thousand combinations; i.e., a string of five or six characters, with about four alternatives for each one.

Problem 2: Semantics

The decision rule discussed above involves only the syntax of the FORTRAN language. It by passes all the richness of semantics. For example, it ignores the simple but important fact that identifiers, and especially variable names, rarely appear only once in a given program. In principal, compound decision theory does not ignore this; it is taken into account by the prior probability of an entire program. Thus, for example, a program containing the variable name HELLO only once is less probable, a priori, than a very similar program containing HELLO several times. In practice, certainly, it is a hopeless task to reflect the multiple appearance of an identifier by directly enumerating prior probabilities.

Structure of the contex-directed analyzer

The previous section outlines the decision-theoretic approach to the utilization of context and points out two severe drawbacks. This section describes the structure of a context-directed analyzer that retains the flavor of the decision-theoretic solution while minimizing combinatorial problems. The analyzer capitalizes on the multiple appearance of variable names, which is a first step toward employing semantic, in addition to syntactic, information to correct classifier errors. We briefly describe the overall operation of the analyzer and then describe its major operations in more detail.

Let us first delineate the current status of our work. The analyzer described is implemented as a LISP program running on an SDS-940 computer. The data for the classifier is an SDS FORTRAN II program which is restricted only in that the I/O lists in input-output statements are simple lists of identifiers. Every statement in the FORTRAN program, except for the COMMENT and FORMAT statements, is subjected to a detailed analysis. For these particular statements the analyzer as yet returns the first choice for each character as the answer.

The analyzer is organized as a two-pass program. During the first pass each statement is identified by type and a table of identifier names is assembled and clustered. During the second pass each statement is resolved and the final classification of the FORTRAN text is made.

In the subsequent discussion we will need to refer to the data structure, illustrated in the GOTO example, that the classifier produces when presented with a segment of FORTRAN text. We will, for no special reason, refer to this data structure as a *P-list*. The i^{th} element of a P-list is a collection of alternatives and confidences for for the i^{th} character in the original text. Thus a P-list contains all the information passed from classifier to analyzer, and the elements in the P-list are in one-to-one correspondence with the characters in the original segment of text.

Statement identification

The identification of statement type is enormously facilitated by the appearance of a "control word" (DIMENSION, IF, etc.) at the beginning of all statements except the arithmetic assignment statement. This property of FORTRAN gives us good reason to suppose that our early assumption that all legal strings are equally likely grossly oversimplifies the matter, at least when the string is at the beginning of a statement. A more realistic assumption would be that each of the 30-odd FORTRAN control words is equally likely at the beginning of a statement, but this would overlook both the relative frequencies of statement types and the arithmetic assignment statement. A simple and not too unrealistic course of action would be to treat all control words as being equally likely, but to reject them all in favor of the arithmetic assignment statement if a sufficiently good match with a control word is not found. This leads directly to the following procedure for identifying statement type.

> Compute the confidence of each FORTRAN control word from the leading segment of the P-list for the statement. If the highest confidence computed exceeds a threshold, then decide the statement is of the corresponding type. Otherwise, decide the statement is an arithmetic assignment.

At this point we must make a remark about the computation of the confidence of a string of characters. If

each confidence were in fact the log of a probability, then the confidence of a string would be (under a conditional independence assumption described in the Appendix) the sum of the confidences of each component in the string. This is, unfortunately, not the case in practice. A reasonable measure of the confidence of a string that has proven quite satisfactory in practice is the normalized, or average, confidence, i.e., the sum of the confidences divided by the length of the string being considered. As an illustration, the confidence of GOTO for the example in the first section is $-170/4 = -42.5$. The confidence of DO is $-220/2 = -110$, under the convention that an alternative not explicitly listed by the classifier has a confidence of—200. The confidence of DIMENSION can be taken as minus infinity, simply because length considerations make it an impossible candidate. The actual implementation of the analyzer differs from the above description only in that confidences of the various control words are compared against a threshold sequentially, and a decision is made if the threshold is exceeded.

Statement analysis

After a P-list representing a statement to be resolved has been identified by type, it is analyzed in order to isolate its natural subparts. The details of these analyses depend upon the statement type, but the following two principles are common to all:

(1) Since the combinatorial explosion is the root of all evil, find delimiters that break the P-list into smaller segments that can be handled by other programs.
(2) Since no single character is reliable, spread the risk in finding a delimiter over a segment of the P-list as long as possible.

A single example will suffice to convey the flavor of the approach. Consider the IF statement. It is syntactically demanded that every IF statement have the form IF[expression] integer_1, integer_2, integer_3. It is easy to strip off the IF from the front of the statement, but we would also like to partition the remainder into a bracketed expression and a list of three integers. The inreliability of single characters make it unwise to seek merely the last right bracket (our character set uses brackets instead of parentheses), so we resort to a more elaborate technique. We start at the tail of the P-list representing the statement, and step along toward the front looking among the alternatives for, successively, a digit, a comma, and a digit. Having found this triple once, we continue to step toward the front looking among alternatives for a second digit-comma-digit triple. When we find it a second time we tentatively declare that the second comma has been passed (reading from right to left) and step along again, now looking for a digit-right bracket pair. When we find this pair, we tentatively declare that the delimiter "right-bracket" has been found and analyze the expression and integer-triple separately. If either of these analyses fails, we assume we have not yet found the delimiter and continue to step along toward the front. If we reach the front of the list without finding the delimiter, then the analysis fails, and the first choice decisions are accepted by default.

Similar analyses are used for other statement types in order to isolate such typical FORTRAN constructions as identifiers, lists of expressions, index controls and the like. These constructions are themselves the subject of further analysis. A subsequent section describes the analysis of what is perhaps the most interesting construction, the arithmetic expression.

The identifier table

Our semantic analysis is concerned only with the multiple appearance of the same identifier in a typical FORTRAN program. While rudimentary, this analysis has proven very successful and is used extensively. The analysis consists of three phases: constructing a table of identifiers (more precisely, the P-lists representing presumed identifiers), clustering the table, and finally using it to resolve FORTRAN statements. Each of these phases will be described in turn.

As currently implemented, assembly of the identifier table begins by restricting attention to statements rich in identifiers. For our purposes, we consider DIMENSION, COMMON, and the various input-output statements as our potential sources of identifier names. As a typical example, let us consider the extraction of identifiers from a COMMON statement. This statement consists of the word COMMON followed by a list of identifiers. One method of finding these identifiers is to search for possible commas in the P-list representing the statement and to assume that everything between commas is an identifier. This approach has proven unreliable because it places too much reliance on single characters. A more satisfactory algorithm searches the P-list exhaustively for all possible occurrences of a string of the form "alphanumeric-comma-alphabetic-alphanumeric . . . alphanumeric-comma-alphabetic." If such a string has a sufficiently high confidence, the segment of the P-list between the two commas is declared to represent an identifier and is added to the table. This algorithm is quite reliable because the confidences of at least five successive characters are computed even if the identifier has length one.

The second phase of semantic analysis involves "clustering" all identifiers of the same length. Clustering accomplishes two things: it prevents the same

identifier from appearing in several slightly different forms in the final result, and it allows us to correct identifier errors even in statements that contributed to the table. The clustering algorithm groups together all P-lists in the identifier table having at least one common alternative for each character. If, for each character, the common alternatives have a sufficiently high average confidence, then all the P-lists in the group are replaced by a single P-list having only the common alternatives. As an example, suppose the identifier table contained the following entries (we suppress the associated confidences for readability):

Entry[1]	*Entry*[2]
N A H E	W R M E
W M S	N A F
L	M
	X

The clustering algorithm would produce the single result

N A M E
W

if the average confidence of the two N's is higher than that of the two W's.

The final phase of identifier analysis is concerned with using the identifier table to resolve ambiguities in the text, and it is divided into two steps: finding a candidate segment of a P-list and matching the segment against the identifier table. Candidate segments of a P-list are found by essentially the same algorithm that assembled the table in the first place; an exhaustive search is made to find segments of the P-list that confidently might represent some identifier. When such a segment is found, it is compared against the table to find the best match. If the match is sufficiently good, the segment of the orginal P-list is replaced by a new segment having only a single alternative for each character.

The match of P-list against table of identifiers fulfills two needs. The obvious advantage is that it (presumably) results in a lower error rate in the final answer. Less obvious, perhaps, is the important reduction in the combinatorial complexity that is achieved by having only a single alternative for each element in a string of characters. We might mention here that one could certainly construct a table of labels as well as a table of identifiers, but this has not yet been implemented.

Resolution of arithmetic expressions

Each FORTRAN construction, such as identifiers, lists of integers, expressions, etc., is the subject of separate analysis. Of these, the analysis of expressions is probably the most interesting because of their complexity and variety. Further, many of the techniques used have been applied in the analysis of simpler constructions.

The expression analyzer has at its disposal a number of techniques which it applies in a fixed sequence. These techniques, in order of application, are (1) an identifier match with the table of identifiers, (2) a compression operation to reduce the combinatorial complexity, (3) a procedure to increase local consistency, (4) a partition of the P-list representing the expression into segments that might represent subexpressions, (5) an exhaustive resolution of the subexpressions, and (6) a reconstruction operation to "uncompress" the final answer. Suppose, then, that a P-list alleged to represent an expression has been obtained. Let us trace the action of the expression analyzer.

The first step is a matching operation against the table of identifiers. As previously described, this operation replaces appropriate segments of the P-list by new segments having only a single alternative for each original character. Once this is done the semantic ability of the analyzer is exhausted, so we take the second step of compressing the P-list. This is done to reduce combinatorial complexity by eliminating syntactically equivalent alternatives. Thus, if there are several letters as alternative to a single character, they may all be replaced by a single generic "X," with some associated confidence. Specification of the associated confidence is a little ticklish, but some heuristic arguments suggest that a good choice is the maximum confidence of the alternative letters. The same procedure is used for digits. Special characters, such as + and $, are left unchanged. As an example, the compression operation would convert the following P-list (suppressing confidences)

A	B	C	+	F	U	N
U	V	W	Q	1	2	3
R		S	1	4		5
						6
						Z

to a compressed version:

X	X	X	+	X	X	X
			X	1	1	1
			1			

The third operation of the expression analyzer is an investigation of local legality. It is interesting to note that, loosely speaking, an arbitrary string of alphanumeric characters can fail to be a legal expression because of either purely local or purely global illegalities.

Global illegalities cannot be detected by inspecting fixed-length segments of the string; typically they result from such things as bracket mismatches. Local illegalities can be detected by inspecting fixed-length segments, the simplest types arising from the juxtaposition of only two characters in an illegal manner, e.g., "* /" or "+]." The local legality check simply verifies that the highest confidence alternatives for pairs of consecutive characters are in fact legal pairs. If an illegal pair is found, then we consider not just that pair of characters, but the 4-tuple of characters centered on the illegal pair. Dynamic programming is then used to select alternatives that produce the legal 4-tuple of highest confidence. The confidences of these alternatives are increased enough to make them first choices. The choice of considering four consecutive characters is a compromise between the desire to make decisions on a global basis and the constraints of combinatorics. The basic advantages of the local check are its speed of operation and conservative nature. It often corrects some errors and never introduces fatal new ones.

The fourth step in the analysis of expressions is the breakdown of long P-lists into shorter ones. We select the operators +, −, /, and* as potential delimiters for partitioning the list. In other words, any character position having one of the above operators among its alternatives is a potential delimiter. A tenatative selection of delimiters is made on the basis of the relative confidences of the potential delimiters and their alternatives, and the segments of the P-list between these delimiters are examined. If each segment can be made into a legal subexpression (by means of dynamic programming) then the segments are strung together and the original expression is resolved. Otherwise, a different selection of tentative delimiters is made. There are a number of pitfalls in this operation. First, consider the expression A + FUN[X + Y]. This is certainly a legal expression, but the segment "FUN[X" delimited by the two plus signs is not a legal subexpression. This is annoying, but not fatal, since a subsequent iteration will presumably partition the expression as A and FUN[X + Y]. A second pitfall is exhibited by the following P-list (as usual, suppressing the associated confidences):

$$\begin{matrix} X\ X + + X \\ X \end{matrix}$$

If the first plus sign were chosen as the delimiter we would have the two simple legal subexpressions X X and + X as first choices of each segment of the P-list, but the concatenation of the two with another plus sign is illegal. This pitfall can be avoided by making a final legality check which, if not satisfied, forces a new selection of potential delimiters. Thus although the method of partitioning expressions is far from perfect, it works well in many cases and does serve to reduce the combinatorial explosion. The partitioning method described has proven useful in many other situations, e.g., partitioning a list of integers by means of commas, etc.

An alternative method of resolving arithmetic expressions is to do a left-to-right parse of the first choices of the P-list until an error is detected. The utility of this method depends largely on the proximity of error commision and error detection. If the two are nearly adjacent a parse could be very useful. Consider, however, a P-list in which the first choice for the first character is an erroneous left bracket. Detection of the error might well occur at the end of the expression, yielding very little useful information. In any event, a parse can tell only that an error has been committed; the localization and correction of the error must be accomplished by other means.

The last step in the resolution of expressions is to "uncompress" the answer. (Recall that all letters appear generically as "X" and all numbers as "1.") The reconstruction is accomplished in a straightforward manner by comparing the compressed answer with the original P-list and replacing each "X" by the highest confidence letter (and similarly for digits).

Summary of analyzer structure

We conclude this section with a summary description of the operation of the context-directed analyzer. The analyzer has a two-pass structure. The first pass determines the type of each statement (more precisely, of the statement represented by the P-list) and produces a clustered table of identifiers. The second pass accomplishes the resolution of each statement—the detection and correction of erroneous first choice alternatives. In this pass each statement is partitioned into subparts

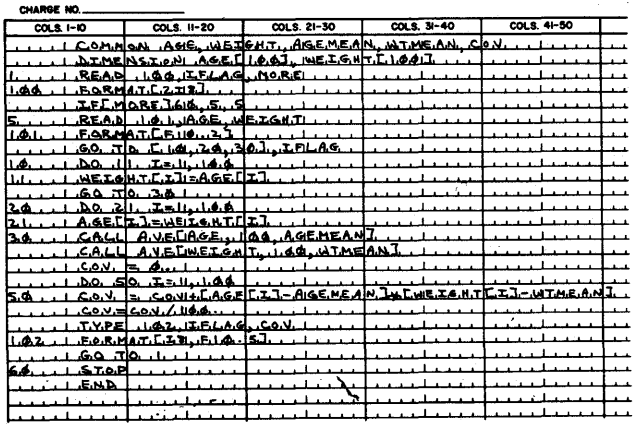


FIGURE 1—A hand-printed FORTRAN program

```
      COMMON AGE,WEIGHT,AGEMEAN,WTMEAN,COV
      DIMENSION AGE(100),WEIGHT(100)
1     READ 100,IFLAG,MOK(
100   FORM/T(2I8)
      IF(MER=)60,5,5
5     RKAD 101,AGE,WEIGHT
101   FORMATCF10.2)
      GO TO (1N,=0130),IFLAG
10    D7 11 I=1,100
1/    WEIGH=(I)=AGECS)
      G3 TO 10
20    D0 21 I=1,100
21    AGE(I)=WEIGHT(I)
70    CALL AVE(AGE,100,AGEMEAN)
      (ALL AVE(WEIGHT,100,WTMEAN)
      (0V=0.
      GD 50 I=1,100
50    C0V = C0V+(AGE(I)-AG(HCAN)*(WEIGHT(I)-UTMEAN)
      C0V=C0V/100.
      TYPE 102,IFLDG,C0V
102   FORMAT(I8,F10.5)
      GO TO 1
60    =T0F
      END
```

TABLE I—First choice of classifier

based on the syntactic requirements of the respective statement types. Each subpart consists of one of a small number of constructs, e.g., an expression or a list of integers. The analysis of each construct often begins with a match against the table of identifiers. If the resulting P-list is combinatorially simple it is analyzed exhaustively by dynamic programming. Otherwise, the P-list is further partitioned into segments by means of delimiters and the segments are examined. Before any exhaustive analysis is made the P-list is compressed to reduce the number of alternatives. The final answer is reconstructed from the compressed answer by comparison with the uncompressed P-list.

Example

Source data

This section describes an experiment illustrating the operation of the context-directed analyzer. The source data for this example came from the hand-printed FORTRAN program shown in Figure 1. The characters on the coding sheet shown were scanned, preprocessed, and classified by the methods described by Munson.[1] Because we purposely wanted data with a moderate error rate, we chose to use only topologically-derived features. Using these features, Munson had previously obtained a nine percent error rate on other data by the same writer. The results on this data were very similar, with 38 out of the 410 characters misclassified for an error rate of 9.3 percent. The particular errors made are shown underlined in Table 1. It is interesting to note that about one-third of these classification errors would not have been detected by purely syntactic methods.

These error rates correspond to the first choice responses of the character classifier, a linear machine. The linear machine classifies a character by computing dot products between a feature vector and 46 stored weight vectors, one for each of the 46 categories. The first choice response is the category corresponding to the largest dot product. The context-directed analyzer uses these dot products to determine alternative choices for each character, together with a measure of confidence for each alternative. The confidences C_i are obtained by normalizing the dot products according to

$$C_i = \frac{S_i - S_{max}}{S_{max}} \qquad i = 1, \cdots, 46,$$

where S_i is the i^{th} dot product, and S_{max} is the largest dot product observed for all of the characters in the source program.

Since the correct category for the character is usually included among the choices having high confidence, it is not necessary to consider every alternative for every character. An empirical study showed that almost invariably the correct category was among those alternatives whose dot products were at least half of the maximum dot product for that character. Thus, only those characters whose confidences met this condition were included in the list of alternatives. This typically reduced the number of alternatives from 46 to 4 or 5, with occasionally only 1, and never more than 10. The price for this simplification was an occasional failure to include the correct category, which was the case for the five doubly-underlined characters in Table 1. Although this introduces extra problems, the reduction in combinatorial complexity is worth the price.

Statement identification

As mentioned previously, the analyzer is organized as a two-pass program. During the first pass, the type of each statement is determined and variable names are collected for the construction of the identifier table. Statement identification was done by comparing the beginning of each statement with the "control" words, IF, DO, READ, etc. For example, the alternatives and the corresponding confidences for the first part of the sixth statement were as follows:

R	K	A	D	−25	−65	−28	−42
A	[	R	O	−49	−65	−62	−52
Z	E		V	−62	−68		−61
	U				−69		
	R				−77		
	L				−81		
	Z				−81		

The average confidence for the first choice selection RKAD was −40. The average confidence for READ was −41, which was sufficiently high to identify the state-

ment as a READ-statement. This matching procedure correctly identified all but one of the 24 statements, including two cases in which the correct category of a control word character was not included in the list of alternatives. However, absence of the correct category caused the ninth statement, a DO–statement, to be erroneously identified as the arithmetic-assignment statement D711I = 1,100. Subsequent analysis failed to resolve this as a legal arithmetic-assignment-statement, however, and the result of this failure condition was that first choice decisions from the classifier were accepted as the final output.

The identifier table

During the first pass, all COMMON, DIMENSION, and input/output statements were inspected to collect potential variable names. This operation was allowed to be somewhat liberal, since the inclusion of spurious identifiers is less harmful than the exclusion of actual identifiers. For example, in the last TYPE-statement the input/output list had the following alternatives:

```
I  F  L  D  G  ,  C  O  V
[     [  A  6     [  D  W
S     6  M  +        Q  U
K     2  N  U        B  +
         ,           G  O
         1           P
```

Because the fifth-choice comma had a fairly high confidence, the program found IFL and G as well as IFLDG and COV as possible variable names. While there is a danger that these fragments might have accidentally matched similar names elsewhere in the program, no such matches occurred. One reason is that long names are tried before short names when the identifier table is used, and this prevents the premature discovery of erroneous matches with short fragments. Another is that completely accidental matches involving names of length greater than three or four are highly unlikely.

The search for possible variable names yielded the following (first choice) possibilities:

```
G  AGC  MOK[  IFLAG  WEIGHT   AGEMEAN
   COV        IFLDG  WTMCAN
   AGE               UEIGHT
   AGE               WEIGHT
   IFL
   COV
```

Even in this simple example the need to cluster the identifier table is clear, since (a) four names were found more than once, and (b) three of these appeared with different first choice spellings. Clustering reduced the identifier table to the following first choice possibilities:

```
G  AGE  MOKE  IFLAG  WEIGHT   AGEMEAN
   COV               WTMCAN
   IFL
```

Of these names, two were spurious (G and IFL), but caused no trouble. Two were wrong (MOKE and WTMCAN), but since only one representative of each was found, they could not be fixed. The remainder (AGE, COV, IFLAG, WEIGHT, and AGEMEAN) were correctly clustered.

Statement analysis

During the second pass, each statement was resolved in turn. Since each different type of statement had to be treated differently, a complete description of how this was accomplished would be tedious. However, the spirit of our procedures can be conveyed by considering the resolution of the long arithmetic-assignment statement. For this statement, the first choices of the classifier were

```
50  COV = COV + [AGE[I] − AG[HCAN] *
          [WEIGHT[I] − UTMEAN] .
```

As with all statements, the label field (columns 1 to 5) was inspected first. Its resolution was trivial, since the first choices were legal. Attention then shifted to the statement field. Starting in column 7, a search was begun for a possible equals sign to be used to break the statement into a tentative variable and a tentative expression. (Had later procedures failed to resolve either of these parts, the search would have been resumed for a possible equals sign further to the right.)

The first character found having an equals sign for an alternative was, in fact, the correct equals sign. At this point, the first step was to resolve the left-hand side of the statement. Since the tentative variable, COV, could have been either a simple identifier (scalar variable) or an identifier followed by a bracketed list of expressions (array variable), a search was begun for a string of the form "alphanumeric, left-bracket." No such string was found, of course, and the tentative variable was declared to be just an identifier of length three. A search through the corresponding part of the identifier table produced a match, and COV was accepted for the name.

The next step was to resolve the expression. Here an exhaustive search of the expression for candidate identifiers was begun at once. Each candidate found was matched against appropriate length entries in the identifier table. This procedure produced five matches, and changed the first choices for the expression from

```
COV + [AGE[I] − AG[HCAN] * [WEIGHT[I]
          − UTMEAN]
```

to

$$\text{COV} + [\text{AGE}[I] - \text{AGEMEAN}] * [\text{WEIGHT}[I] - \text{WTMCAN}],$$

which, even though it contains the error in WTMEAN, is a syntactically valid expression. Thus, in this case the expression was resolved by the first operation, the use of the identifier table. The remaining operations, which have been very useful in other instances, were not needed, and hence were not performed. Since both the variable and the expression were now resolved, these parts were joined by an equals sign and appended to the results of the label-field analysis to yield the final resolution of the statement.

When similar procedures were applied to the other 23 statements, 28 of the 38 errors were corrected, reducing the error rate from 9.3 percent to 2.4 percent. The final output of the analyzer is shown in Table 2, where the 10 remaining errors are underlined. Three of these errors were due to the appearance of WTMCAN rather than WTMEAN in the identifier table, and three more were due to other problems with identifiers: MOKE, MERE, and S. A better method of using the identifier table, in which a final determination of variable names is postponed until all matches are made, would no doubt yield improved results.

Of the remaining four errors, one was in a FORMAT-statement, one in the DO-statement control word, and two involved labels. The FORMAT error was due to the fact that we have yet to implement that part of the program that resolves FORMAT statements. The DO error was caused by the missing alternative, and its correction would require the use of much more sophisticated methods for identifying statement types.Both label errors, however, could easily be cured by using a table of labels similar to the table of identifiers. Thus, roughly half of the 10 uncorrected errors could be resolved by relatively straightforward additions to our present program; the remainder would be difficult indeed to fix.

DISCUSSION

This paper has been concerned with techniques for using context to detect and correct character recognition errors. A few concluding remarks and observations about these techniques and their implementation are in order.

Our first observation is that the addition of new techniques to the context-analyzer program can continue virtually without limit. Many of these additions are straightforward. For example, tables of library subroutine names or statement labels could be incorporated and used in an obvious fashion. Other strategies, which humans employ with remarkably little effort, are much more difficult to implement. For example, the determination of statement type is currently made by matching the leading portion of the P-list against the various control words. A short control word results in a greater risk that the statement type will be misidentified, yet a human easily identifies statement types by their general appearance or gross structure, as well as by the (possibly misclassified) control word. This appreciation of global structure has been one of the more difficult abilities to give the analyzer.

Another observation is that the basic strategy employed by the analyzer should change with variations in the error rate of the input data. The support for this observation rests on intuitive, rather than experimental grounds, but it seems clear that elaborate procedures that may be required for very poor data are unnecessarily inefficient on very good data. While the present analyzer can cope with a certain amount of error-rate variability by automatic adjustment of thresholds, there is no provision to change the basic nature of the operations as a function of the quality of the input data.

A third observation is that there will always exist FORTRAN programs that are unlikely to be resolved successfully. One need only consider the contrary programmer who defines three separate variables as SS5S5, S5S5S, and S5SS5 to appreciate this. Whenever there can be errors in the input, there is a chance of errors in the output. In a practical system, one would want to provide the user with more than the final decision of the recognition system. For example, diagnostic messages could be given to aid the user in finding and correcting errors, whether they were committed by the classifier, the analyzer, or the user himself.

It is difficult to assess the usefulness of our techniques on the basis of an exploratory investigation. A

```
      CØMMØN AGE,WEIGHT,AGEMEAN,WTMCAN,CØV
      DIMENSIØN AGE(100),WEIGHT(100)
1     READ 100,IFLAG,MØKE
100   FØRMAT(2I8)
      IF(MERE)60,5,5
5     READ 101,AGE,WEIGHT
101   FØRMAT(F10.2)
      GØ TØ (10,20,30),IFLAG
10    D7 11 I=1,100
11    WEIGHT(I)=AGE(S)
      GØ TØ 30
20    DØ 21 I=1,100
21    AGE(I)=WEIGHT(I)
70    CALL AVE(AGE,100,AGEMEAN)
      CALL AVE(WEIGHT,100,WTMCAN)
      CØV=0
      DØ 50 I=1,100
50    CØV=CØV+(AGE(I)-AGEMEAN)*(WEIGHT(I)-WTMCAN)
      CØV=CØV/100.
      TYPE 102,IFLAG,CØV
102   FØRMAT(I8,F10.5)
      GØ TØ 1
60    STØP
      END
```

TABLE II—Final output of analyzer

thorough evaluation of the performance of the analyzer can be made only by testing it on a large number of FORTRAN programs produced by a variety of authors. Unfortunately, we were unable to undertake a data-processing project of this magnitude.

An equally difficult question concerns the extendability of the reported techniques to other problem domains. These techniques can be characterized by three qualities: risk-spreading in decision making, partitioning of a large decision problem into a hierarchy of subproblems, and continual checking of internal consistency. It seems clear that our basic approach applies more or less directly to other programming languages, and perhaps could be used with natural language in tightly constrained situations. The conjecture that the general approach, at least, can be applied in more general problems has a certain piquancy, but it remains only a conjecture.

We have, however, been able to achieve a substantial reduction in error rate for a particular application. In our opinion, it would have been difficult to obtain a comparable improvement by applying more conventional context analysis methods which do not take advantage of the special nature of the problem.

AKNOWLEDGMENTS

The authors wish to thank their colleagues at Stanford Reasearch Institute, and most particularly to thank Dr. John H. Munson for many stimulating and fruitful discussions.

This work has been supported by the United States Army Electronics Command, Fort Monmouth, New Jersey under Contract DA 28-043 AMC-01901(E).

APPENDIX

In this Appendix we derive the decision rule that classifies strings of alphanumeric characters in an optimal (minimum probability of error) fashion. By appropriately interpreting our result, we arrive at the decision rule described in the text.

Suppose that we are given some string of n characters to classify. Our problem is to determine a string of n categories that minimizes the probability of misclassification. If we let the vector X_i denote the set of measurements made on the i^{th} character and θ_i denote the category selected for the i^{th} character, then it is well known from Bayesian decision theory that the minimum probability of error is achieved by the following rule:

Select the categories $\theta_1, \cdots, \theta_n$ which maximize the posterior probability $p(\theta_1, \cdots, \theta_n|X_1, \cdots, X_n)$.

In other words, the posterior probability is computed for every possible assignment of θ_1 through θ_n, and the most probable assignment is taken as the decision.

By Bayes' law of inverse probabilities we can write the posterior probability as

$$p(\theta_1, \cdots, \theta_n|X_1, \cdots, X_n) = \frac{p(X_1, \cdots, X_n|\theta_1, \cdots, \theta_n)\, p(\theta_1, \cdots, \theta_n)}{p(X_1, \cdots, X_n)}. \quad (1)$$

This shows that the computation of the posterior probability depends upon both the prior probability $p(\theta_1, \ldots,\theta_n)$ and the conditional probability $p(X_1, \ldots, X_n|\theta_1, \ldots,\theta_n)$. To simplify the computation of the conditional density, we make the reasonable assumption that the manner in which a character is formed depends only upon the category of the character and not on the categories or the measurements of any surrounding character. This assumption is equivalent to an assumption of conditional independence, namely that

$$p(X_1, \cdots, X_n|\theta_1, \cdots, \theta_n) = \prod_{i=1}^{n} p(X_i|\theta_i). \quad (2)$$

At this point we must make some assumptions about the performance of the character classifier that provides the input to the context-directed analyzer. If this classifier were designed for the optimal classification of characters without regard to context it would compute, for every θ_i, the posterior probability

$$p(\theta_i|X_i) = \frac{p(X_i|\theta_i)\, p(\theta_i)}{p(X_i)}.$$

During the design of the classifier it was tacitly assumed that all classes are equally likely a priori, so that $p(\theta_i) = 1/46$. We therefore make the bold assumption that the classifier computes, for all 46 values of θ_i,

$$p^*(\theta_i|X_i) = \frac{p(X_i|\theta_i)\, 1/46}{p(X_i)}. \quad (3)$$

Substituting (3) and (2) into (1), we obtain

$$p(\theta_1, \cdots, \theta_n|X_1, \cdots, X_n) = \frac{p(\theta_1, \cdots, \theta_n)}{p(X_1, \cdots, X_n)} \prod_{i=1}^{n} 46\, p(X_i) p^*(\theta_i|X_i).$$

Now for given measurements $X_1,\ldots,X_n$ we are interested in maximizing this quantity over $\theta_1,\ldots,\theta_n$ so we can ignore constants and factors depending solely on X_i and obtain the following optimal compound decision rule:

Select the categories $\theta_1, \cdots, \theta_n$ for which

$$p(\theta_1, \cdots, \theta_n) \prod_{i=1}^{n} p^*(\theta_i|X_i) \text{ is maximum.}$$

We can, of course, take any monotonic function of this quantity and maximize it instead. Taking logarithms we can select the $\theta_1,\ldots,\theta_n$ which maximize

$$\log p(\theta_1, \cdots, \theta_n) + \sum_{i=1}^{n} \log p^* (\theta_i|X_i) .$$

If we define $\log p^* (\theta_i|X_i)$ as being the confidence that the measurements X_i indicate class θ_i, then we may reasonably define the confidence of the string to be

$$\sum_{i=1}^{n} \log p^*(\theta_i|X_i) .$$

The optimal decision rule, then, computes the confidence of each string of length n, biases each string confidence by adding the logarithm of the prior probability of the string, and selects as the answer that string having the highest biased confidence.

REFERENCES

1 J MUNSON
Experiments in the recognition of hand-printed text: Part I—Character recognition
In this volume

2 B GOLD
Machine recognition of hand-sent Morse code
IRE Trans on Information Theory Vol IT-5 pp 17–24 March 1959

3 W W BLEDSOE J BROWNING
Pattern recognition and reading by machine
Proc EJCC pp 225–232 Dec 1959 Also in *Pattern Recognition* L Uhr Ed pp 301–316 Wiley New York 1966

4 L D HARMON
Automatic reading of cursive script
In Optical Character Recognition Fischer et al Eds pp 151–152 Spartan Washington DC 1962

5 A W EDWARDS R L CHAMBERS
Can a priori probabilities help in character recognition
J ACM Vol 11 pp 465–470 October 1964

6 G CARLSON
Techniques for replacing characters that are garbled on input
AFIPS Conf Proc Vol 28 pp 189–192 Spring Joint Computer Conference 1966

7 C K McELWAIN M B EVENS
The degarbler—A program for correcting machine-read morse code
Information and Control Vol 5 pp 368–384 1962

8 C M VOSSLER N M BRANSTON
The use of context for correcting garbled English text
Proc ACM 19th National Conference paper D2 4–1 D2 4–13 1964

9 K ABEND
Compound decision procedures for pattern recognition
Proc NEC Vol 22 pp 777–780 1966

10 K ABEND
Compound decision procedures for unknown distributions and for dependent states of nature
In *Pattern Recognition* L Kanal Ed Thompson Book Co Washington DC 1968

11 J RAVIV
Decision making in Markov chains applied to the problem of pattern recognition
IEEE Trans on Info Thy Vol IT-13 pp 536–551 October 1967

12 R O DUDA P E HART J H MUNSON
Graphical-data-processing research study and experimental investigation
Fourth Quarterly Report Contract DA 28-043 AMC-01901 (E) SRI Project ESU 5864 Stanford Research Institute Menlo Park Calif March 1967

The design of an OCR system for reading handwritten numerals

by WILLIAM S. ROHLAND, PATRICK J. TRAGLIA
and PATRICK J. HURLEY

International Business Machines Corporation
Rochester, Minnesota

INTRODUCTION

The problem of transcribing computer input data from a human-sensible form to a machine-sensible form has grown with the computer industry. The "input gap" has also grown as the time to process a given batch of data in the CPU (central processing unit) has become smaller and smaller when compared to the time needed to manually transcribe that same batch of data. In short, today's data processing systems are capable of accepting, using, and disposing of data at much faster rates than available input devices are capable of providing, without some form of preliminary off-line conversion or transcribing process.

Optical character recognition (OCR) has been given much credit for helping bridge the "input gap." This is because optical character readers have been used for several years for the automatic transcription to computers of data printed in certain specified type fonts on certain specified types of input devices such as high-speed line printers, typewriters, credit-card imprinters, and a variety of printing presses, cash registers and adding machines. This would include sales checks; airline, bus, and train tickets; utility bills; premium notices; charge account statements, and countless others too numerous to mention. One input device for character recognition that had not been successfully exploited during the early years of OCR was the common lead pencil.

This paper traces the steps taken in exploring the practicality of an input system consisting of a usually unconcerned, intractable human being; an always too sharp or too dull pencil; and an optical reader designed to accommodate many of these variables. The initial objectives, as defined by Market Planning, will be discussed. The technology choices will then be examined, and the limitations and problems of the system discussed. Finally, relative performance as a function of training, motivation of, and feedback to, the writer will be presented.

This paper deals with only one aspect of the IBM 1287 Optical Reader (Figure 1). Although this machine possesses the capability of reading many type fonts printed by a variety of input devices, only the *N*umeric *H*and*w*riting Feature (NHW) will be discussed. The discussion presented here also applies to the numeric handwriting feature of the recently announced IBM 1288 Optical Page Reader.

Initial objectives

Several initial design objectives for developing a hardware capability for the reading of numeric handwriting were set forth.

- *The minimum character set should be the ten numerics plus five control symbols, preferably alphabetic characters. (C, S, T, X and Z were chosen)*

  Although some applications require, or prefer, full alphanumeric capability, a definite majority of the applications can be handled with the stated minimum set. Dollar amounts, stock numbers, merchandise classes, departments, dates, clerks, customers and creditors can all be identified by numbers. The limited number of control symbols are necessary for tagging various categories of numeric information, i.e., negative balance, sub-total, taxable amount, etc.

- *Acceptable character shapes should be those*

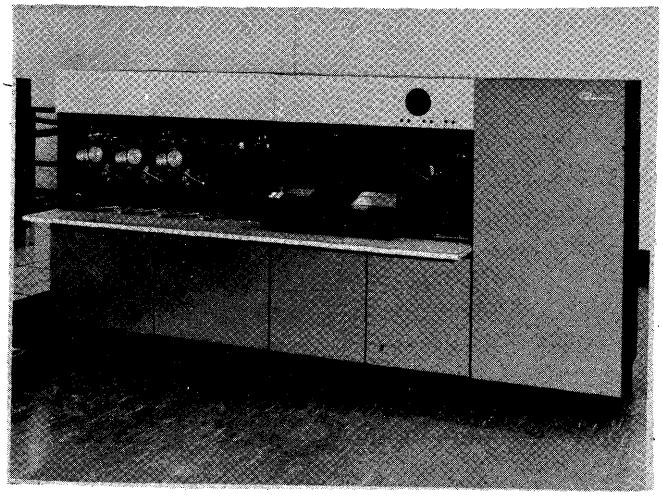


FIGURE 1—IBM 1287 optical reader

naturally formed by the majority of the population.

There is some disagreement as to whether one "writes" or "prints" numeric characters. On the other hand, the difference between handwritten and hand-printed alphabetic characters is generally recognized. For clarification, the character shapes are defined as handwritten numerals and hand-printed alphabetic symbols (not cursive script).

Although it is highly desirable through training techniques to limit the distribution of acceptable character shapes, these techniques should not force the writing of unnatural character shapes.

- *Within the limits of good human engineering, the constraints printed on the document to guide the writer relative to the location, size, and aspect ratio of the character, shall be minimum.*

  This is based on the premise that the simplest constraint will tend to result in the simplest training and maximum effectiveness in writing.

- *No special printing devices must be required, other than a common No. 2 pencil.*

  There should be no requirement for special inks, pencil leads, or for papers other than OCR bond papers.

- *The objectives for recognition rates (characters per second), recognition performance (character reject and substitution rates) and product cost are interrelated to a high degree, with many trade-offs possible among these three factors. General limitations, however, must be set for each factor.*

  The recognition rates must be such that, when effective throughput rates are calculated, with provision for transport time, format control, and reject rescans, these effective throughput rates represent a good payoff to the customer.

  Reject and substitution rates should be the same or lower than equivalent error rates of equal impact in the manual transcription process. For example, the character substitution rate should be no higher than keypunching error rates, which are comprised primarily of transposition errors. Customer satisfaction with a given level of rejects and substitutions will depend to some extent on the effectiveness of his system checking and correction routines.

  The product cost, after all trade-offs have been made, must obviously be compatible with a marketable price.

Technical description—IBM 1287 Optical Reader

The numeric handwriting function of the IBM 1287 Optical Reader will be described in three parts:

- Scanner and Format Control,
- Data Extraction Process, and
- Recognition Decision.

Scanner and format control (Figure 2)

A CRT (cathode ray tube) flying spot scanner was chosen for the 1287 primarily because of its format and scan pattern versatility. This versatility was required to accommodate a wide range of document sizes and formats, and to provide the scan patterns for mark reading and for a variety of stylized type fonts, in addition to those required for NHW.

The scanner is composed of a 5-in. P36 CRT (P36 is the phosphor type), two mirrors, a f/2.8 2:1 magnifier lens, a 1½-in. S11 response monitor PMT (photomultiplier tube), a 5-in. S11 response document PMT, a PMT power supply, and a high voltage power supply which provides both the focus potential and post-accelerator anode potential for the CRT. Mechanically, the light-tight

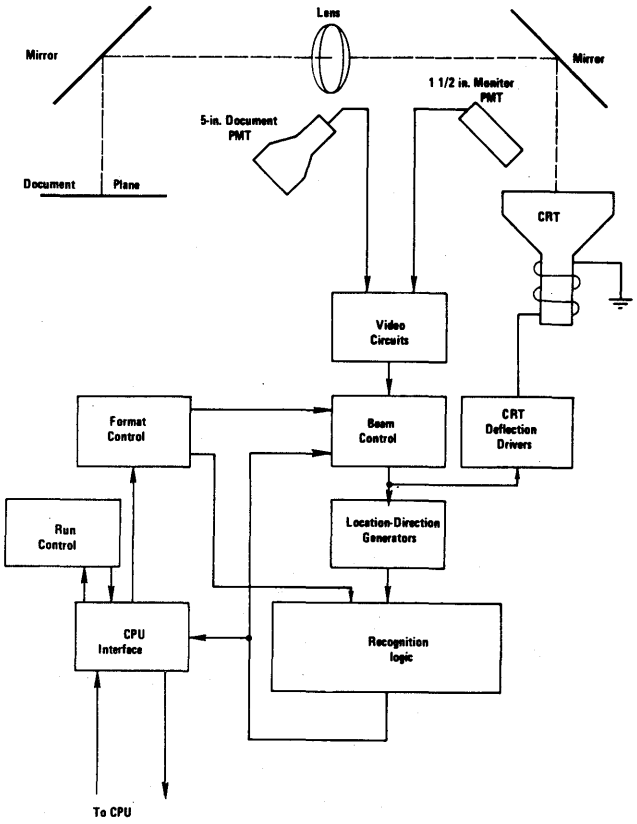


FIGURE 2—Schematic of scanner and format control

scanner housing is mated at the document transport plane to a vacuum bed plate. A clutched belting arrangement transports the document to the vacuum bed plate through a pair of light-seal brushes.

Within the scanner housing, the CRT spot of light is reflected 90 deg. by the first mirror, and magnified and focused at the document plane by the lens via the second mirror. Light reflected from the document is sensed by the document PMT which, in turn, provides an intensity-modulated signal to the video amplifier.

The CRT spot is constantly and directly monitored by the monitor PMT. The state of the art of phosphor deposition on CRT screens precludes a perfect screen; therefore, variations in spot intensity at the CRT screen, which are caused by phosphor graininess, are detected by the monitor PMT and the output from this PMT is subtracted (in the video amplifier) from the output of the document PMT.

The video circuitry amplifies the video signal, compensates for long and short term drift in scanner components, and stores peak electrical excursions for clipping functions and other analog-to-digital conversions. The output of the video circuitry goes to beam control.

Beam control is that portion of the logic which maintains control of the CRT beam at all times. This block of circuitry receives input from the format control circuitry and from the recognition logic, as well as from the video circuitry. The beam control provides input to the CRT deflection drivers and to the location-direction generators.

The feedback of beam control data to the CRT deflection system completes the optical-electrical flying-spot scanner loop. The CRT beam action, therefore, is really dependent upon what the document PMT had seen as a result of beam action a half-microsecond earlier.

The location-direction generators, which provide basic vital data to the recognition logic, will be described in detail in connection with data extraction.

The beam control circuitry can be controlled to generate sine and cosine functions in such a manner as to produce a circle scan (Figure 3). In addition, the circle size can be selectively attenuated while in black so that scanning progresses along a black edge. If the circle is allowed to proceed unattenuated at diameter X until it strikes black, and is then attenuated to a diameter of X/2 for

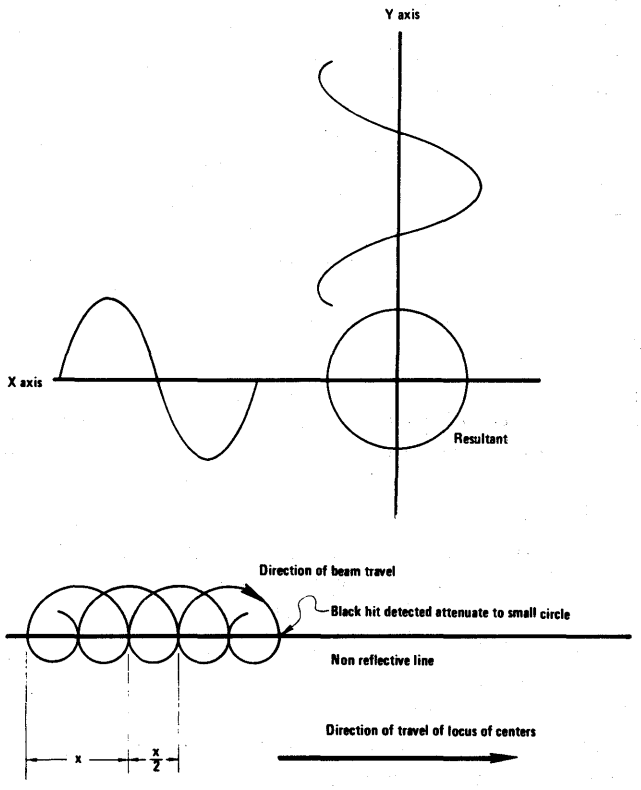


FIGURE 3—Circle scan method

180 deg., and the process repeated, the scanning along a line will progress at the rate of X/2 per circle.

Two modes of circle scan are utilized by the 1287 in the NHW mode. The search for the first character in the field is accomplished by forcing a horizontal right-to-left circle scan. As real black video is encountered, curve following is initiated with the full flying-spot scanner feedback loop in operation. The result is that the beam progresses generally clockwise around the outer contour of the character.

Many of the 1287 functions are under program control. These include the run control, the scanning format, direction of scan, type fonts to be read, reading modes to be employed, and special format conditions. Two-way communication between the 1287 and the CPU (normally System/360 Models 25 through 50) is handled by the CPU interface circuitry. This control may best be illustrated by following a sequence of events in the processing of a document.

When the CPU has determined that the 1287 is in a "ready" status, as indicated by the run control (via the CPU interface), a command is issued to the run control to feed a document. The run control then takes over and feeds, separates, and aligns the next document and properly registers it in the light-tight scan station. The CPU is then advised by the run control that a document is in the read station and the 1287 is prepared to scan the document.

The CPU then issues a series of format words, one for each field of data to be scanned, as required by the 1287. Each format word contains four hexadecimal bytes. The first 3 bytes contain the vertical and horizontal start, and the horizontal stop respectively. (The CRT beam can be addressed to any set of coordinates within a 256 $\times$ 256 grid.) The fourth byte identifies the type font to be read in that field, determines the scan direction, how blanks should be transmitted, and whether on-line or off-line correction is to be effected should a reject character appear.

It should be noted that the first format word transmitted for a new document results in addressing the document reference mark. A special hardware routine is initiated to locate the edges of the reference mark with respect to the coordinates defined in bytes 1 and 2, and to generate a registration error voltage which is used to modify the coordinates of all fields subsequently addressed on this document.

As this information is being transmitted from the CPU, it is stored in the format control circuitry—the first three bytes in a digital-to-analog converter and the fourth byte in digital storage. At the conclusion of the transmission, the scanner is ready to begin a seek operation to a point to the right of the field to be scanned, as defined by bytes 1 and 2.

When the CPU replies with a signal indicating that it is ready to accept the transmission of recognition information, the format control signals the beam control to seek to the XY coordinate indicated by the first two bytes. When the beam reaches that point, a right-to-left artificial follow is initiated by the beam control to search for the first character in the field. When the last character in the field has been scanned and the beam has progressed horizontally to the location indicated in the byte 3 of the format word, the beam is forced into a large raster or idle mode.

The beam remains in large raster until the CPU transmits the format word for the next field to be scanned. This process is repeated until all fields on the document have been scanned. At that time, the CPU transmits a document eject command. The run control executes the command by moving the document just scanned into the stacker transport and by moving the next document into the scanning station.

Data extraction process

Because of the wide variability of handwritten numerals, it is highly desirable to extract a relatively small number of measurements that are highly insensitive to minor variations in the shape of the same character. On the other hand, these measurements must dichotomize the character set in order to have discrimination value. There is a very practical limit, however, on how far a given measurement can go in meeting these two requirements. As we shall see later, if the variation in shape for a given character is allowed to degrade beyond a reasonable limit, it can no longer be distinguished from the degraded form of another character.

A form of measurement that generally meets these requirements is a sequential arrangement of combinations of locations and vectors (or directions) generated as a result of contour following. The process of contour following with a circle scan and selectively attenuated circles has been described.

The handwritten numeral is found on the document by means of the artificial follow search rou-

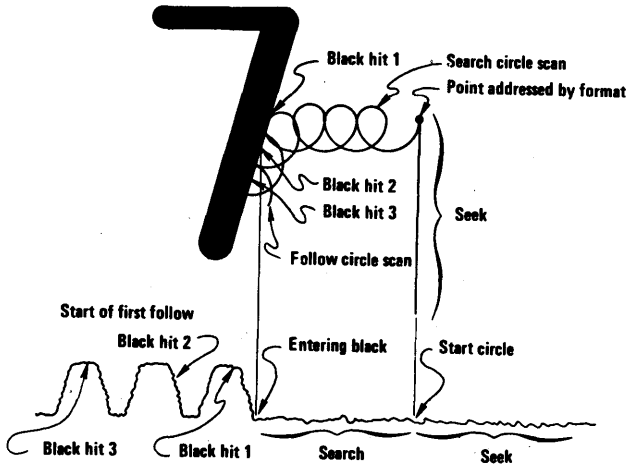


FIGURE 4—Contour following a character

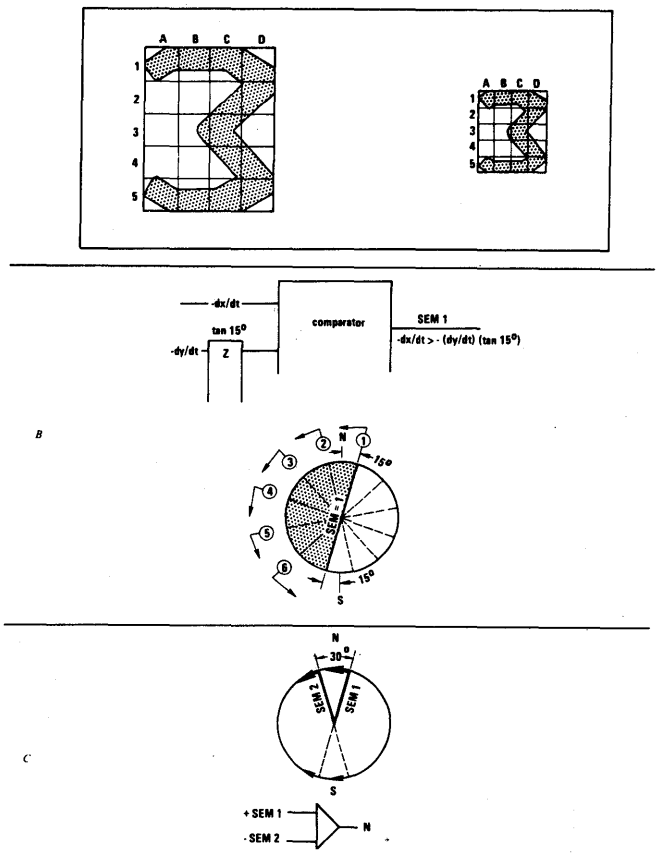


FIGURE 5—A four by five matrix
B Voltage comparator
C "ANDing" Sem 1 and Sem 2

tine (Figure 4). When real video data is encountered, the video-deflection feedback loop causes the beam to follow the outer contour of the character in a clockwise direction. The beam is permitted to make a "first follow" or complete trip around the character to exactly locate the character, normalize the measurement circuitry to the size of the character, and optimize threshold circuitry. The follow circle scan frequencies are filtered out of the deflection signals thereby providing signals representing the locus of circle centers for subsequent recognition measurements.

During the "first follow," the filtered horizontal and vertical deflection signal excursions about a zero reference voltage are tracked, and their extremes are stored. The stored voltages, representing the horizontal and vertical extremities of the character, are applied across resistor divider networks which divide the horizontal dimension into four zones (A, B, C and D) and the vertical dimension into five zones (1, 2, 3, 4, and 5) (Figure 5A). During the "second follow," location information is provided by comparing the current referenced deflection voltages to the levels stored in the four by five matrix.

The instantaneous direction of travel, with an angular resolution of 30 deg., of the circle center is obtained as follows: The filtered XY deflection signals are differentiated to generate the signals +dx/dt, —dx/dt, and —dy/dt. Each of these three signals is then effectively multiplied by the tangent of 15 deg. Voltage comparisons of appropriate pairs of these six signals yield six digital signals which represent six semi-circles, each one displaced 30 deg. from the next. For example, the output of the voltage comparator comparing the amplitudes of —dx/dt and —dy/dt (tan 15°) is active whenever the direction of circle center travel is in the semi-circle bounded by 15 deg. west of due south to 15 deg. east of due north (refer to the shaded area in Figure 5B). Any particular 30 deg. direction segment, or any combination of 30 deg. segments, can be generated by "ANDing" two of the 12 signals composed of the six semi-circles signals and their inverses. The 30 deg. segment representing "north" is generated by "ANDing" together + (semi-circle 1) and — (semi-circle 2) as illustrated in Figure 5C.

In order that information about the duration of travel in a particular direction might be utilized for characters of varying size, the time to follow around a second time is normalized by making the follow circle scan diameter linearly dependent on the amplitude of the stored voltage representing the character height.

Two types of measurements, known as feature tests and supplementary scans, are made on the character.

Feature tests make use of the parameters of

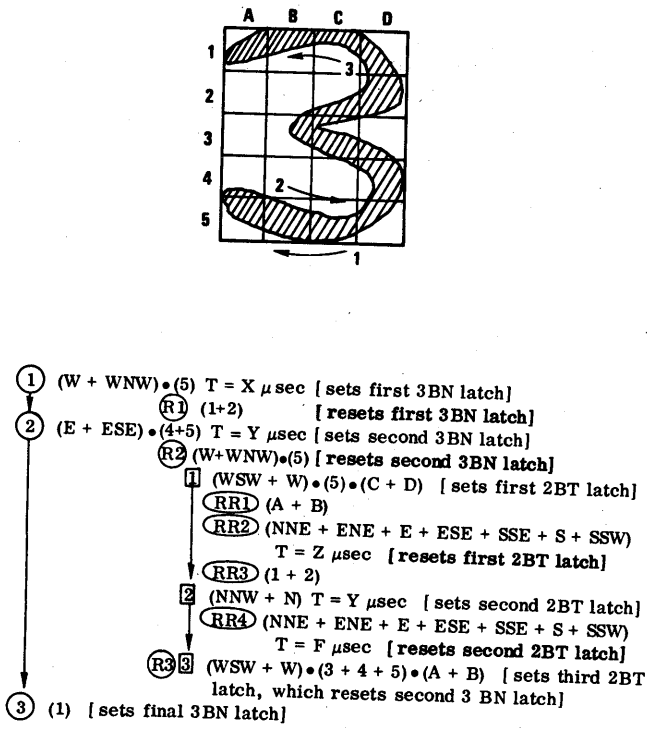


FIGURE 6—Typical feature test

location, direction, time, and sequence while following the outer contour of the character to identify character feature shapes. These feature shapes are subordinate to an entire character shape and many are common to several characters.

A typical feature test will be described in detail to illustrate the manner in which position and direction information is used (Figure 6). This feature test is called 3BN—"three bottom, normal"—because it is intended to recognize the most common variations of the bottom horizontal stroke of the numeral 3 (or 5). Three sequential conditions are required to satisfy 3BN. They must all occur during "second follow." First, the circle center must travel west or west-northwest in matrix row 5 for x microseconds. Next, there must be y microseconds of travel east or east-southeast in matrix rows 4 or 5. If the beam travels into matrix rows 1 or 2 before satisfying the second condition, the first condition is reset, and must be satisfied again to reinitiate the sequence. This "reset" condition prevents such shapes as "open topped" zeros or ones with a tick on the right side from passing the test. The final requirement is that the beam travel into matrix row 1.

There are two events that will reset the second condition if they occur before the final requirement is met. They are, 1) if the beam travels west or west-northwest again in row five, which prevents the test being prematurely passed during "first follow," and, 2) if the third condition of the feature test "two bottom-like-three" (2BT) is met. This last reset condition makes use of a portion of the feature test designed to recognize two bottoms that droop down toward the lower right, somewhat like the upper right hand portion of a normal three bottom. This sort of thing is done fairly often throughout the feature test logics. Sometimes, instead of using part of another feature test, the reset condition is itself sequential. The only reason for asking for row 1, as the last requirement for 3BN, was to make it possible to implement the last two reset conditions.

Nearly all recognition decisons can be made on the basis of only feature test data. Only when the feature tests do not yield a high-confidence recognition decision are supplementary scans employed to "look inside the character" for additional information. Some of the conflict pairs that can be resolved by supplementary scans are the skinny zero vs the fat one; a nine vs an eight with a narrow bottom loop; a zero vs an eight without dimples on the sides; and, a fat-topped seven vs a thin-topped nine. The first two examples would be resolved by a horizontal supplemental scan looking for two black hits vs one hit. Similarly, the third and fourth examples would be resolved by a vertical supplementary scan.

The recognition decision

Compared to what has been described thus far concerning beam and format control, trigonometric scan patterns, generation of locations and directions, and the development of feature tests, the recognition decision logic may appear to be somewhat trivial. Basically, it is a combinatorial logic statement for each character in the set, operating on the output of the feature test circuitry, that provides the decision directly (unless a conflict must be broken by the use of a supplemental scan). Based on the characters involved in the conflict, the recognition logic will select the type of supplemental scan required. The logic statement for any given character usually consists of a number of paths representing the most commonly encountered variations of that character's shape. For example, the logic tree for the nine logic includes a path for nines with open tops (9), a different path for nines with an opening on the left side (9), another path for nines with a three-like bottom (9), as well as paths for the more con-

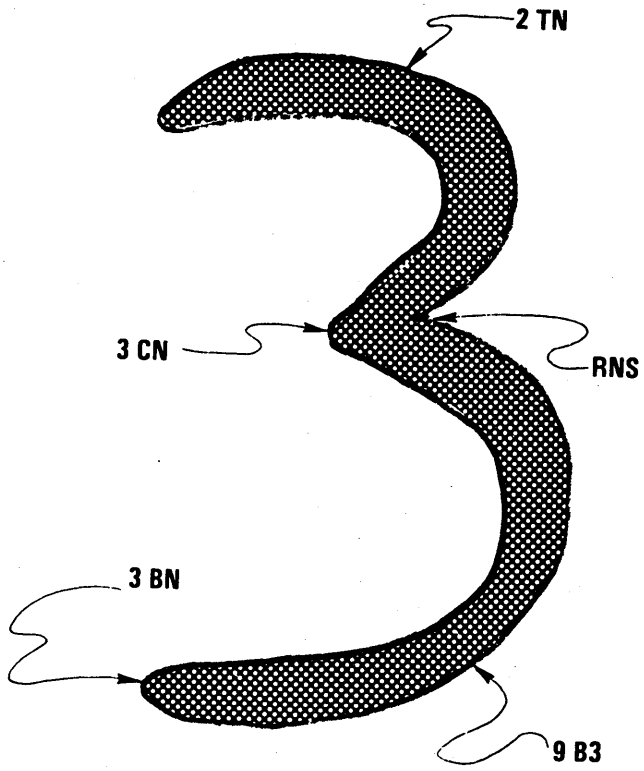


FIGURE 7—Feature tests satisfied by "3"

ventional closed loop-straight stemmed nines.

Figure 7 shows a well-formed character 3 and some of the feature tests which it satisfies. A fairly horizontal top start produces 2 TN (2 Top Normal) which is commonly found in the characters 2, 3, and 7. A number of other feature tests, representing a variety of other top start conditions would be just as acceptable. The RNS (Right Notch Straight) is a strong characteristic in the character 3, and to a lesser degree in the 8. The 9B3 (9 Bottom Like 3) is very common for the character 3 and some forms of the 9. The 3BN (3 Bottom Normal) ending is typical for the five and some forms of the nine. All other normal 3 endings also yield acceptable feature tests in this area. The 3CN (3 Cusp Normal) adds considerable weight to the decision for the 3.

This diagram shows only those feature tests satisfied by this particular shape of the character 3. There must obviously be a number of feature tests that remain unsatisfied in order to prevent other character shapes from substituting (or satisfying the 3 logic). In the same manner, some of the sequence tests that are strongly satisfied by the 3 should show up as inhibit conditions in the logics for other characters.

When a character has been recognized, the beam is directed to a point immediately to the right of right center (referred to as the initial point). Right-to-left artificial follow is then initiated with the video blanked until the beam has passed the area occupied by the character just recognized. Artificial follow then continues until the next character is encountered. If the recognition logic is unable to make a decision based on available measurement data, the beam is returned to the initial point and a second recognition attempt is made. Up to two rescans are made, with clipping level adjustments made in instances where specific degradations of the character are indicated. If the character cannot be confidently identified after two rescans, the CPU is so informed and the recognition process is continued on the next character.

Document format and writing implements

This section presents a typical handwriting document format showing the field and character locators determined to be optimum by a series of extensive human factors experiments and field studies, and discusses the proper choice of writing implements.

Document format

Figure 8 is one possible format for a retail sales check. The document reference mark, which can be located anywhere on the document within the scanning area, is used to provide the link between all field locations on the document and the format control system, and to compensate for variables such as paper cutting, printing registration, and document stopping tolerance. On this document, the reference mark is the dark "L-shaped" mark in the upper right-hand corner.

All the fields on this document are horizontally aligned, but may be both horizontal and vertical on the same document if this is desired in the application. The only requirement on the placement of data fields is a 0.125-in. clear band between any edge of the field and the edge of the document or another field. Each field is comprised of one or more character locators, which are rectangular boxes printed in a reflective ink (shaded area in Figure 8). As determined by human factors studies on how people normally and comfortably write, the aspect ratio of the boxes should be three-to-four (width to height), with the height ranging from 0.240 to 0.320 in.

A space of 0.020 in. between boxes is recommended in order to minimize linking of adjacent

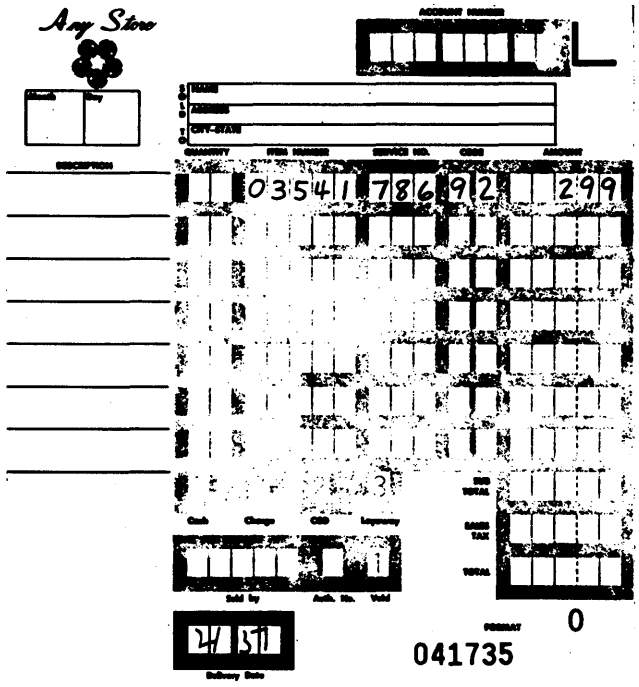


FIGURE 8—Possible format for retail sales check

characters. The lower left-hand field of Figure 8 illustrates the confusion that can result both to the human reader and the OCR reader when handwritten characters are linked. The field was intended to contain the digits 2 1 5 1. Without the preconditioning that the boxes provide, the field could easily be interpreted as 4 5 1, or even 4 3 7.

Constraints other than a rectangular box were considered and tested early in the development of the current technology, but these were discarded because of the additional burden placed on the writer and their interference with normal writing habits. Generally, it was concluded that the fewer rules the writer must remember about the formation of each character, the greater will be his acceptance of recommendations for good writing discipline.

Writing implements

The ideal writing implement should provide a continuous stroke of uniform width and density throughout the character. The written character should be largely independent of the pressure applied to the writing implement, should be highly insensitive to smudge and smear, and should have a minimum of extraneous non-reflective matter.

A number of writing implements were investigated for their suitability in providing the desired characteristics. No known implement studied consistently provided all of these characteristics, and therefore the selection of the implement was a matter of a best compromise.

Table I presents the conclusions of an evaluation of a variety of fine-line pencils and pens in terms of the characteristics desired in a writing implement. A plus (+) in the matrix should be interpreted as an advantage for that implement, a minus (—) as a disadvantage, and a zero (0) as no net advantage or disadvantage. Note that it is next to impossible to assign a value to each of the pluses and minuses, and that they have widely varying values. Inasmuch as a minus in only one characteristic could nearly rule out a given implement, the scores are not directly additive. It should also be noted that the applicability of a given implement varies with the application, the writing environment, the training emphasis of the writer, and other factors.

A primary conclusion is that the best overall writing implement is the fine-line mechanical pencil with a medium lead (No. 2-2½). All mechanical pencils have a minor maintenance disadvantage in that they can run out of lead, but normally the lead can be quickly replaced by a spare carried inside the pencil.

One runner-up is the wood pencil with a medium lead. Its disadvantage is that the point breaks or becomes dull (reliability), and needs to be sharpened (maintenance). A second runner-up is the fine-line mechanical pencil with a hard lead (harder than No. 2½). Its disadvantage is its lighter stroke density and the range of density, particularly under the condition of lightly applied pressure. This factor can be overcome to some extent, however, with proper emphasis on training.

Soft lead pencils have the disadvantage of susceptibility to smudge and smear and to leaving extraneous deposits of graphite, compared to medium and hard lead pencils.

Ball-point pens are characterized by their uniform stroke width and insensitivity to applied pressure on the one hand, and by skipping (lack of stroke continuity), smear susceptibility, extraneous ink deposits, and running out of ink (reliability) on the other hand.

The advantages of the fountain pen are its continuity of stroke and uniform stroke density. However, it tends to smear wet ink, and at times leaves a double track. It also runs out of ink (reliability) and needs to be filled (maintenance).

The felt pen has some key advantages, such as continuous uniform stroke (both width and density) and insensitivity to pressure. It has some

equally key disadvantages. Its (absolute) stroke width relative to normal character size is too large causing a general blobbing of the character. It leaves extraneous deposits of ink, and the pen tip dries quickly unless it is covered.

In most applications the ordinary medium wood pencil will satisfy the need for a writing implement. A medium lead mechanical pencil adds reliability and convenience. In applications where a more permanent recording is required, certain types of ball-point pens can be successfully used under proper conditions.

Human factors

Motivation

In most endeavors, motivation can make the difference between success and failure, or at least the difference between highly successful results and moderately successful results. Case studies

Desired Characteristic	Writing Implement									
	Wood Pencil - Soft (No. 2)	Wood Pencil - Medium (No. 2 - 2 1/2)	Wood Pencil - Hard (No. 2 1/2)	Mech. Pencil - Soft (No. 2)	Mech. Pencil - Medium (No. 2 - 2 1/2)	Mech. Pencil - Hard (No. 2 1/2)	Ball Point Pen - Easy Flow	Ball Point Pen - Other	Fountain Pen	Felt Pen
Continuous Stroke (constant pressure)	+	+	+	+	+	+	-	-	+	+
Uniform Stroke Width (constant pressure)	-	0	+	-	0	+	+	+	-	+
Absolute Stroke Width	-	0	0	-	0	0	0	0	-	-
Uniform Stroke Density	0	0	-	0	0	-	0	0	+	+
Smudge/Smear Resistant	-	0	+	-	0	+	-	-	-	0
Lack of Extraneous Deposits	-	0	+	-	0	+	-	0	-	-
Pressure Insensitive	0	0	-	0	0	-	+	+	0	+
Good Stroke Starting	+	+	+	+	+	+	0	-	0	0
Lack of Maintenance	-	-	-	+	+	+	0	0	-	0
Reliability	-	-	-	+	+	+	-	-	-	-

\+ This characteristic is an advantage of this implement.
0 This characteristic is neither an advantage nor a disadvantage of this implement.
\- This characteristic is a disadvantage of this implement.

TABLE I—Suitability of writing implements for NHW.

on the application of numeric handwriting as direct input to data processing systems have verified some interesting data. These case studies were conducted in a variety of industries to determine the effectiveness of motivation, training, and feedback, and the interrelation of these factors. In all cases, the gauge of effectiveness was actual performance on a prototype model of the IBM 1287 Optical Reader in reading documents generated in the case studies. Only the conclusions from the studies will be presented.

Motivation must first be instilled at the user's top management level. If the enthusiasm for and confidence in a program is generated there, and the need for positive involvement is communicated downward through all levels to the operational level, the probability of success is greatly enhanced. In the case studies where management interest and involvement were obvious to the lower level employees, the response was one of equal interest and enthusiasm. In post-test interviews that were conducted among participating highly unionized employees, comments such as "If it's good for the company, I want to do well" were common.

In general, the case studies demonstrated that highly motivated employees working under poor environmental conditions yielded results that were at least as successful as those produced by moderately motivated employees working under excellent environmental conditions.

Some employees were motivated by the mere fact that management was interested enough in the program to provide a training session on the writing of proper shapes, particularly when those shapes looked like the characters they had always written. Many others were motivated through properly applied feedback where they knew that the quality of their output was important to the success of the system.

Most frequently, the motivation was in some intangible form, such as the sincere desire to comply with a management plan, when that plan had been presented to the employees in terms of the benefit to be derived. Today, monetary rewards in the form of bonuses and discounts are offered as motivation in at least one application. However, most forms of motivation discussed here cost only a small amount of management time and effort, but they produce handsome payoffs.

Training

Because of the limitless variety of character shapes encountered in numeric handwriting, some limit must be imposed on the number of variations of a given character shape that will be allowed to define that character. Needless to say, the greater the number of variations allowed for any particular character, the more difficult and costly it becomes to maintain sufficient separation between the decision logic for that character and the logics for all other characters in the set.

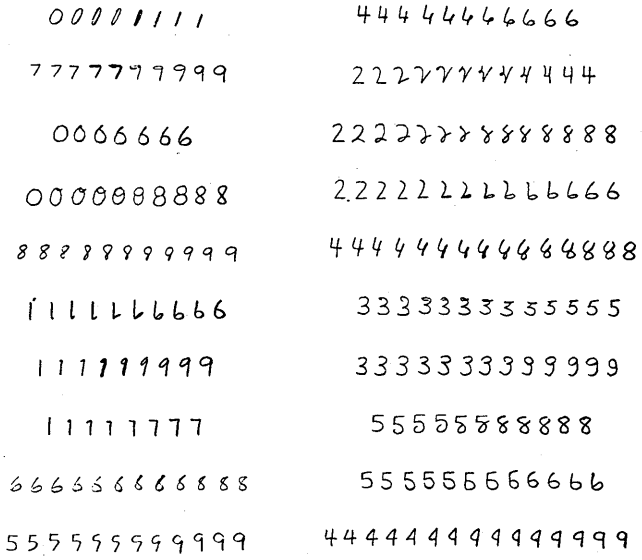


FIGURE 9—Examples of character degradation sequences

Figure 9 illustrates some common examples of character degradation sequences, in which progressively more degraded forms of one character pass through a "twilight zone" and emerge as progressively less degraded forms of another character. The few shapes on either end of the sequence are both human and machine sensible, but those falling toward the center of the sequence cannot be confidently recognized, particularly when they do not appear within a sequence that limits the choice. The only way to limit the variations in handwritten character shapes that must be handled by an optical reader is to motivate and train the writer to exercise a reasonable degree of care in writing.

In developing the 1287, a practical compromise was sought between recognition hardware complexity and the degree of care required of the writer. Happily, a compromise was found which requires relatively little conscious attention to writing rules. The "model" character set chosen as a handwriting "font" does not differ from the normal character shapes taught in most grammar schools. Furthermore, considerable effort was devoted to making allowances for the most com-

monly encountered reasonable deviations from the "model" shapes. For example, although the "model" shape of the numeral (2) does not show a loop at the junction of the bottom stroke with the upright stroke, a common variation of this shape which does have a loop at the junction (2) is accepted by the recognition logic.

Human factors studies indicated that some training would be beneficial to writers to help them achieve an optimum writing style. The small degree of constraint imposed, and the ease of learning the "model" shapes (which involved little more than a review of writing habits learned in grammar school) led to the development of a short, easy to understand self-instruction manual. This training is easily given, including supervised practice if desired, in less than 30 minutes.

The model shapes are shown on the top line of the Any Store document in Figure 8. It may appear unnecessary to require employees, who will be writing for direct entry into data processing systems, to take instructions on how to form handwritten numerals that are no different than were learned in elementary school. However, the primary purpose of the instruction was to impress the writer with these five basic rules:

- Write the character, just filling the box (the 1287 can handle characters that are less than half the height of the smallest recommended box height).
- Avoid linking characters.
- Avoid leaving gaps in line strokes.
- Write simple shapes without fancy strokes or curls.
- Make loops closed and rounded.

Feedback

The instruction session provides the writer with these basic rules and a feel for allowable deviations. Through immediate feedback during the instruction session he also receives an indication of his initial performance. As in all servomechanisms with feedback removed, the writer tends to become lax and sloppy if occasionally he does not receive some word about his performance. As indicated earlier, the feedback provides a motivating effect in demonstrating to the writer that his output has a bearing on the success of the total system. From a practical standpoint, feedback in the form of helpful hints from his supervisor will tend to correct "open-loop eights," "thin-topped nines," a tendency to write very light characters, or any other individual quirk.

Another type of feedback is the computer-generated report card, a listing of error documents by clerk or employee number. Table II is a relative performance chart illustrating the importance of motivation, training, and feedback for machine recognition of handwritten numerals. This chart shows only relative performance under the various

TABLE II—Relative reader performance as a function of motivation, training, and feedback

Level Of Supervision	Level Of Training	1. Very High		2. Average		3. Low	
		r	s	r	s	r	s
I Very Close	A Detailed initial training and continuing feedback	1	1	5	7.5	10	15
	B Detailed initial training; early feedback only, probably problem oriented	2.5	2.5	6	10	12.5	17.5
	C Initial training by manual; little or no feedback	4	4	9	15	15	20
II Loose to Moderate	A	2.5	2.5	7.5	10	12.5	20
	B	4	4	9	12.5	15	30
	C	6	5	11	15	20	40
III Little, if any	A						
	B	4	5	11	15	25	50
	C	7.5	15	15	20	30	75

combination of these factors. Each number may be considered a reject and substitution rate multiplier in relation to the rates obtainable under the most favorable conditions.

CONCLUSION

Any system involving the reading of handwritten characters is subject to a myriad of human variables. It is impossible for even the most sophisticated optical reader to compensate for all of them if no control is exercised. As indicated in Table II, the user frequently has the degree of freedom to provide the environment compatible with the system performance he desires.

Customer satisfaction with reading performance is highly variable with the application and the degree of control that the customer has over the quality of his input. Character reject rates of a fraction of a percent are achievable from input prepared by unskilled workers on a routine job in a controlled environment.

Figure 10 illustrates the range of character shapes beyond the "model" shapes that are read correctly by the IBM 1287 Optical Reader. Its significance is that the success of a handwriting reader application can be greatly enhanced if the reader is designed to read a broader range of character shapes than what the writer is asked to write.

In conclusion, it must be stated that the practicality of reading handwritten numerals by machine in a variety of industry applications is a demonstrated fact by virtue of wide customer acceptance. Its application in the future seems almost without limit.

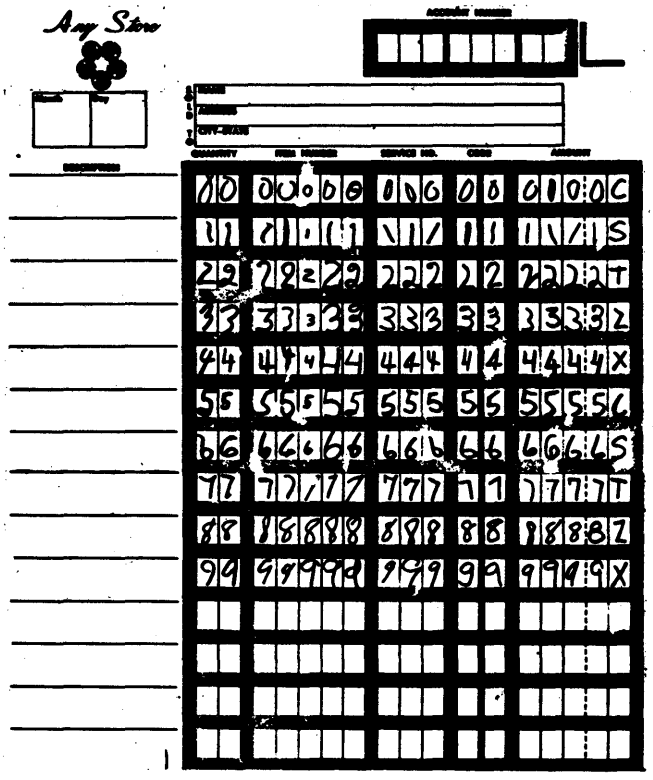


FIGURE 10—Range of character shapes beyond "model" shapes that are read correctly by the 1287

ACKNOWLEDGMENTS.

The authors would like to acknowledge Mr. Douglas C. Antonelli for his Human Factors contributions, and the many others from the Optical Reader Development departments of the IBM Systems Development Division Laboratory, Rochester, Minnesota, who helped develop the 1287 Optical Reader.

REFERENCES

1 D C ANTONELLI
Optical character recognition of numeric handprinting
Case Studies and Results, IBM Rochester Minnesota (unpublished report

2 T L DIMOND
Devices for reading handwritten characters
Proceeding of EJCC 232–237 1957

3 M N CROOK D S KELLOG
Experimental study of human factors for a handwritten numeral reader
IBM J of Research and Development 7 No 1 January 1963

4 E C GREANIAS et al
The recognition of handwritten numerals by contour analysis
IBM J of Research and Development 7 No 1 14–21 January 1963

5 N SEZAKI H KATIGIRI
Character recognition by the follow method
IEEE Proceedings 510 May 1965

The dynamic behavior of programs*

by I. F. FREIBERGS

McGill University
Montreal, Canada

INTRODUCTION

A computer system consists of several resources for which users' programs are competing: CPU time, primary and secondary storage and Input-Output devices. More and more computer systems, which are being marketed currently, allow for resource sharing among several jobs, in order to obtain a better utilization of all available equipment, or to provide better service to the users by reducing their job "turnaround" time, or both.

Some of the approaches tried so far in order to achieve these aims are:–

multiprogramming, as in Univac 1108, IBM 360, i.e., the sharing of core memory by several users' programs;

remote-access computing via teletype-like terminals, as in the RAX system on IBM 360, or Dartmouth College's time-sharing system on GE265, i.e., "slicing" of the CPU's time between the terminal users;

on-demand paging. Only the active parts of programs, or pages, are kept in core memory, as in CDC3300, SDS940, IBM 360/67, GE645. This is achieved by dividing up the available core memory into blocks of a certain size. Blocks of storage might then be allocated to program pages on an "on-demand" basis.

The performance of the above three types of systems becomes difficult to estimate, as compared to batch processing systems, since each program can no longer be considered per se, but is heavily dependent on the priority and scheduling rules of the operating environment, as well as on the type and volume of other programs to be processed. In time-shared systems, the performance will also be influenced by the size of the time-slices and by the resulting systems overhead for program swapping between core and auxiliary storage. In paged systems additional performance parameters are introduced by page size and page turning strategies.

Such systems are generally considered to be too complex and non-deterministic in nature for analytical study methods. The alternative for studying these systems is simulation, validated by subsequent direct measurement.

As an exception, it must be pointed out that remote-access low speed terminal systems have been studied by queueing theory. A summary of the models and the measures used is available.[1] Simulation models of remote-access systems have also been developed, at MIT,[2] and at SDC.[3]

Multiprogramming on Univac 1107[4] as well as an IBM 7094/44 directly-coupled system [5] have been simulated.

The IBM 360/67 system has been simulated at Stanford.[6] Paging strategies have been investigated at IBM, both experimentally using the M44/44X system,[7] and by simulation.[8]

In all such models, and particularly in ones dealing with page-turning, crucial assumptions have to be made about the dynamic behavior of programs under execution, in particular about frequency of references to data in memory, and about the distance between successive instruction and data fetches. It has been said that page-turning can be either very useful or disastrous, depending on the type of program to which it is applied.[9] Up to now, few direct investigations have been carried out, and performance estimates of systems, such as the IBM 360/67, were based on certain theroretical assumptions about the behavior of compilers and of the users' programs of different classes. Recently, at SDC a study of actual memory requirements and simulated page demand rates has been carried out for a fixed page size of 1024 words on some interpretive type programs (LISP, META5), with some interesting conclusions.[10]

In the Spring of 1966 it was decided to obtain em-

*This research was supported by the National Research Council of Canada through Grant A-4081.

pirical data about the dynamic behavior of programs, typical to McGill University Computing environment, from the point of view of CPU time and memory utilization.

The data thus obtained would also be used as input to subsequent simulation studies about various paging and scheduling strategies.

Method

A trace-like program was developed for the IBM 7044 which executes interpretively any program, instruction by instruction, recording for each the operation code and the actual address of the instruction. For instructions referencing data in memory, the actual address of the data word was also recorded, together with an indicator as to whether it was a data fetch or a data store.

Supervisor calls (SVC) were not executed interpretively, since it was felt that the design philosophies behind supervisor routines would differ too widely, so that the IBM 7044 supervisor routines would not have a sufficient degree of generality for subsequent study of other systems. Instead, the entry point in the supervisor was noted, and the time spent in the supervisor was measured in clock pulses of 1/60th of a second each. From the entry point it was then possible to determine the type of SVC which caused the suspension of program execution. Most of the SVC's occurred for input or output (I/O) operations.

Since the programs investigated were not specifically organized for execution with paging, dividing the core memory into 32 equal size pages of 1024 words regardless of program organization would result in an overestimation of page boundary crossings. For this reason a memory map, or layout, was taken for each program investigated, indicating its functional parts, namely the regions occupied by the supervisor, the input-output control blocks and buffers, the problem program, its subroutines, system-supplied subroutines, common data areas, etc. Memory was then subvided into a variable number of pages of 1024 words or less, but making sure that each functional part begins on a new page boundary. Table I shows a typical memory layout with page allocation. Since any program would consist of the above functional parts, regardless of the computer system used, it was felt that the results of this investigation would be of more general value, beyond the IBM 7044 memory organization.

The output of the trace-like program was recorded on magnetic tape, and was subsequently analyzed and summarized by an analyzer program.

After various attempts, the best way to summarize in pictorial form the information obtained was found to be a series of "Snapshots" of memory between successive supervisor calls. Typical output of the analyzer program is shown in Table II, which shows the sequence of steps for an in-core Fortran compiler.

TABLE I—Memory layout of an in-core Fortran compiler

Page No.	Description	Symbol	Page Size (Words)	Page Address (In Octal) Start	End
1	File Control	F	101	14323	14467
2	Control PGM	C	1024	16113	20112
3	Control PGM	C	567	20113	21201
4	Loader	L	1024	21202	23201
5	Loader	L	1024	23202	25201
6	Loader	L	1024	25202	27201
7	Loader	L	387	27202	30004
8	Program	P	1024	30005	32004
9	Program	P	1024	32005	34004
10	Program	P	1024	34005	36004
11	Program	P	1024	36005	40004
12	Program	P	1024	40005	42004
13	Program	P	1024	42005	44004
14	Program	P	1024	44005	46004
15	Program	P	1024	46005	50004
16	Program	P	1024	50005	52004
17	Program	P	1024	52005	54004
18	Program	P	1024	54005	56004
19	Program	P	1020	56005	60000
20	Buffer	B	941	76122	77776

TABLE II—"Snapshots" showing memory utilization between successive SVC's for an in-core Fortran compiler

F C C L L L L P P P P P P P P P P P P B*	Time In Msec.	Interrupts Cause	Interrupts Clock Pulses
P F I . . . S . . . . . I	0	Load	165
. S G . . . . P P P . . P S P . P . . .	16	I/O	0
P S S . . . . P S S . . P S . . G . . .	1	GET	0
P S S . . . . P F S P . P S . . G . . G	2	PUT	1
P S S . . . . P F . . . . . P	1	PUT	0
P S S . . . . P I S . . . . . G . . P	1	PUT	0
P S S . . . . . F . . . . . P	1	GET	0
P S S . . . . P I S . . . . . G . . G	1	PUT	0
P S S . . . . . F . . . . . P	1	GET	0
P S S . . . . P S S S P P S . . P . . G	7	PUT	1
P S S . . . . . F . . . . . P	1	GET	1
P S S . . . . P S S S . S . . . G . . G	4	PUT	0
P S S . . . . . F . . . . . P	1	GET	0
P S S . . . . P S S S S S P . . P . . G	9	PUT	1
P S S . . . . . F . . . . . P	1	GET	0
P S S . . . . P S S S S S P . . P . . G	10	PUT	0
P S S . . . . . F . . . . . P	1	GET	0
P S S . . . . P S S S . S S . . P . . G	11	PUT	0
P S S . . . . . F . . . . . P	1	GET	0
P S S . . . . P S S S S S S . . P . . G	19	PUT	0
P S S . . . . . F . . . . . P	1	GET	0
P S S . . . . P S S S I S . . . P . . G	2	PUT	0
P S S . . . . P S S F P P S S S S . . P	62	PUT	1
. P . . . . . G G P . S S . . .	34	PUT	0
. G . . . . . G G G . S S . . .	18	PUT	0
. G . . . . . G G G . S S . . .	5	PUT	0
. P . . . . . G . S S . . .	2	PUT	0
P S S . . . . . S F . . .	1	GET	0
P S S . . . . . G	1	TYPE	23

* Symbols for page identification as per Table I.

A header line identifies the pages of each functional part of the program. Each snapshot line begins with a layout of memory, split up into pages as described above. The meanings of the symbols used in the snap-

shots are (for each page) :

•	page not used during this interval	
I	Instructions fetched	
F	Instructions and data words fetched	Page content unchanged
G	Data words fetched	
P	Data words stored and fetched	
S	Instructions fetched and data words stored and fetched	Page content changed

Next the time (in msec.) spent in uninterrupted computation before the SVC occured is given. The average instruction time for the IBM 7044 was found to be 4.4 microseconds. Finally, the reason for the supervisor call is indicated with the number of clock pulses spent in the supervisor before program execution was resumed.

Snapshots were chosen between successive SVC's rather than for a fixed number of instructions, because at each SVC the scheduling algorithm has the option of deciding whether to continue with the present program or to switch to a different one.

Results

A summary of the classes of programs investigated so far is shown in Table III. This table also shows a percentage breakdown by type of instruction within each class of programs. Of interest here is the 34% average proportion of branch instructions, which may result in references to instructions on a different page in memory. Also of interest is the 51% average ratio of data words to instruction words.

TABLE III—Classes of programs investigated

Class of Program	Total No. Of Instructions Traced (In Millions)	Percentage of Different Types of Instructions				Ratio Of Data Words To Instruction Words (%)**
		G (%)	P (%)	R (%)	B (%)	
FORTRAN Execution (11 programs)	4.02	36	20	11	33	57
String Processing	.28	38	23	15	24	61
Simulation (GPSS)	1.29	28	27	20	25	55
List Processing (SLIP)	1.13	26	23	19	32	49
FORTRAN Compilation In-Core Compiler (7 Programs)	1.74	28	22	14	36	54
Cobol Execution	1.89	28	15	14	43	45
AVERAGE		29	22	15	34	51

G = Instructions requiring a data word fetch in memory. (5 microseconds)*
P = Instructions requiring a data word store. (5 microseconds)
R = Instructions referring to registers only. (3 microseconds)
B = Branch, or jump, instructions. (3 microseconds)

* Average instruction time for IBM 7044.
** Data words G+P+MOVE instructions

The percentages of Table III will vary somewhat, depending on the instruction repertoire and on the number of registers available in a particular computer. For instance in the IBM 360 with 16 general purpose registers, as opposed to 2 Arithmetic registers and 3 Index registers in the IBM 7044, the proportion of instructions referencing data in memory was expected to be reduced in favor of more register-to-register type instructions.

To verify this some IBM 7044 programs were rerun with 360 Fortran G and Cobol F. The percent increase in the proportion of R type instruction was found to be between 40% (Fortran) and 340% (Cobol). The percentage of B type instructions was decreased between 56% (Fortran) and 64% (Cobol) to compensate for this. The proportion of branch instructions for which the branch is taken was found to be around 90% on the 360. The instruction times obtained in this investigation can then be adjusted accordingly.

By comparing the "snapshots" of the various programs investigated, several results emerge.

A program does not execute very many instructions between successive SVC's. This is illustrated in Figure 1, which shows the cumulative percentage of the number of SVC's vs. the number of instructions executed between successive SVC's. For most classes of programs 50% of the time this number lies somewhere between 100 and 1000 instructions, i.e. less than 5 msec.

Figure 2 shows the cumulative percentage of SVC's vs. the number of instruction pages required between successive SVC's. It can be seen that this stabilizes around 2 to 3 pages, except for the in-core Fortran compiler which requires 5 pages most of the time.

Data page requirements are higher than those for instruction pages and practically coincide with total page requirements. This is illustrated in Figure 3. Most programs need 4 to 6 data pages between successive SVC's with the execption again of the in-core Fortran compiler, which requires between 5 and 12 data pages.

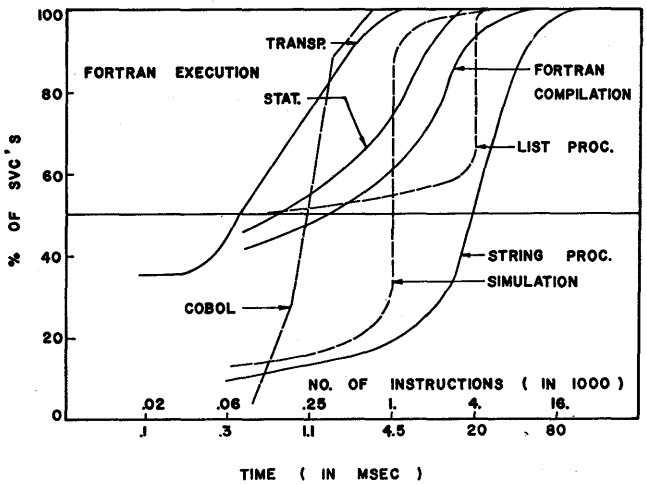


FIGURE 1—Cumulative frequency of the number of instructions executed between supervisor calls

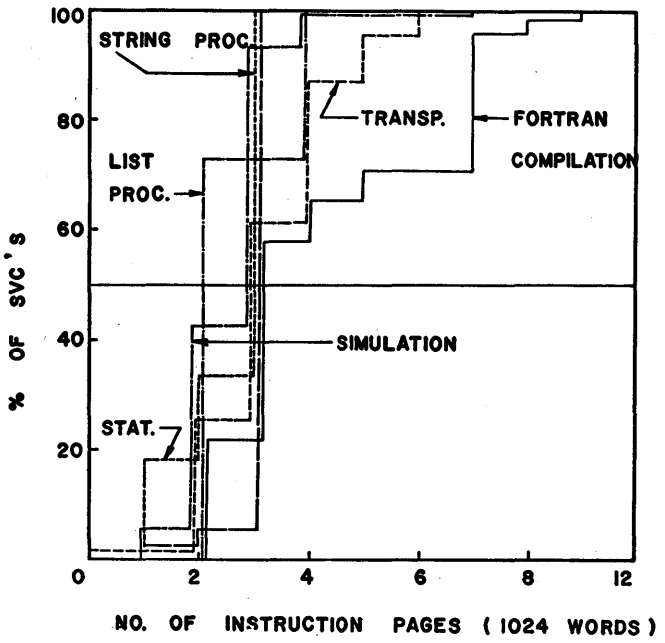


FIGURE 2—Cumulative frequency of the number of instruction pages required between supervisor calls

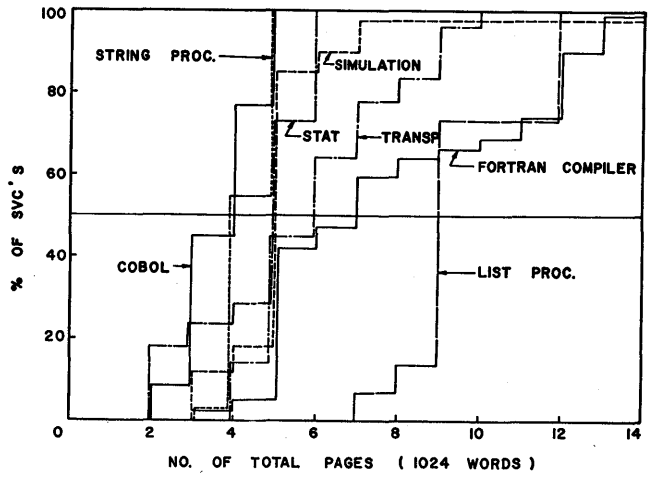


FIGURE 3—Cumulative frequency of the total number of pages required between supervisor calls

In order to obtain an estimate about the total memory requirements of the class of jobs processed by the McGill Computing Centre, all the jobs submitted during a certain 24 hour period were examined. For 127 of these jobs which reached the execution phase, a memory layout was obtained. The results of the analysis are shown in Figure 4, i.e., the cumulative percentage of the number of jobs vs. their total memory requirements (for the entire duration of the job).

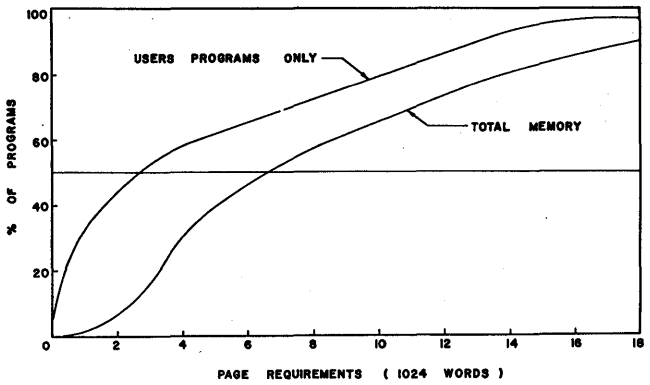


FIGURE 4—Cumulative frequency of the total number of pages required for a 127 job sample at McGill University

It can be seen that 50% of the jobs require less than 7 pages, made up as follows:-

Problem program including its subroutines	3 pages
System supplied subroutines	2 pages
I/O Buffer areas	1 page
Common data area (for larger programs)	1 page

DISCUSSION

Contrary to the popular view, long compute sequences without any interruption seem to be the exception rather than the rule. The number of instructions executed betweed supervisor calls is of the order of 10^2 to 10^3. Since most of the SVC's are due to I/O requests, assuming a blocking factor of about 10 with a fully overlapped I/O channel operation, return to the interrupted program could be immediate in 9 out of 10 cases. This would result in a sequence of 10^3 to 10^4 instructions (5 to 10 msc) about 50% of the time.

On the other hand, the corresponding memory requirements appear to be larger than generally expected, in agreement with the SDC study.[10] They are of the order of 2 to 3 instruction pages and 4 to 6 total pages. An exception here is the Fortran Compiler whose page requirements are about twice as large.

From these considerations it seems clear that the systems overhead for a "one-page-on-demand" strategy would be prohibitive. To avoid this, at least 3 pages for instructions and data, plus one page for I/O buffers, plus 2 pages for systems supplied subroutines (which could be shared code) should be made available to any program at the outset, as well as during each successive period of activity.

From Figure 4 it can be seen that such a memory assignment policy would accommodate the entire program of 50% of all jobs entering the execution phase, so that there would be no further paging overhead for the entire duration of these jobs.

Another question worth raising at this point is whether the additional cost of implementing a fixed page size hardware system is warranted, as opposed to a multiprogrammed system where the entire program is brought into memory.

These costs should also be weighed against those of acquiring more Large Capacity Storage, especially in view of the large memory requirements of Compilers.

SUMMARY

In this paper the results obtained from an interpretive instruction by instruction execution of different classes of programs (Fortran compilation and execution, Cobol, GPSS, SLIP) on an IBM 7044 have been presented.

Memory and CPU time requirements between successive supervisor calls have been analyzed, with the outcome that most of the time instruction sequences are rather short and more than one page (1024 words) of memory is required.

The data obtained can be used as realistic input to simulation models of multiprogrammed or fixed page size computer systems.

REFERENCES

1 G ESTRIN L KLEINROCK
Measures models and measurements for time-shared computer utilities
Proc 22nd ACM Nat Conf pp 85–96 August 1967

2 A L SCHERR
An analysis of time-shared computer systems
Research Monograph No 36 The MIT Press Cambridge Mass 1967

3 G M FINE P V McISSAC
Simulation of a time-sharing system
Mgt Sci 12 6 pp B180–B194 February 1966

4 G K HUTCHINSON J N MAGUIRE
Computer systems design and analysis through simulation
Proc of the 1965 Fall Joint Computer Conf pp 161–167 1965

5 J H KATZ
Simulation of a multiprocessor computer system
Proc 1966 Spring Joint Computer Conf pp 127–139 1966

6 N R NIELSEN
The simulation of time sharing systems
Comm ACM 10 7 pp 397–412 July 1967

7 R W O'NEILL
Experience using a time sharing multi-programming system with dynamic address relocation hardware
Proc 1967 Spring Joint Computer Conf pp 611–622

8 L A BELADY
A study of replacement algorithms for a virtual-storage computer
IBM Sys J 5 2 pp78–101 1966

9 J B DENNIS E L GLASER
The structure of on-line information processing systems
Proc of the 2nd Congress on the Information System Sciences p6 1964

10 G M FINE C W JACKSON P V McISSAC
Dynamic program behavior under paging
Proc of the 21st ACM Nat Conf pp 223–228 August 1966

Resource allocation with interlock detection in a multi-task system

by JAMES E. MURPHY

International Business Machines Corporation
Poughkeepsie, New York

INTRODUCTION

In a multiprogramming environment, special care must be taken to insure that, for a given physical record on a data set, only one task at a time is involved in the following sequence:

1) Read the physical record (with update in mind);
2) Find the logical record to be changed;
3) Change the logical record;
4) Write the "new" physical record.

For example, suppose there exists on a (direct access) data set a physical record, R, consisting of the logical records A, B, C, and D, and that there are two programs, P_1 and P_2, contending for the resources of the system.

Suppose further that the following sequence of events is allowed to take place:

1) P_1 reads R and obtains A, B, C, D in its buffer;
2) P_2 reads R and obtains A, B, C, D in its buffer;
3) P_1 changes A so that its buffer contains A^1, B, C, D;
4) P_2 changes D so that its buffer contains A, B, C, D^1;
5) P_1 writes the "new" physical record so that

$$R^1 = A^1, B, C, D;$$

6) P_2 writes the "new" physical record so that

$$R^1 = A, B, C, D^1.$$

As a result of P_2's action, the update to the data set made by P_1 has been erased as though it never existed. Any recovery via an audit trail at a later time would duplicate this error. This example demonstrates the general need for protecting the integrity of system resources and insuring validity of the work done by the resources' users.

There are three degrees of protection required by resources and their users in a multiprogramming environment. These are:

1) If a resource is to be altered by a task, then the entire sequence of events essential to the alteration of the resource must be protected from any interference which would affect the completion of the change.
2) If a task is using a resource but not changing it, then that task requires protection against any changes in the resource which would affect the validity of the information the task receives from the resource.
3) If a task uses a resource without changing it, then no protection is required when changes to the resource do not affect the ultimate use of information received from it, and the task in no way interferes with other users of the resource.

All tasks wishing to use a resource seek control of it by name through a resource management facility. Requests for a given resource are queued on a first-in, first-out (FIFO) basis. The concepts of shared (S) vs. exclusive (E) control are used. Any number of tasks may use a resource simultaneously if they have all requested shared control. A task requesting exclusive control will be the only task using the resourse once it gains control (except for those tasks falling into category (3) above). Tasks coming under category (1) would request exclusive control; tasks in category (2) would request shared control; and tasks in category (3) would use the resource with-out requesting control.

Embedded in the typical approach to resource allocation, however, are certain rigid constraints designed to avoid all possibility of system interlock, i.e., the situation in which two or more tasks place each other into a permanent wait state.

Suppose that a task is allowed to request control of a resource without releasing control of another resource

it has previously obtained. Here the expression "enqueue" (ENQ) will refer to the requesting of a resource, and "dequeue" (DEQ) will mean the releasing of a resource (both actions performed through queue management).

Assume the following sequence of events, where P_1 and P_2 are programs running concurrently in a multitask system and R_1,R_2 are resources:

a) P_1 ENQ's R_1 for E (exclusive control);
b) P_2 ENQ's R_2 for E;
c) P_2 ENQ's R_1 for S (shared control);
(At this point P_2 is placed in a wait state);
d) P_1 ENQ's R_2 for S (Now P_1 is placed in a wait state).

The queues for R_1 and R_2 would appear thus:

R_1	R_2
P_1, E	P_2, E
P_2, S	P_1, S

FIGURE 1

Now P_1 and P_2 are each waiting for a resource which the other holds. Under these conditions both programs would remain inactive until one or both were terminated by control programming, either through a timeout mechanism or other such safeguard.

Interlocks can occur only if a task is allowed to wait for a resource without releasing all resources it previously controlled. However, this is only a necessary and not a sufficient condition for interlock. Usually, resource control is handled as though the two conditions were equivalent, and tasks must request multiple resources in parallel or in some pre-defined sequence.

This paper discusses a more flexible resource management which examines each request in the context of all currently pending requests and determines the existence or absence of an interlock.

Queue management

For each resource, maintain a queue containing the name, mode, and rank for each task which has requested control of the resource but which has not yet released it. Within this queue:

TASK NAME identifies the task requesting use of the resource.

MODE refers to the type of control desired—"E" for exclusive, "S" for shared.

RANK indicates the task's relative position in the queue. This is assigned as follows:

A) Enqueuing
1) The first task in the queue is assigned control of the resource in the desired mode, and is assigned a rank of 0 of that resource.
2) If the new task requests mode S and the mode of the last task in the queue was also S, then the rank of the new task is equal to that of the old. Otherwise, the rank of the new is one greater than that of the old.
3) If its rank in the queue is 0, then the task may use the resource without waiting for another task to finish with it.
4) If its rank is not 0, the task's request for the resource must be set aside for analysis of interlocks.
5) This same procedure is followed for each of the resources requested simultaneously by the task.
6) Interlock analysis is now performed. The algorithm and the procedures followed when an interlock is detected are described later in this paper.
7) If the task is allowed to wait for the requested resources without releasing those it already controls, its name, rank, and mode are entered in the resource queues. In this case, the wait count of the task is increased by one for each resource for which it has a rank greater than zero.

B) Dequeuing
If a task releases control of a resource, then:
1) The name of the task requesting a DEQ is removed, along with its mode and rank, from the queue, for the specified resource.
2) If there are other tasks in the queue having rank zero, or if the queue is empty, the dequeuing process is finished.
3) The rank of each task in the queue is decremented by one, and the wait count is decremented by one for each task whose rank is now zero.
4) When the wait count for a task is zero, it is removed from the wait state and proceeds sharing or controlling resources in the requested mode.

Example: Let us suppose the following sequence of requests occurs for resource R.

1. Task B requests exclusive control
2. Task C requests shared control
3. Task A requests shared control
4. Task D requests exclusive control
5. Task E requests exclusive control

At the end of this time, the entries on the queue for

resource R would be:

NAME	MODE	RANK
B	E	0
C	S	1
A	S	1
D	E	2
E	E	3

FIGURE 2

When task B releases R, the entries become:

NAME	MODE	RANK
C	S	0
A	S	0
D	E	1
E	E	2

FIGURE 3

"Waits-for" as an order relation

Definition: Let the symbol, p_{ij}, represent the rank of task T_j on the queue for resource R_i.

Definition: If $P_{ai} > P_{aj}$ for some a, $1 \leq a \leq m$ where m is the number of resources being used in the system, then task T_i waits for task T_j to finish with resource R_a. The expression $T_i \overset{a}{>} T_j$ is defined to symbolize this relationship.

If T_i waits for T_j to finish with resource "a" and T_j waits for T_k to finish with resource "b", then we have $T_i \overset{a}{>} T_j$ and $T_j \overset{b}{>} T_k$, or $T_i \overset{a}{>} T_j \overset{b}{>} T_k$. Clearly T_i waits for T_k to finish with resource R_b because the *minimum* length of time T_i must wait for resource R_a is the length of time during which T_j controls resource R_a plus the length of time during which T_k controls resource R_b. This might be expressed in the form $T_i \overset{a,b}{>} T_k$.

However, in this paper we are only indirectly concerned with the resources for which a task waits. Our main interest here is that a task must wait. To determine for which other tasks a given task is waiting, requires knowledge of the resources involved. In line with this, make the following definition:

Let T_i and T_j be any two tasks currently resident in a multi-task system. Then say that T_i waits for T_j, expressed $T_i > T_j$, if and only if there exists a series of resourses, r_k, $1 \leq k \leq m$, and a series of tasks, T_h, $1 \leq h \leq m-1$, such that $T_i \overset{r1}{>} T_1 \overset{r2}{>} \cdots \overset{ra}{>} T_a \cdots \overset{rm}{>} rmT_j$.

Lemma: The relation "waits-for", ">", is transitive and is an order relation.

Note: Transitivity of the "waits-for" relation may be proven formally by introducing the variable, T, for time. Of significance here are the times when a task gains control of a given resource, when it releases the resource, and when the task gains control of the last resource for which it is waiting.

Definition: A (system) interlock is said to occur if there is a task, T_0, related by competition for resources to other tasks, T_i, in a system such that a chain of wait-for relations $T_0 > T_i \ldots T_m > T_0$ exists.

Matrix representation of "waits-for"

The ordering of the set, T, of the n tasks in a system by the relation ">" may be portrayed in an n x n matrix, W, the Wait Matrix, constructed as follows:

Let the n columns and the n rows be named by the tasks in the system, $T_0 - T_{n-1}$. Let the elements W_{ij} be assigned the value 1 if T_i has a rank higher than that of T_j on the queue for any resource R-i.e. $T_i \overset{R}{>} T_j$. If there exists no resource, R, for which $T_i \overset{R}{>} T_j$, then let $W_{ij} = 0$. Note that the elements along the main diagonal are all zeros.

Construction of the precedence matrix

For a more complete, and more complex, model of the task-resource system, construct a matrix, P, (the precedence matrix) as follows. Let the n columns be named by the tasks in the system, $T_0 - T_{n-1}$. Let the rows be named by the m different resources currently being used, $R_0 - R_{m-1}$. Let the m x n elements of the matrix $p_{ij} = (R_i, T_j)$, each contain the rank (or place) on the queue for R_i which has been assigned to task T_j. Clearly many of the elements will be empty (blank or null, as opposed to zero.)

Going across any row i, corresponding to resource R_i, we find an entry in column j, corresponding to every task T_j which is in the queue for R_i. A zero in column k of row i indicates that task T_k has control of resource R_i. Multiple zero entries in row i indicate that several tasks share control of that resource. Duplicate non-zero (and non-null) values indicate tasks which will share control at a later time. Similarly $p_{ik} < P_{ij}$ indi-

cates that T_k is ahead of T_j in line and that T_j will have to wait for T_k.

Going down any column j, corresponding to task T_j, we find an entry in rows $i_1,\ldots,i_e$ corresponding to every resource R_i which T_j controls or for which it is queued and waiting. An entry (rank) of zero in row k of column j indicates that task T_j now has the use of resource R_k. Non-zero (and non-null) entries in column j indicate those resources for which task T_j is waiting.

Illustrations

Assume a system with four tasks, $T_0 - T_3$, and six resources, $R_0 - R_5$, with all queues initially empty, and that the following sequence of requests has been processed by queue management:

1) T_0 request E for R_0 and S for R_5;
2) T_1 requests S for R_0, R_3, and R_5;
3) T_2 requests E for R_1 and R_2, and S for R_3;
4) T_3 requests E for R_3, R_4, and R_5.
5) T_0 releases R_5.

At this point, the queues for the six resources are shown in Figure 4.

SOURCE	R_0		R_1	R_2	R_3			R_4	R_5	
TASK NAME	T_0	T_1	T_2	T_2	T_1	T_2	T_3	T_3	T_1	T_3
MODE	E	S	E	E	S	S	E	E	S	E
RANK	0	1	0	0	0	0	1	0	0	1

FIGURE 4

Figure 5 summarizes the state of the system in a precedence matrix. In Figure 5, two tasks, T_0 and T_2, are not waiting for any resources and are able to process. T_1 is able to process when T_1 releases R_0. T_3, however, must wait to process until both T_1 and T_2 releases R_3 and T_1 releases R_5.

	T_0	T_1	T_2	T_3
R_0	0	1		
R_1			0	
R_2			0	
R_3		0	0	1
R_4				0
R_5		0		1

FIGURE 5

	T_0	T_1	T_2	T_3
T_0	0	0	0	0
T_1	1	0	0	0
T_2	0	0	0	0
T_3	0	1	1	0

FIGURE 6

The wait matrix is shown in Figure 6.

Interlock detection

To demonstrate the interlock detection procedure, continue with the system described in the previous section, using the state of the system shown in Figures 4 and 5.

Starting at this point, we will examine two cases as T_0 requests a resource. In each case, T_0 must wait for a resource to become available while another task is able to continue processing and T_0 and the remaining two tasks are in a wait state. In one case, as resources are released by the task which is running, subsequent tasks are able to process. In the second case, if the request were to be honored, the remaining three tasks would still be in a wait-state once the task which was still running released its resources. In this situation, only a new task could run, and it could only continue so long as it demanded no resources held by any of the three totally interlocked tasks.

Case 1: Suppose T_0 asked to share R_2. Figure 7 shows the resulting resource queues, and Figure 8 shows the new precedence matrix. ("Δ" indicates the affected entries.) The significant steps in determining if the request may be honored and T_0 allowed to wait for R_2 are as follows:

a) $T_0 \overset{R_2}{>} T_2$;
b) $T_2 \not> T_i$ for any value of i;
c) $T_0 \overset{R_2}{>} T_i$ for any $i \neq 2$;
d) $T_0 \overset{R_j}{>} T_i$ for any $j \neq 2$.

Thus there exists no sequence $T_0 > \cdots > T_0$, and the request may be honored without fear of interlock.

RESOURCE	R_0		R_1	R_2		R_3			R_4	R_5	
TASK NAME	T_0	T_1	T_2	T_2	$T_0\Delta$	T_1	T_2	T_3	T_3	T_1	T_3
MODE	E	S	E	E	S	S	S	E	E	S	E
RANK	0	1	0	0	1	0	0	1	0	0	1

FIGURE 7

	T_0	T_1	T_2	T_3
R_0	0	1		
R_1			0	
R_2	$\Delta 1$		0	
R_3		0	0	1
R_4				0
R_5		0		1

FIGURE 8

Case 2: Suppose that, instead of R2, T_0 requests shared control of R_4. The resulting resource queues and the precedence matrix are described in Figures 9 and 10. ("Δ" indicates those entries which are new).

Significant steps in concluding that the request can not be honored follow and are keyed to the arrows in Figure 10.

a) $T_0 \overset{R_4}{>} T_3$;
b) T_3 is waiting for control of R_3;
c) $T_3 \overset{R_3}{>} T_1$;
d) T_1 is waiting for control of R_0;
e) $T_1 \overset{R_0}{>} T_0$.

RESOURCE	R_0		R_1	R_2	R_3			R_4		R_5	
TASK NAME	T_0	T_1	T_2	T_2	T_1	T_2	T_3	T_3	$T_0\Delta$	T_1	T_3
MODE	E	S	E	E	S	S	E	E	S	S	E
RANK	0	1	0	0	0	0	1	0	1	0	1

FIGURE 9

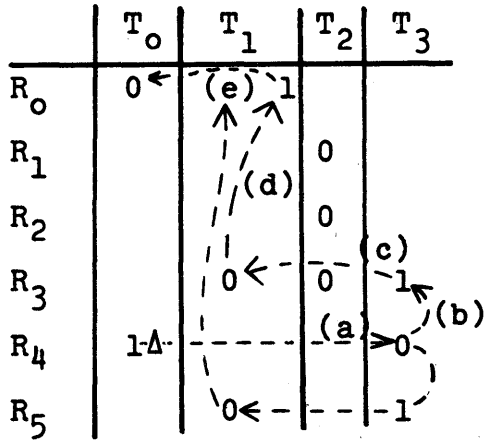


FIGURE 10

Thus there exists a sequence $T_0 > \cdots > T_0$ and the request must be denied. In this case the wait-chain is $T_0 > T_3 > T_1 > T_0$. If T_0 were allowed to wait for R_4 without first releasing R_0, the tasks T_0, T_1, and T_3 would be in a permanently interlocked wait state.

Since there is only one new resource requested and one old resource being held, no further paths need be followed. Note that there is an alternate branch which parallels the first path, but which does not lead to any new knowledge.

For a simpler view of the interlock, use the wait matrix shown in Figure 11. Scanning across the row from T_0, we find that $T_0 > T_3$. Scanning across T_3's row, we find $T_3 > T_1$. Scanning T_1's row, we see that $T_1 > T_0$. Thus $T_0 > T_3 > T_1 > T_0$, and an interlock exists.

It is apparent from even this simple example that an algorithm using the wait matrix would be far simplef to implement and use than would one based on the precedence matrix. However, since the information which can be provided with the wait matrix (i.e., whether or not an interlock exists) is only a subset of the information available by using the precedence matrix, this discussion will concentrate on the latter.

The detection algorithm

The algorithm to trace through the chains of wait-relations in the precedence matrix may be regarded as making a series of horizontal and vertical movements through the elements of the matrix. If a valid series of movements leads back to the column from which the series began, then an interlock has been detected.

The rules governing movements through the matrix are as follows:

1) Searching for positions to which to move is done down columns from the top and from left to right along rows.
2) Movement alternates in direction. A position occupied via a horizontal move must be vacated by a vertical move and vice versa.

	T_0	T_1	T_2	T_3
T_0	0	0	0	1
T_1	1	0	0	0
T_2	0	0	0	0
T_3	0	1	0	0

FIGURE 11

3) Movement along a row (within the queue for a resource) can be made only to a position having a (non-null) place-value (rank) less than that of the position the algorithm currently occupies. This locates a task in the queue ahead of the task whose column is now being occupied
4) Movement down a column through the list of resources contended for by a task can be made to any position having a non-zero (and non-null) place-value. This locates resources for which the task whose column is occupied must wait.
5) If, during a horizontal move, the column corresponding to the original task is reentered, an interlock has been found.
6) A sequence of moves stops when an interlock is found, or when no position is open in the specified direction.
7) A new sequence is started at those points from which multiple paths lead.

To insure that all moves are taken before vacating the spot currently held, all positions which may be occupied next are selected and their coordinates saved for later use. In the list with the coordinates of these "next" positions is placed the direction in which movement will be made.

When all the "next" moves from a position have been listed, the position is vacated and a new one taken from the list of "next" positions. The jobs is complete when the "next" position list is empty.

Describe the precedence matrix as a two-dimensional array indexed by the variables R and T, with each element of the matrix consisting of three entires:

1) P, the rank of task T on the queue for resource R;
2) H, the row flag, turned on to show that this element has been accessed horizontally:
3) V, the column flag, turned on to show that this element has been accessed vertically.

Define the "NEXT list as a last-in, first-out stack, indexed by the variable i, having the entries:

1) RNEXT, to save the R-coordinate of selected elements of the precedence matrix
2) TNEXT, to save the T-coordinate of selected elements of the precedence matrix
3) NEXTMOVE, containing "H" or "V", depending whether the direction of movement from this entry is to be horizontal or vertical when its coordinates are selected from the NEXT list.

With all indexes, assume a value of 0 to reference the first element.

It is assumed that, prior to entering the analysis routine, queue management has:

1) Checked that analysis is really needed in that:
   a) At least one resource was already controlled by the task; and,
   b) At least one of the newly requested resources has been assigned a rank higher than zero;
2) Stored the number of tasks in TMAX;
3) Stored the number of resources in RMAX;
4) Constructed the matrix;
5) Stored the column number corresponding to the requesting task in TINIT ($0 \leq$ TINIT $\leq$ TMAX -1);
6) Stored the row number corresponding to any one of the resources already controlled by the requesting task in RINIT ($0 \leq$ RINIT $\leq$ RMAX -1).

RESOURCE	R_0		R_1	R_2		R_3			$R_4\Delta$		R_5	
TASK NAME	T_0	T_1	T_2	T_2	T_0	T_1	T_2	T_3	T_3	T_2	T_1	T_3
MODE	E	S	E	E	S	S	S	E	E	S	S	E
RANK	0	1	0	0	1	0	0	1	0	1	0	1

FIGURE 12

Example

Suppose the request in Case 1 of Section VII has been made (and validated) and the state of the system is as shown in Figures 7 and 8.

Assume that T_2 now requests shared control of R_4. The new queues and precedence matrix are described in Figures 12 and 13.

	T_0	T_1	T_2	T_3
R_0	0	1		
R_1			0	
R_2	1		0	
R_3		0	0	1
R_4			1Δ	0
R_5		0		1

FIGURE 13

In this case it is obvious that an interlock exists, because no task remains in a run condition. However, as

WHEN ADDED TO LIST	RNEXT	TNEXT	NEXTMOVE	WHEN REMOVED
(2)	4	2	H	(4)
(5)	4	3	V	(7)
(9)	3	3	H	(21)
(10)	5	3	H	(11)
(12)	5	1	V	(13)
(14)	0	1	H	(15)
(16)	0	0	V	(17)
(18)	2	0	H	(19)
(22)	3	1	V	(24)

FIGURE 14

has been shown in previous examples, an interlock can exist even though another task is running.

Since T_2 must wait for a resource and already controls other resources, analysis is needed.

Now queue management sets TINIT = 2, TMAX = 4, and RMAX = 6. Figure 14 traces the maintenance of the "next" list. The numbers in Figure 14 are keyed to the following text which outlines the algorithm as it processes the matrix (Figure 13);

1) The column under T_2 is scanned.
2) An open position is found in the row with R_4.
3) No further entries in column 2.
4) Latest entry taken from "next" list.
5) Row 4 scanned, with an entry found in column 3.
6) End of Row 4.
7) Latest entry taken from "next" list.
8) Column 3 scanned.
9) Entry found in Row 3.
10) Entry found in Row 5, then end of row.
11) Latest entry taken from "next" list.
12) Row 5 scanned, with the only entry in column 1.
13) Latest entry taken from "next" list.
14) Column 1 scanned, with only eligible position in Row 0.
15) Latest entry taken from "next" list.
16) Row 0 scanned, with an eligible position in column 0.
17) Latest entry taken from "next" list.
18) Column 0 scanned, with an eligible position found in Row 2.
19) Latest entry taken from "next" list.
20) Row 2 scanned, with the only open position found in Column 2, corresponding to TINIT. The routine enters "2" in the message list for queue management.
21) Latest entry taken from "next" list. (In this case, the only one left in the list is the entry placed during step 9.)
22) Row 3 is scanned, and the first valid next position found in column 1. This entered in the "next" list.
23) Continuing the scan of row 3, another valid next position is found in column 2. Since this corresponds to the requesting task, the row number (3, for R_3) is placed in the message list for queue management.
24) The next entry is taken from the "next" list.
25) This entry indicates that the position in row 3, column 1 is to be occupied and a vertical scan made. However, the vertical flag in this position was set during the process implied in step 14. Therefore the entry is discarded.
26) The "next" list is empty and the process in complete.
27) R_2 and R_3 ("2" and "3" in the message list) must both be released before the request for R_4 can be honored.

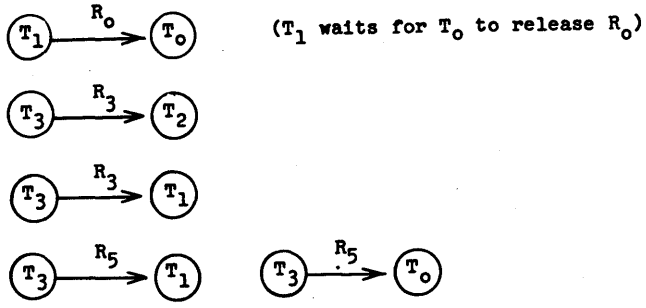
Representation as a directed graph

Alternatively, interlock detection may be viewed as isolating loops (cycles) in a directed graph. This is so because a system of tasks and resources, inter-related through queues, may be represented as a directed graph.

Let the tasks in such system be represented by the vertices (or nodes) of a directed graph. Let an arc be drawn from node T_i to node T_j, oriented toward T_j, if and only if T_i must wait in the queue behind T_j for resource r_k. Associate with the arc $\overrightarrow{T_i\ T_j}$ the symbol r_k to complete the representation.

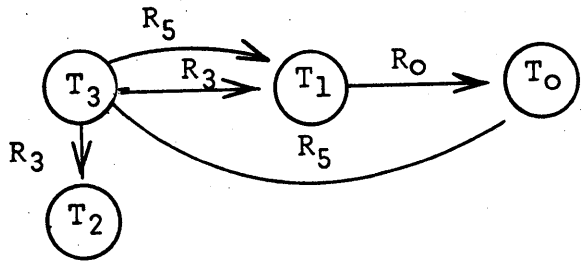
The total set of these arcs, connected into a minimal network, illustrates the inter-relationships of all the tasks in the system with respect to their competition for available resources.

Using the illustration in an earlier section, first construct the set of individual arcs for each "wait pair."

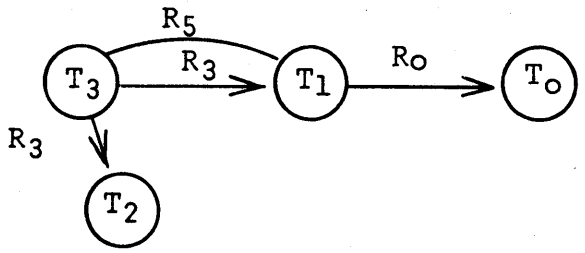


Construct the minimal graph, retaining duplicate

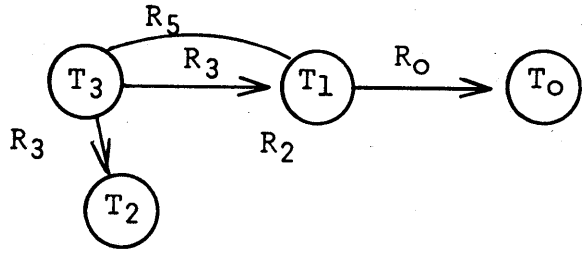
arcs between the same pair of nodes for completeness.



Note that the tasks which are still able to process are the sinks of the graph. If T_0 releases R_5, the graph becomes:

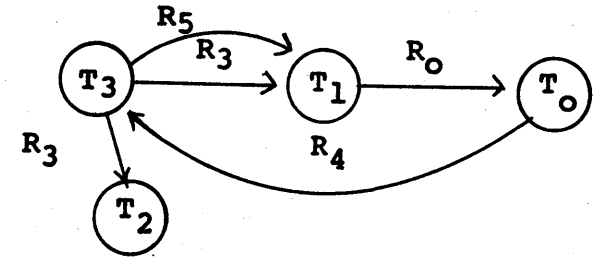


If T_0 were now to request R_2, the graph is:



Even though T_2 is now the only task able to process, no interlock exists. When T_2 completes, T_0 may run. When T_0 completes, T_1 may run, and then T_3.

Suppose, however, that instead of R_2, T_0 requested R_4 (S or E). The graph would then be:



T_2 is still free to process. However, an interlock exists: for when T_2 completes, no other task may run. Each is waiting for a resource held by another of them.

There are several techniques from Graph Theory for the detection of cycles in a directed graph,[1] some oriented toward matrix representation. In view of the relatively small number of elements in this application, adaptations of these might be highly efficient.

Also of potential usefulness from Graph Theory is the idea of the length of the longest path between any pair of vertices. This could be used as a rough estimate of the length of time a task would have to wait for a resource. The task could decide whether to use an alternate, even temporary, resource, if appropriate.

CONCLUSIONS

This paper has presented a queue management technique which detects interlocking requests for system resources. Two variations of matrix manipulation were shown, offering a choice between completeness or compactness.

Either approach offers more flexibility and security than traditional resource control. Conventionally, the possibility of system interlock constrains a task to:

1) Release all currently-controlled resources prior to requesting others;
2) Request multiple resources in parallel; and/or
3) Use resources in a strictly-defined order.

The wait-matrix, with elements consisting of a single binary digit, requires very little storage. It yields only a yes/no answer as to whether an interlock exists, but it provides the information with a minimum of programming. When a task's request for a resource is denied, it has to release all those it already controls and then re-request control of them in parallel with the new resource.

The precedence matrix is bulky, especially in complex systems with many resources defined. However, the algorithm based upon it can isolate the minimal set of resources which must be released by the task.

In addition, the algorithm is easily modified to reverse the direction of search through the wait-chains. This action permits it to identify the minimal set of resources for which the request for control must be denied.

Topics for further investigation are suggested by showing that the set of tasks in a system is a partially ordered set under the relation "wait-for"; and that the system of tasks and their requests for resources may be represented as a directed graph.

REFERENCES

1 C C YUNG
The connectedness of directed graphs and applications
PhD Thesis Columbia University New York 1966 p 71

A dual processor checkout system

by KENNETH C. SMITH

Martin Marietta Corporation
Denver, Colorado

"The brain of the higher animals, including man, is a double organ, consisting of right and left hemispheres connected by an isthmus of nerve tissue. It looks as though in animals with an intact calosum (*nerve tissue*) a copy of the visual world as seen in one hemisphere is sent over to the other, with the result that both hemispheres can learn together a discrimination presented to just one hemisphere. When this connector between the two halves of the cerebrum was cut, each hemisphere functioned independently as if it was a complete brain. . . ."

"Knowing that the answer was wrong, the right hemisphere precipitated a frown and a shake of the head, which in turn cued in the left hemisphere to the fact that the answer was wrong and that it had better correct itself! . . . Taken together, our studies seem to demonstrate conclusively that in a split-brain situation we are really dealing with two brains, each separately capable of mental functions of a high order. . . ." [1]

As human sciences probe deeper into the physical and psychological makeup of man; and as other sciences extend man's technical creations, there frequently seems to exist a parallel between the man himself and his inventions. Not only do these parallels occur often but they frequently occur to an extent much more gross than could have been postulated. Such is the case with a "Dual Processor Checkout System." Although seemingly a new technical creation it is closely parallel to its creators in its decision and controlling mechanisms.

Control and decision making is achieved by two Sigma 7 computers which act independently in monitoring test article data but coordinate efforts to diagnose data and transmit commands. Functions performed by only half the brain and results of tests are shared such that each computer may learn from the other. Now we may discuss a more technical description of the checkout system.

General discussion

Two Scientific Data Systems, Sigma 7 processors and five 16k, 32 bit word memory banks were chosen for the basic computer components. Connected directly into the memory with control from direct I/O lines are two PCM links, analog channels, and two identical discrete data links. Command links to the test article are provided redundantly. This configuration provides a high probability of completing a test should a checkout equipment malfunction occur; and a high probability of test article error detection to guarantee human safety.

The software consists of three main elements: Executive Subsystem, Data Monitoring Subsystem, Command/Control Subsystem. The Data Monitoring Subsystem processes PCM data links, discrete data links, data associated interrupts and notifies the remainder of the subsystems of anomalies which may have occurred from the expected data flow. Specially built hardware helps to filter a raw discrete data rate of 1MHZ and a PCM data rate of 384 KHZ per link. All data is monitored at all times to provide a complete test article profile.

The Command/Control Subsystem interprets output from a user oriented test language translator (subject of another paper for this conference) which provides the test engineer with full control over vehicle testing in a language he may easily use. Tests may be written which send com-

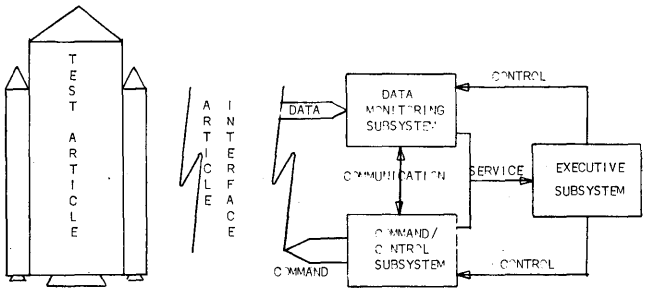


FIGURE 1—Checkout system online software

mands to the test article and set criteria in the Data Monitoring Subsystem as to what results are expected. The Command/Control Subsystem then acts on the recovered data as to how to proceed further.

Executive software fundamentally consists of a modified version of the SDS Basic Control Monitor herein described as BCM. Each CPU is normally administered by BCM which handles all the checkout system processing functions. Processor work loads are divided into modular software elements called tasks and scheduling these tasks, core allocation, I/O processing, trap processing, self tests, etc., become service functions of BCM. The Executive Subsystem provides these services and overall control of processing to the other subsystems.

Reliability and integrity are fundamental to the design philosophy and cannot be compromised. All data coming into the checkout system must be processed by both CPU's as are commands transmitted to the test article. This minimizes any possible data error or incongruity between CPU's to the extent that a disagreement between CPU's will stem from a discernible hardware malfunction. The important points being first to detect an error at the earliest possible opportunity and second, to take proper recovery action for continuation of the test with one or two CPU's on line.

A self-test program is the basic element of integrity which guarantees the success of the system and exists as the lowest priority task in the scheduler. This means that whenever a CPU is idling it is in fact testing itself and associated memory for operability. Several types of tests are conducted on the computer processor and memory banks: Instruction tests are executed on the CPU. Read/write tests are performed on the memory modules and periodic sum-checks are made on software programs. Self test is also used in processing Sigma 7 traps and in this way a CPU may determine which particular part of the hardware has caused a failure.

Responsibility rests with each CPU to cross examine the validity flags, which, if they are set, indicates a critical failure in the CPU whose flags are being tested. The validity flag is set whenever a possible malfunction may have occurred and is reset when the proper recovery has been executed. The weight of making a command decision rests upon the good CPU which must learn the major computer component failure of the other computer system. This may result in new processing functions or a small modification of the test presently being conducted.

The functions which will be described consist of that functional hierarchy which manages these learning and controlling processes between CPU's. With the internal computer programs and hardware being tested as they are, all that remains to maintain integrity is to control the data inputs and outputs. The methods of doing this will later be described in detail. Such is the spectrum of programs which guarantee system integrity.

Figure 2 depicts a generalized control and error flow for the Executive Subsystem and portrays the hierarchy of the Executive Programs. CPU/CPU communication is processed through the Dual Processing Controller with the exception of testing the validity flag. This and CPU/CPU syn-

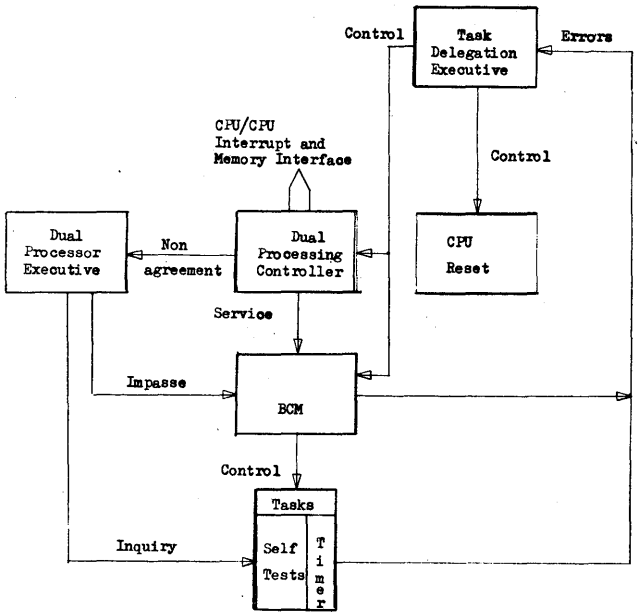


FIGURE 2—Checkout system executive subsystem

chronization tests are conducted on a periodic basis so that an inter-CPU link will not be required to detect a faulty processor. The Check-out System receives a BCD range time which is required for data synchronization purposes. A binary doubleword image of this time is maintained in each CPU for basic clocking purposes. These three sources must be in agreement for the Checkout System to be fully operational and any disagreement is indicative of a computer or data error. In this case a computer may be taken off line or a related checkout system function discontinued.

Errors resulting from any data disagreements are passed to the Dual Processor Executive which diagnoses inconsistencies on a detailed level. Whenever the computer systems disagree upon a data input, a program failure or some hardware failure, exhaustive testing must be initiated to uncover the error. Because this is difficult it is imperative that programs compare data wherever the eventuality may arise that different functions may be performed as a result of the data change.

Should a discrete error be detected in one Data Monitoring Subsystem but not in the other then that inconsistency will be reported immediately to the Dual Processor Executive, this indicates a hardware malfunction if the criteria for the Data are pre-checked. It may be parenthetically added that this particular error seldom arises since all critical discretes (those which are crucial to the test articles successful performance) are triply voted in hardware before being evaluated by the software, indicating that a single discrete failure will not be detected as an error—only a dissenting vote.

The Dual Processing Executive isolates the malfunctions in the Checkout System and determines the proper posture to continue testing. Any decisive errors, hardware or software, from all executive subsystem programs are sent to the task Delegation Executive which reconfigures the software to be compatible with the hardware failures. If a PCM link failed, then all programs pertaining to that link would be disarmed, a message would be printed and processing would continue. CPU Reset is entered if the Task Delegation Executive has determined that the functions assigned to this Executive Subsystem cannot be completed. This obviously assumes that the computer memory and processor are operational to a large degree so that it may turn itself off. Memory or CPU failures would be observed in the verification flags which help to alleviate this problem.

Executive programs reflect the Checkout System redundancy philosophy. Stated simply, all critical command/control loops are triply redundant. Whenever a command is to be sent out to the test article, that command is first voted by both CPU's; the command is transmitted only if both CPU's agree. The command is sent down three separate transmission links to the pad equipment and ultimately to the test article. Critical data are monitored in an analogous manner. Commands will generally result in a discrete event change through the Data Monitoring Subsystem where the discrete events are again voted by the CPU's. Should the CPU's disagree upon an anomaly, then this would result in control being passed to the Dual Processor Executive where such inconsistencies are processed. Each CPU is self-reliant to the extent that it can fully complete a test independently and run asynchronously with periodic data and timing checks. This helps to minimize the data crossover, simplifies analysis in case of an error and ultimately reduces the complexity of the software programs.

Thus far, little uniqueness has been suggested for software implementation in the checkout system. Functions such as dynamic allocation of memory, program scheduling (with a hardware priority interrupt algorithm), high speed mass storage buffer techniques, on-line data analysis and display have seldom been seen in one real-time system but are generally known and understood. It is not the intent of this discussion to discuss the total Checkout System Executive uniqueness but it must be mentioned (even if trite) that the Sigma 7 dictated some unique implementation by virtue of its real time characteristics. Now, we may proceed to a more specific treatment of the dual system.

Task delegation executive

Ultimate hierarchal control resides with the Task Delegation Executive. All noteworthy information of system status is available here and all decisions to be based on failures will be made here. A failure may result in two general actions: (1) elimination of that particular function from the processing chain in that CPU; (2) elimination of a CPU. Each action that is to be taken is fully dependent on the present system state. For-

tunately, however, there are only two basic states; Redundant Mode with both CPU's executing the same test to provide reliability; Test Mode with each CPU operating independently. This minimizes the number of switch positions.

Switching is accomplished on three program levels. At the first level is the arming and disarming of tasks (scheduled entries) and interrupts. Both these areas contribute to the change of data flow (input and output). At the second level is a gross trap vector change affecting all Executive Services such as I/O, inter CPU communications, etc. The third level is more refined and lies within the service trap processors as a trap may branch to one of several programs for processing. The Task Delegation Executive must weigh each of the failures and turn the switches to the proper position. Notification of the other CPU and operator/test conductor notification are other functions completed.

Dual processing controller: Transmit/receive logic

Communication between the CPU's is accomplished using interrupts and common memory. Implementation incorporates both a simplex and duplex interrupt structure. The former technique is used where ultimate action will result in a processor being taken off-line. This is a high priority communique for highly significant messages. The remainder of communication is implemented through a duplex algorithm where an acknowledgement is always required to complete the loop. In both cases messages are placed in mail boxes which are scrutinized when inter-processor interrupts occur. For the duplex system, timers are set up upon initiation of a transmission which determine the greatest allowable time that may be delayed before receiving an acknowledgement.

The inter-processor time delays, as set by the timers, are only dependent upon interrupt priorities and their processing. These times are not difficult to determine, and not dependent upon total processor loads. A second set of timers is used where necessary to guarantee that queuing functions are ultimately processed. A failure to do so implies a serious system non-agreement.

Later examples will demonstrate the technique where these timers are used. A disadvantage in a duplexed system is that it implies a loss of speed and lost processor time due to bookkeeping requirements. This lost time is felt to be less important than the integrity gained since the number of command transmissions to the vehicle is not great over a one second interval. There are also frequent occasions where the CPU's must be synchronized which implies CPU lost time by forcing one CPU to wait for another. The result of the technique is a synchronization of events rather than of a timing window. A communication technique of this type allows each CPU to run asynchronously most of the time which is important for this system since the loads on the processors are not identical. Areas of non-redundancy such as CRT displays, PCM evaluation and recording, and analog data collection can cause processing imbalances.

Redundant functions, such as critical anomalies and commands, are always agreed upon by the CPU's so that any subsequent action will be the same in both CPU's. This then must be done in a synchronous manner.

Dual processing controller: Inter-processor services

Three basic types of services are provided: memory check, hardware register check and much of the intra CPU service linkage. These services may be put together into several combinations to process any particular circumstance which may arise. All these services are requested through a trap processor that a task may initiate. This being the only manner in which a slave program (one that does not have full access to the instruction repertoire) may enter the master mode and subsequently do I/O, trigger interrupts, etc. The tool used to provide this linkage has been discussed in the previous paragraphs in general and will now lend to a more specific treatment of technique.

As has been previously mentioned, the CPU's run asynchronously and for this reason queuing type techniques are used so as not to inhibit the processors operating upon other tasks. To compare data between processors, then, some memory allocation scheme must be implemented so that data may be passed from CPU to CPU. Inherent in the design is a large segment of memory which has been designated for dynamic allocation herein termed "Free Storage." A free storage block (FSB) may be assigned to a program for an indefinite length of time.

Inter-CPU communication is separated into three priorities. These priorities are assigned to

a service function and dictate some of the implementation. At the highest priority level, are the intra-CPU services to be allowed between CPU's. Expected processing delay and acknowledgement times are short and no synchronization is required. Therefore, no queuing is used.

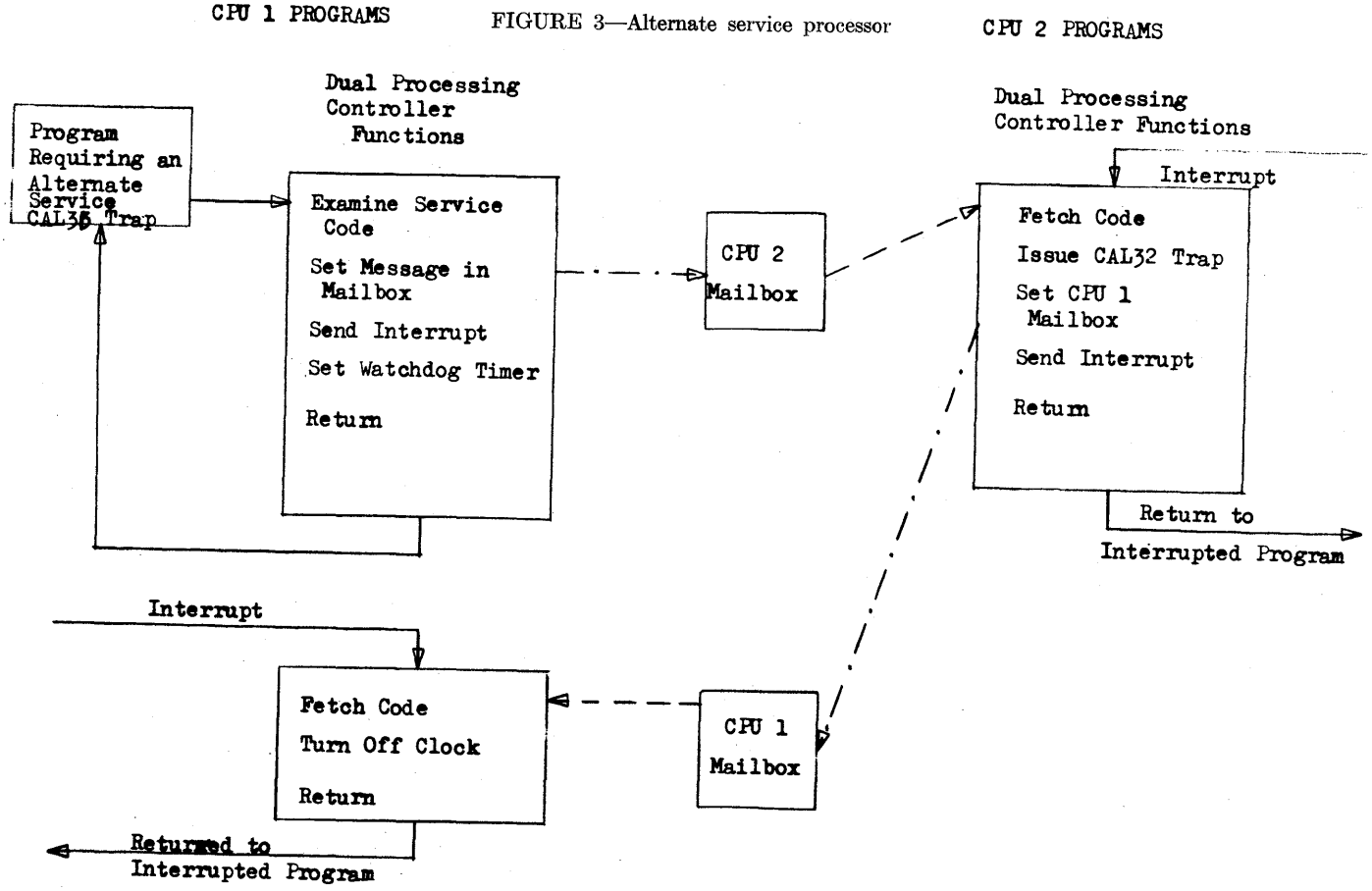
The second transmit/receive pair is assigned to command functions. Execution delay time due to interrupting functions is again expected to be nonexistent but because of a synchronous processing characteristic some queuing is required. Data cross checking is executed at the lowest level of priority with normally expected delay times in the milliseconds region. Therefore extensive queuing is required.

EXAMPLE 1

All inter-CPU services do not require a FSB and, in fact, this is the rule rather than the exception. Traps which provide scheduler services between CPU's are of such a nature and no additional linkage is needed than that already provided within one CPU. No queuing is necessary because the message may be acted on immediately requiring no synchronization at all between CPU's. A task such as PCM processing may be active in CPU 2 and desire to communicate an anomaly to CPU 1. The task would form all the normal setup required to activate a task within CPU 2 but would execute an inter-CPU trap instead. The scheduler in CPU 1 will in turn activate the requested task depending on its level in the task priority structure. Any address modification will be done in the trap processor if the addressing needs to be changed. This is a serious consideration since the memory banks are numbered oppositely from each CPU with bank 3 being addressed the same from both CPU's. A buffer area relative to CPU 2 will be displaced by a constant (for that location) so that CPU 1 may address the correct buffer.

Figure 3 depicts a typical inter-CPU scheduler service processing. A task being processed by

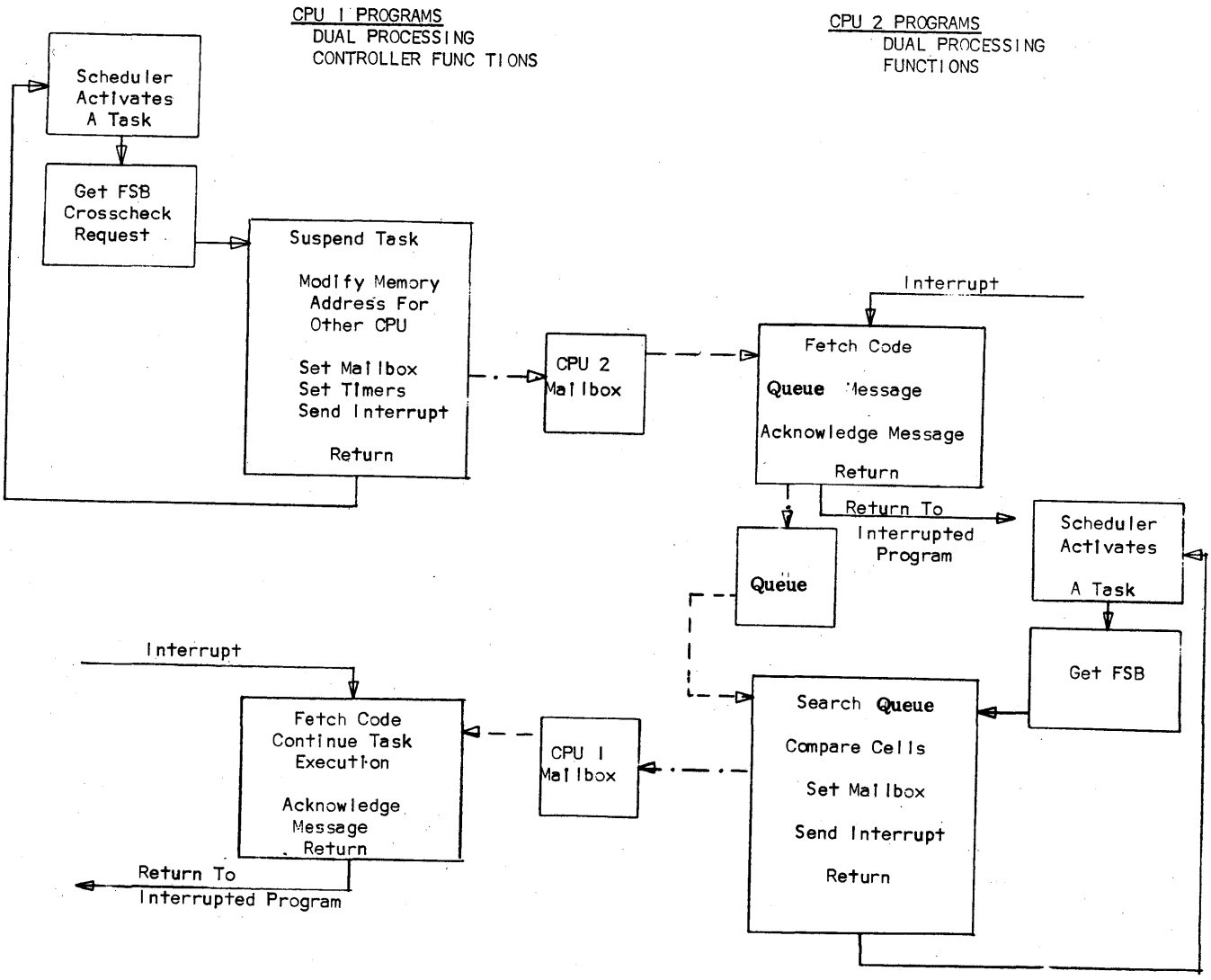
FIGURE 3—Alternate service processor



CPU 1 desires to transmit a display message in CPU 2 and therefore initiates a trap to perform that function. The message is deciphered by the Dual Processing Controller. A special code is placed in CPU 2's mailbox which will direct it to perform the desired function. Argument addresses are modified for CPU 2 usage. Then, an interrupt is generated for CPU 2 to look in a specific mailbox location. Before CPU 1 exits it must set a timer to be certain that CPU 2 has received the message. Control is returned back to the requesting task to continue processing.

Now CPU 2 comes into operation through the interrupt, immediately performs the desired function, sets the acknowledge code in the mailbox, and triggers an acknowledge interrupt to CPU 1.

FIGURE 4—CPU/CPU memory crosscheck



When the interrupt routine is exited, the scheduler will consider the new request and service it at the proper priority level. A CPU 1 interrupt routine next comes into activity, turns off the timer and exits back to the interrupted program.

EXAMPLE 2

The case above is the more frequent of the types of inter-CPU services but far from the most interesting. Data compares demonstrate a greater level of complexity and require a FSB to be accomplished. Following Figure 4 will assist in comprehending the design implementation of the inter-CPU data compare service. Since the work scheduled for the processors in asynchronous, we may assume that the schedular activates a task in CPU 1 which requires a check of data in memory (the synchronous case of both CPU's executing the same task at the same time cannot be ignored for implementation but is done so here for simplicity. The proper trap is set and control is given to the Dual Processing Controller which first suspends the task. By suspending execution of this task it may be observed that the mailbox entry to be made is unique as to the task which requested service. All queued entries are keyed to a task since it has been temporarily ignored by the scheduler and no new service can be requested from that task. The memory address of the FSB is modified to be read by CPU 2 and the message is set in the mailbox for CPU 2. CPU 2 is notified of the pending message by an interrupt from CPU 1 and then CPU 1 returns to normal execution with the service requesting task suspended.

CPU 2 receives the interrupt, looks into the mailbox and decides to queue this message for later reference. An interrupt is returned to the initiating CPU to complete one duplex loop. Again, CPU 1 is initiated and cleans up any bookwork that may be pending such as turning off one timer.

In this type of service an additional clock is required to assure that the request to compare data will not be ignored indefinitely. This is precisely the type of error the checkout system is designed to process and timer runout could be indicative of a major malfunction. More will be said on this area later in the discussion of the Dual Processing Executive. It is observed that except for the initial memory compare request all processing proceeds asynchronously under interrupt control.

The second half of the sequence may now take place! CPU 2 has the same task activated which also wishes to compare data. The queue is searched, an entry found and the data cells are compared. The results of this compare are sent back to CPU 1 which continues execution of the suspended task if the results agree. If they do not, an error mode results which sends control to the Dual Processing Executive to diagnose the problem. Acknowledgement is sent back to CPU 2 from CPU 1 to complete the duplex loop. Control is returned to the task if it is still the highest active task in the schedule and if the data agree.

EXAMPLE 3

Implicit in describing the CPU/CPU service functions is a method to compare commands that are to be sent to the vehicle. Although simple in concept, the actual implementation is somewhat more formidable than the example previously described for data cross-checking. Three distinct steps are required to transmit a command and these commands are intimately connected to the redundant philosophy that both processors participate in transmitting commands. This condition may be overridden but only if one processor is on line, otherwise agreement is mandatory in the command/control loop. This is not to say that a program may not be generated to test the test article up to checkout system processing limits such as generating a frequency of 10 KHZ or better to determine phase shift in a vehicle subsystem. In this situation no inter-CPU communication is required at the command level.

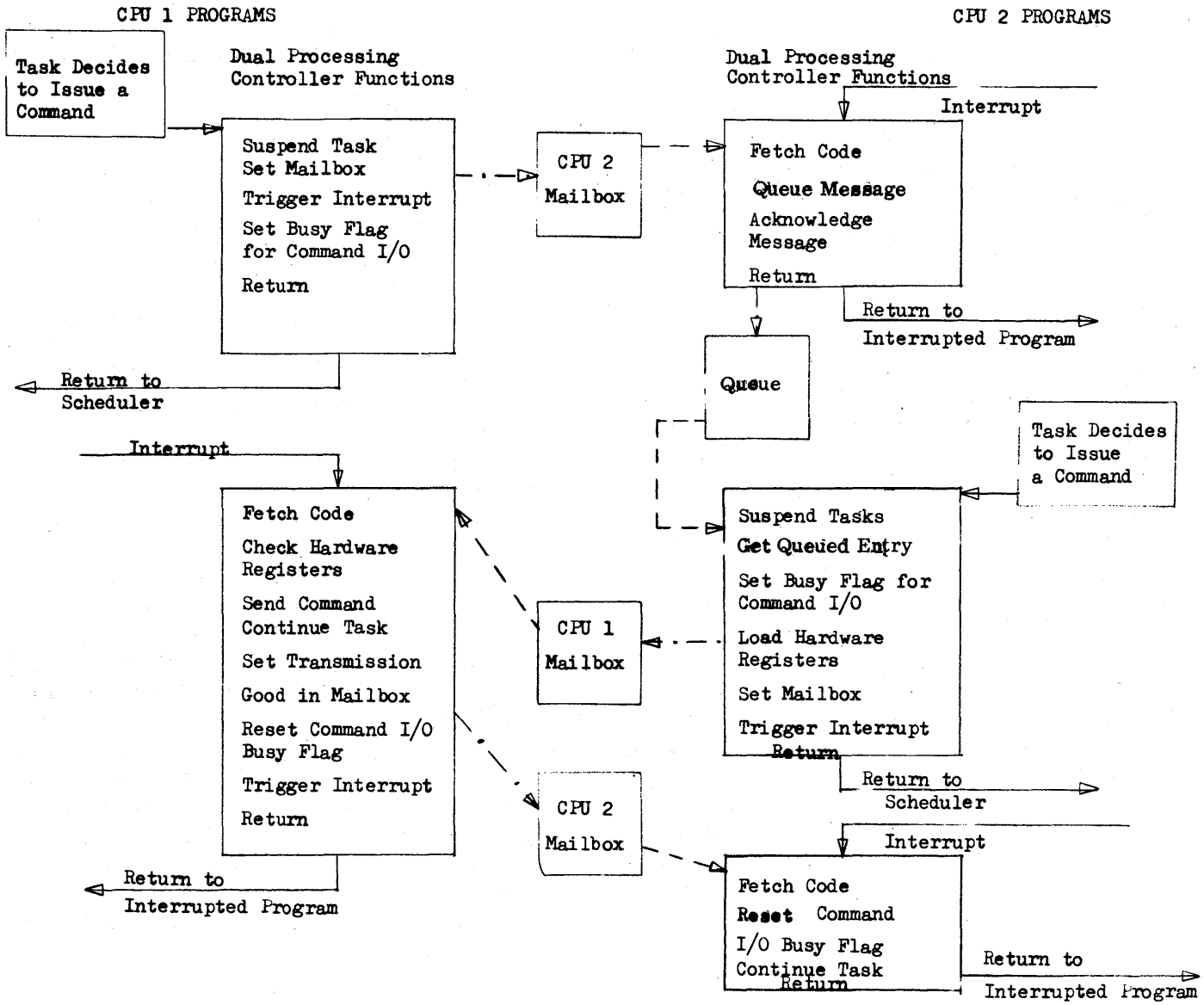
Implementation has been in such a way that a minimal amount of inter-CPU communication is required for synchronization and is somewhat similar to that as demonstrated in the previous example. Following Figure 5, a task which is to send a command causes a trap to the special I/O handler routine. The Task Delegation Executive has previously set this trap processor to respond for a Redundant Mode. In this case the task requesting service does not know how many CPU's are on line or which CPU is executing the program. The proper processing path is automatically taken. An interrupt is sent to CPU 2 which queues the message and transmits the acknowledge back. At a later time CPU 2 has the same task activated as in CPU 1 which determines to send the same command. CPU 2 observes CPU 1 is already waiting and therefore proceeds with

command execution by loading the proper hardware command registers. A message is placed in the mailbox for CPU 1 to check and send the command to the test article, which CPU 1 does. The task which was suspended is continued and a transmission complete or incomplete is sent to CPU 2. If the command is sent, then control is returned back to the task when the interrupt routine finally releases control. Control is sent to the Dual Processor Executive should this not be the case, if the command should be bad or the CPU's do not agree on register content. CPU 2 continues the suspended task when it gets an interrupt which indicates the transmission has been sent. Control is then returned to the task which was suspended.

Dual processing executive

The function of this program is to determine which computer system has malfunctioned, not an obviously simple assignment with only two voters. We must recall, at this time, that all data inputs and outputs are agreed between CPU's before they are allowed to occur. Also important is the awareness a CPU must have to its peripheral

FIGURE 5—Command comparison service



equipment (peripheral to the processor and memory that is). To this end all Martin built hardware has extensive error detecting capability which compliments the SDS equipment.

Using this knowledge, that the only malfunction left must be in one of the computers themselves, then a self-test is initiated in both computer systems. A period of time is passed to allow completion of the test and then the validity flags are tested, each CPU testing the other. Should an error be detected then the malfunctioning CPU will be taken off line. The responsibility for this function resides with the good CPU.

Far, far down the processing path lies the possibility that an error will not be detected. To realize how far down it is necessary to point out that all critical data, that which can force a processing decision, are majority voted; and that the programs for testing a test article have been frequently run before. Therefore, it is not expected to find software errors in this environment. All data loads are well analyzed and programmed for. However, should this eventuality arise then the test article will be "safed" by both systems to assure that nothing and no one will be harmed.

Reset

This program attempts to clean up a bad CPU and place itself off line. At the same time the good CPU also performs a reset function on the bad CPU which will prohibit that CPU from taking any further action by isolating the processor from all memory. This will assure that no further interaction will be allowed by an uncontrollable computer.

Technique effectiveness

One method of evaluating the effectiveness of the implementation technique is to examine the trade-offs available within the implementation and make this information available for future use. Core used for the total dual processing control and error processing scheme for all the processes described amount to a total of 1200 locations. Sigma 7 processors must have unique first memory modules to process traps and interrupts which accounts for the reverse addressing scheme previously mentioned. From this same consideration, and because of the reliability factor, CPU's are not allowed to access these modules in a dual processor configuration. Therefore, mailboxes reside in the second two memory modules and any message sent out is also recorded so that lost messages can be recovered to the fullest extent. Eight interrupts are assigned for inter-CPU linkage and each interrupt has two mailbox locations with the exception of one which is not used for duplex operation. Six of the interrupts are tied in pairs with the seventh available for interrupt error recovery. A timing chart would also be in order. Two basic considerations are made: (1) elimination of the duplex system, (2) elimination of queuing.

	Nominal Total Time (usec)		
CPU Service	W/O Duplex	W/O Queuing	Normal
Crosscheck	369.3	469.3	639.3
Command Transmission	320.0	370.0	520.0
Alternate Service	186.0	246.0	246.0

Times shown do not take into consideration expected delays due to asynchronous operation nor is an individual processor share of the time shown. In general, execution times may be divided in half to obtain these times. Therefore, to process a crosscheck service each CPU would require about 320 usec processor time for a queued, fully duplex technique.

Observations

The computerized dual processor solution to automatic checkout is not a concept that has evolved over night but after years of experience in the field. A pilot study was undertaken to aid in feasibility which resulted in a basic checkout system design criteria. Data, dynamics and interfaces are all well defined and the problems left became those of fitting a computer into this environment. Uniqueness of the checkout system stems from its reliability requirements and the method of implementation. The Dual Computer System with its asynchronous characteristics allows either computer to control the article being tested. Moreover, system information must be made available for automatic switchover to occur. It is also recalled that both computers take an active part in test execution. In this light, the similarity with the human counterpart can be observed.

Software and hardware design and implementation optimization are observed on two levels: (1) design criteria for the checkout system where trade-offs were made, (2) frequent testing of very well defined processes. Implementation errors can

show up during preliminary article testing since many tests are dynamic enough to exercise the checkout system to its fullest capability. This is in addition to normal program checkout.

The Dual Processor Checkout System has been implemented with the idea that integrity cannot be compromised and at the same time system dynamics must not be restricted. The system must respond to 95 discrete changes in 10 milliseconds of which all changes are not critical in nature. However, this will result in only a few commands being generated. Asynchronous processing is the answer to the overall problem allowing each computer the greatest latitude possible.

The Dual Processor Checkout System with its two brained system, requires close coordination to achieve the desired ends. This is done through an isthmus of memory which is stimulated by interrupts. A capability exists for each processor to gain information and commands from each other. Should one half of the checkout system fail the other half is fully capable of carrying on normally but must have to re-learn menial, non-critical tasks. Specifically, PCM monitoring may have to be re-initialized so that correct test criteria may be set, which indicates a learning phase. This is also true of analog testing. Finally, the connection between the processors may be severed in which case they may operate independently.

BIBLIOGRAPHY

1 M S GAZZANIGA
The split brain in man
Scientific American August 1967

An operating system for a central real-time data-processing computer *

by PAUL DAY and HENRY KREJCI

Argonne National Laboratory
Argonne, Illinois

INTRODUCTION

A detailed study of the laboratory data-processing needs of our Chemistry Division has shown that about 25 unrelated experiments (see Table I) will require or greatly benefit from real-time computer service. An assessment of the nature of these requirements and a careful study of the capabilities of the Sigma 7** computer clearly indicate that this work load could be handled by such a central computer. This load will use up to 10% of the I/O capacity of the system and require about 40% of the Central Processor time to satisfy the real-time needs of all the experiments if they are running simultaneously. An additional 30% of the Central Processor capability would be used to perform final processing and analysis of the data at the end of each experiment. Further examination of the problem indicated that while a comparable amount of money would have to be spent whether we purchased individual computers for each experiment or a single Sigma 7 system, the service that could be provided by the central system would be far superior to the individual small computer service. In addition to the usual features found in third generation computers, the Sigma 7 has true independent Input/Output processors, independent memory modules and a high speed random access device (3×10^6 bytes/sec transfer rate), all of which are essential for the efficient operation of a large data-collection and processing system.

Analog information from the various laboratory instruments will be digitized at the remote site and transmitted directly to the central computer memory. The data from each of these remote instruments will be processed by a separate program residing in the central computer. In the initial stages of our installation, most of the experimentalists will require this collection and storage of data for processing at the termination of a run which may last anywhere from seconds to days. On the other hand, some of these experiments, such as nuclear detector multiparameter analyzers, will be partially controlled by the computer. These experiments will be provided with amplifier gain stabilization and live displays of computer modified spectra. There is little doubt that as the experimentalists become more cognizant of the capabilities of a central computer, more and more experiments will be modified to become increasingly interactive with the computer during an experiment.

The very nature of laboratory research requires that the experimentalist be allowed to change his operating parameters during an experiment and make interrogations regarding its progress. Therefore, in addition to the data lines, each remote site will contain at least a teletypewriter which will always be interactive with the computer. Using this device, the experimentalist will be able to initiate the loading of his program and provide the relevant program parameters by responding to a series of questions posed by the computer.

Upon termination of an experiment, the accumulated data will be completely processed by a user-selected processing program. This processing will be done in a low priority background mode. Additional background processing will be run on an open-shop basis and small batch jobs will run one-at-a-time during the many milli-seconds per second that the Central Processor is not

*Work performed under the auspices of the U.S. Atomic Energy Commission.

**Manufactured by Scientific Data Systems, Santa Monica, Calif.

handling the real-time processing. In addition, there are some very long-term (hundreds of hours) routine computations that are of important theoretical interest which will be carried out as a "sub-background" job. While these calculations could be done more readily on a larger and faster computer, the time available on larger systems is better spent on doing more advanced research in this computational area.

Design goal

The specific requirements of the users in a typical chemistry research facility cover a broad spectrum: (a) Data rates vary from one byte/sec to 100K bytes/sec during millisecond periods. (b) Real-time operations, except for data storage requirements, vary from zero to sophisticated analysis requiring about 10% of the Central Processing Unit's (CPU) time throughout the operation of an experiment. However, there is no instance in which a high request rate and a large computational load combine to require more than 0.1 sec per second of CPU usage. (c) While some experiments are completely invalidated by the loss of a single data point, others would not be degraded significantly even if 20% of the data were lost. (d) Also, a few experiments must have a sophisticated response from the computer in less than a second and others could wait all day, the loss being the experimentalist's time and patience.

The computer will be used as a data buffer by each of the experiments, some of which will require moderate calculations on a real-time basis. Most of the real-time programs require the evaluation of mathematical functions and the use of many common sub-routines. For example, over half of these programs require a rather large spectrum-analysis routine. Considerable memory can be saved by coding these routines as "pure procedures" (re-entrant code) and having one copy resident in memory shared by many resident programs. As a result, in the majority of the cases (20), the data buffer area will require about four times more core than the specialized portion of each program. Since most of the experiments are transmitting data in an asynchronous manner, these large data areas must be dedicated for the duration of an experiment; only a 20% saving in core would be realized if "overlay techniques"[1] were employed on the code area. Therefore, at the expense of buying sufficient memory to keep these real-time programs resident during an experiment, the Operating System is relieved of the burden of re-establishing appropriate memory-residence for the majority of the programs. A few (5) of the experiments will transmit their data in bursts at intervals of 30 seconds or longer and then require a rather large program for analysis and a non-critical response (time-wise) back to the laboratory. These service requests are handled using overlay techniques and are processed one at a time on a first-come first-served basis in a common memory area on a low priority basis.

A simulation program has been written to assess the compatibility of the CPU requirements of the various users. This program depicts on a graphical plot (Figure 1) the dedication of the CPU as a function of time. The input parameters to the program include the switching overhead time, data rates, buffer size, computation time per service request, number of I/O requests per service, and the elapsed time between an I/O request and its completion. Included in the program is a routine which simulates the randomness of the individual events in those experiments where this is applicable. Table II summarizes the results of simulating 21 minutes of Sigma 7 operation. As can be seen, each service request was satisfied before the succeeding request was signalled (zeros in column labeled "No. Inst. Lost"). This gives one assurance that the demands of all of the proposed users will be satisfied by the CPU when they are running simultaneously. The simulation also shows that it is practical, when a broad spectrum of requirements is to be satisfied, to assign a priority to each task and let it run to completion or until it is interrupted by a service request from a job of higher priority. Upon completion of the higher priority task, the interrupted task is resumed. When a real-time program is to be loaded, its priority will be assigned based on the current computer work load, the service frequency, computational time and its required response time.

Inasmuch as the sequence of tasks carried out by the CPU will be determined by the priority assignments, some means must be undertaken to insure that a high priority program does not exceed its allotted running time (preventing programs of lower priority from running) due to a program error or an unusual data sequence. Also, all of the real-time programs will be associated with research projects whose data-processing requirements change from time to time, implying that these programs will require modification on

USER	INSTRUMENT	INVESTIGATION
Abraham	Digital potentiometer	Low temp. heat capacity
Atoji	Neutron spectrometer	Crystallography
Carnall	Multi-channel anal.	Optical absorption spectra
Diamond	Atomic beam	Nuclear spins
Engelkemeir	Multi-parameter anal.	Heavy element decay studies
Fields	Multi-channel anal.	Decay scheme studies
Friedman	Multi-parameter anal.	Proton reaction studies
Glendenin	Multi-parameter anal.	Fission studies
Hines	Multi-channel anal.	Decay scheme studies
Holt	Mass spectrometers	Routine mass analysis
Katz	Mass spectrometer	Deuterated compounds
Katz	Cryogenic NMR	Deuterated compounds
Osborne	Digital potentiometer	Low temp. heat capacity
Peterson	Neutron spectrometer	Crystal structure
Siegel	X-ray spectrometer	Crystal structure
Steinberg	36-det. count lab	Decay and spectra studies
Steinberg	Multi-parameter anal.	Proton-nuclei reactions
Studier	Time-of-flight m. s.	Carbonaceous chondrites
Wexler	Mass spectrometer	Charge states of mol. frag.
Unik	Multi-parameter anal.	High res. fission studies
Weil	Nucl. mag. res.	Structural studies

TABLE I—Work load summary

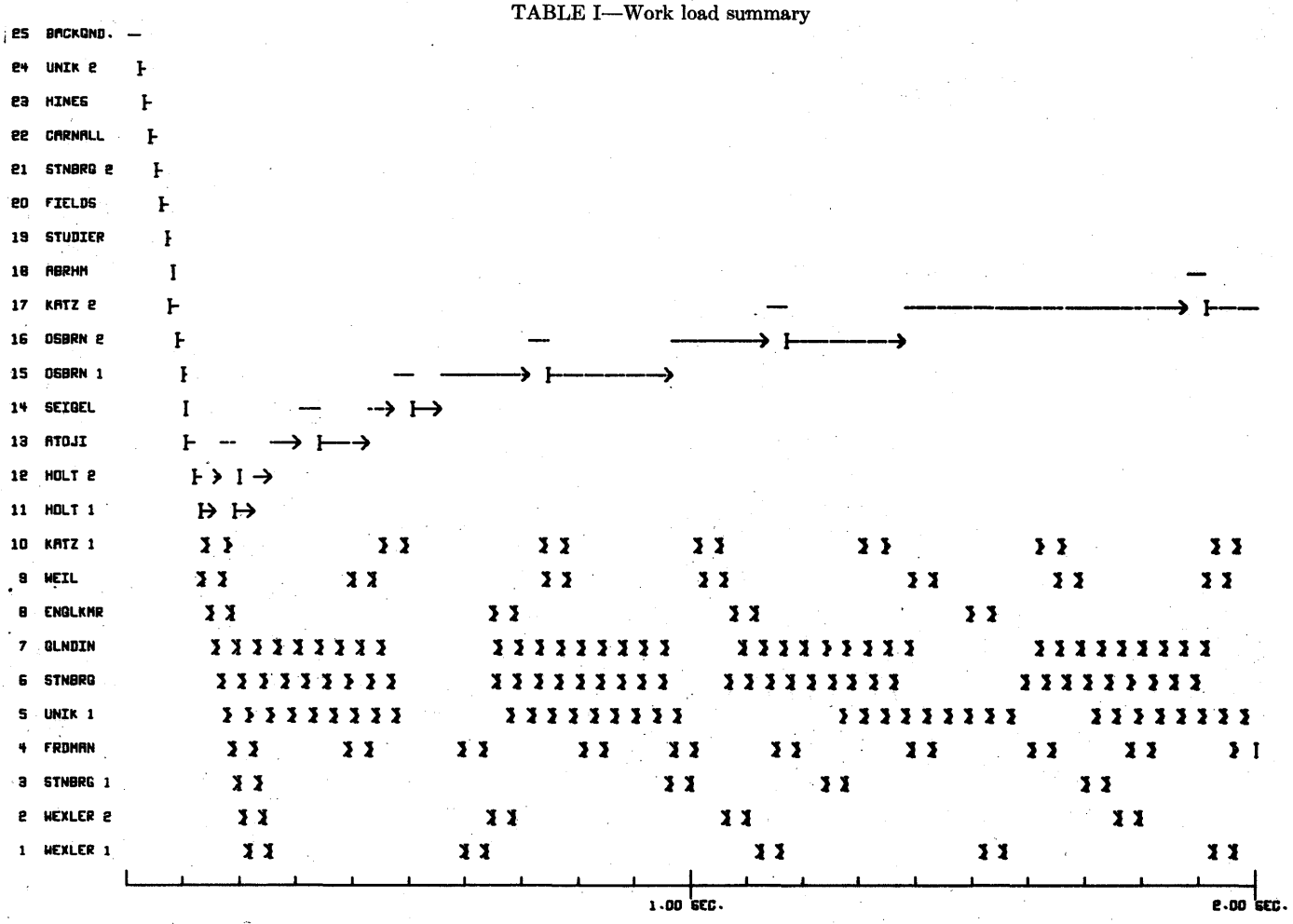


FIGURE 1—Simulation of CPU usage as a function of time. Increasing time along horizontal axis and tasks of decreasing priority along vertical axis.

occasion. With 25 such programs resident in memory at a time, some of which have not been thoroughly tested, it is absolutely necessary that these programs be prevented from writing over each other.

In addition to being required for the Operating System, auxiliary storage must be used for several other purposes. The real-time processing for a single multi-parameter experiment requires the maintenance of eight updated spectra. Each spectrum (2048 words) must be updated during a data buffer processing which takes place at intervals of about one second. Limiting core residency to one table at a time requires the reading and writing of the eight tables once each second. The simultaneous running of the four planned multi-parameter experiments will require an aggregate data transfer rate of 512K bytes/sec between memory and mass storage. Also, the raw data from the multi-parameter experiments must be written on magnetic tape. The remainder of the remote sites will generate considerably less data (less than 65K bytes per experimental run), which can be readily handled by a mass storage device with a capacity of several million bytes. To facilitate remote loading of real-time programs, their object forms must reside in mass storage.

21.712 MIN. TOTAL RUN TIME

.32 MSEC./SWAP. OVERHEAD

PRIORITY AND LINE DESCRIPTION	TIME REQ. TO SERV. AN INT.	AVE. TIME BETWEEN INTERRUPT	PERCENT OF TOTAL TIME	NO. INTS. SERVICED	NO. INTS. LOST	TIMES INTERRUPTED
1 WEXLER 1	1.50	249.56	.60	5220.	0.	0.
2 WEXLER 2	1.20	249.56	.48	5220.	0.	12.
3 STNBRG 1	2.00	247.66	.81	5260.	0.	82.
4 FRDMAN	2.50	100.49	2.49	12964.	0.	349.
5 UNIK 1	2.70	55.65	4.85	23408.	0.	939.
6 STNBRG	2.50	55.78	4.48	23355.	0.	1082.
7 GLNDIN	2.20	55.71	3.95	23382.	0.	1175.
8 ENGLKMR	2.10	250.42	.84	5202.	0.	544.
9 WEIL	1.80	150.18	1.20	8674.	0.	834.
10 KATZ 1	3.10	150.36	2.06	8664.	0.	1641.
11 HOLT 1	16.00	29606.75	.05	44.	0.	77.
12 HOLT 2	16.00	31016.60	.05	42.	0.	72.
13 ATOJI	60.00	9578.65	.63	136.	0.	833.
14 SEIGEL	50.00	12771.54	.39	102.	0.	498.
15 OSBRN 1	150.00	14803.38	1.01	88.	0.	1276.
16 OSBRN 2	150.00	15147.64	.99	86.	0.	1261.
17 KATZ 2	400.00	18609.96	2.15	70.	0.	2736.
18 ABRHM	150.00	14803.38	1.01	88.	0.	1300.
19 STUDIER	270.00	28319.50	.95	46.	0.	1204.
20 FIELDS	2000.00	93049.79	2.15	14.	0.	2840.
21 STNBRG 2	2000.00	217116.16	.92	6.	0.	1225.
22 CARNALL	2000.00	162837.12	1.23	8.	0.	1611.
23 HINES	2500.00	162837.12	1.54	8.	0.	2026.
24 UNIK 2	3000.00	108558.08	2.76	12.	0.	3641.
TOTAL FOREGROUND			37.60	122099.	0.	27258.
BACKGROUND			57.00			73765.
OVERHEAD			5.40			

TABLE II—Real time simulation-summary

While the most efficient code is written in assembly language, the experimentalists should not be required to learn a particular machine language, but be permitted to write their own programs in the more familiar FORTRAN language, if they prefer. Therefore, FORTRAN capability should be provided for the real-time user as well as for the batch processing users.

Although immediate final analysis is useful to the experimentalist, it does not have the urgency of the real-time operations and therefore should be executed in a "background mode" which will operate at a low priority level. The actual final processing requests will be placed in a background queue by the data-collection phase program upon completion of an experimental run. The remaining CPU time should be used for "open shop" batch processing and long-term computations.

The Supervisor offered by Scientific Data Systems (Batch Processing Monitor) for the Sigma 7 was designed to handle a number of real-time tasks in the foreground mode (high priority) while running a sizable batch-processing operation in the background mode (low priority). The principal shortcomings of their system are that the practical number of foreground tasks is considerably less than ten, there is inadequate memory protection, no provision is made for foreground program "overrun time" traps, and that it contains an inefficient Rapid Access Device (mass storage) handler. Rather than attempt to modify a Supervisor that was designed without large-scale real-time operation as the primary goal, it was decided that a fresh start, with the detailed requirements of the experimentalist in mind, would be a better approach.

Hardware configuration

The Sigma 7 is a third-generation machine with an 850 nano-second memory and an instruction set whose execution times range from 1.2 μ sec for an "add immediate" to 25 μ sec for a "long form floating divide" (16 decimal digit). The Central Processing Unit (CPU) has sixteen programmable registers, of which seven may be used as index registers. The instruction counter, "memory write key," and other information germane to the currently running program reside in a 64 bit CPU register called the Program Status Doubleword (PSD). When a program is to be interrupter, the 16 registers and the current PSD must be saved, and the PSD of the interrupting program must be set in the CPU. This can be accomplished with a series of three instructions (24 μ sec). When the interrupted program is to be resumed, the 16 registers and the PSD must be reloaded. This is also accomplished with three instructions (22 μ sec). The CPU also has a priority interrupt structure which responds to signals from internal clocks, I/O terminations and up to 240 external sources. A memory "write protection" feature associated with the CPU compares the "write protect key" (two bits in the PSD) with the portion of a 512 bit "write-protect lock" image (private CPU register) associated with the page of memory (512 word) being referenced. An attempt to modify memory with an improper "lock and key" match results in a fault trap. This checking operation proceeds in parallel with instruction interpretation and thus does not lengthen instruction execution time.

The hardware configuration for our system is shown in Figure 2. For the remote terminals and the standard peripherals, data transfer with the main memory is accomplished through one of the two "Multiplexor Input/Output Processors," each of which can service up to 32 simultaneous users with a throughput of about 400 K bytes/sec. For high-speed data exchange with mass storage at speeds approaching full memory speed, one "Selector Input/Output Processor" will be used. Both types of I/O processors operate independently of the CPU once an I/O operation has been initiated. They also contain internal hardware which permits them to execute a succession of I/O commands from core memory ("command chaining") without CPU intervention once they have the address of the first command in a chain.

The advantages of independent input-output processors would be obviated if there were only a single set of addressing hardware for all of memory. Our configuration will contain four separate memory banks of 16K words each. Each bank of memory is accessed through a port, one port being dedicated to the CPU and one to each of the I/O Processors. Each of the banks contains its own reading and writing hardware. Therefore, the CPU can operate at full computational speed when instructions are being executed from one bank of memory while a mass storage device is transferring data through the "Selector I/O Processor" at near full memory speed into another bank.

The mass storage device will be an SDS "Rapid

Access Device" (RAD) which consists of a rotating disk (35 msec. rotation period) with fixed heads that transfer data at 3×10^6 bytes/sec. The 64 bands of 82 sectors (1024 bytes) each have a total storage capacity of about 5×10^6 bytes.

Operating system structure

Overview

Our system is not a generalized data acquisition system, but an Operating System in the fullest sense of the term as recently described by Wood,[3] consisting of a Supervisor and a set of Processors. The Supervisor was written to provide a well defined multi-program environment in which a large number of programs may operate without mutual interference; the Processors (assembler, compiler and loader), which run under the Supervisor, are the current SDS versions. Each program is fully protected from all other programs both in space as well as time. By implementing the Sigma 7 "write-protection" feature, a program is prevented from writing in a portion of memory which is not assigned to it. In addition, all I/O is executed via I/O handlers (part of the Supervisor) which fully check the validity of each request. Using the CPU internal clock, the system also keeps control of the time that it spends on each program and can abort a job which exceeds its allotted time (or proceed to the next program if a time-slice has been consumed in a time-sharing situation). The Supervisor is structured in such a manner that it is event-driven from three types of signals; request for service from a remote site, the completion of an I/O operation, or the run-out of an internal clock. With this structure it is then quite feasible to have a mix of tasks running simultaneously which include: real-time data-collection and/or processing programs requesting service via an I/O completion or external signal; a group of time-sharing conversational terminals which are time-sliced via an internal clock; and a modest batch-processing operation.

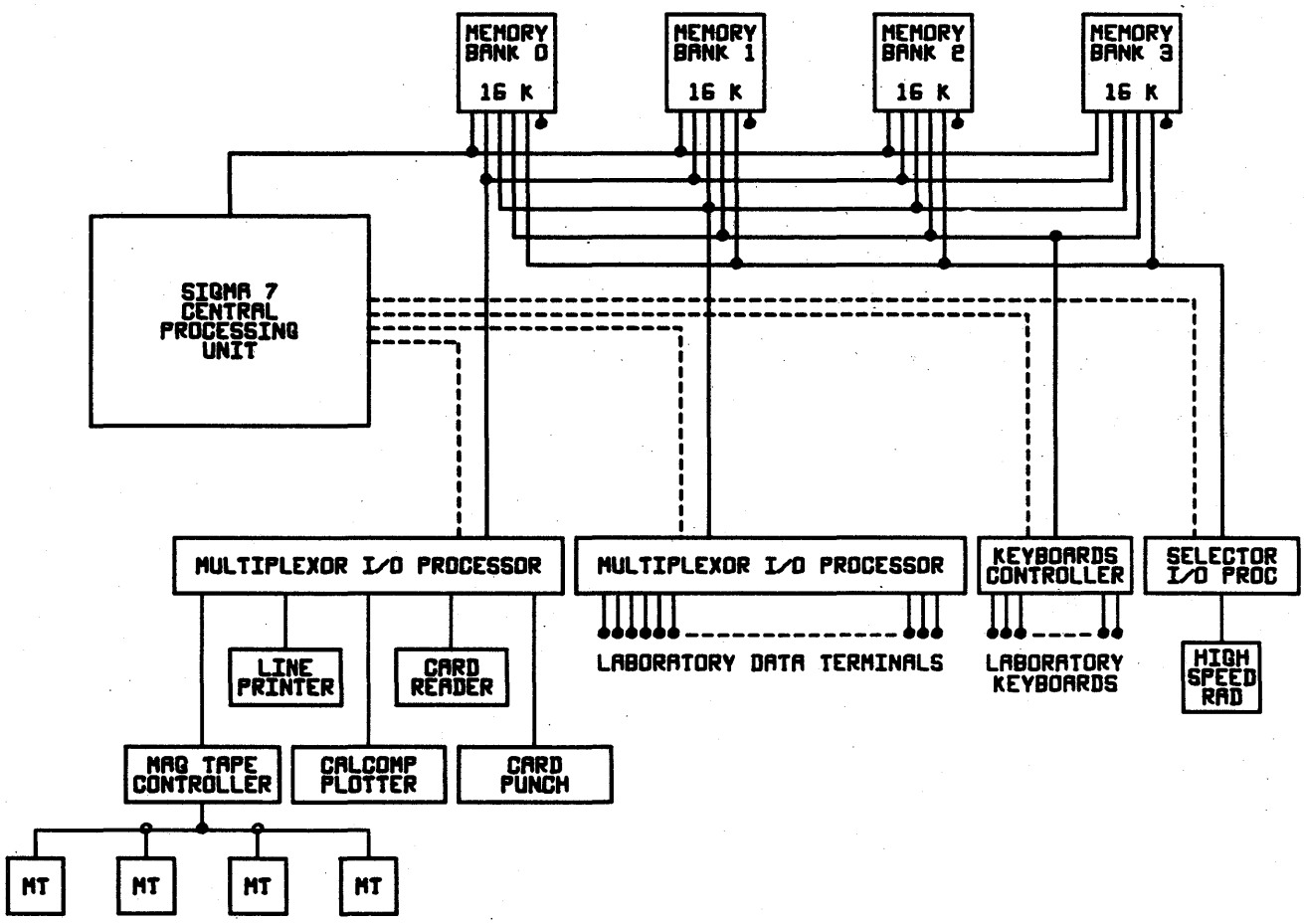


FIGURE 2—Hardware configuration

Data buffering

As indicated in an earlier section, the variety of tasks that must be performed on a real-time basis is of such a nature that they may be assigned a priority and performed on a run-to-completion basis (interruptable only by a higher priority task). The task that each program must perform generally consists of transferring the data that accumulates in memory to auxiliary storage and/or performing some modest computations and returning the results to the remote site. Since response times of less than a second are not required in most of the cases, and in order to keep the program switching rate down to a reasonable level (about a hundred per second), the data will be accumulated in blocks (under control of a Multiplexor IOP) into dedicated portions of memory associated with each program. The size of these blocks will depend on the average data rate, the peak rate, the amount of processing required and the urgency for computer analysis. In most of the programs under consideration, the experiment can proceed before a block of data is analyzed or transferred to auxiliary storage. Thus, to facilitate an uninterrupted flow of data, the data buffer areas are divided up into a minimum of two equal blocks and the "command chaining" feature of the I/O Processors is incorporated in the following manner. The first I/O command fills half of the buffer and upon filling this area the command-chaining flag directs the input/output processor to get the next command which initiates input into the other half of the buffer. Simultaneously, with the completion of the filling of the first half, the I/O interrupt of the CPU is triggered, signalling the Supervisor that service is needed. With the expected mix of tasks, the associated program should obtain sufficient CPU usage to process the buffer and extend command chaining in a circular manner before the second half of the buffer fills. As a result, the experiment can continue to transmit its data to the computer independently of the detailed needs of the other users. In the event the task is not completed before the second half of the buffer is full, the I/O for this terminal will halt because command chaining had not yet been set; the processing program will have to start the I/O operation again when the appropriate task has been completed on the other half of the buffer. For data inputs that vary a great deal in rate during the course of an experiment, this double-buffering technique can be extended to any level. Typically, these buffers are a few hundred words long with a filling time of about one second.

Software priority structure

The CPU task sequence can be readily related to the priority chain in one of two ways. The rapid switching capabilities of the external priority interrupt structure could be used if the requirements of the tasks can be estimated with sufficient accuracy and remain constant for extended periods. However, if the real-time needs change, the priority assignments must be modified. Although changing hardware priority assignments is a relatively simple changing of cables, it could often mean stopping some experiments while the cables were changed. This would be intolerable. To avoid this problem, a software priority interrupt system was designed and implemented which allows priority alteration in microseconds and has the potential for a Supervisor controlled dynamic priority re-assignment. Figure 3 depicts the priorities as they are established in the present Supervisor. Any task of higher priority can gain the services of the CPU (via an interrupt) after the request has been processed by the "Interrupt Supervisor" portion of the Supervisor.

Program description table

The Supervisor runs, services and controls the various resident programs by referring to a Program Description Table (PDT) associated with each program. These tables contain the Program Status Doublewords, the "write-protection" lock image, the current and allotted running times, space for saving the register contents when interrupted and various control bytes which assist the Supervisor in keeping track of the current status of the program. All of the I/O commands and the memory bounds are also stored in the table. The I/O handlers use this information in processing I/O requests. The integrity of these tables is insured by making them inaccessible to the user programs ("write-protected").

Input/output handlers

All input-output operations are effected by appropriate handlers which not only facilitate data buffering but also permit programs of lower priority to run while a program of higher priority is waiting for the completion of an I/O

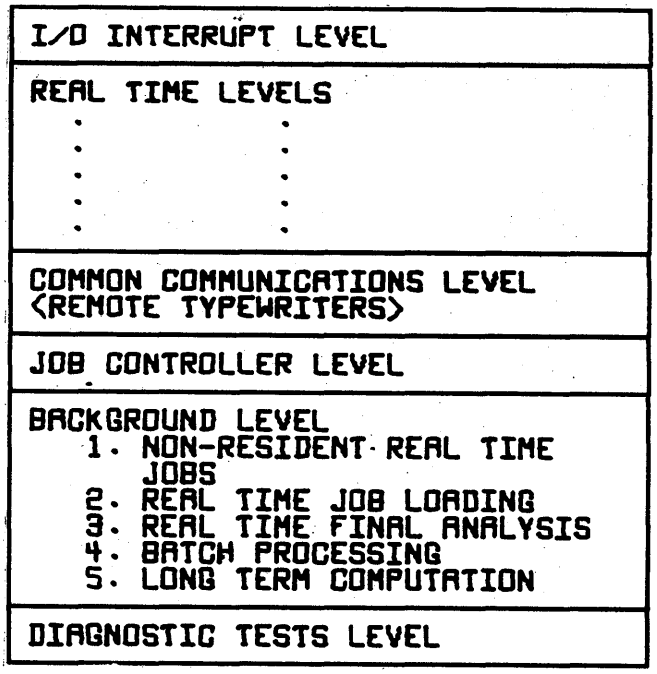


FIGURE 3—Priority structure of the operating system.

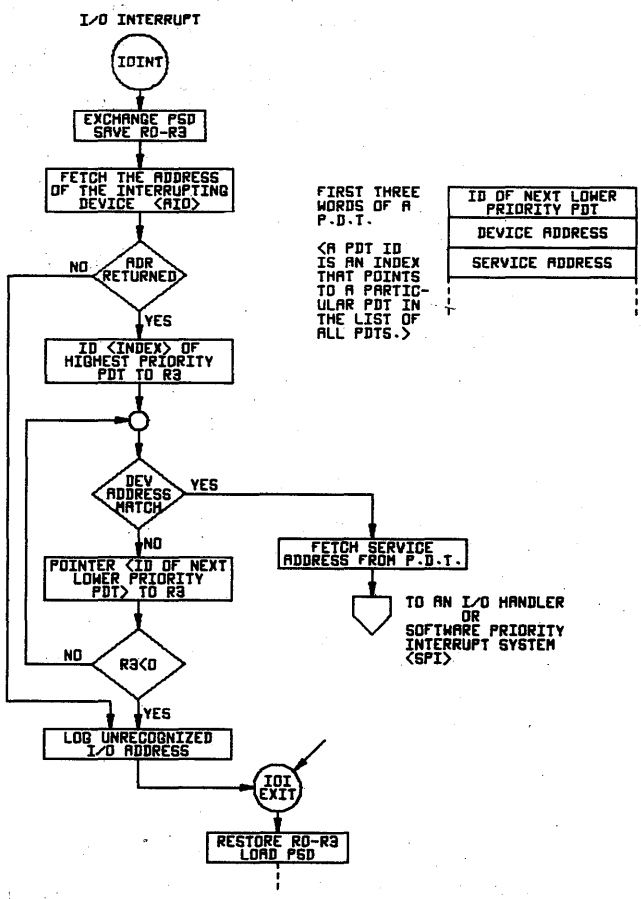


FIGURE 4—Interrupt supervisor (priority table scan).

operation. Three types of I/O requests are provided.

1) The handler returns control immediately after processing a request;
2) The handler returns control after a requested I/O operation is completed, allowing lower priority programs to run in the interim;
3) The handler returns control if, or when, the previous request (on this device) is completed.

Interrupt supervisor

As specified previously, the system will be event-driven by the numerous data sources being buffered into core memory. A full data buffer for a particular experiment will trigger the I/O Interrupt (CPU hardware interrupt), which will cause the Interrupt Supervisor to ascertain the source of the interrupt and what action should be taken. Figure 4 depicts this interrogation and scanning of the PDT's. Figure 5 indicates the manner in which the Interrupt Supervisor determines the current state of the requestor's program and whether the request is of higher priority than the current task. If it is not, an indicator ("waiting") is set in the new task's PDT and the interrupted job is resumed. However, if the new task is of higher priority, the current state of the registers, the program status doubleword (PSD) and the current time are stored in the interrupted program's PDT. The PDT for the new program is then used to set the current time and the new "write-protect" state; the new program is then started with the PSD from its table. The "Service Completed" branch in Figure 5 depicts the series of events that take place when a real-time task has been completed. In this case, the Supervisor finds the next lower priority task waiting for service and initiates or resumes this task.

Communications

The remote terminal keyboard I/O is handled in essentially the same manner as the other I/O except there is one PDT (therefore one priority level) for all of the terminals. During those periods that the CPU is not performing computations (many milliseconds per second) and a communications request is pending, the communications supervisor will scan a "remote keyboard" table in a circular fashion. Upon finding a flag in this table, it transfers control to the

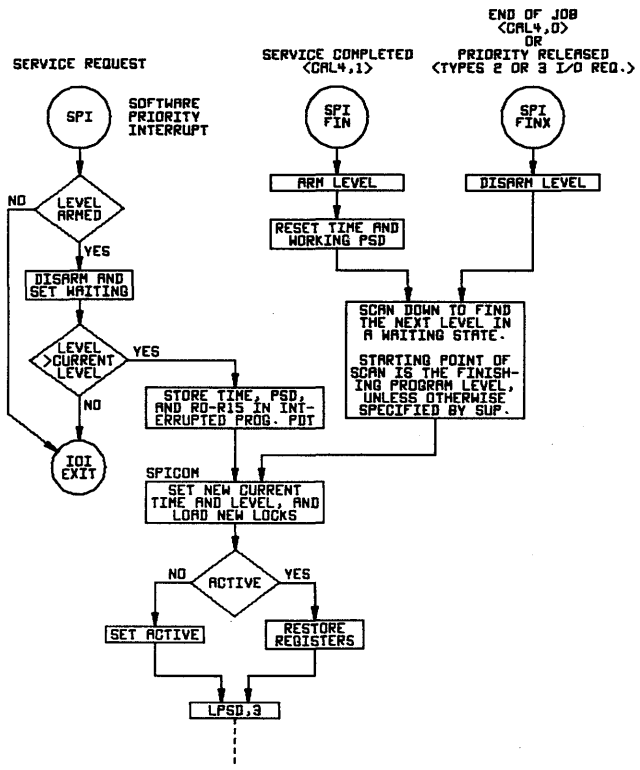


FIGURE 5—Interrupt supervisor (software task switching).

communications section of the program associated with the particular remote keyboard. Operation of this communications section of the program is carried out at a priority level just above the background priority, thus not inteferring with any real-time operations.

The operation of each remote keyboard is controllable from three different routines. (1) When a remote site is not actively sending data to the computer and therefore its program is not residing in memory, the keyboard is under the control of a supervisor level program which will accept instructions from the keyboard for the loading of the communications section of the desired real-time program. (2) Interrogation by the initial communications program establishes the required parameters and reserves peripheral devices as required. Contingent upon memory and peripheral availability, the main running program will be loaded and the system will then be ready to start the experiment. (3) During the running of the experiment, a part of the communications program will be resident to permit the modification of parameters during the experiment.

Job controller

The "Job Controller" section of the Supervisor handles the loading of real-time programs requested by the remote terminals, and places requests for final analysis in a queue for processing on a first-come-first-served basis. These operations entail finding appropriate memory space, reserving special peripherals if required, and setting the appropriate information in the associated PDT. The same functions, with the exception of job queueing, are performed for the open-shop user via instructions from the card reader. The Job Controller also loads the appropriate processor (assembler, compiler, file management routine, etc.) as required to satisfy the user's request.

Background

The background area is used for several purposes, as Figure 3 indicates. While the background level has only one PDT, the tasks have a priority to each other as indicated in the figure. A request for a higher priority task to be performed in the background area than the one currently being run, will cause the Job Controller to "checkpoint" the task in progress. As a result, up to four partially completed tasks may be stacked on the RAD at one time. Of highest priority is the "non-resident real-time" user, whose requirement for service is not urgent; these requests arrive at infrequent intervals (30 seconds or longer) and require a rather large program (10 to 20K words). Next lowest in priority are "real-time final analysis" tasks which are requested by the users' real-time programs at the completion of an experiment. Below these is the "real-time loading" operation. All three of the above uses of the background level are individually queued so that all the requests for a particular level of service are completed before the next lower priority task level is undertaken or resumed. The level indicated by "batch-processing" is associated with the open-shop input through the card reader; it is at this level that all assemblies, compilations, file management and general computations are performed. The lowest level, "long-term computations," will use overlay techniques rather extensively in order to fit into a memory availability of about 20K words.

Finally, during periods when the CPU is not required by any other function, the "machine

diagnostics" will run; these consist of a simple set of system testing instructions to find intermittent trouble in the hardware.

SUMMARY

An Operating System has been designed and written which provides a multi-program capability for about 25 core-resident programs. Some of the unique features of this system are: absolute isolation of individual programs in space as well as time; low system overhead (5%); selective loading of programs from mass storage by remote terminals; and the ability of simultaneously running real-time processing, conversational time-slicing and batch-processing. The System has been debugged as thoroughly as possible on a Sigma 7 configuration with standard I/O devices. Memory requirements consist of about 6K words for the Supervisor, 38K words for all of the resident real-time programs (including data buffers) and about 20K words for the background processing. The System will be able to service the relatively large work load described (25 experiments) with a modest hardware configuration by taking advantage of commonality of requirements, utilizing re-entrant routines.

REFERENCES

1 R J PANKHURST
Comm ACM 11 119 1968
2 SDS SIGMA 7 Computer Reference Manual 90 09 50C May 1967
3 T C WOOD
FJCC 209 1967

Holographic read-only memories accessed by light-emitting diodes

by D. H. R. VILKOMERSON, R. S. MEZRICH, and D. I. BOSTWICK

RCA Laboratories
Princeton, New Jersey

INTRODUCTION

Though it has been recognized for some time that photographic material has the highest information packing density of any present storage medium, preholographic techniques[1] to utilize this capacity required a lens system. Lenses were necessary to use the fine detail storable on film, and precise mechanisms were needed to place this fine detail before a photodetector. Such photographic systems provided an easily made, very large, fixed store in a small size at low cost, but suffered from two major drawbacks:

1. Long access time due to the mechanical nature of the access, and
2. Sensitivity to dust and scratches due to the easily obscured nature of the fine detail carrying the information.

Using the photographic emulsion in a different way, specifically to record holograms[2] rather than photographs, eliminates these drawbacks. The principles of holography are outlined in the next section; to understand how a holographic memory works requires only the knowledge that the information on the hologram is stored in the photographic emulsion in such a way that the hologram, when illuminated by a laser beam, produces a real image at a particular area in space. That is, holograms store not only the pattern of the image, but its spatial position as well; in a sense the hologram "contains" both an image and a lens that focuses that image at a particular area in space. It is this property of imagining at a particular area in space that eliminates mechanical parts from this photographic memory; a series of holograms carry the information-images such that they all image to the same area in space. Then to read out, instead of mechanically positioning the proper photograph in front of a lens and photodetector, the pfoper hologram is illuminated, imaging a pattern onto the photodetector plate, which decodes the optical pattern into a parallel readout of a binary number. Thus, the access time is no longer limited by mechanical processes.

The nature of the hologram also eliminates the second drawback, that of dirt or film imperfection sensitivity. It is shown in a latter section that every small area of the hologram contributes to the amplitude of every part of the complete pattern. A piece of dust or a scratch lowers the total "signal," but only by the ratio of the obscuring area to the total area of the hologram. By making this hologram area large compared to dust size, dust and scratch immunity are ensured.

We, as well as others[3,4] have developed a general hologram memory system utilizing these characteristics of holograms, as shown in Figure 1. Each hologram would store an image corresponding to points on a photodetecting array. Because of the hologram's spatial memory, one matrix of detecting positions could be used in common for all the holograms. The presence of a light point at a particular location would serve to indicate a one in a particular digit of a word. Each hologram "page" stores a different block of data, corresponding

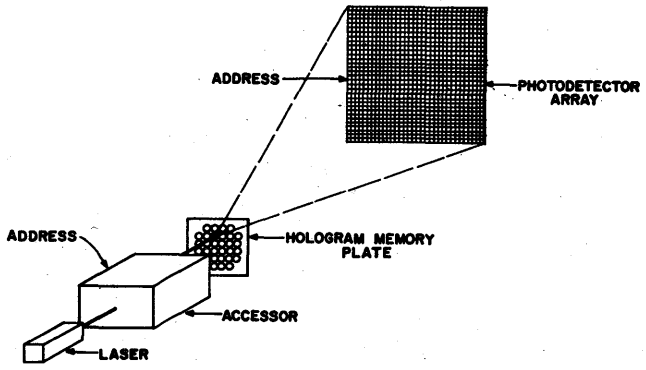


FIGURE 1—The general structure of the hologram read-only memory

to a number of words. A word would be selected by accessing a particular hologram and reading out a particular word from the block of data for that hologram page. Since the full information capacity of photographic emulsion can be utilized, hundreds of millions of bits per square inch of holographic medium can be stored. There is no mechanical positioning required in the system, since only a light beam is moved; therefore high speeds are possible.

Using gas lasers and a deflection system capable of many spot positions, on-line storage of billions of bits at near main frame memory access time seems possible. To efficiently use such a memory system, the organization of the computer would have to be changed to something embodying the "firmware" concept, [5] i.e., incorporating into the hardware the extensive software that now is transferred in and out of the computer.

Of course the difficulty of absorbing such a memory system into present machines is not the only one. Much effort on deflection systems and photo-detection systems must be expended before the huge capacities and high speeds inherent in such a hologram memory system can be realized.

What we wish to treat in this paper is a more modest version of the general hologram memory system. We use an array of GaAs light emitting diodes as an accessing means, that is, we associate a hologram with each element of an x-y array of GaAs diodes; then turning on a particular diode produces the data field of that hologram on the photodetecting array. Figure 2 shows such a memory system.

As will be discussed below, the characteristics of GaAs diodes, in either their lasing or non-lasing mode, restricts the capacity of such a memory to 10^5— 10^6 bits. However, because the GaAs diodes can be turned on in less than 10 nanoseconds, access times in such a memory system can be well below 100 nanoseconds. Moreover, the restrictions on capacity also indicate greater tolerance to alignment. Since the only connection between the accessing, storage, and detecting portions of this memory system are optical, it becomes easy to *change* the stored information. All that is required is removal of one mosaic of holograms and its replacement with another.

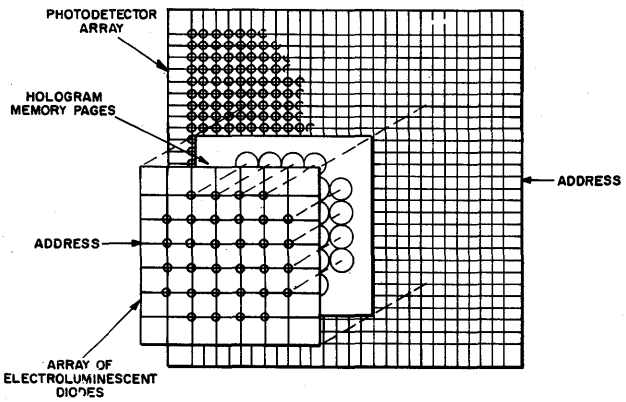


FIGURE 2—Exploded view of light emitting diode accessed hologram ROM.

With this GaAs diode array system a very fast, medium capacity, read-only memory with changeable contents becomes realizable[6]. Since many of the existing third generation computers contain microinstructions in read-only stores of about the same capacity as that indicated for the diode-accessed holographic memory, it would seem that the existing read-only memories could be replaced by this type of holographic memory; such a system could be an order of magnitude faster and allow for increased flexibility of CPU configuration by easy change of the microinstruction repertoire.

The components of a light-emitting diode accessed holographic read-only memory

As indicated in Figure 2 there are three elements of this holographic memory system: the holographic storage medium, the GaAs diode array, and the photodetecting array. We will examine these three elements in turn, attempting to indicate how the parameters of each part influence the others and the total system characteristics.

Holography—General considerations

The basis for the choice of holography as the approach to a read-only memory is its ability to achieve high bit packing densities, with great redundancy, and its lens-like nature. For an intuitive grasp of how the hologram, through the storage of interference patterns, functions we consider Figure 3 in which a hologram is made of a point light source. (It is understood that the analysis for many point sources follow immediately from arguments of superposition.) In this configuration the film is an equiphase plane with regard to the reference beam, i.e., the phase is the same at every point of the film. However, the phase of the object beam varies over the film surface. If we look at an expanded view of the film, as in Figure 4, we can see that there will be regions of constructive and destructive interference over the film area. The spacing between these regions (actually lines) can be shown to be given by

$$d = \lambda/\sin\theta \qquad (1)$$

where λ is the wavelength of the light, and θ is the angle between the reference beam and a ray from the object as seen from the small area of the film. When the

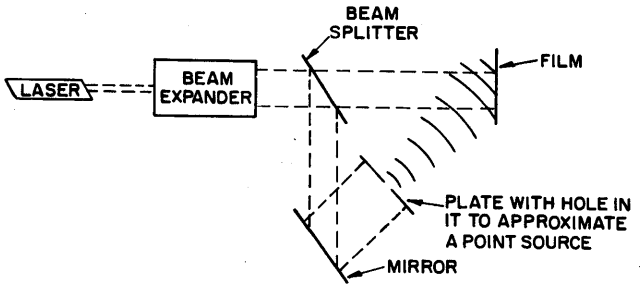


FIGURE 3—Holography of a point source

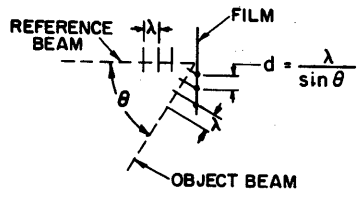


FIGURE 4—Interference at film to form a hologram

film is developed it will have the patterns of constructive and destructive interference recorded on it. The spacing between lines will be on the order of a few microns, and will vary slightly over the area of the film due to the difference in angles of the rays reaching the top and bottom of the film. When the developed film is illuminated by a laser beam in the same direction as the original reference beam, the film will act as a slightly modulated diffraction gratings with the results shown in Figure 5. Three beams are produced. The first is the undiffracted beam (called the O'th order beam) and contains no information. One will be diffracted at an angle θ (upwards in the figure) and will appear to be coming from a point source behind the hologram. This is referred to as the virtual beam, and is not used in the memory system to be discussed. The third beam will be diffracted at an angle $(-\theta)$ and will come to a point focus at a position conjugate (with respect to the film) to the original point. This is referred to as the real beam and is the one used in the memory system.

A number of advantages of holography are apparent from the above discussion.

The first is its high storage capacity. Since no lenses need be used, either in formation or playout, it is possible to utilize the full storage capabilities of photosensitive materials, which can be as high as 10^9 bits per square inch. The ultimate capacity of holographic store can be found from the following arguments. The capacity of the hologram is inversely proportional to the size of the real image spot mentioned above. The size of the spot can be found by applying the theory of thin lenses (the hologram can be shown to be analogous to a thin

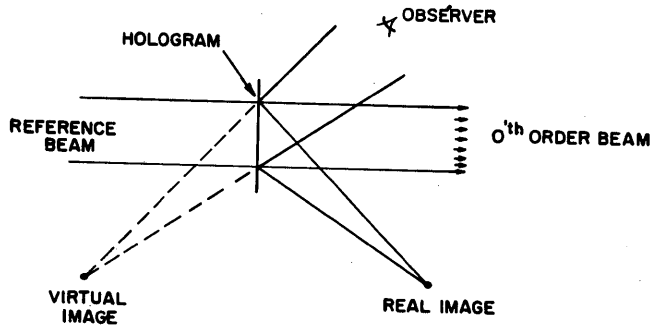


FIGURE 5—Playback of a hologram of a point source

lens as regards image resolution) and find the size of the spot (d) given by

$$d = \frac{2f\lambda}{a} \tag{2}$$

where a^2 is the hologram area, λ is the wavelength of the illumination and f is the hologram to spot distance. In steradians, the angular spot size is:

$$= 4\,\lambda^2/a^2 \tag{3}$$

We see that the larger the hologram the smaller will be the spot and thus the more information we can store. To find the total capacity of the hologram (P) we divide the total angular extent of the image (an array of spots) by the angular extent of each spot to get (we allow half the area for guard space)

$$P = \frac{1}{2}\,\frac{\theta^2\,a^2}{4\,\lambda^2} \tag{4}$$

where θ is usually about one radian ($\approx 60°$). Typically, $a = 1$ mm, $\lambda = 5 \times 10^{-4}$ mm, $\theta = 1$ radian to get

$$P = \frac{1}{2} \times 10^6 = 5 \times 10^5 \text{ bits/hologram} \tag{5}$$

or, this would store at a density of 3.1×10^8 bits per square inch. Clearly by holography one can utilize almost the full storage capability of the recording medium.

The second advantage is the hologram's dust and dirt immunity. Since light from the point source is spread over the entire hologram's surface (thus ensuring interference patterns over the entire film surface), any part of the hologram will reproduce the same image as any other paft of the hologram. It can be seen that the only effect of dust and scratches is to reduce the active area of the hologram. This effect manifests itself only in a

reduced image intensity (in proportion to the stored area) until the area is reduced to the point that aperture limiting reduces resolution to intolerable levels.

The third advantage of holography is its lens-like nature. This is readily apparent from Figure 5 if we consider the real image. The hologram has not only stored the spot, but also its position in space; the hologram, effectively, has a "built in" lens. While the hologram has the advantage of a "built in" lens in being able to project an image to a particular point in space, it does not have the disadvantage of a lens system in magnifying errors in position. In a lens system, any error in position of the subject is exaggerated in the image by the magnification of the system (which is quite high in conventional high density systems); in the holographic system, any positional error of the hologram is not magnified, the image is effectively "locked" to the hologram. Thus the holographic storage system is tolerant of positioning errors. It is this feature, coupled with the holograms inherent redundancy, that allows a memory with removable media.

One further point, one that will be important later, is the hologram's efficiency; that is, the ratio of the total power into the real image to the total power incident on the hologram. It has been shown theoretically that efficiencies as high as 33 percent can be expected, and in the laboratory this figure has been approached for simple systems. A more practical figure is about 5-10 percent, as will be discussed in a following section. The power per bit (or spot) in the image is then given by the total power into the hologram times the hologram efficiency, divided by the number of bits (spots) in the image, or

$$I_{pt} = \frac{I_0 E_{ff}}{N} . \qquad (6)$$

For a more complete analysis of general holographic considerations see Ref. 2

Holograms—Special considerations

In the introductory discussion of holography, it was supposed that the reconstruction would be done with the same kind of light used during the hologram construction; that is monochromatic, spatially coherent light from a laser. Accessing holograms by means of a light emitting diode however, imposes restrictions on the quality of the playback to be expected, for the light from such a source is neither monochromatic nor spatially coherent and is usually at a different frequency than the light which formed the hologram. We should now like to examine the problems caused by the use of light emitting diodes in hologram reconstruction and see the extent to which they limit the capacity of such a system.

The hologram can best be viewed as a collection of optical gratings, and as such the governing relationship for reconstruction is (see eq. 1)

$$\sin \theta = \lambda/d \qquad (1)$$

where λ is the wavelength of the reconstruction illumination, d is the grating spacing and θ is the angle at which the image-forming light leaves the hologram. (Incident illumination is assumed to be perpendicular to the hologram.) It should be noted that the grating spacing (d) is a function of the wavelength of the illumination used during hologram construction and of the angle the light beam made with the film. The relationship determining the grating spacing is essentially eq. (1) above. Following this discussion, and with the aid of eq. (1), we are now in a position to discuss the first limitation of a Light Emitting Diode accessed hologram, wavelength shift.

Gallium arsenide (GaAs) light emitting diodes emit light at a wavelength of about 9000 Å. High resolution photographic film, of the type needed in holography is only sensitive to visible illumination (3000 Å—7000Å). As a result, the hologram must be formed with visible illumination (for example 6328Å) and played back, or reconstructed, with infrared light (9000Å). Examining eq. (1) we see that if no precautions are taken severe distortion will result in the image, for a change in λ will cause a change in θ, the playback angle. Furthermore, the larger the angle the greater will be the distortion. Precautions can be taken however, to reduce the effect of wavelength shift. They are discussed in detail elsewhere,[7] and will be briefly described here. Since the wavelength shift to be experienced is known, and since the distortions as a function of wavelength shift is known the first precaution is to simply predistort the information to be holographed. A second step is to use as the reference beam in reconstruction one that converges to a point in the image plane. This technique is known as Fourier Transform Holography and can be shown to have distinct advantages over other holographic techniques in minimizing distortion due to wavelength shift. Since neither of these techniques are perfect, the third, and perhaps most important from the viewpoint of hologram capacity, is to limit the largest angles over which the image is stored. Typically the angles are limited to 45° to the hologram normal (compared to the 60°—70° normally used).

Temporal coherence, or the degree of monochromaticity, is the second effect to be considered. Referring back to eq. (1), we can see that a lack of monochromaticity (i.e., $\lambda = \lambda_0 \pm \Delta\lambda$) will result in a loss of resolu-

tion in the hologram image, for a spread in wavelength will cause a related spread in output angle. A detailed analysis shows that with a given degree of temporal incoherence (i.e., a given $\Delta\lambda$), the maximum number of spots that can be recorded or the number of bits (P) that can be stored, is given by

$$P = \left(\frac{\theta \max}{\tan \theta \max}\right)^2 \left(\frac{\lambda}{\Delta\lambda}\right)^2 \quad (7)$$

as long as the size of the hologram (a) is less than

$$a \leq \frac{2}{\tan \theta \max} \frac{\lambda^2}{\Delta\lambda} \quad (8)$$

where θ max is the maximum angle, as seen from the hologram, over which the image is played out, λ is the mean wavelength of the illumination, and $\Delta\lambda$ is the wavelength spread. As examples, with a mean wavelength of 8500Å, a $\Delta\lambda$ of 30Å, and a θ max of 45°, (all values for lasing diodes) the number of bits that can be stored is 5×10^4. If now $\Delta\lambda$ is 300Å (the value for non-lasing diodes), the capacity is reduced to 500 bits per hologram.

The third, and perhaps most important limitations arise from the spatial coherence of conventional light emitting diodes. The GaAs diodes is an extended source of light, each segment of the source emitting light independently of every other segment. As a result there is no unique, time-invariant representation of the emitted radiation that can be made that relates one part of the wavefront with another. It is this that is meant by spatially incoherent illumination. Such illumination imposes a severe limitation on the capacity of a holographic store, for it can be demonstrated that if spatially incoherent light is used to illuminate a hologram, and if the source as seen from the hologram is of a given angular extent, then the smallest image from the hologram will have the same angular extent. As a result the number of bits per hologram (the number of resolvable spots in the hologram's image), which is inversely proportional to the smallest resolvable spot in the image, is reduced. There are a number of ways to overcome this limitation. The first is by the use of a lens to reduce the angular extent of the source as seen by the hologram. The second, and more elegant, is the use of small, lasing GaAs diodes. These diodes, as will be discussed below, have a high surface brightness, near monochromatic output, and if made very small have spatially coherent light emission.

There are other factors that could be considered, for example the effects of emulsion shrinkage. However, this and other effects are not peculiar to this application of holography and so will not be discussed here. Complete discussion can be found in the literature. See for example Ref. 7.

Light emitting diodes

There are actually two types of light emitting diodes to consider, incoherent and coherent (lasing) diodes.

We will first consider incoherent light emitting diodes and then turn out attention to coherent (lasing) diodes.

Incoherent diodes emit light, when forward biased, at a mean wavelength of 9100Å, with a spectral width of 300Å. The diodes typically have a low power efficiency, about 2%, requiring large currents to achieve high light output. In our experiments, using a commercially available diode, four amps are required to obtain 40mw of emitted radiation. The diode response time, current in to light out, is relatively long, typically 100—150 ns. It is suspected that the slow speed of response is due to a mechanism involving deep traps which must be filled before light of appreciable intensity will be emitted.

Surface brightness, a figure of merit commonly used to compare different sources of illumination, is 40 mw/mm^2 which is large when compared to conventional sources but low, as will be seen, when compared to a lasing source. Surface brightness is important in that it ultimately determines, regardless of the kind of optics used, the power per point in the image. A low surface brightness therefore, will result in a low storage capacity in the hologram memory.

As mentioned previously, such a source is spatially incoherent. Furthermore, due to its low surface brightness, the source size must be large to achieve reasonable power levels. We can see therefore, that such a source will always impose a severe limitation on the capacity of such a memory (especially if a high speed memory is desired).

Coherent (lasing) diodes emit light at a mean wavelength of 8500Å, with a spectral width of less than 30Å. Above threshold (approximately 50 ma) they have a high differential power efficiency about 30 %, allowing low drive currents. The speed of response is very high, the diodes typically turn on in less than ten nanoseconds.

Surface brightness is very high, typically 8×10^3 mw/mm^2. Thus even with incoherent illumination, small source sizes can be used and still yield appreciable power levels, implying high storage capacities. Encouragingly however, light emission has been achieved for small ($\approx$1/2 mil) lasing diodes which is completely coherent and implies that capacities of up to 30×10^3 bits per hologram can be achieved. The small size of these diodes also allows one to fabricate an array of light sources which is quite compact. Arrays of 1 mil diodes on 10 mil centers are contemplated for future models of the memory.

The main disadvantage of coherent (lasing) GaAs diodes is that presently they must be used in a liquid nitrogen enviornment. With the availability of reasonably low cost closed cycle liquid nitrogen refrigerators this is not a real disadvantage, mainly a psychological one.

The photodetecting array

Figure 6 shows a word-organized photodetecting array suitable for the Light-Emitting Diode holographic memory system. The total analysis of this matrix will be published elswhere; in this section we will outline how this matrix works and its characteristics.

A word is selected by turning on the switching diodes of a particular (horizontal, in the case of Figure 6) line by closing its switch (a transistor). Then the photocurrent generated by light falling on the back-biased photodiodes (which can have typically 70% quantum efficiency) on the vertical lines has a path through the sense amplifier, which has low imput impedance, through ground and the conducting word line. Photocurrents generated in the photodiodes associated with an unselected word line see a high impedance path and so do not contribute to the sense signal. The risetime of such an array has been calculated to be

$$\text{R.T.} = 5 \times \text{wn } R_{\text{switch diode}} C_{\text{photodiode}}$$

where 5 represents the ratio of a selected word sense signal to worst case unselected word signal, wn the number of bits in the detecting matrix, $C_{\text{photodiode}}$ the capacity of a photodiode, and $R_{\text{switch diode}}$ the on resistance of the switching diode. Putting in typical values of 5 pf for photodiode, 1 ohm resistance switching diodes and a 1000 bit array, a 25 nanosecond risetime is indicated.

One of the features of this array is the isolation of the selection current, which flows horizontally through the switching diodes, and the signal currents, which flow through the vertical line. Since the large currents associated with driving the GaAs diodes are electrically separate, due to the optical coupling, high speeds without drive noise spikes in the sense output are achievable. Moreover, this simple diode structure is fairly easy to integrate, and the photodetecting array can be built up of smaller subsections. Such photodetecting arrays have been built at out laboratories and operated successfully.

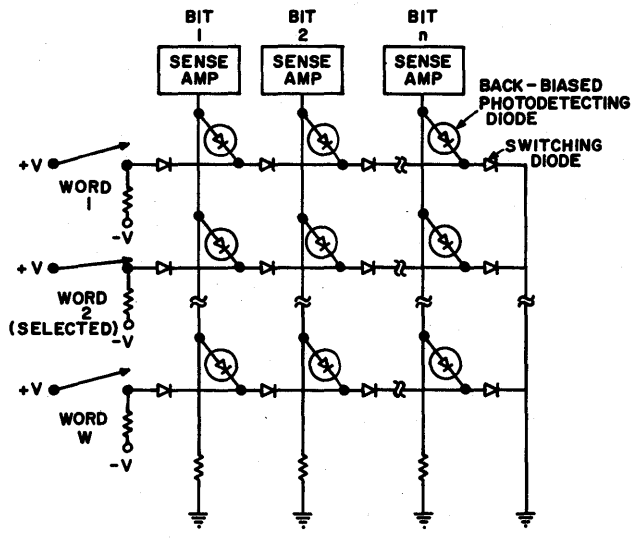


FIGURE 6—A word-organized photodetecting array

Discussion of experiments

Experiments were performed to demonstrate the implications of the previous statements. To illustrate the salient features of the concept, a small working model of a holographic read-only memory accessed by light emitting diodes was built.* The model consisted of four holograms accessed by four GaAs light emitting diodes. The hologram stored 26 bits, with the output of the hologram projected onto a 26 bit photodetector array as described in the previous section. The outputs of the photodetector array were amplified to a suitable level, threshold detected, and strobed out. Each component will be examined in turn.

To compensate for the wavelength shift, as discussed previously, each hologram stored a predistorted array of spots as shown in Figure 7, with the output of each hologram a square array. The hologram was made with the configuration schematically

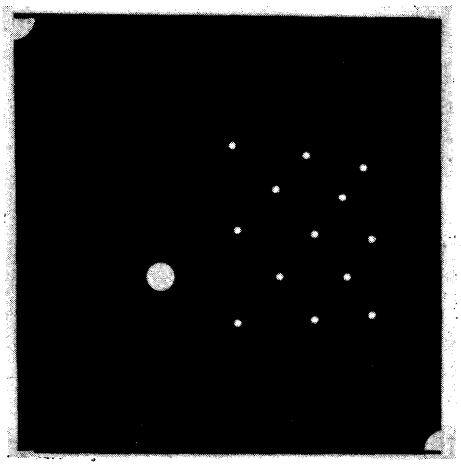


FIGURE 7—Input spot array, showing reference beam position. Notice lack of squareness

* For ADP/ECOM, Fort Monmouth, New Jersey.

shown in Figure 8, using light from a He-Ne laser ($\lambda = 6328$Å). The reference beam diverged from a point 1″ in front of the hologram surface and at an angle of 2° to the hologram normal.

The sources of illumination for hologram reconstruction were commercial GaAs light emitting diodes. The diodes were driven by a pulse amplifier capable of delivering 4 ampere pulses with a 50% duty cycle at 10 Mhz. The pulses was able to deliver the four ampere pulse within 25 nanoseconds at the initiate pulse, including delay plus current risetime.

The desired diode was selected by means of a switch. A simple lens was used to collect the light emitted by the diode to provide the convergent beam needed for undistorted hologram reconstruction. As discussed earlier the playout beam converged to a point at the plane of the image 1″ from the hologram.

The image of each hologram was projected onto a photodetector array, of the type discussed previously, with each hologram imaging on the same photodetector array. The light sensitive elements of the array were hpa 4207 diodes. The word diodes were hpa 1006 high conductance diodes. Separate word lines could be selected either mechanically, by means of switches, or electronically.

The output of each bit line was fed into a low input impedance, high gain, wide bandwidth, amplifier. The amplifier had a total delay plus rise time of about 50 nanoseconds, an input impedance of 50 ohms, and a transfer ratio of 1.5 volts per 1 μwatt of illumination incident on a photodetector. The output of the amplifier was strobed and then applied to a threshold detector.

Sense amplifier output of two bits read in parallel is shown in Figure 9. There are a number of things of interest to be seen in the figure. The first is the high one to zero ratio obtained from this kind of memory system.

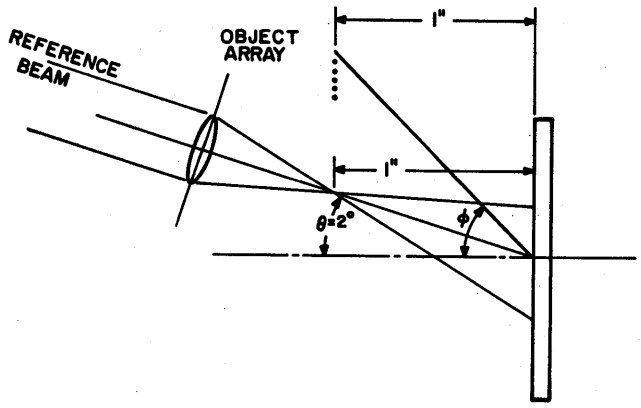


FIGURE 8—Recording arrangement

The second is the long rise time of the light emitting diodes. Figure 10 illustrates the access time ofthe system. The top trace is the input to the light emitting diodes, pulse amplifier (the address pulse), the second trace is the sense amplifier output, and the bottom trace is the processed output. The access time shown is 200 no; again it can be seen that amost all of the delay is caused by the slow turn on time of the light emitting diodes, as seen in the second trace. As shown the pulse duration is 500 ns. This length was chosen for pictorial reasons only, and is not representative of pulse duration achievable. As we have said above, the main speed limitation is the time of response of the light emitting diodes, which with incoherent diodes is about 150 ns, while with coherent light emitting diodes it is about 10 ns, or less.

The minimum detectable signal, the value of which was a prime objective of this study, for it ultimately determines the speeds and capacities that can be achieved in future systems, was found to be 2.5×10^{-14} watt-seconds. (This corresponds to a 1/4 μwatt signal of 100 nanoseconds duration, we give the result in terms of minimum energy for reasons of generality.) This value would allow a 10:1 signal to noise ratio, or an error rate of 10^{-8}, which is minimally acceptable. With this figure it is possible to determine the energy needed

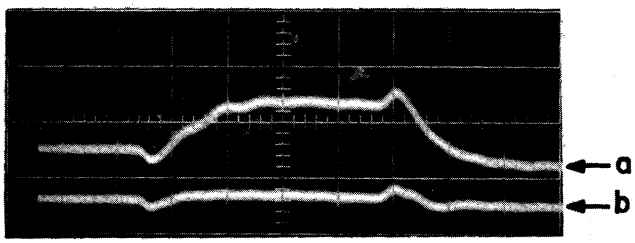


FIGURE 9—The sense amplifier output for a one (a) and zero (b). Vertical scale ½ volt/cm Horizontal scale 100ns/cm

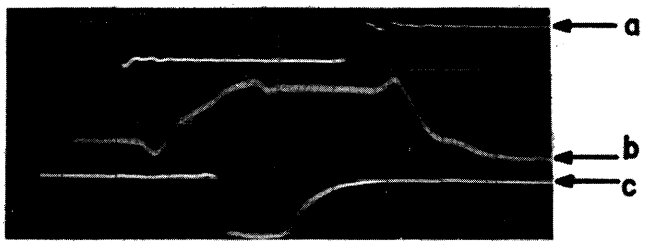


FIGURE 10—Operating waveforms of hologram read-only memory using incoherent light-emitting diodes
(a) Initiate pulse
(b) Sense amplifier output, showing rise time of light-emitting diode
(c) Strobed output
Horizontal scale 100ns/cm

to access a hologram memory system, for if N is the number of bits, E_{ff} is the hologram efficiency, then the energy needed is

$$E_{TOT} = \frac{N \times 2.5 \times 10^{-14}}{E_{ff}} \text{ watt-seconds .}$$

The hologram used in this model had an efficiency of 5%, which is typical for the type of holograms used. This means the minimum energy needed in a hologram memory is

$$E_{TOT} = N \times 5 \times 10^{-13} \text{ watt-seconds .}$$

The storage capacity of the holograms used has been shown to be most severely limited by the low surface brightness and lack of spatially coherent radiation of the light emitting diodes. The maximum number of bits that can be stored with such a source is below 200 bits per hologram if access times of .5 μsecs are desired. The large physical size of the light emitting diodes does not allow compact spacing of the array of sources (and thus of holograms) and thus limits the capacity of a complete system, for the conditions of a maximum hologram output angle of 45° and a hologram array size larger than the photodetector array are not compatible. As a result, if the holograms can not be spaced compactly, only a small number of holograms can be used, limiting the system capacities. All of the above indicate that coherent (lasing) diodes must be used to make the concept realizable.

CONCLUSIONS

From the discussion in this paper, the advantages of a hologram read-only memory are manifest. Digital information can be stored with high bit density and with great redundancy. The need for fine, highly corrected lenses is obviated; accordingly, the need for precise, mechanical placement does not exist. Easy replacement of the storage medium is possible since the only link between the medium and the rest of the system is light, requiring no electrical or mechanical connections. The most important advantage, however, is that through the use of holography large information stores can be achieved with relatively few electric components. In a properly made system, the number of components needed in a conventional memory is essentially divided by the number of holograms used: each hologram projects its images onto the same detector array, reducing the need for extensive electronics.

While these advantages have been apparent to us and others for some time, a total system has not been implemented due to a lack of an efficient means of accessing the hologram at high speed. It has been the purpose of this paper to describe a technique to achieve such a means, the use of an array of light emitting diodes to access the holograms.

The use of light-emitting diodes as a hologram selector severely reduces the capacity of this type of hologram memory when compared with that using a gas laser. However, the use of these diodes allows very rapid access to the memory store, something not yet achievable with the gas laser systems.

This is the philosophy of this class of read-only memories: sacrifice capacity for the speed and easy changeability of contents. Experimentally we have demonstrated the feasibility of making holograms for such systems and devised photodetecting arrays of requisite sensitivity and speed. As both our analyses and experiments have indicated, the critical element in this type of memory is the accessing array. Incoherent diodes possess neither the surface brightness nor short turn-on time required for this memory system. With lasing diodes, capacities of 10^5—10^6 bits at cycle times below 100 nanoseconds appear achievable with present hologram and photodetecting array techniques. However all present lasing diodes need to be cooled to 70—150°K. With further improvement in the device characteristics, an order of magnitude larger capacities become possible. Such memory systems can be evaluated on the basis of the analysis presented in this paper.

REFERENCES

1 R J POTTER
Optical processing of information
Spartan Books Baltimore 1963

2 D GABOR
Microscopy by reconstructed wave-fronts
Proc Roy Soc London A197 454 1949

3 H FLEISHER
Application of interference photography to optical information storage
Presented at Holography Seminar Colorado State University January 1967

4 L K ANDERSON et al
A high-capacity semipermanent optical memory
Presented at Conference on Laser Engineering and Applications Washington DC June 1967

5 A OPLER
Fourth generation software
Datamation 13 22 1967

6 R CHAPMAN M FISHER
A new technique for removable media read-only memories
Proc of 1967 Fall Joint Computer Conference Anaheim Calif 1967

7 D BOSTWICK D H R VILKOMERSON R S MEZRICH
Techniques for removal of distortions in hologram images caused by a change in playback wavelength
Presented at Spring Meeting of Optical Society of America March 1968

Semiconductor memory circuits and technology

by WENDELL B. SANDER

Fairchild Semiconductor
Palo Alto, California

INTRODUCTION

In the past few years the use of semiconductor flip-flops for a significant portion of the memory of a computing system has been approaching a practical reality. The pressures and commitments within semiconductor laboratories toward achieving a competitive edge in the main frame memory market are increasing. The memory field offers a new market area as opposed to the displacement of existing products in a new form. Furthermore, the manufacturing process for semiconductor memories is characterized by the mass production of similar items; this process is in exact accordance with the present production methods in the semiconductor industry and is far less painful than that of the LSI logic field, which threatens to lead to an endless product proliferation at small volumes. Although the memory field is in its infancy and largely speculative, it is possible to make a brief survey of the technology and the concepts being pursued in the developmental laboratories. A further review of semiconductor memory can be found in Reference (1).

Chip technologies

At the present time three distinct chip technologies are being considered; bipolar, p-channel MOS and complimentary MOS. The bipolar process used for semiconductor memory is similar to conventional processing including multi-layer metal. Memory circuits are repetitive and can be relatively non-critical of component value permitting exploitation of the bipolar process in new ways to achieve a marked improvement in component density. An excellent example of this is a cell described by BTL[2] at the 1967 ISSCC. The p-channel MOS technology offers a conceptually simple technology capable of producing memory cells at good density.

Progress in MOS technology is proceeding more or less independently from the particular needs of semiconductor memory and the major effort in p-channel MOS memory circuit techniques is simply to exploit the technology to its fullest. For example, it is desirable to use bipolar circuits to provide high level drive and low level sensing to achieve the best memory system performance from the MOS cells. [3,4,5]

Complimentary MOS is the integration of both p-channel and n-channel MOS devices on the same chip. It is the least well developed of the three technologies discussed in terms of production capability. There is a wide variety of fundamental approaches to complimentary MOS being developed including thin film field effect transistors,[6] silicon on saphire [7], and simple diffused silicon. Due to the need for production compatibility and low cost, the all-silicon system is the most likely near-term production process and in fact, is the only process with standard products of any kind presently available. Complimentary MOS is viewed by many (but not all) in the semiconductor field as being applicable primarily to special memory applications where the inherently low power is essential. It is not likely to be cost competitive in memory applications with either bipolar or p-channel MOS for the foreseeable future.

Packaging technologies

There are three basic technological factors in interconnection of the monolithic chips into a memory module; interconnections from the chip, second level interconnections, and chip sealing against contamination. There is a maximum size

of chip for any given technology that can be fabricated with 100% yield over the chip. This number is fairly small; on the order of 64 to 256 bits/chip with present technology. This represents a level of testing required before proceeding to interconnecting the chips (or wafer region) with a higher level interconnection.

The most straightforward way to handle these good chips is to test the wafer, ink the bad chips, dice the wafer, and throw away the bad die. However, at least one manufacturer, Texas Instruments, has taken an alternative approach, wherein the good areas are mapped and a special interconnection mask for the wafer is made and applied to interconnect these regions.[8,9] This approach will not be discussed in detail here. The following discussion assumes the handling of small chips acquired by wafer sort and dicing. The attachment of the chip to the next level interconnect can be handled by individual lead bonding of pads on the die to a next level interconnect or may be formed by anyone of several batch attachment techniques. Since memory chips tend to have a large number of leads, batch attachment is most attractive.

Some of the batch connection techniques are:

1) Ultrasonic bonding of aluminum bumps on the chip to the substrate metalization
2) Thermo compression bonding of aluminum bumps on the chip to the substrate metalization
3) Solder reflow of solder bumps on the chip to the substrate metalization (Figure 1)
4) Form solder coated beams extending beyond the edge of the chip which are bonded to the substrate with the die either face up or face down.

A comparative evaluation of methods 1, 2 and 3 may be found in Reference 10. The beam lead approach of 4) was developed by Bell Telephone Laboratories and is described in reference.[11]

The attachment is usually made with the die mounted face down on the substrate, however, techniques for face up mounting are being developed for both beam lead structures and by batch interconnection of face up chips after die attachment.[12] The major factors to consider in the attachment systems are the economics, attachment yield, repairability and heat dissipation. Ease of repair will probably be a dominant factor if many chips are attached to a single substrate. In general, the face down bonding technologies are easier to repair, but the face-up technologies allow better inspection of bond quality and better heat dissipation.

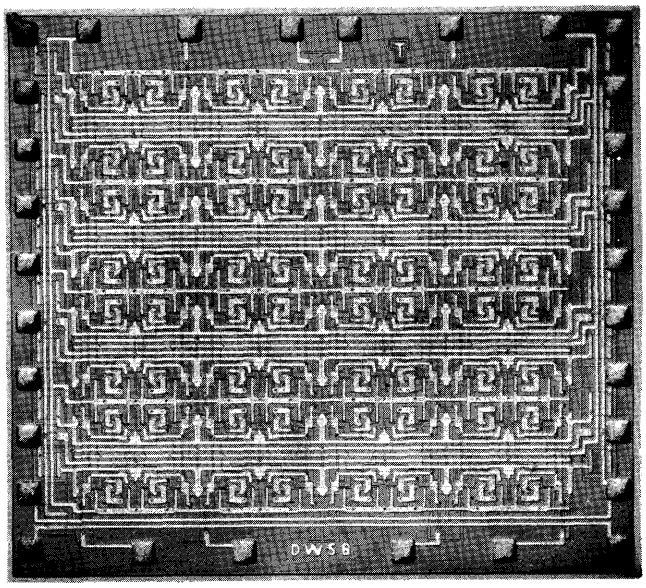


FIGURE 1—Chip with solder bumps

The substrate (Figure 2) for next level interconnection can be anything from a single chip package to dozens of uncased chips on a multilayer substrate. The best economic potential lies in multiple attachment of uncased chips to the substrate. Single layer metalization on the substrate is highly desirable from an economic standpoint but may not be adequate in all cases. The substrate material is usually either alumina with

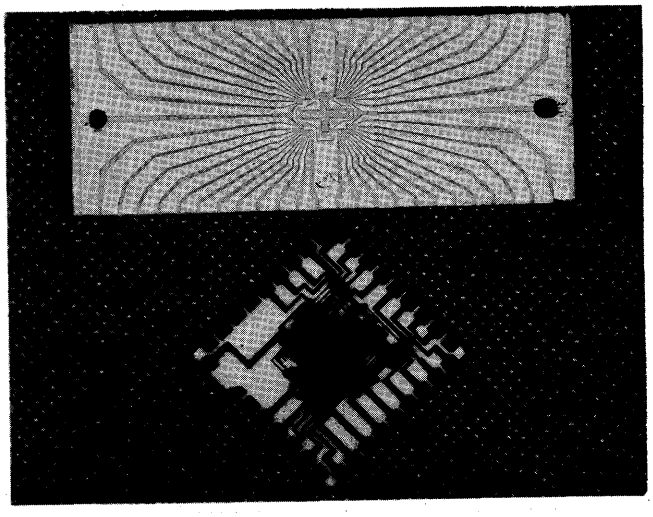


FIGURE 2—Single and double layer substrate

gold plated molymanganese interconnections or silicon with aluminum or moly-gold interconnections. The alumina is a common ceramic well understood for the purpose whereas silicon is more fragile but can provide very high interconnection densities.

Chip sealing is required to prevent surface contamination which can degrade the circuits. Chip sealing can be accomplished by a sealing cap on every chip, by sealing the entire multichip substrate (Figure 3), or by a sealing coat of silicon nitride on the chip. The nitride sealing method is by far the most attractive but is not yet a well developed production technology for integrated circuits. The most common and practical alternative is sealing the entire module. This requires a large area seal that is not easy but can be done.

Memory circuits

Read-write cells

Bipolar

Figure 4 illustrates the most common bipolar memory cell in linear select and coincident select form. This cell operates with the word line at a low potential for standby. When the word line is raised the information in the cell is read out by sensing the current in the bit line, or is written into by holding down one bit line, thus forcing the transistor on the held down side to be turned on.

The major disadvantage of this cell is that the standby power dissipation is higher than the power dissipation when addressed. This problem

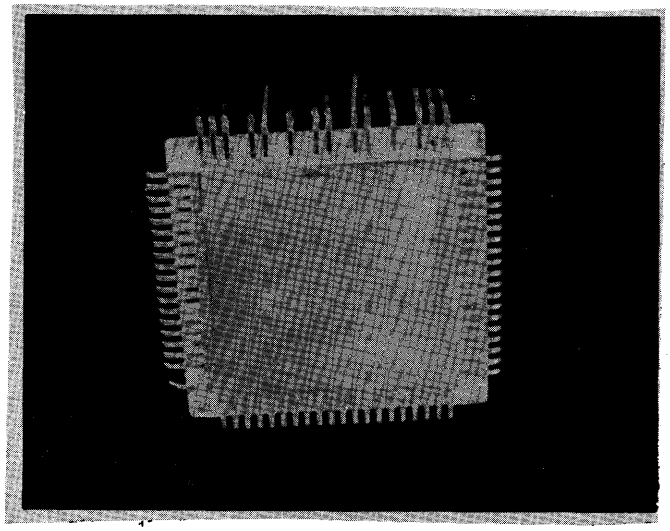


FIGURE 3—Multichip module assembly

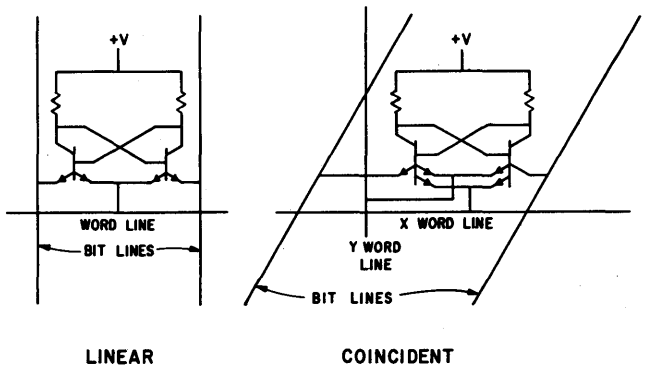


FIGURE 4—Bipolar cells

can be alleviated for the linear select cell by treating both the illustrated word line and the +v line as word lines. Both lines can be raised for addressing and minimum supply voltage for standby operation can be utilized. [8]

Other cell circuits can be used [13,14,15] and four layer devices have been proposed [16] but the cell of Figure 4 will probably be the most common bipolar cell for the near future in large memory arrays due to its simplicity.

MOS cells

Figure 5 illustrates the most commonly used MOS cell in both [17,12,13,14] linear select and coincident select form. The cell is operated by turning on the MOS transistors to connect the cell to the bit lines. The bit lines can then be sensed to determine which state the cell is in. For writing into the cell a differential voltage is impressed across the bit lines to force the cell into the desired state. To achieve the best performance from this cell the word lines (or x and y lines) are driven from powerful bipolar drive circuits, thus providing very fast addressing to the cell. If the

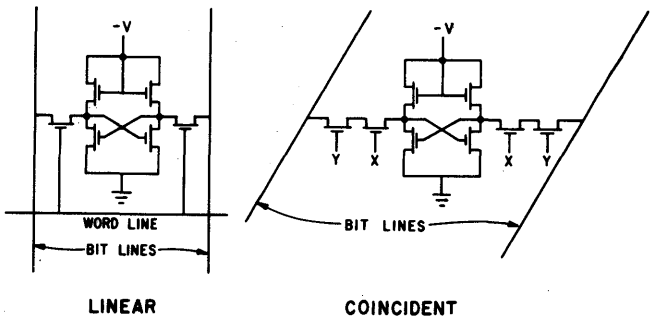


FIGURE 5— MOS cells

bit lines are held negative during the sense time then a differential current will appear on the bit lines as soon as the word line is on. This differential current can be sensed by a bipolar differential thresholding circuit with very little voltage change on the bit lines. Thus, the poor capabilities of MOS devices to drive large capacitance has been defeated by using bipolar drive and sensing. The resistance of the MOS resistors is of little importance in this cell since signal current is drawn through the ON switch and the write voltages (again bipolar) will force both sides of the cell to the correct potential. Therefore, these load resistors need only provide leakage current or be supplied by asynchronously pulsing the —v line. In this way, standby cell power can easily be in the microwatts. Peripheral circuit power and transient power are the major sources of power dissipation in the MOS memory.

Complimentary MOS

Figure 6 illustrates a complimentary MOS memory cell. This is not suggested as an optimum cell but is representative of the complexity of complimentary MOS cells.[5,6] In this cell transmission gate A is normally on to close the flip-flop loop. To read the cell transmission gate B connects the cell to the bit line and the state of the cell can be sensed. To write into the cell transmission gate A is turned off and transmission gate B is turned on so that the flip-flop will be forced to the state impressed on the bit line.

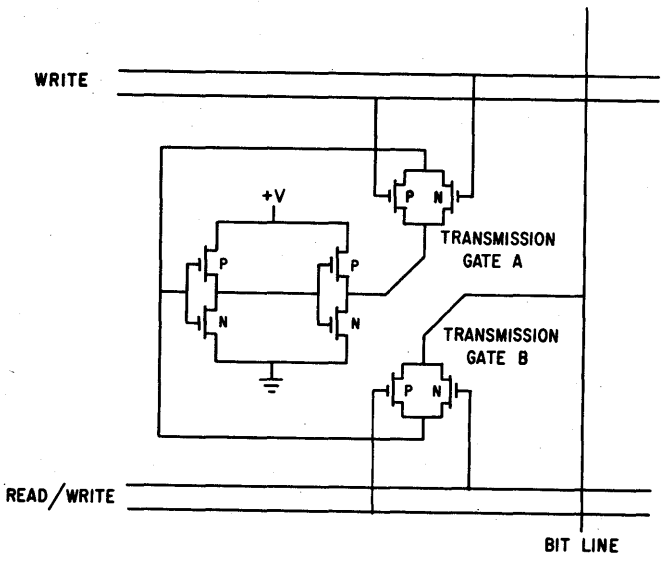


FIGURE 6—Complimentary MOS cell

Comparative evaluation

In all three technologies the memory organization will be repetitive arrays of dense cells and specialized peripheral circuitry. Only the complimentary MOS is likely to be treated as having logic compatible signal levels; the MOS and bipolar will be used with very specialized bipolar interface circuits. Table I is a brief comparison of the three cell technologies assuming present pilot line technology. The designs are assumed optimized for cost with speed a secondary factor. The speed here means the cycle time in a complete memory system of the order of 10^5 bits.

TABLE I—Cell technology comparison

	Cell Area (sq. mil)	Speed (n sec)	Power (cell only) (mw)
Bipolar	10	100	0.5
p-channel MOS	15	250	0.01
Compl. MOS	80	200	10^{-6}

The cell density of bipolar and p MOS are close enough that cell cost will not be a determining factor. The complimentary MOS density, however, is poor enough to shut it out of a raw cost race. Furthermore, it offers no major advantage over p-channel MOS in speed; therefore, the first conclusion that can be drawn is that complimentary MOS is most suited in micropower applications where very low standby power is essential. This market area is sufficiently large to assure development of complimentary MOS memories.

The bipolar vs p MOS trade-off is more involved. The bipolar has a speed advantage with no direct cell cost penalty; however, the 0.5 mw power level causes definite thermal problems. A 4K memory module could be assembled on less than 1 sq. inch of substrate but would dissipate 2 watts of power. Thus, simple air convection cooling is inadequate although other cooling methods can easily handle this power density. The p MOS cell does not have this problem and can be packaged to the mechanical limit with simple cooling.

The big market is in the low cost/good performance area. There is no clear cut winner between

bipolar and p MOS for memory cells as the tradeoffs are too close. Both will be developed and a final edge of one over the other will require some dramatic development in one of the technologies.

Bipolar technology will always have an edge in speed, however, because speeds much faster than 100 n sec. system cycle time are achievable. Therefore, bipolar cell technology is assured a niche in the memory market.

On-chip decoding

As cell density increases there is increasing difficulty in getting the interconnections off the chip. For example, a 4 mil by 4 mil linear select cell will have leads on 4 mil centers in one direction and 2 mil centers in the other. These leads can be brought off on alternate sides around the chip giving 8 mil pad to pad centers on two sides and 4 mil centers on the other two. Coincident select cells require only 8 mil centers on all sides but the cells will be larger with a given technology.

These lead centers are manageable in terms of actually making the connections to the substrate but the substrate problem can be serious. It is very difficult to place 4 mil centers on a ceramic substrate, although such placement is not too bad on silicon. Unfortunately, the unscrambling around the chip will be a maze and will almost certainly require complex multilayer substrate interconnection.

If the lead density can be significantly relieved, then single layer substrates can be considered since part of the interconnections can be reflected onto the chips if more pads are available. This all leads to the consideration of on-chip decoding. On-chip decoding must be very simple since it must be duplicated on many chips. The simplest decoding is diode networks in bipolar and series gating in p MOS. In both cases on-chip address inversion is prohibitive. In bipolar the complexity is too severe and in p MOS the speed penalty is too severe. Assuming on-chip decoding with non-inverting logic, three choices of input are available:

1) True and complement binary signals. For 2^n bits 2 n lines are required. The internal decoding gates are n input gates.
2) Multi-dimensional decoding. For 2^n bits two dimensional decoding requires $2(2^{n/2})$ lines (n even) and a decoding gate fan-in of 2: Three dimensional decoding requires as little as $3(2^{n/3})$ lines with a gate fan-in of three. Multidimensional decoding can be continued to the point that each dimension is only 4 lines wide (plus possibly one 2 line dimension) where the number of lines required is only 2n and the number of gate inputs is $\frac{n}{2}$ (n even) or $\frac{n+1}{2}$ (n odd). Thus, the number of inputs is the same as for true-complement inputs but the decoding gates are simpler.
3) Combinatorial Decoding. If there are m input lines and the internal decoding gates have n inputs then $\binom{m}{n}$ independent signals can be decoded. For example, with 12 inputs and 3 input gates the number of lines that can be selected on chip is $\binom{12}{3}=220$ whereas the multidimensional case would also use 12 inputs and 3 input gates but only select $2^6=64$ lines.

Of the three input structures the multidimensional decoding is superior to true-complement input and the input code is very easy to construct. The combinatorial decoding is the most efficient on the chip but the input code is very complex to construct and the number of select lines is not inherently a power of 2. Since the multidimensional decoding is sufficient to solve the lead problem (for example 16 lines for 256 bits vs 48 lines for the undecoded linear select cell) the complication of combinatorial decoding is not required.

Figure 7 illustrates decoding networks for the bipolar cell. Both word line decoding and bit line decoding are possible. On the word line all diode inputs must be high to permit the selected word line to rise. On the bit lines all inputs must be high on the selected bit line pairs to eprmit the cell current to be coupled through the output diodes to the bit line bus. Note that by using both word line and bit line decoding the decoding is broken into two networks so that for 256 bits, two 4×4 decoding networks can be used requiring only 2-input gates.

Figure 8 illustrates decoding for the p MOS structure. Series gating is the most desirable form of decoding since the series gates can be made with low on impedance and driven by bipolar drivers. Word line decoding is again possible, however, the drive voltage of the series gate must be somewhat higher than the highest potential expected on the line and since the word

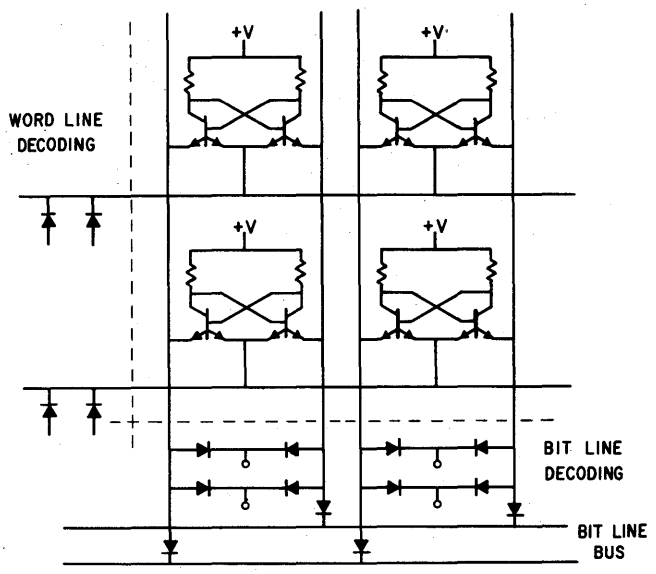


FIGURE 7—Bipolar cell decoding

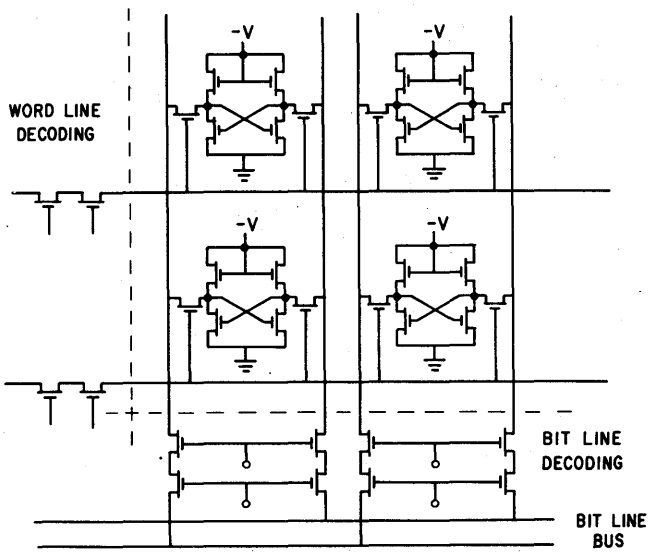


FIGURE 8—MOS cell decoding

line is already driving a series gate the input voltages must be quite high. On the bit lines, however, the series decoding gate drive required is the same as the normal word line so that bit line decoding is more desirable since the voltage swings are somewhat smaller.

Peripheral circuit consideration

The peripheral circuitry for semiconductor memory has many of the attributes of peripheral circuitry for magnetic memory. In particular the largest possible fan-outs and fan-ins are desirable such that powerful drivers and sensitive sense circuits are used. This provides a minimum peripheral circuit overhead cost.

The p MOS word lines and bit lines are generally capacitive requiring drive voltages in the 5 to 20 volt range with sense currents in the order of 0.1 ma. The high voltage capacitance drive required leads to active pull-up drivers. Transient power at these voltages can be the dominant power requirement of the system.

Bipolar cell drive circuits require a low voltage swing (in the order of 1 volt) but currents in the range of 0.1 to 0.2 ma per bit, therefore, bipolar cells use a low voltage current driver and driver power is small compared to cell power. Bipolar sense circuits must sense 0.1 ma signals with small bit line perturbation so the sense circuit requirement are quite similar between bipolar and p MOS cells.

Other memory forms

Other memory structures than simple read-write are possible. The most commonly considered are associative memories,[19,20] multiport memories and read-only memories. Of these three, read-only memories are making the biggest splash in the market place.

Both bipolar and p MOS read-only memories are available from semiconductor manufacturers today. Both contain complete logic compatible decoding on the chip and use a fixed connection pattern applied during fabrication. Thus, there is a tooling charge to get the pattern desired. The bipolar version is somewhat faster but the p MOS version is somewhat lower cost. The p MOS memory is stored by selectively creating or connecting to a dense array of MOS transistors. The bipolar memory is stored by selectively connecting to an array of diodes or emitter followers. The actual array densities can be quite similar, however, the logic compatible p MOS decoding is much more dense than the logic compatible bipolar decoding.

An economic example

Table II provides an illustration of the economic factors in semiconductor memory. This is an example of potential 1972 costs in high volume production. A 4096 bit module is assumed using 16, 256 bit bipolar memory chips, 4 decode-drive chips and 2 sense-digit drive chips. The packaging

is assumed as a single layer interconnect ceramic substrate with repairable upsidedown chip attachment. The module would have about 70 n/sec read or write cycle time and would dissipate about 2.5 watts.

TABLE II—Module cost

Cost Center	Cost Each	Total/ Module	Cost/Bit (cents)
256 bit chip	$1.00	$16.00	0.4
Decode/drive chip	0.75	3.00	0.075
Sense/digit chip	1.00	2.00	0.05
Substrate	10.00	10.00	0.25
Attach cost	.50	11.00	0.275
Test	2.00	2.00	0.05
		44.00	1.1
25% yield loss		11.00	0.275
Repair		1.00	0.025
Retest		1.00	0.025
		57.00	1.43
5% Yield loss		3.00	0.075
		60.00	1.5

Table II illustrates module cost alone and does not include system packaging or power. The potential for system packaging compatability and at least partial power supply compatibility are favorable factors in semiconductor memories.

CONCLUSIONS

Semiconductor memory is looming as contender for a major portion of the computer main frame memory market. Complimentary MOS and bipolar technologies are assured of a niche in the micro power and very high speed areas respectively. The major battle will be between bipolar and p channel MOS for the low cost (and high volume) segment of the market.

The semiconductor and packaging technologies required are in an advanced state of development and a major impact should be seen within the next few years.

REFERENCES

1 D A HODGES
Large-capacity semiconductor memory
Proceedings of the IEEE Vol 56 No 7 July 1968

2 J E IVERSON J H WOURINEN JR B T MURPHY D J D STEFAN
Beam-lead sealed-junction semiconductor memory with mini ma cell complexity
IEEE Journal of Solid-State Circuits Vol SC-2 No 4 pp196–201 December 1967

3 P PLESHKO L M TERMAN
An investigation of the potential of MOS transistor memories
IEEE Transactions on Electronic Computers Vol EC-15 No 4 pp 423–427 August 1966

4 D E BREWER S NISSIM G V PODRAZA
Low power computer memory system
AFIPS Conference Proceedings vol 31 pp 301–393 FJCC 1967

5 J H FRIEDRICH
A coincident-select MOS storage array
Digest of Papers pp 104–105 ISSCC 1968

6 J R BURNS J J GIBSON A HAREL K C HU R A POWLUS
Integrated memory using complimentary field-effect transistorsl
Digest of Papers pp 118–119 ISSCC 1966

7 J F ALLISON F P HEIMAN J R BURNS
Silicon on sapphire complimentary MOS memory cells
IEEE Journal of Solid State Circuits Vol SC-2 No 4 pp 208–212 December 1967

8 R S DUNN G E JEANSONNE
Active memory design using discretionary wiring for LSI
Digest of Papers ISSCL 1967 pp 48–49

9 R S DUNN
The case for bipolar semiconductor memories
AFIPS Conference Proceedings Vol 31 pp 596–598 FJCC 1967

10 P SCHARLT T COLEMAN K AVELLAR
Flip component technology
Proceedings 1967 Electronic Components Conference pp 269–275

11 M P LEPSETTER
Beam lead technology
Bell systems Tech J vol 45 pp 233–253 February 1966

12 J MARLEY J H MORGAN
Direct interconnection of uncased silicon integrated circuit chips
Proceedings 1967 Electronic Components Conference pp 283–290

13 G B POTTER J MENDELSON S SIRKIN
Integrated scratch pads sire new generation of computers
Electronics vol 39 No 7 pp 119–126 April 4 1966

14 H A PERKINS J D SCHMIDT
An integrated semiconductor memory system
Proceedings—Fall Joint Computer Conference 1965 pp 1053–1064

15 I CATT E C GARTH D E MURRAY
A high speed integrated circuit scratch pad memory
Proceedings Fall Joint Computer Conference 1966 pp 315–331

16 R P SHIRELEY
SMID A new memory element
Proceedings Fall Joint Computer Conference 1965 pp 637–647

17 J D SCHMIDT
Integrated MOS transistor random access memory
Solid State Design January 1965 pp 21–25

18 A W BIDWELL
A high speed associative memory
Digest of Technical Papers ISSCC 1967 pp 78–79

19 R IGARASH T KAROSARA T YAITA
A 150-nanosecond associative memory using integrated MOS Transistors
Digest of Technical Papers ISSCC 1966 pp 104–105

2-1/2D core search memory

by PHILIP A. HARDING and MICHAEL. W. ROLUND

Bell Telephone Laboratories, Inc.
Naperville, Illinois

Usual memories allow addressing of individual word lines with each word line containing an assemblage of bits. Bit detectors which can sense all words are utilized to read the word contents. Associative memories allow bit lines to be addressed, as well as word lines, to determine which set of words match the input bit states. Such memories are useful because they eliminate time consuming word hunting in table look up operations, and because they allow easy access to specific words highlighted by activity or flag bits.

Unfortunately, most associative memories proposed are expensive. Possibly circuits based on large scale integration may allow low cost associative operations, but they are not economically feasible today. Core associative memories rarely have been proposed; those described have complex memory mat structures.[1] In most cases, the cost penalties far outweigh the usefulness of the true associative array.

The semiassociative solution, one in which a segment of any single bit line in an array can be addressed to read out the corresponding bit locations for a number of words may be the compromise that finds a wide range of applications.[2] This type of memory, defined as a single bit search memory, is symbolically illustrated in Figure 1. Such a memory may be operated in the ordinary sense; one of the "n" unique addresses can be selected to read the "m" bit word contents. Similarly, a unique address may be chosen and the m bits can be independently written. The figure illustrates the selection of address "A_i" either for reading or writing the "m" bit contents.

In the search mode, a single block of K address locations associated with any single word bit, B_j, may be selectively read or written. The figure highlights the B_jth bit of words 1 through K in block 1 and the B_ith bit words of SK through (S + 1)K in block S as possible search words. The entire word field of n can be arranged in n/K blocks of K words each to facilitate the search mode. It has been found that a modified 2 wire or 3 wire 2-1/2D Core Memory can economically

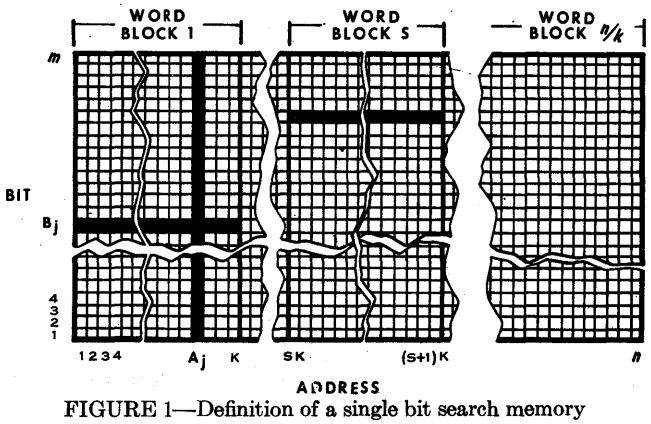


FIGURE 1—Definition of a single bit search memory

achieve the single bit search memory function if only one m-bit normal address word or one K-bit search word is selected during one memory cycle.

2-1/2D, 2-wire memory

A single bit search memory can be thought of as an extension of the conventional, 2-1/2D, 2-wire core memory shown in Figure 2, which was reported on in an earlier paper.[3] In a 2-wire, coincident current memory, the readout voltage from a particular core must obviously be sensed across one of the two drive wires intersecting the core. In the memory of Figure 2, the core array is composed of a front and rear plane, with the readout voltage being sensed differentially across a pair of selected bit wires, one in each plane. The center-tapped connection of the bit readout transformer also forces the bit drive current to divide equally between the front and rear planes. The group selection circuit provides a virtual ground to the selected pair of memory wires in each bit while simultaneously isolating the non-selected wires, which are multipled to the readout transformers at the top of the array. Hence, any noise voltages induced on the nonselected wires are not coupled into the readout. The group selection circuit is formed of diode rails connected in a driven bridge configuration

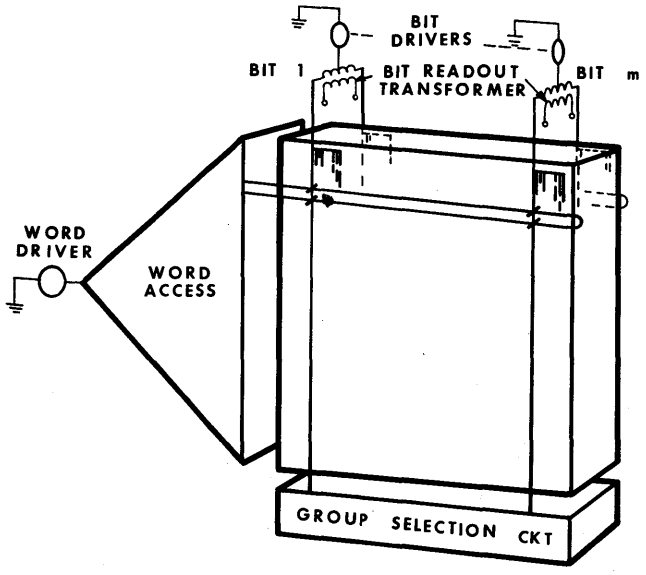


FIGURE 2—2-1/2D core memory

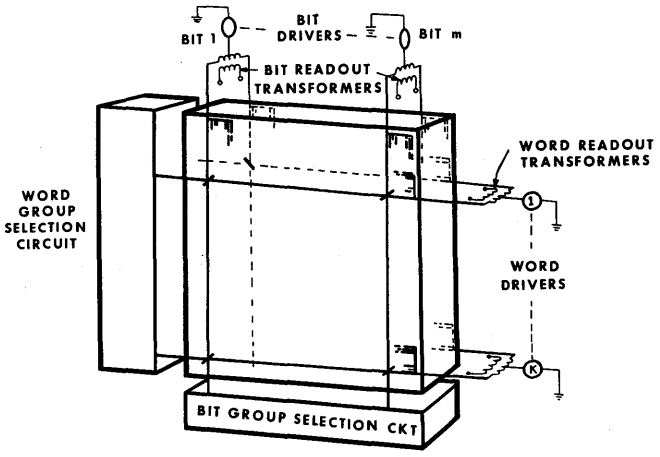


FIGURE 3—A single bit search core memory

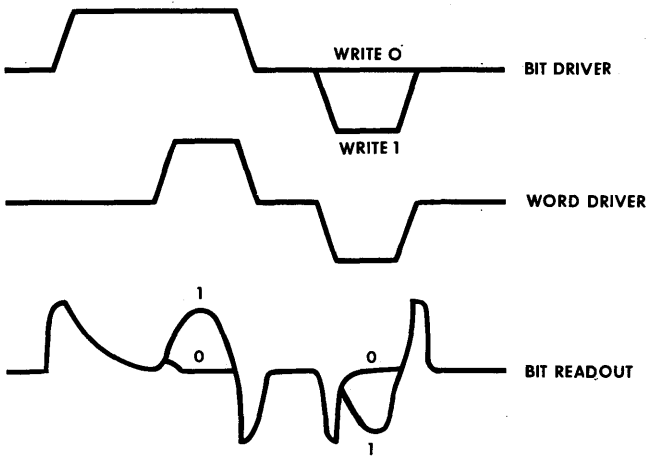


FIGURE 4—Memory address sequence: bit driver current word driver current, bit readout

of the type described in Reference 4.

The word access consists of a diode matrix driving folded word loops each of which intersects two cores on a given bit line. Since the word and bit currents will add in one of the cores and cancel in the other, the direction of word current is used to select one core or the other, thus reducing the number of word access circuits required by a factor or two.[4] The looping of the word wire has a number of additional advantages. The cores can be oriented in-line, rather than in a diamond pattern, increasing packing density. The shuttle voltages due to word current tend to cancel. And finally, no more than half of the cores on a bit line can be disturbed by word current into a worst case delta noise state.[4]

A word is read out of this memory in the following fashion. Bit read current is applied first causing a large noise spike in the readout. When this noise has expired, the word current is applied, reading out the state of the m bits of the word. The word is rewritten into memory by reversing the word current and applying reverse bit write current to those locations where a "one" is to be stored. This timing is indicated in Figure.4.

Single bit search memory

The word access of Figure 2 supplies current to only one selected word loop. At little additional cost, we can perform the same selection process with an access that is virtually identical to that used in the bit dimension, as shown in Figure 3. We can select a pair of word lines (rather than a single loop) by energizing the appropriate word driver and word group selection circuit. Note that the cores on the front and rear planes are oppositely phased with respect to the word current so that only one core is selected per bit. However, we now have the added flexibility of being able to readout in the word dimension. That is, rather than energizing m bit drivers and then a single word driver and finally sensing the readout on m pairs of bit wires, we energize all K word drivers first and then a single bit driver while sensing the readout on K pairs of word wires. Thus, we can readout the state of K different word locations of a given bit in a "search" mode, or all m bits of a word in the normal or "address" mode. The timing for the search mode is illustrated in Figure 5. The readout wave form is the same in either mode, but appears across the bit transformers in the address mode and across the word transformers in the search mode.

4 plane single bit search

The word wiring scheme of Figure 3 lacks the shuttle

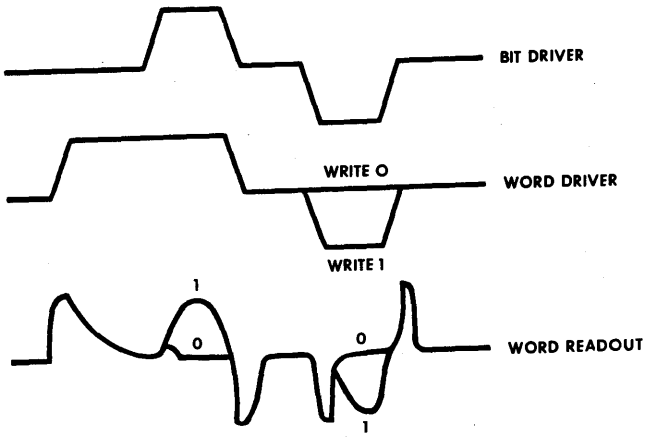


FIGURE 5—Memory search sequence, bit driver current, word driver current, word readout

and delta noise reduction advantages of the word line looping of Figure 2. These advantages can be regained by using the four-plane configuration of Figure 6. The word and bit lines are wired so that only one plane receives a simultaneous word and bit current. This is indicated in Table 1.

TABLE 1—4 memory plane wiring

CORE SELECTED	BIT LINES
PLANE 1	PLANES 1, 2
PLANE 2	PLANES 1, 2
PLANE 3	PLANES 3, 4
PLANE 4	PLANES 3, 4

WORD LINES
PLANES 1,4
PLANES 2,3
PLANES 2,3
PLANES 1,4

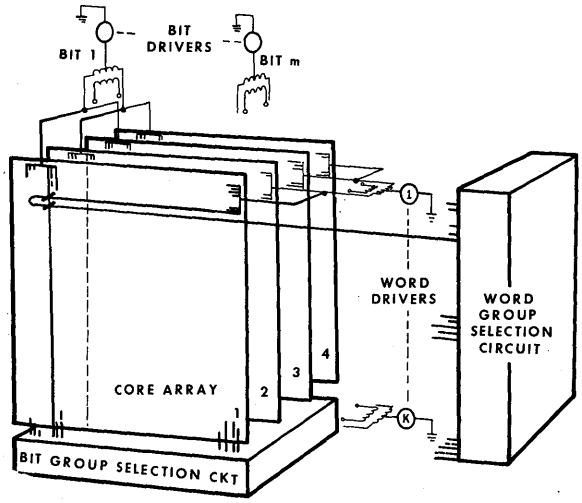


FIGURE 6—Improved search memory access

Search memory operation

Let us consider the operation of a search memory of n words, having m bits per word, and with the word access divided into K segments (each segment having an independent driver, detector, and register cell). The block diagram of such a memory is shown in Figure 7.

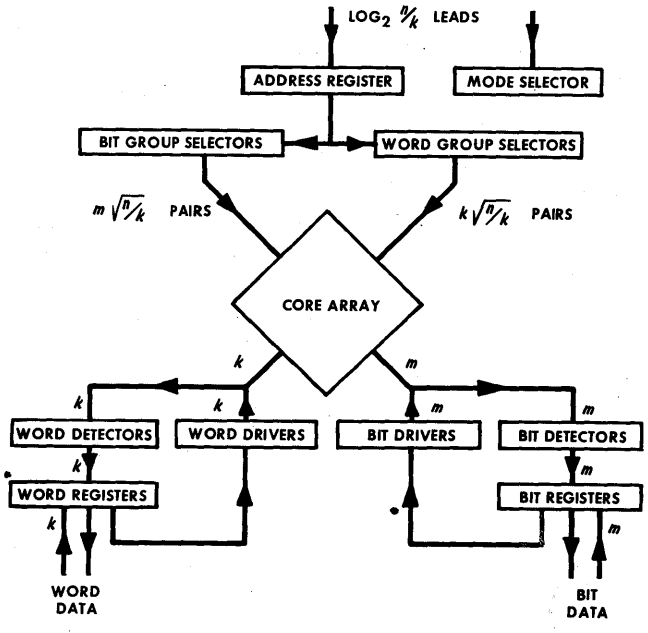


FIGURE 7—Single bit search memory block diagram

In a normal or address mode, the address to be interrogated is supplied in two parts; $\log_2 \frac{n}{K}$ address bits are supplied to the address register which controls the bit group and word group selector circuits. As an example let us assume equal bit and word group selector circuits. Then the bit group selector circuit decodes 1/2 $\log_2 \frac{n}{K}$ inputs and selects one of the $\sqrt{\frac{n}{K}}$ groups of m bits each. The word group selector decodes the remaining 1/2 $\log_2 \frac{n}{K}$ inputs and selects one of the $\sqrt{\frac{n}{K}}$ groups of K bits each.

The remaining 1 out of K address selection is performed via the word data input, which energizes the appropriate 1 out of K word drivers. All m bit drivers are atcivated and readout is accomplished via the m bit detectors.

In a search read mode, $\log_2 \frac{n}{K}$ address bits are again supplied to the address register to control the group selector circuits. However, now the roles of the word data and bit data inputs are interchanged. All K word drivers are energized, corresponding to the K addresses over which the search is to be made, while only a single bit driver is energized, dependent on which bit is to be searched. Readout is accomplished via the K word detectors.

Flag bit memory

As mentioned earlier, one of the attractive applications of a search memory is in the reduction of hunting for an active or "flagged" word. If a single bit of the word is reserved to indicate activity in the remainder of the word, that particular bit can be called a "flag" bit. If one performs a search read on the flag bit, then those locations in the word register which are set will correspond to the words which are "active," and subsequent normal reads can be used, with the active word locations automatically stored in the word register, to determine the entire contents of the active words. If multiple words are flagged in a word block, some form of priority selection may be necessary. In the case of few active words, the flag bit search can reduce the hunting time by a factor of K. In practice, the reduction can be as great as one or two orders of magnitude.

Ripple search

In the case where more than a single flag bit is required to locate a desired address in memory, a "ripple search" technique can be employed. This simply involves sequentially searching through the flag bits, eliminating all addresses which mismatch on any of the desired flag bits. The matching operation is readily accomplished by using a word register such as the one shown in Figure 8. The register is initially set to the "'1's" state by a timing pulse. If the readout from a given word detector mismatches the match bit B_J, the corresponding flip-flop in the word register is reset. On the subsequent search read, the match bit becomes the second desired flag bit B_{J+1}, and those readouts which mismatch will again reset their corresponding word register flip-flops. Thus, any word flip-flop which remains set after all of the flag bits have been searched corresponds to an address whose flag bits match all of the desired flag bits.

The following example further illustrates the ripple search technique. Suppose that the desired flag bits B_J through B_{J+3} are 1001 and that the memory contents

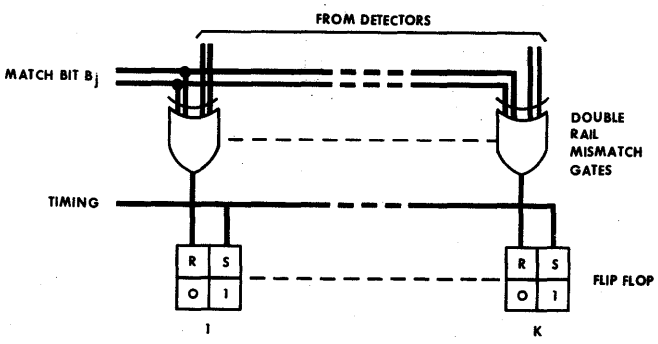


FIGURE 8—Ripple search word register

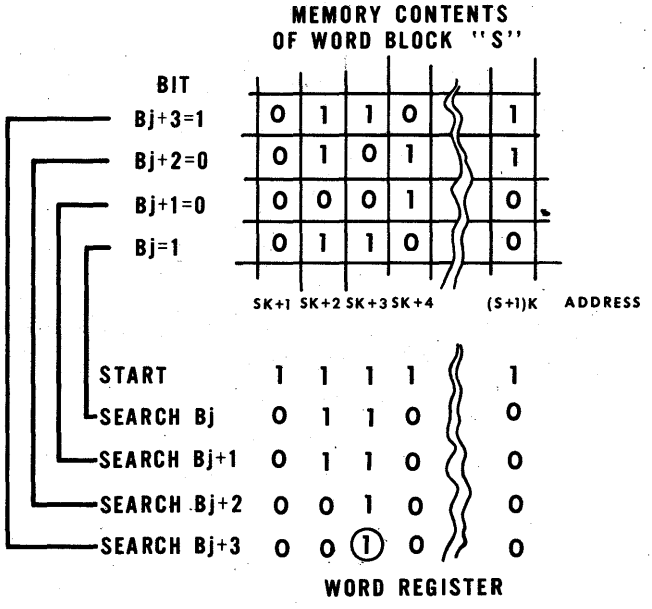


FIGURE 9—Ripple search sequence memory contents and register contents

being searched at address block S is as shown in Figure 9. The word register is initially set. Flag bit position B_J is then searched, looking for those addresses in which a "1" is stored. Since only addresses SK + 2 and SK + 3 match, the remaining word register positions 1, 4 and K are reset. The next search is made on position B_{J+1}, looking for locations containing a "0." The only mismatch occurs at the SK + 4 address or 4th bit of the register, but since this position was reset on the previous read, no change occurs. Note that 1st bit remains reset although it matched on B_{J+1}. Subsequent search reads on B_{J+2} and B_{J+3} indicate that only address SK + 3 matches on all four desired flag bits.

Multiple bit searching

A completely associative array has the flexibility of addressing a number of bit line segments, r, matching the memory contents with the input bit data, and selecting the words whose contents agree with the r data input bits. Such a memory which is fully associative over K words at a time is symbolically defined in Figure 10. Here the memory is segregated into r regions; a search word can be simultaneously written into or read out of each region in the search mode, but the memory can select a normal word, A_i, in the standard address mode.

A block diagram representation of this scheme is shown in Figure 11 where each of the search memory blocks are identical to the single bit search memory described in Figures 7 and 8; the only deviation is the reduction of the m data bits to m/r and the reduction of the core array by a factor of r. Since there are r memory blocks, the total number of cores and bit circuits do not

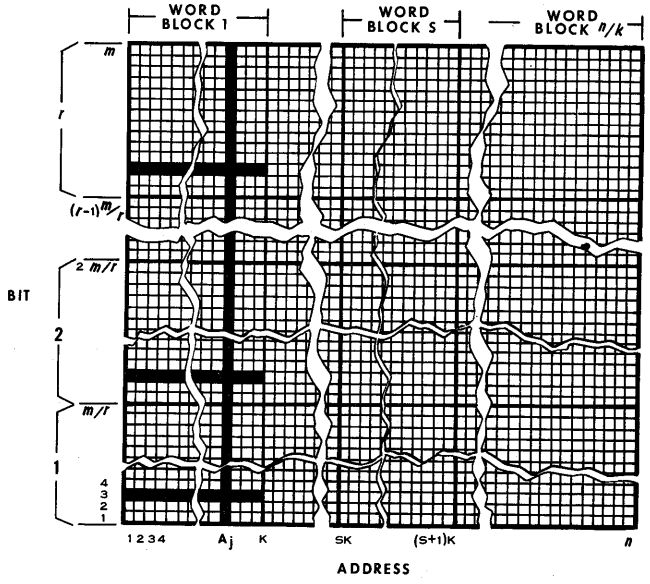


FIGURE 10—Definition of r bit search memory

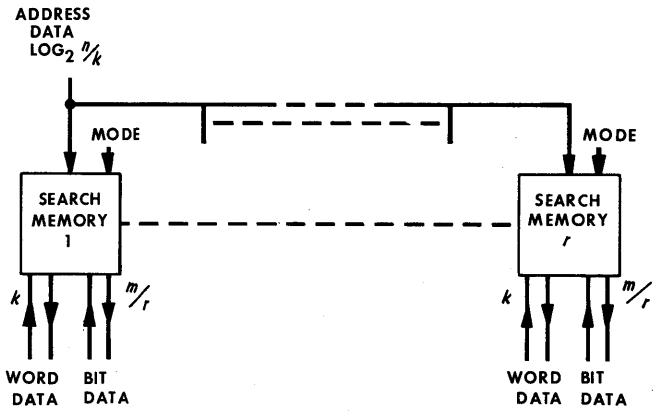


FIGURE 11—r bit search memory block diagram

increase over the single bit search, but the K word detectors and word register circuits increase by a factor of r. Thus, it is possible to simultaneously read or write rK bits in the search mode. Although this approach appears at first to be expensive, there are a number of characteristics which make it economically attractive. First, the r search memory blocks can be recombined to form a common memory array and access with a more optimum bit to word line aspect ratio. Secondly, if the basic memory size is large, it is necessary to segment the array and replicate the circuitry in any event in order to maintain high speeds and good noise margin.

The relative costs of an 800,000 bit and a 6,400,000 bit memory as a function of the number of simultaneous search bits are plotted in Figure 12. For example, the single bit search mode for a 800,000 bit memory costs 10% more than a standard no searching 2-1/2D core memory. A simultaneous search capability of up to 8 bits will double the cost of the store. However, a 6,400,000 bit store cost is hardly affected by the search mode feature.

Applications

A single bit search memory has been proposed which which can locate and read out a "flagged" word in 2 memory cycles. By means of the appropriate word logic, the memory can be used to make a ripple search in a few more memory cycles. In addition to the searching operations listed, this type of memory can be used in "pattern recognition" computers; it can be used as a serial to parallel or parallel to serial converter for massive streams of data; it can be used for matrix transposition and a number of possible multiple word operations such as simultaneous write into K words.

Extension of this memory into an r bit search allows true associative operation for word block sizes in the range of 10 to 100 words without a serious cost penalty

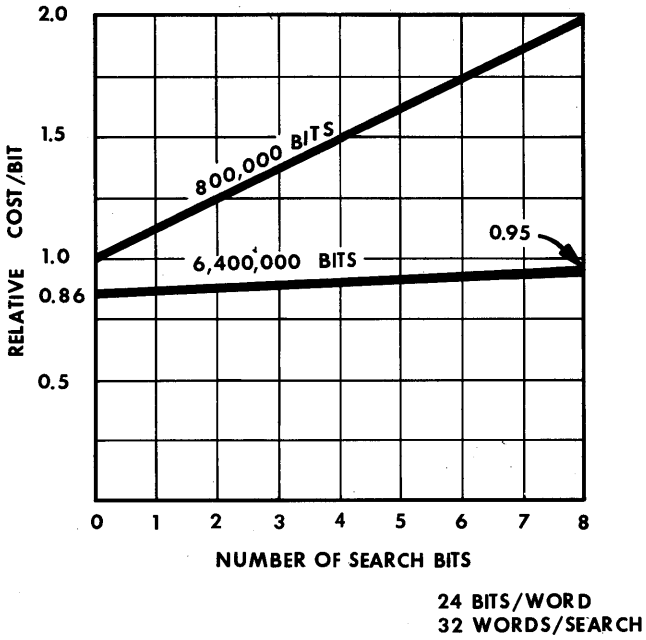


FIGURE 12—2-1/2D core address and search memory costs: relative cost/bit vs number of search bits for an 800,000 and 6,400,000 bit memory

for large sized memories.

The memory described is basically a 2-1/2D-2-wire core store; it therefore has the speed, size and cost limitations inherent to such systems. Microsecond operation for a few million bits is entirely feasible. However, it is also possible to extend the scheme to 3-wire systems for increased speed capability.

REFERENCES

1 W L McDERMID H E PETERSON
A magnetic associative memory system
The IBM Journal of Research and Development Vol 5 No 1 January 1961 pp 59–62

2 H S STONE
Associative processing for general purpose computers through the use of modified memories
1968 Fall Joint Computer Conference

3 P A HARDING M W ROLUND
Novel low cost design for 2–1/2D storage systems
1967 Solid State Circuits Conference Digest of Technical Papers Vol X IEEE Catalog No 4C–49 pp 82–83

4 P A HARDING M W ROLUND
Bit access problems in 2–1/2D 2–wire memories
1967 Fall Joint Computer Conference Proceedings Vol 31 pp 353–362

Design of a small multiturn magnetic thin film memory

by WILLIAM D. SIMPSON

Texas Instruments, Incorporated
Dallas, Texas

INTRODUCTION

Since Pohm* introduced the concept of multiturn windings as a means for improving the efficiency of planar thin film memory elements and thereby lowering the cost, very little, work has been published on memories using this technique. Planar thin film memory elements typically require word currents on the order of 500 ma and bit currents on the order of 150 ma while signals are in the range of 1 to 2 mv. Multiturn windings can be used for sense lines and/or word lines to improve the efficiency of planar film elements because the drive requirements are inversely proportional to the number of turns while the signal output is theoretically directly proportional to the number of turns. This paper describes the design of a 74000 bit planar thin film memory using a multiturn sense-digit structure but using single lines for word drive.

Design goal

The prime purpose for design of this system was to obtain a planar thin film memory concept which would be compatible with standard integrated circuits and yet retain relatively high performance characteristics. Cost, power, and reliability as always were also important considerations.

Film characteristics

The magnetic films used in this system are continuous sheets of 1100 A Ni-Fe-Co film evaporated on electro-chemically polished aluminum substrates approximately 3 in. square (Figure 1). A layer of SiO is deposited over the film for protection. Typical values for Hc and Hk are 5.0 and 4.8 respectively. The aluminum substrate serves as a ground plane and also plays an important role in obtaining virtually creep free films.

System characteristics

The memory is organized into 1024 words of 72 bits each, using 24 film planes in two 3 × 4 back-to-back arrays as shown in Figure 2. Word

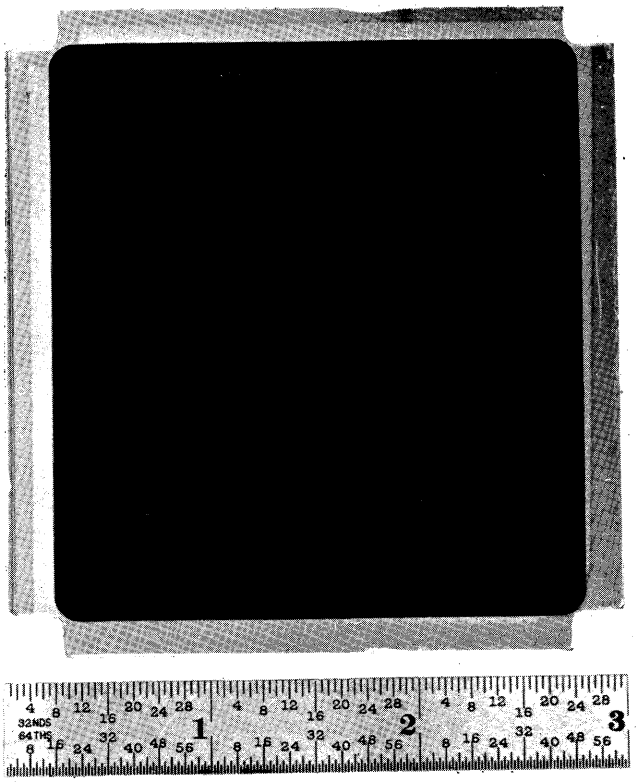


FIGURE 1—Film plane

*Pohm, A. V., "Magnetic Film Scratch-Pad Memories," IEEE Transactions on Electronic computers, Vol. EC-15, No. 4 August 1966.

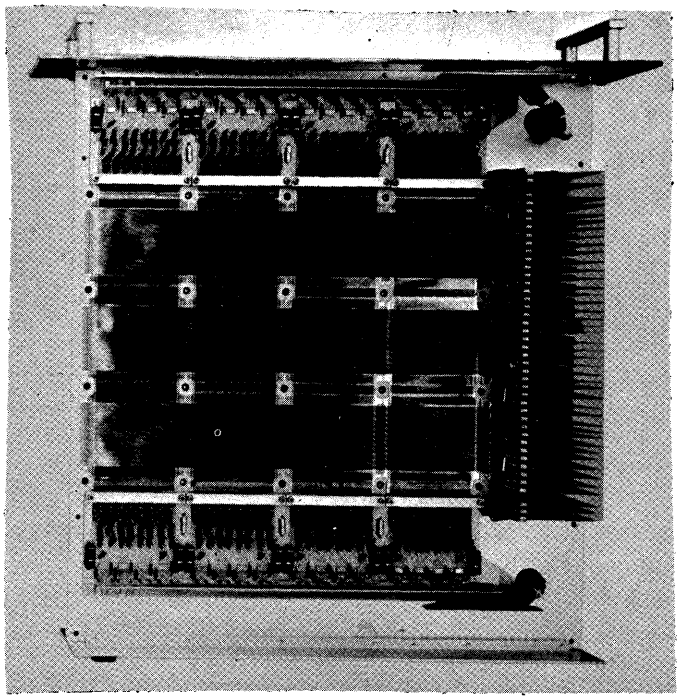


FIGURE 2—Film plane array

lines are etched from copper-Mylar** laminate and are 10 mils wide on 20 mil centers, approximately 10 inches long. Sense-digit lines are made from insulated 4 mil round wire wound into 4-turn flat coils approximately 13 inches long held in place by Mylar cladding. These flat coils shown in Figure 3 are on 80 mil centers. Since there are two

**Trademark of E. I. du Pont de Nemours & Co.

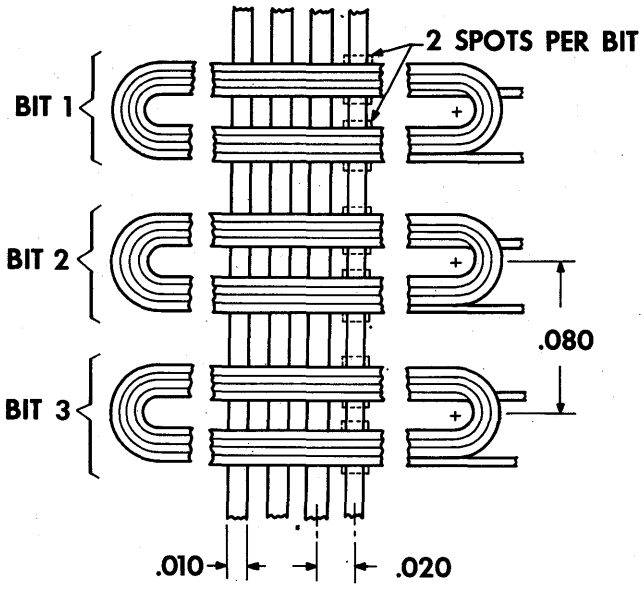


FIGURE 3—Memory element geometry

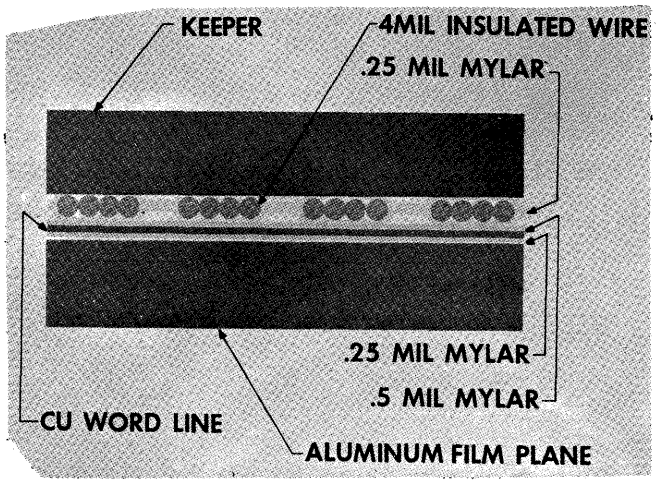


FIGURE 4—Bit geometry cross section

crossings of each coil for each word line, there are necessarily two memory elements per bit. Furthermore, since the flat coil serves both as the sense signal pickup coil and the bit drive line, the two elements receive opposing easy-direction fields and are always magnetized in opposite directions. Thus, the signals from the two elements switched by rotation in the conventional DRO mode are additive in the coil and appear in differential mode at the coil output terminals. As Pohm indicated, the sense line does not behave like a transmission line but like a lossy inductive pick-up coil. If this were not the case the signals would not be additive at the output terminals because of time delays in the 13 in. long coil.

The memory cell cross section, shown in Figure 4, consists of the aluminum plate with film on the upper surface, an overlay of word lines followed by an overlay of sense-digit lines, the word

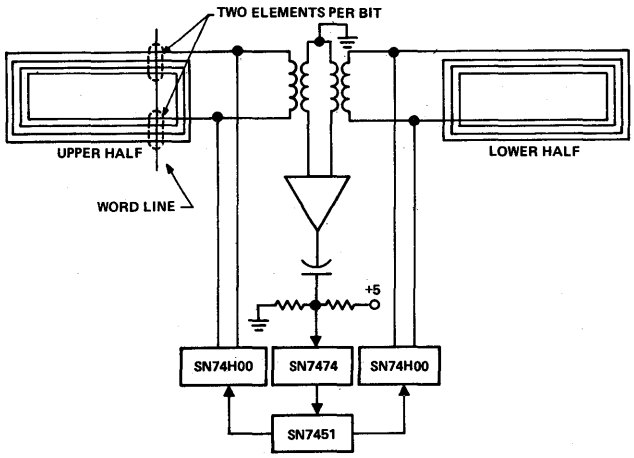


FIGURE 5—Sense-digit channel

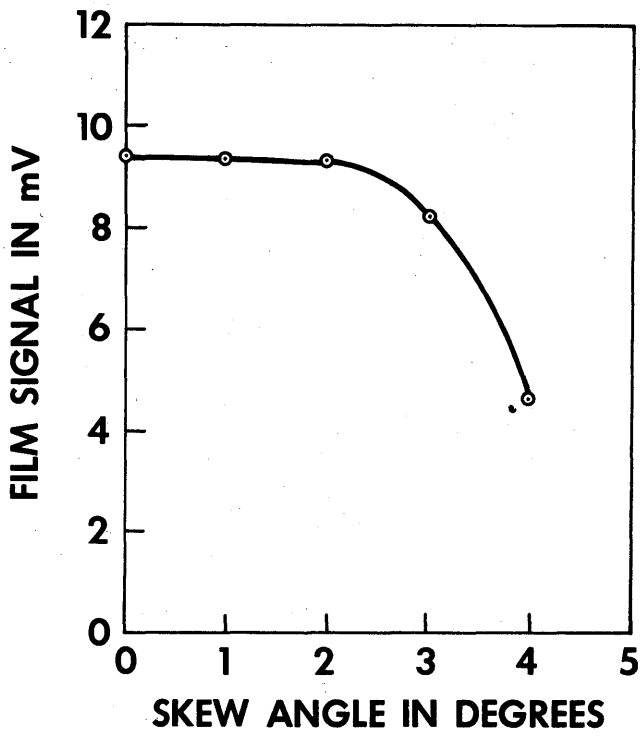


FIGURE 6—Film element skew tolerance

lines being along the easy-axis of the film and the sense-digit lines being along the hard-axis of the film. A ferrite keeper overlay pad is used as a semi-flux-closure path and as part of a pressure pad to keep the lines flat against the film. As shown in Figure 5, a single amplifier is used in each bit channel for sensing signals from the two 12 plate arrays by using differential transformers on the ends of the sense-digit windings. Capacitively coupled read-noise, generated by the word drive pulse, is automatically cancelled in the flat coil by virture of the fact that this signal is common mode at the transformer input terminals. Furthermore, in the write portion of the cycle, noise generated by the bit drivers is similarly cancelled by virtue of the fact that similar signals from the two halves of the memory are present in common mode fashion at the input terminals of the amplifier. Two important results of the two element per bit storage cell are that continuous films can be used since there is no requirement for dummy sense lines for noise cancellation (and therefore no need to etch away film), and skew tolerance has been increased as shown in Figure 6. Skew tolerance is increased because the skew angle for both elements of a bit will normally be the same. Therefore, since skew manifests itself in assymetrical signals in each half of the sense line, the two elements have compensating effects. Figure 7 is a plot of signal output versus bit drive current for the two element memory cell as the result of a creep disturb test in which 10^6 disturbs are applied from each side of a given memory cell. The bit current threshold for writing is 35 ma and the threshold for disturb is 95 ma. Signals at the sense line output terminals are from 9 to 12 mv when word current of 500 ma is applied with rise time less than 15 ns.

Memory electronics

The sense-digit channel electronics shown in Figure 8 are fully integrated. An SN 7510 wideband amplifier is used to amplify the sense signal to a level compatible with the D-type flip-flop of the series 74 TTL family. Note that the output of the amplifier is ac coupled to a bias network which sets the quiescent level at the midpoint between the two logic levels such that the bipolar film signals can easily be detected by the flip-flop. The digit driver is formed by two gates connected at either end of the flat sense-digit coil; the bipolar digit currents are obtained by driv-

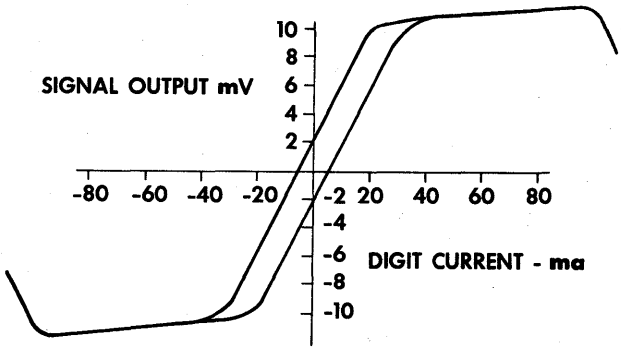


FIGURE 7—Disturb tolerance characteristic

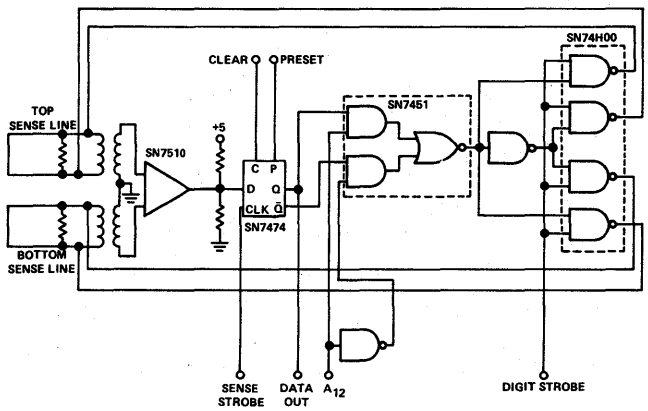


FIGURE 8—Sense-Digit electronics

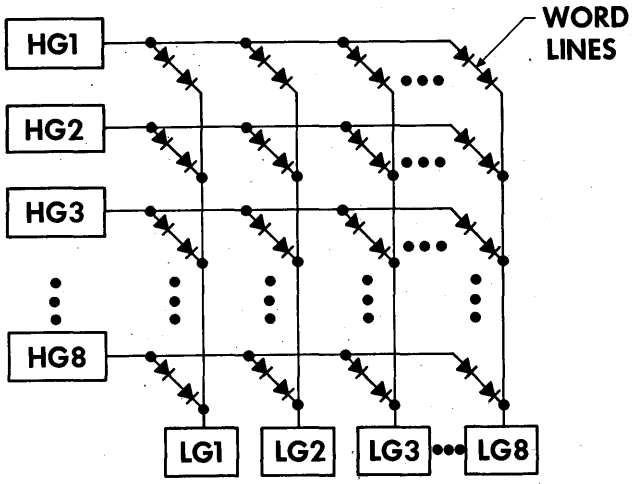


FIGURE 9—Diode selection matrix

ing one gate high and the other gate low simultaneously. The word drive is accomplished through the use of integrated high-gates and low-gates, and sixteen diode matrix selection arrays as shown in Figure 9. Two interleaved arrays drive 128 word lines in each octant of the memory.

Special integrated circuits developed at Texas Instruments accept TTL inputs and have high current transistors for the output to provide 500 ma word currents with rise times faster than 15 ns. Two high-gates and two low-gates are included in one flat pack. Thus, 64 flat packs are

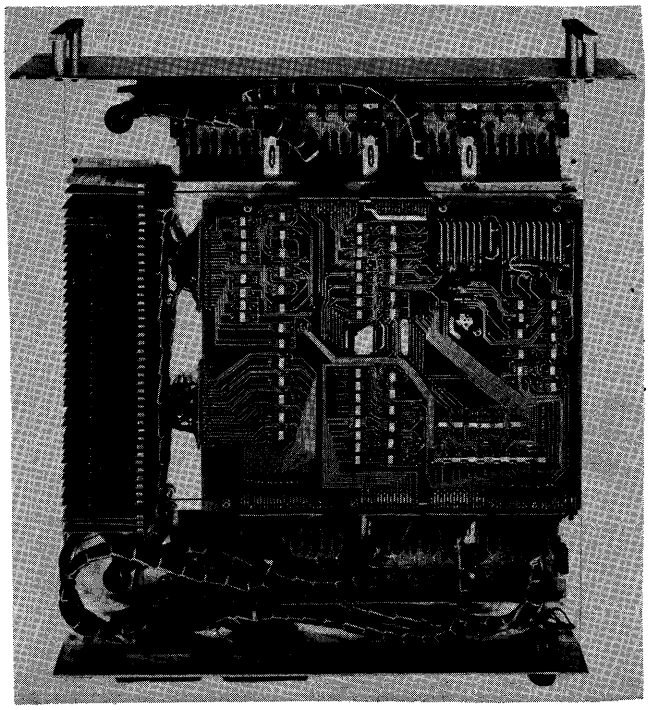


FIGURE 10—Memory photograph

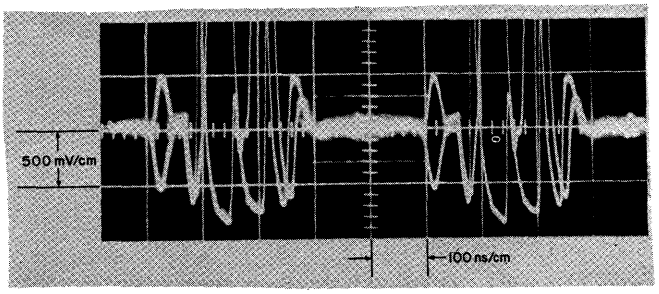


FIGURE 11—Sense amplifier output waveform

required for selection of 1024 words. Diodes are included on both ends of the word lines to prevent "half select" currents in unselected lines during word current transients.

All of the logic circuitry, is accomplished using standard series 74 networks. System timing is achieved through the use of tapped delay lines so that no one-shots or other adjustable devices are required. In fact, there are no controls or other variable elements in the entire memory.

Physical specifications

The memory package shown in Figure 10 is designed for 19 in. rack mounting with a panel height of 3½ in. and a total depth of 20 in. The weight is approximately 35 lbs. The power dissipation of the memory is less than 60 watts. This relatively low power level is primarily due to the saturated stage nature of TTL networks used throughout.

Memory performance characteristics

The sense amplifier analog output is shown in Figure 11. Signals for both 1s' and 0s' are shown superimposed for one particular bit channel with all 1024 words of the memory operating. Note that the recovery pattern indicates a minimum cycle time of 350 ns can be achieved using this technique, although the memory was originally designed for application as a 500 ns cycle time system. Similarly, the design goal for access time of 250 ns was also achieved.

CONCLUSION

A planar magnetic thin film memory has been designed and built by Texas Instruments using all integrated circuits for electronics achieving a cycle time faster than 500 ns, and an access time of 250 ns. The memory is organized as 1024 words by 72 bits in order to balance the costs of the word drive circuits against the sense-digit cir-

cuits. The inherent advantage of this particular organization is that the computer can achieve speed advantage not only because of a fast repetition rate, but also because four 18 bit words are accessed simultaneously. (Comparable core memory designs are ordinarily organized 4096 words of 18 bits each.) The outlook is for higher speed (faster than 150 ns) memories in similar organizations to be developed in planar magnetic films. The cost of these memories will be competitive with 2½ D core memories of the same capacity but the organization and speed can be considered to offer at least a 4:1 improvement in multiple word accessing and a 3:1 improvement in speed. As a result of this, more computers will be designed to take advantage of the long word either by extending the word length of the computer itself or by ordering instructions and data in such a manner that sequential addressing will be required a large percentage of the time.

An adaptive sampling system for hybrid computation

by GEORGE A. RAHE

Naval Postgraduate School
Monterey, California

and

WALTER KARPLUS

University of California
Los Angeles, California

INTRODUCTION

The concept of sampling is central to the operation of all systems in which analog information is to be processed by a digital computer. In conventional hybrid computing and data-processing systems the continuous analog signal is represented by an amplitude-modulated pulse train in which the pulses occur at fixed intervals of time. Such synchronous sampling facilitates control by the digital computer clock and requires a minimum amount of equipment. In many applications, however, it is important to minimize the number of samples employed to represent the analog signal. For example in telemeter applications, it is important to economize transmitter power by limiting the number of samples transmitted over long communication links.

In hybrid computation, power conservation is not a primary objective, but high sampling rates often tax severely available digital computer memory capacity and the band-width of data channels. Not infrequently, high sampling rates limit the number of analog channels which can be accommodated by a given analog-digital interface. Accordingly, a variety of so-called data-compression techniques have been proposed, techniques which are designed to reduce the number of samples which must be transmitted across an analog-digital interface without exceeding specified error bounds.

The system described in this paper represents a novel approach to this problem. It differs from conventional data-compression techniques in that the analog signal is modified or subjected to an approximation prior to sampling. The theory underlying this method is first briefly developed below, followed by description of a hybrid computer mechanization of the data compression system.

Redundancy in synchronous sampling

In most data processing and hybrid computing systems, the sampling rate is dictated by specified error bounds upon the reconstructed signal. If the continuing analog signal is sampled and processed by a digital computer, it must be possible to reconstruct a continuous signal from the samples so that the maximum difference between the reconstructed and the original signal nowhere exceeds a specified tolerance. In accordance with the well-known sampling theorem, this sampling rate is based on the largest magnitude and the highest frequency components the signal is expected to attain. Actually, the analog signal may never attain these maximum magnitudes or frequencies, or it may attain them for only brief periods of time. Therefore, the utilization of fixed sampling usually leads to a large number of unnecessary samples, samples which can be eliminated without deteriorating the quality of the reconstructed signal. In essence, the synchronous train of samples contains redundant information, and the various data-compression schemes are intended to minimize this redundancy. A number of proposed data-compression methods involve the suppression of redundant samples. In that case the analog signal is sampled, and the sample is analyzed to determine whether samples can safely be omitted from the signal transmitted to the digital computer. An alternate approach, proposed in this paper, involves the utilization of an adaptive sampling system so that the analog signal is sampled only as often as neceessary, but all samples actually taken are transmitted to the digital computer. The sampling interval is therefore continuously and automatically controlled as a function of the analog signal activity.

Accuracy constraints

The accuracy demanded of a sampled data system of the type considered in this paper is dictated first of all by the characteristics of available hardware. Thus, it is unreasonable to attempt to reduce the reconstruction error below the combined magnitudes of the various error sources inherent in the hybrid system. These error sources include particularly the drift, zero-offset, phase-shift, and noise in the analog portion, and the quantization error (related to the word-length) on the digital computer.

The purpose to which the data is to be put also dictates the form of reconstruction and therefore the control laws for the sampling operation. Consider for example the operation of graphic CRT terminals in hybrid computation where analog signals are to be displayed on one or more such terminals. Commercially available terminals represent a function by a series of straight line segments to an accuracy of from 1%to 5%. Line segment generators require only the origin and terminus of a line segment to produce the required line. For the purposes of this type of application, a sampling control law is required which will provide a reconstruction by linear interpolation to a predefined maximum error, with the minimum number of line segments, and in real time. In addition the approximation must be continous at the end points of the line segments in order not to be objectionable to the user.

Definition of the approximation — continuous secant

The nature of the reconstruction to be considered here and some of the properties of the approximation will now be described. Consider that the function $\phi(x)$, continuous on a closed interval (a,b), is to be approximated by line segments $P_i(x)$ over subintervals

$\Delta_i = (x_{i+1} - x_i)$ i = 1, 2, $\cdots$ n such that[1]

$$\text{Max}|\phi(x) - \sum_{i=1}^{n} P_i(x) \leq E \quad a \leq x \leq b \quad (1)$$

where

$$P_i(x) = a_i + b_i x \qquad x \epsilon \Delta_i$$

$$P_i(x) = 0 \qquad x \epsilon \Delta_i$$

and

$$\sum_{i=1}^{n} \Delta_i = (a, x_1) + (x_1, x_2) + \cdots (x_{n-1}, b) = (a, b)$$

$$a_i = \phi(x_i)$$

$$b_i = \frac{\phi(x_{i+1}) - \phi(x_i)}{\Delta_i}$$

The set of line segments $P_i(x)$ which form the best approximation to $\phi(x)$ are defined to be those which minimize the number of segments n for a given predefined tolerance E. The definition that $\phi(x)$ is continuous in the mathematical sense will of course present no restriction on mechanizable functions.

It can be readily shown[3] that a minimum of nonunique number of line segments on a closed interval $a \leq x \leq b$ is made up of the set determined by the maximum line segment with orgin at a point (a) and maximizing the length of each succeeding line segment adjoining them at their end points until a segment is determined which contains the point (b).

The problem of finding the optimum sampling points is reduced to determining the largest value of Δ_i in any interval which satisfies the predefined error.

Determination of approximation interval

Before proceeding with the derivation of the control laws for determining the maximum approximation interval, it will be advantageous to consider the nature of the function $\phi(x)$ to be approximated. The continuous function $\phi(x)$ to be approximated on a certain finite interval has, in most cases, a fixed direction of concavity (upward or downward) which changes only a finite number of times. Such functions will be referred to here as "piecewise convex or concave".

Without loss in generality, the study of concave function $\phi(x)$ is reduced to that for convex function $-\phi(x)$ and defined as follows: Definition: A function $\phi(x)$ is convex on the interval (a,b) provided that[2]

$$\phi(\mu\beta + (1 - \mu)\alpha) \leq \mu\phi(\beta) + (1 - \mu)\phi(\alpha)$$
$$(\alpha, \beta) \subset (a, b) \qquad (2)$$
$$\mu \epsilon (0, 1)$$

The plot of a convex function $\phi(x)$ in cartesian coordinates, therefore, is characterized by the property that any arc of the plot has all of its points located not higher than the secant chord that joins its end points.

For the purposes of exposition only, consider $\phi(x)$ twice differentiable on the interior interval $x_i < x < x_{i+1}$ then

$$\frac{d^2}{dx^2}\phi(x) \geq 0 \qquad (3)$$

A value of $x = x_{i+1}$ is sought such that the maximum deviation of $\phi(x)$ from the secant $P(x)$ is equal to E in the interval $x_i \leq x \leq x_{i+1}$, where

$$P(x) = \phi(x_i) + \frac{\phi(x_{i+1}) - \phi(x_i)}{x_{i+1} - x_i}(x - x_i) \quad (4)$$

then there exists a x_{i+1} such that:

$$P(x) - \phi(x) = E \qquad x_i < x < x_{i+1} \quad (5)$$

Substituting equation 4 into equation 5

$$\phi(x_i) + \frac{\phi(x_{i+1}) - \phi(x_i)}{x_{i+1} - x_i}(x - x_i) - \phi(x) = E$$

or

$$\frac{\phi(x) - \phi(x_i) + E}{x - x_i} = \frac{\phi(x_{i+1}) - \phi(x_i)}{x_{i+1} - x_i} \quad (6)$$

defining

$$M_3(x) = \frac{\phi(x) - [\phi(x_i) - E]}{x - x_i} \quad (7)$$

and

$$M_2(x) = \frac{\phi(x) - \phi(x_i)}{x - x_i}$$

It is seen from Figure 1 that the slope of the tangent line UV is equal to the minimum value of $M_3(x)$ so that the value of $x = x_{i+1}$ is sought such that:

$$\frac{d}{dx} M_3(x) = 0 = \frac{d}{dx}\left\{\frac{\phi(x) - \{\phi(x_i) - E\}}{x - x_i}\right\} \quad (8)$$

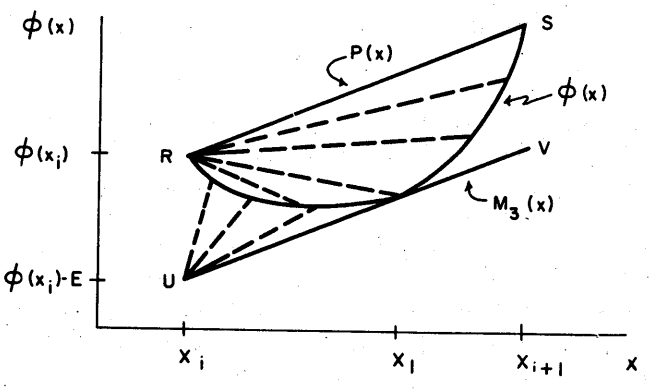


FIGURE 1—The continuous secant method.

taking the derivative

$$-[\phi(x) - \{\phi(x_i) - E\}](x - x_i)^{-2} + \frac{d\phi(x)}{dx}(x - x_i)^{-1} = 0 \quad (9)$$

From which

$$\frac{d\phi(x)}{dx} = \frac{\phi(x) - [\phi(x_i) - E]}{x - x_i} = M_2(x) \quad (10)$$

For

$$x = x_{i+1}$$

So that the value of $x = x_{i+1}$ is that value of x for which

$$M_2(x) = \text{Min}M_3(x) \qquad x \epsilon \Delta_i \quad (11)$$

as is shown in Figure 2a.

In an entirely parallel development for the case where $\phi(x)$ is concave:

$$\text{Defining } M_1(x) = \frac{\phi(x) - [\phi(x_i) + E]}{x - x_i} \quad (12)$$

then x_{i+1} is that value of x for which

$$M_2(x) < \text{Max}M_1(x) \qquad x \epsilon \Delta_i \quad (13)$$

as is shown in Figure 2b.

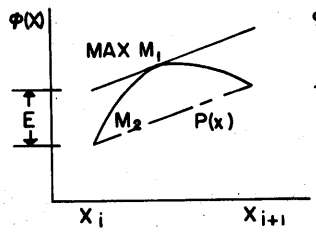


Figure 2b

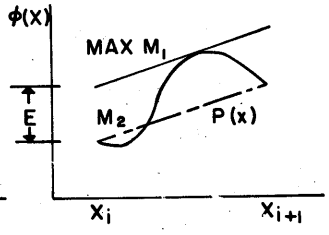


Figure 2d

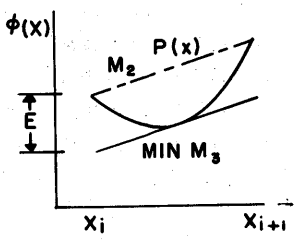


Figure 2a

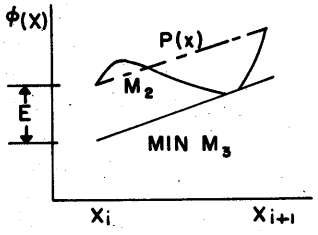


Figure 2c

FIGURE 2—Behavior of the secant method for (a) convex interval; (b) concave interval; (c) concave-convex interval; (d) convex-concave interval.

Consider the case now where $\phi(x)$ is not strictly concave or convex in an interval. If the curve is concave convex as in Figure 2c, the maximum excursion of $\phi(x)$ from $\mathrm{MaxM_1}(x)$ must have been less than E in the concave portion. Since $M_2(x)$ at $x = x_{i+1}$ must be greater than $\mathrm{MimM_2}(x)$ in the interval, then the error in approximation over the concave portion must be less than E. Which is to say that the interval is effectively convex to within a predefined error E. A similar argument follows directly for the case where $\phi(x)$ is convex-concave, as shown in Figure 2d.

Summary of control laws for the determination of samling points

The general control laws for determination of the optimum sample points may now be summarized as follows:

Defining

$$M_1(x) = \frac{\phi(x) - [\phi(x_i) + E]}{x - x_i}$$

$$M_2(x) = \frac{\phi(x) - \phi(x_i)}{x - x_i} \qquad (14)$$

$$M_3(x) = \frac{\phi(x) - [\phi(x_i) - E]}{x - x_i}$$

and $x > x_i$.

The sample point x_{i+1} is given by the minimum value of x for which either of the following logical equations is satisfied:

I. $$M_2(x) \leq \mathrm{MaxM_1}(x) \qquad (15)$$

II. $$M_2(x) \geq \mathrm{MinM_3}(x)\ . \qquad (16)$$

Mechanization of the hybrid interpolator

A primary concern in the development of any sampling control law is ease of mechanization. A functional diagram of the mechanization of the proposed hybrid interpolator is given in Figure 3. While an all analog mechanization encounters certain difficulties, modern hybrid multipliers are suited to divide by a parameter as restricted as time. Equipment will allow updating of time at a 500 kHz rate. The suitability of this method was verified by simulation on a hybrid computer system.

By contrast, the operation of a pure digital interpolator is illustrated in Figure 4. The interpolator

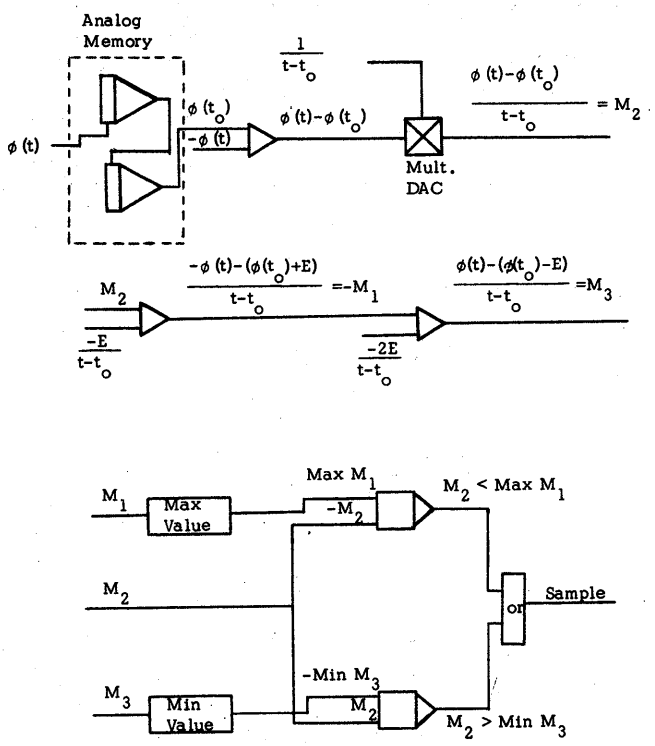


FIGURE 3—Mechanization of the control laws for the secant method.

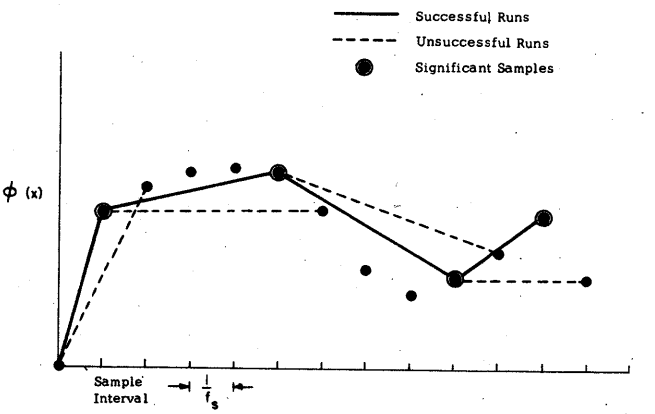


FIGURE 4—A digital linear interpolation.

forms a straight line with f(0) as the origin and f(2) as the terminus, and computes a value for the intervening sample f(1). If that value is within tolerance a new line is formed with f(3) as terminus and both intervening values must be computed and compared with the actual values. If either value is not within tolerance f(2) is transmitted and becomes the origin of the next line. This procedure has two significant failings: first, the operation requires at least one subsequent value of the function in order to make an approximation, and second each intervening point

must be reapproximated at every new interval. It follows then that for a line lenght of n intervals, s = n/2 (1 + n) intermediate points must be calculated and compared with the actual sample values. Even for relatively small values of n the computation time prohibits its use in most applications.

A determination of efficacy

Choice of test signal

In order to compare various sampling systems it is necessary to define an appropriate test signal. Two considerations prompt the selection of a signal which is a worst case for adaptive sampling: a) The resulting compression provides a lower bound on system performance; and b) it provides a measure of system susceptibility to the generation of more samples than a suitably formulated synchronous system.

Since the adaptive system samples on the basis of both amplitude and frequency, a worst case would be one which is characterized by a flat power spectral density over the total predicted bandwidth (a maximum information signal).

Such a signal which is easily generated and easily described in both statistical and deterministric terms can be constructed from a sum of sine waves.[5]

$$\phi(t) = \sum_{n=1}^{N} a_n \sin(\omega_n t + \theta_n) \qquad (17)$$

For eight or more non-harmonic related sine waves, the probability density function becomes indistinguishable from the Gaussian one.

Comparison of sampling rates

The sampling rate for a fixed maximum error for a number of sampling methods was determined on an IBM 7094 computer. The results of that study are presented in Table I. This summary indicates that even under worst case conditions the proposed system demonstrates a marked reduction in the number of samples required for reconstruction to a fixed predefined error. Certain familiar reconstructions were included for the same relative RMS error for comparison purposes.[3,6]

SAMPLING RECONSTRUCTION METHOD	SAMPLES/ CYCLES OF HIGHEST FREQ.	RMS ERROR (%)	PREDEFINED TOLERANCE % FULL SCALE
Butterworth 4-stage filter	8.0	2.0	——
Linear phase 4-stage filter	15.0	2.0	——
Zero-order hold	628	——	±.5
First-order hold	62.8	——	±.5
First-order (Synch). interpolator	22.2	——	±.5
Continuous Secant	6.7	1.76	±.5

TABLE I—Relative sampling rate (fr) and RMS error vs. predefined tolerance for test signal input[3,6]

Source noise

The operation of a sampling system in the presence of source noise is often neglected since noise over the entire predicted bandwidth works to reduce sample reduction and also precludes the successful operation of many proposed systems which rely on measurement of the derivatives of the function to be sampled.

In the absence of noise, adaptive sampling can be expected to reduce the sampling rate by additional factor of ten when the signal occupies only one tenth of the predicted bandwidth. However, noise can be expected to occupy the entire band and the effect of the presence of this noise is paramount to the evaluation of adaptive system.

In order to evaluate this effect, noise with a flat spectral density, a Gaussian distribution and RMS level equal to half the predefined tolerance and cut off at 18 db per octave at a frequency ten times the highest signal frequency was added to the signal. The effective sampling rate even at this excessive noise level was increased by only twenty percent over the rates determined for the noise free case.

CONCLUSIONS

An algorithm has been presented for the determination of a continuous polygonal approximation which results in the least number of samples for a given predefined maximum error where the end points of the line segments are restricted to lie on the function. The method is suitable for general application since it requires no apriori knowledge of the properties of the function to be sampled. The method has been shown to provide a reduction in the sampling rate even under worst case conditions and to operate effectivelv with little degradation in performance when the signal is corrupted by noise.

ACKNOWLEDGMENT

The studies described in this paper were sponsored in part by the National Science Foundation under a grant to the Department of Engineering, University of California at Los Angeles, and by the U. S. Naval

Ships Systems Command under a contract with the Department of Electrical Engineering, Naval Postgraduate School, Monterey, California. The authors also gratefully acknowledge the courtesy extended to them by the Electronic Associates Inc. Computing Center, El Segundo, California, and by Mr. J. Magnall formerly the director of that center.

BIBLIOGRAPHY

1 DE LA VALLEE-POUSSIN
Lecons sur l'approximation des fonctions d'une variable realle
Paris 1919

2 J W YOUNG
General theory of approximation by functions involving a given number of arbitrary parameters
Trans-American Math Soc 23:331–334 July 1907

3 G A RAHE
Adaptive sampling
PhD Dissertation UCLA 1965

4 L W GARDENHIRE
Redundancy reduction the key to adaptive telemetry
Nat Telemetering Conf Los Angeles California June 1964

5 R K SISKIND
Probability distributions of sums of independent sinusoids
Technical Memo No 83 Systems Technology Laboratories Inc Ingelwood California March 1961

6 D D McRAE
Interpolation errors
Radiation Inc Reports 1 pt 1 May 1961

APPENDIX I.

Synchronous sampling in hybrid computation

The reconstruction of signals in hybrid computation have been restricted in general to the simple zero and first-order holds shown in Figure A.1. Since the error in computer systems is generally required to remain within a predefined tolerance E, the synchronous sampling rate is dictated by the smallest interval in which this tolerance can be reached.

From Figure A.1, the reconstruction by a zero-order hold is seen to be given by the value of the functions at the last sampling instant nT. The output of the zero-order hold $\phi_{ZOH}(t)$ is given by:

$$\phi_{ZOH}(t) = \phi(nT) \qquad nT < t < (n+1)T \qquad (A.1)$$

for which the construction error $e_{ZOH}(t)$ is given by

$$e_{ZOH}(t) = \phi(t) - \phi_{ZOH}(t). \qquad (A.2)$$

For an input signal $\phi(t)$

$$\phi(t) = A \sin \omega t \qquad (A.3)$$

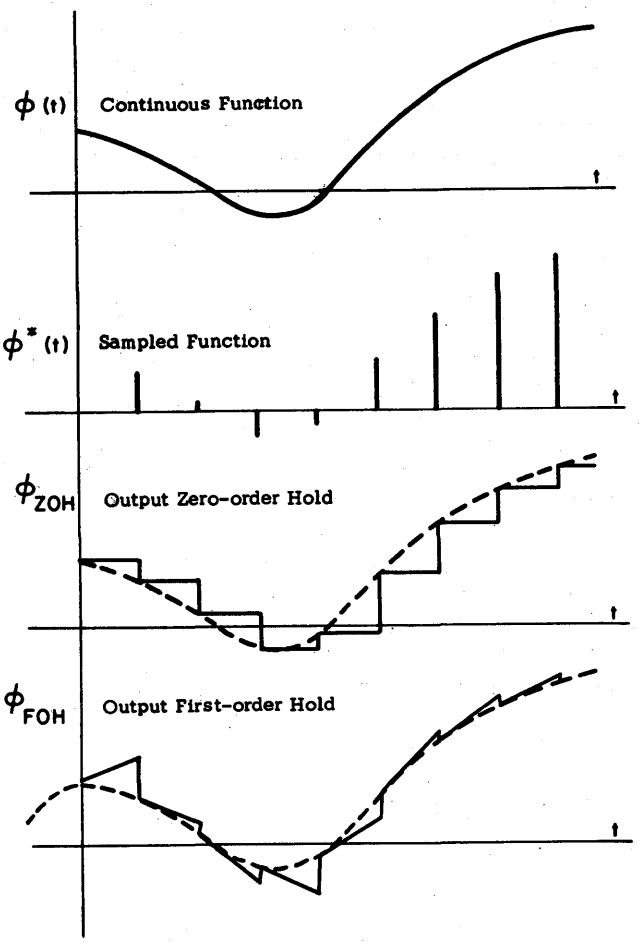


FIGURE A. 1—Zero and first-order holds.

the maximum full scale relative error becomes

$$e_{max} = \text{Max} \left| \frac{e_{ZOH}(t)}{2A} \right| = \frac{A\omega T}{2A} \qquad (A.4)$$

where n = the number of samples/cycle
T = sampling period

Since

$$n\omega T = 2\pi \qquad (A.5)$$

then

$$e_{max} = \pi/n \qquad (A.6)$$

and for a predefined tolerance E, the sampling rate n is given by

$$n > \frac{\pi}{E/2A} \qquad (A.7)$$

Choosing a value of relative error which is used to evaluate the Secant Method, e max < .5 percent, the sampling rate (n) for the zero order hold is found to be:

$$n > 638 \text{ samples/cycle.} \tag{A.8}$$

The number of samples required when reconstruction is performed by a first-order hold is derived in much the same fashion. The output of the first-order hold can be written as follows:

$$\phi_{FOH}(t) = 2\phi(t - T) - \phi(t - 2T) \tag{A.9}$$

and the error for a first order hold e_{FOH} becomes

$$e_{FOH} = \phi(t) - 2\phi(t - T) - \phi(t - 2T) \tag{A.10}$$

In the case of the sine wave input

$$\phi(t) = A \sin \omega t$$

the maximum full scale relative error is given

$$\text{Max} \left| \frac{e_{FOH}(t)}{2A} \right| \leq \frac{\omega^2 T^2}{2} = \frac{2\pi^2}{n^2} \tag{A.11}$$

$$N > \frac{\sqrt{2}\,\pi}{\sqrt{E/2A}}$$

and the sampling rate for e max < .5 percent is therefore

$$n \# 62.8 \text{ samples/cycle} \tag{A.12}$$

Similarly the sampling rate for a linear interpolator can be shown to be:

$$\text{Max} \left| \frac{eLI(t)}{2A} \right| \leq \frac{\omega^2 T^2}{16} = \frac{\pi^2}{4n^2} \tag{A.13}$$

$$n > \frac{\pi}{2} \frac{1}{\sqrt{E/2A}}$$

Again for a maximum relative error of .5 percent

$$n > 22.2 \text{ samples/cycle.} \tag{A.14}$$

A new solid state electronic iterative differential analyzer making maximum use of integrated circuits

by BRIAN K. CONANT

University of Arizona
Tucson, Arizona

INTRODUCTION

The feasibility of really fast hybrid computation was demonstrated by the development and application of The University of Arizona's ASTRAC II.[4,8,10] But, no machine commercially available to date has the required mode-control switching speed and low—impedance computing networks; and most computers do not have the required amplifier bandwidth. The development of the LOCUST system represents an attempt to design a truly producible very fast computer at moderate cost.

The new machine was developed as a project-group Ph.D. dissertation, a somewhat novel concept in engineering education: The writer acted as a project engineer on the overall design, test, and application of the LOCUST computer and helped to supervise four M.S. thesis projects[14,17,19,21], plus several term-paper projects[1,2,5,20], which contributed significant components. The computer, including all printed circuit cards, was built by undergraduate student technicians, using Motorola and Fairchild integrated circuit modules and discrete components.

The LOCUST system is an all solid-state iterative differential analyzer making maximum use of integrated circuits (Figure 1). The machine comprises 34 free amplifiers of which 16 can be used as integrator/track hold circuits, plus 18 amplifiers permanently committed to 6 high-speed multiplier/dividers, and 4 comparators (56 amplifiers total). The new machine is capable of solving a sixteenth-order linear or non-linear differential-equation system up to 2000 times per second for iterative and random-process computations under digital control.[4] Linear errors are within 0.2 percent up to 10 kHz. Special "slow" summing networds also permit operation as a slow analog computer. The following design features are of special interest:

1. Maximum use of both linear and digital monolithic integrated circuits enhances computer performance and still reduces parts and assembly costs.

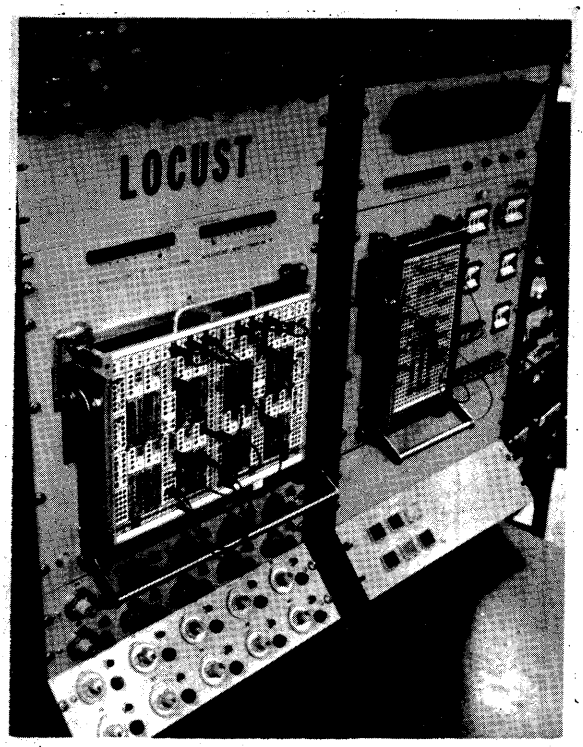


Figure 1—The LOCUST computer, built in the University of Arizona's Analog/hybrid Computer Laboratory. Linear and digital integrated circuits enhance computer performance and still reduce parts and assembly costs

2. New mounting and shielding techniques, including a technique for shielding low cost unshielded patchbays, were developed (Figure 2).
3. Low level current-mode digital logic modules (Motorola, MECL, integrated circuits) eliminated digital noise in the analog portion of the computer. The low-level logic swing (0.8 V) along with the balanced-current nature of the non-saturating current mode logic serve to reduce radiation and, more importantly, computer ground-system disturbances.

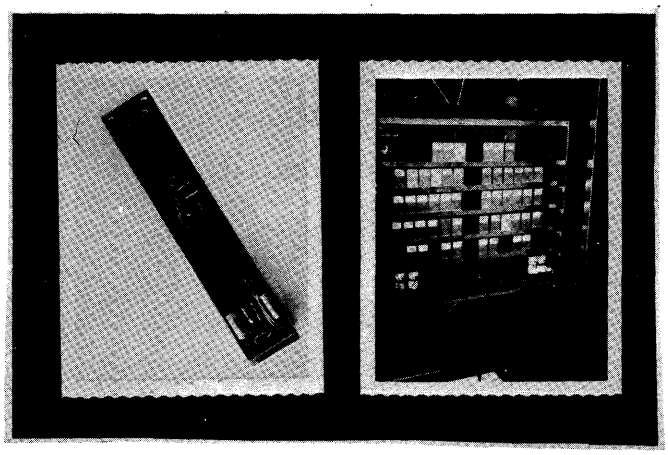


Figure 2—Analog-circuit boxes and digital-circuit cards plug directly into the rear of inexpensive low-leakage plastic patchbay receivers. Rows and columns of patchbay springs used for grounds, power connections, and logic also serve as analog-patchbay shields. Simple metal patchboards with shielded patch-cords have patch holes only for the actual analog-computer terminations for an uncluttered appearance. Summer-integrator patching is logic controlled and does not require cumbersome bottle plugs

4. New solid-state devices have substantially improved amplifier and mode-control switch performance.
5. Free digital logic has been made more convenient as a result of ASTRAC II experience; and the LOCUST/digital computer linkage is designed as an integral part of the logic-control system.
6. New patch panel design affords exceptional convenience for setup and logic control.

LOCUST analog computing elements

High-speed computing requirements

To appreciate the technical problems associated with fast computatiom, consider the following requirements.[9] To keep dynamic errors of ordinary phase-inverting amplifiers, integrators, and mode-control switches within 0.1 percent for 10 kHz signal frequencies.

1. The amplifier gain-bandwidth product must exceed 20 MHz.
2. Computing resistances should be less than 10kΩ; this requires high-current amplifier output stages.
3. The voltage rate of change of a 20 volt peak-to-peak 10 kHz sine wave is $2\pi \times 10^5$ volts per second. To insure a track-hold sampling accuracy of 0.1 percent, mode-control and track-hold timing must be within 15 nanoseconds.

The LOCUST operational amplifiers

The LOCUST iterative differential analyzer satisfies the gain/bandwidth requirements of fast analog computation with the aid of a new high-performance amplifier developed at The University of Arizona as an M.S. thesis project; [14] the reference describes its design in great detail. The performance of the new amplifiers is summarized in Table 1a.

The block diagram of Figure 3a shows the basic amplifier design, which employs a three-channel feed-forward circuit. Beginning from the amplifier input, the channels are: the high-frequency channel directly to the wide-band class AB output stage, the intermediate-frequency channel through a Motorola MC1433 integrated-circuit operational amplifier, and the low-frequency channel through a Fairchild μA726C hot-substrate preamplifier.

The bandwidth and output-current limitations of the MC1433 integrated-circuit amplifier are overcome by cascading it with a high-current output stage and by feeding forward the high frequency signals directly to the output stage from the summing junction. Because

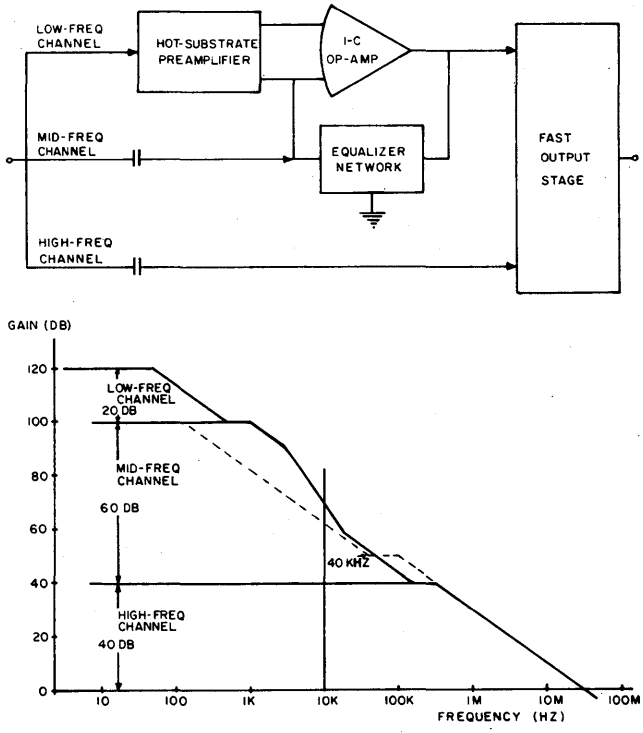


Figure 3 —A LOCUST three-channel feedforward amplifier (a)and its frequency response (b). *The wideband class AB output stage* has low open-loop output impedance, full ± 10-V, 30-mA output to 5 MHz, and provides two convenient input points for the summation of the signals from the high-frequency channel and the intermediate-frequency amplifier. *The intermediate-frequency integrated-circuit amplifier* accurately controls the gain and frequency response of the intermediate-frequency signals by feedback through the three-terminal equalizer network. *The hot substrate differential preamplifier* provides low-offset/low-drift amplification of low frequencies down to d-c

of the low-offset low-drift properties of a new integrated circuit hot-substrate differential amplifier, a chopper stabilizer channel is unnecessary.

Figure 3b illustrates the frequency response of each channel and their combination. Dotted lines show the roll-off of a more conventional 6dB/octave feed-forward amplifier; the "zero" needed at 40 kHz lowers the gain at 10 kHz. An improvement in the open-loop gain of only 6 dB at 10 kHz will decrease the dynamic error of a phase inverter operating at 10 kHz by a factor of two.

Figure 3c is a complete schematic diagram of the new amplifier.

A limiter circuit (Figure 4a) is included in the LOCUST amplifier to insure quick recovery from overload, prevent the summing junction from leaving virtual ground when an overload occurs, and thus completely eliminate the need for amplifier POTSET relays. The combination of resistor R and the parallel diodes reduces the effect of the capacitance of the zener diodes.

The amplifier has a novel silicon-controlled rectifier overload-indicator circuit (Figure 4b). Positive or negative voltage levels which exceed the threshold level of the SCR will trigger the SCR and thus light the overload lamp. The lamp stays lit until the SCR is reset by front-panel pushbutton mode control logic.

The entire amplifier is intended for easy assembly on

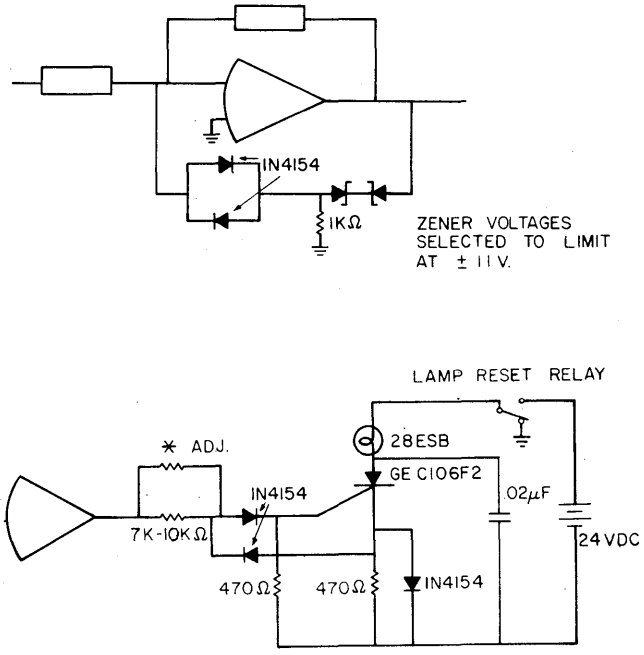
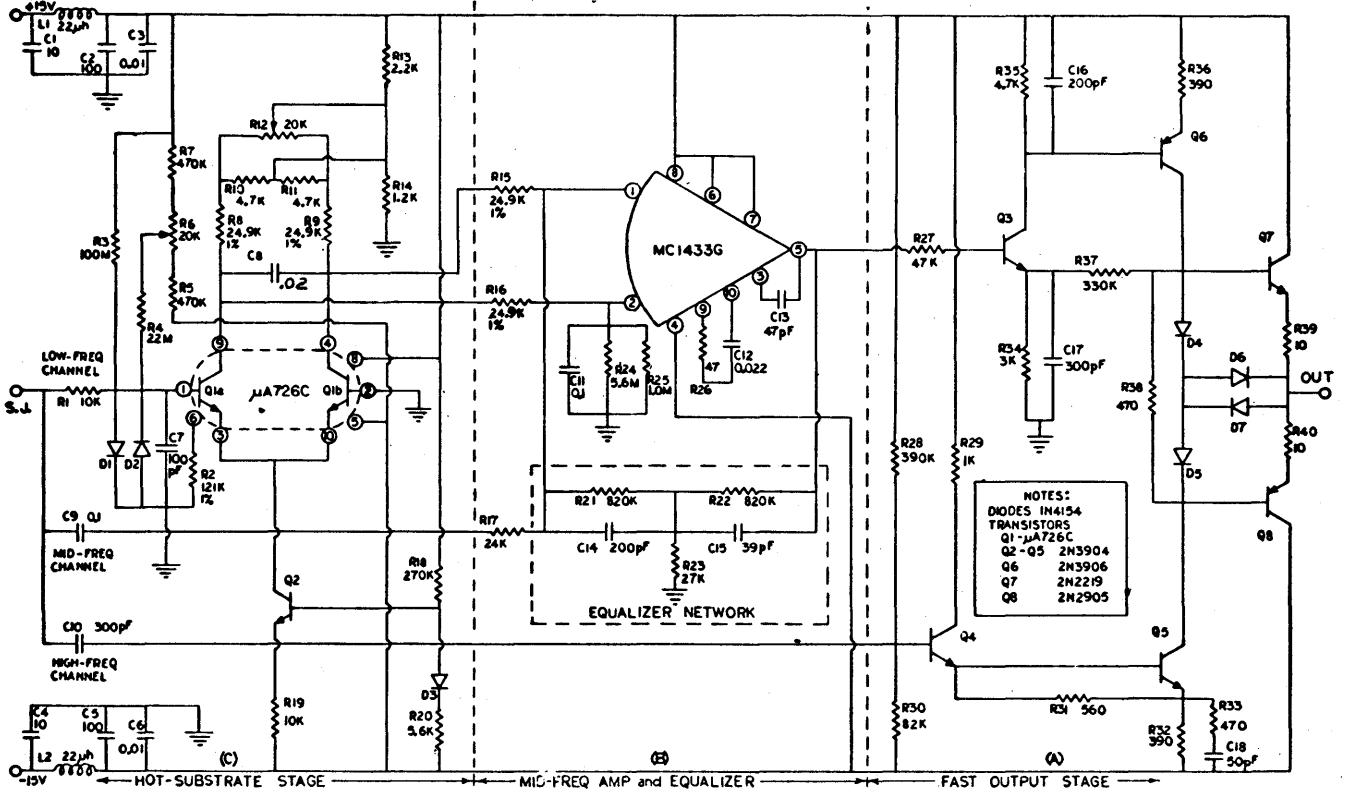


Figure 4—LOCUST amplifiers have overvoltage limiters with low capacitive leakage (a) as well as output-current limiters so that no POT-SET relays are needed. (b) shows the new LOCUST SCR overload-indicator circuit with visual and logic outputs

Figures 3c—Complete LOCUST d-c amplifier circuit



a printed-circuit board with a minimum of adjustments; this is a necessary requirement for The University of Arizona's Analog/Hybrid Computer Laboratory, which utilizes student technicians for the assembly and testing of the completed amplifiers.

New integrator/track-hold circuit

The LOCUST integrator/track-hold module employs the LOCUST amplifier together with the Burr-Brown Model 9944, a new special-purpose electronic switch module designed especially for the LOCUST system (Figure 5a). The Model 9944 consists of a Model 4010 high-speed (nonsaturating) electronic switch circuit, suggested by G. A. Korn [9] and designed by J. Naylor. This is a switched feedback amplifier having an FET input stage for low current-drift (Figure 5b), integrating capacitors (0.01, 0.1 and 1.0 μF), a high-impedance summing network for "slow" (real-time) simulation, relays with MECL input-level drivers for time-scale changing relays, and track-hold summing and feedback resistors. The important specifications of the integrator/track-hold combination are summarzied in Table 1b.

Figure 6 illustrates the operating modes of the integrator/trackhold circuit. In HOLD or COMPUTE (Figure 6a), the switch is effectively out of the circuit when the integrator inputs are not used, or integrates the voltages on the integrator inputs.

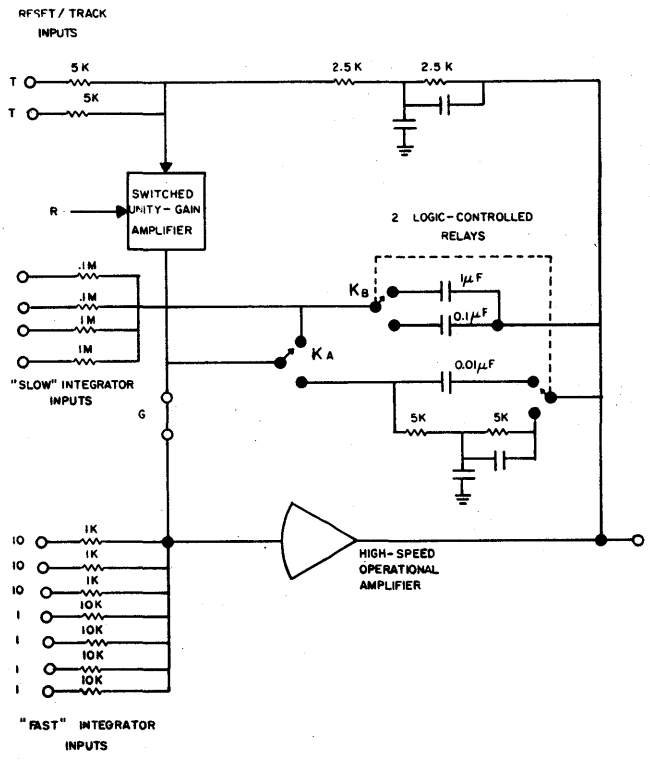


Figure 5a—LOCUST integrator/track-hold circuit. Three logic-switched capacitors and "fast" and "slow" input summing networks provide five time scales with integrator gains between 1 and 10^5 sec^{-1}

In TRACK or RESET (Figure 6b), the track inputs are connected and the integrator inputs are shorted out by the internal impedance Z_s of the switch. Z_s must be small compared to the parallel combination of integrator inputs so that signals of the integrator inputs essentially do not appear at the output. If *one* integrator input of 10 volts is to contribute less than 10 mV error in RESET, one must have

$$\frac{Z_s}{R'} < \alpha\, 10^{-3}\left(1+\frac{R_o}{Z_1}\right)^{-1}$$

where α is the voltage gain of the switch and Z_1 is the parallel combination of R_1 and R_2 (Figure 6c). In LOCUST the smallest computing resistance is 1 kΩ. Therefore,

$$Z_s < 0.33 \text{ ohm.}$$

To allow for paralleled integrator inputs, Z_s should be less than 0.1 ohm.

Referring again to Figure 6, the size of the storage capacitor C is determined by two conflicting requirements:

1. Input errors due to voltage and current offset, integration and switching spikes decrease with increasing C.
2. The small-signal frequency response in TRACK and also the maximum voltage rate of change in TRACK improve as C decreases.

Resistors R_0 and R_1 are low resistance (5kΩ) to minimize stray capacitance effects. Capacitor C_0 is required to decrease amplitude peaking in TRACK; the system might even become unstable if C_0 is omitted. When small values of R_0 and R_1 are used, C_1 is usually not required.

The circuit switches into COMPUTE (HOLD) in typically 40 ns, and two integrators switch within 10 ns of each other, thus meeting the third requirement noted earlier. The LOCUST mode-control switch is, to the best of the writer's knowledge, faster than any commercially available mode control switch by a full order of magnitude.

Analog multiplier/divider and function generator

The LOCUST analog multiplier/divider is a Burr-Brown Model 9943 Fast Quarter-square Multiplier/Divider very similar to that designed by R. Whigham

for ASTRAC II at The University of Arizona.[19]

The multiplier contains five submodules; three wide-band amplifiers, and two precision absolute-value complementary squaring networks. The diode function-generator type squaring modules are temperature compensated and have internal Zener-regulated reference voltages to eliminate the need for adjustments.

The multiplier/divider is installed in LOCUST analog-module can and plugged directly into the rear of the patchbay. The patching field permits the unit to be patched as a 4–quadrant multiplier or as a 2–quadrant divider. Specifications are listed in Table 1c.

The LOCUST plug-in diode function generator (Figure 7) [20] plugs into the front of the analog patch panel and covers the patching field of an associated dual-amplifier model. The circuit (Figure 7b) utilizes two amplifiers to provide both positive and negative slopes. Six breakpoints are fixed at approximately ±1, ±3, and ±5.5 volts. Slopes and d-c level (parallax) are set by means of eight 15-turn wire-wound potentiometers. Input diodes D1 and D2 reduce input loading while shunt diodes D3 and D4, together with the low impedance levels used throughout LOCUST, improves frequency response. The maximum current load presented to an amplifier is 18 milliamperes.

The error for a given frequency depends on the function set up. In general, the error is less than 0.5 percent of half scale for input frequencies below 5 kHz with slopes less than 2 volts per volt. The large signal amplitude frequency response for an SCi standard straightline function (Simulation, 1963) exceed 1 MHz. Function-generator voltage drift is approximately 2 mV/degC.

Circle-test performance

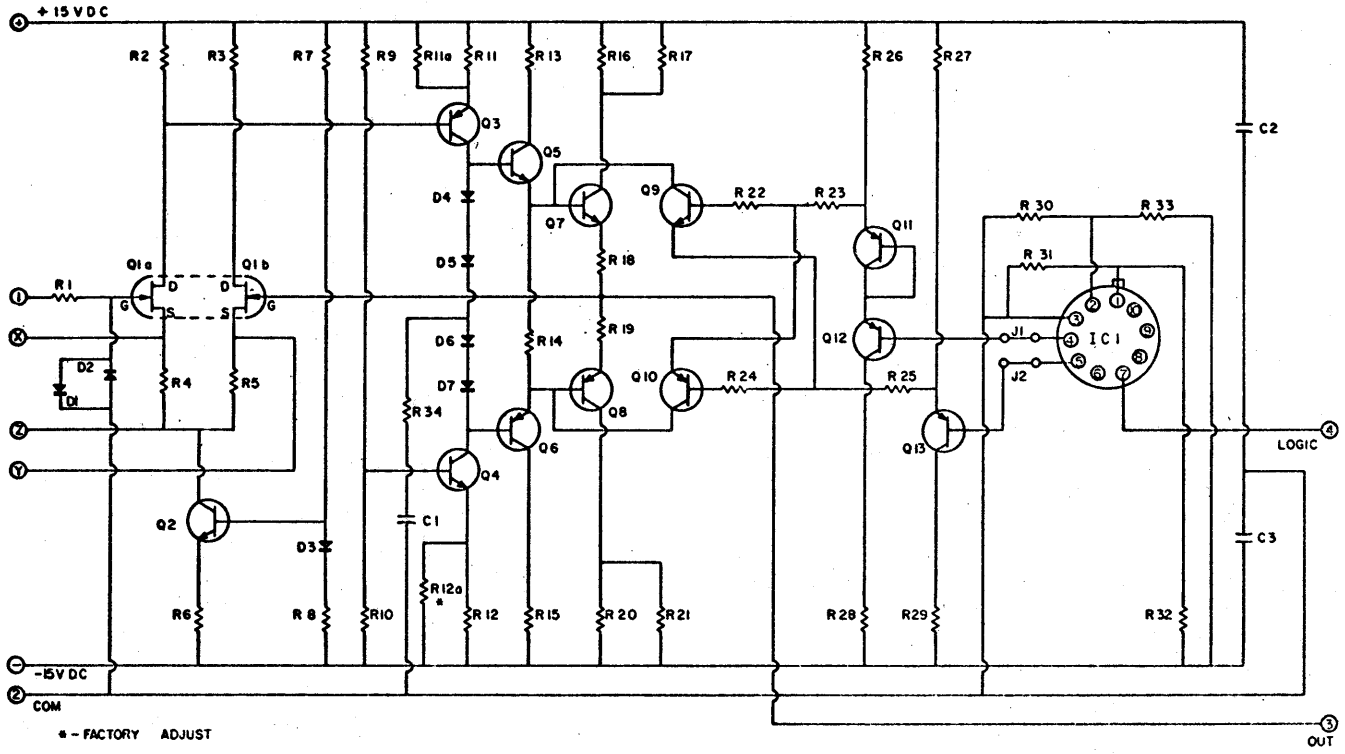
The combined effects of phase shift in linear computing elements can be measured by applying the circle test.[8] The actual computer solution to the ideal harmonic-oscillator equation is

$$(p^2+\omega^2)=0, \quad Y(0)=0, \quad \dot{Y}(0)=B$$

$$Y = B\, e^{-\omega\delta(\omega)\tau} \sin\omega\tau$$

where p is the operator $\frac{d}{dt}$, and $\delta(\omega)$ is the phase shift due to the integrator phase inverter and coefficient potentiometer. The solution amplitude decays exponentially with time for a positive net phase error (lead), while a negative net phase error (lag) causes the amplitude to

Figure 5b—The new ultra-fast mode-control switch is itself a switched operational amplifier with FET input. A current-mode-logic gate (IC1) switches an integrator into COMPUTE or HOLD within 40 n-sec, with at most 10 nsec between integrators



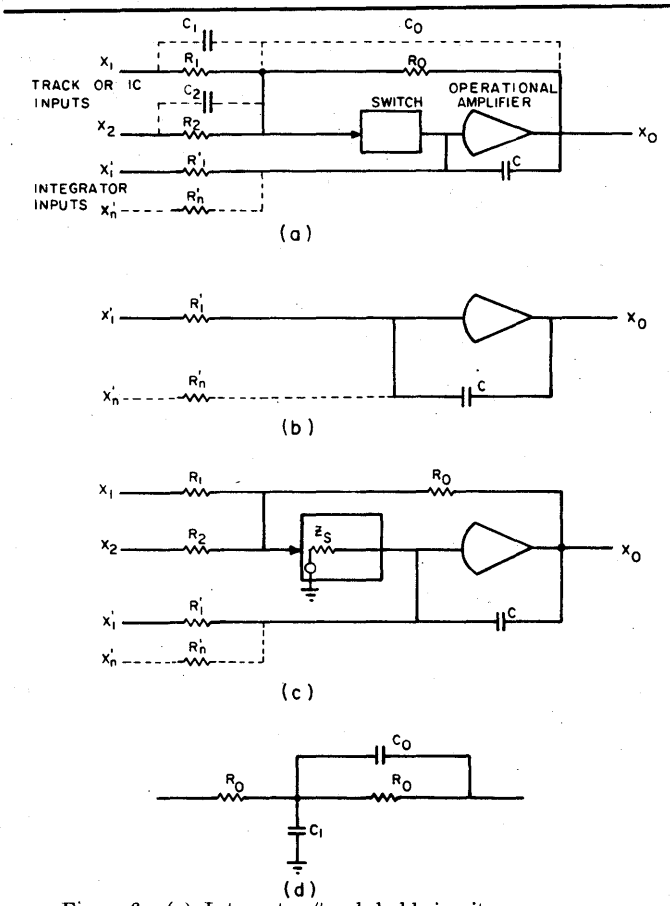


Figure 6—(a) Integrator/track-hold circuit
(b) Equivalent circuit in HOLD or COMPUTE
(c) Equivalent circuit in TRACK or RESET
(d) Modified feedback network replacing impedance R_o of (a)

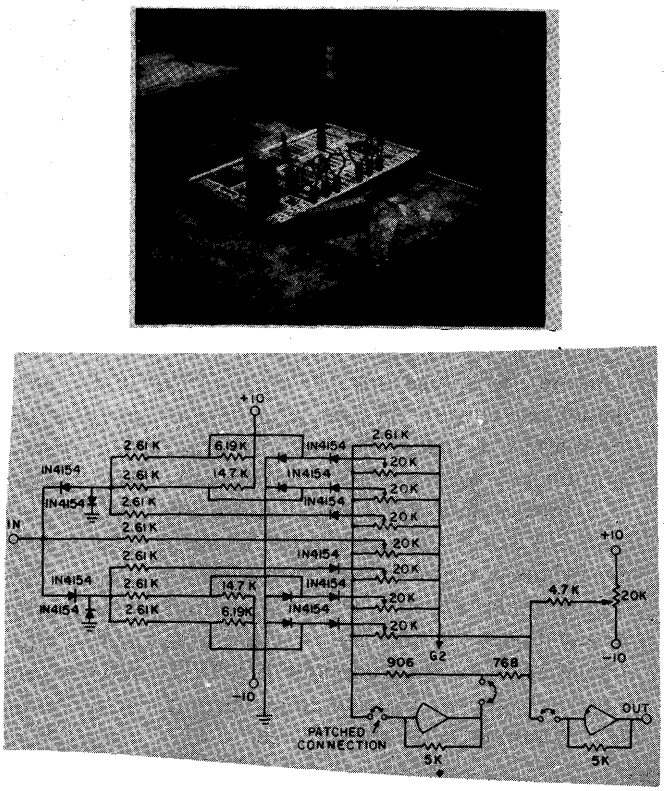


Figure 7— (a) LOCUST patchbay plug-in diode function generator and (b) its schematic diagram

increase exponentially. Figure 8 illustrates the computer setup and the combined phase shift for the LOCUST elements.

LOCUST digital system

Basic iterative differential analyzer control requires periodic RESET pulses to control switches and integrators. More flexible control circuits permit one to change repetition rates, to vary the length of the COMPUTE and/or RESET period, and to select the timing of sampling pulses for accurate analog/digital conversion readout and analog storage. Subroutine counters may also be used to change the computer program after a preset number of iterations or other events.

The LOCUST digital system provides many of these functions in the form of "packaged" subroutines selected by front panel control (Figure 8). This feature liberates the operator from designing and patching frequently used subroutines with the patchable digital logic and leaves him free to concentrate on a much wider class of operations, for detailed iterative subroutine control design is far from easy for most analog computer users. Less frequently employed subroutines can still be patched on the digital patchbay, which contains free logic gates, flip-flops, decade counters and shift registers. Table 2 lists the digital logic cards used.

The LOCUST digital system also includes digital patchbay terminations for:

1. Inputs for individual integrator/track-hold capacitor and feedback resistor selection.
2. 4 analog comparator outputs
3. 12 logic-state lamp inputs
4. 4 free pushbutton logic outputs.
5. A logic output common to all amplifier overload circuits
6. Inputs to level changers (noise drivers) which provide ±10-V outputs under logic control
7. Trunk lines to a PDP-9 digital computer interface equipment
8. A/D converter control inputs.

These digital system features incorporate some new operator conveniences not found in ASTRAC II: (1) each integrator or track-hold capacitor selection re-

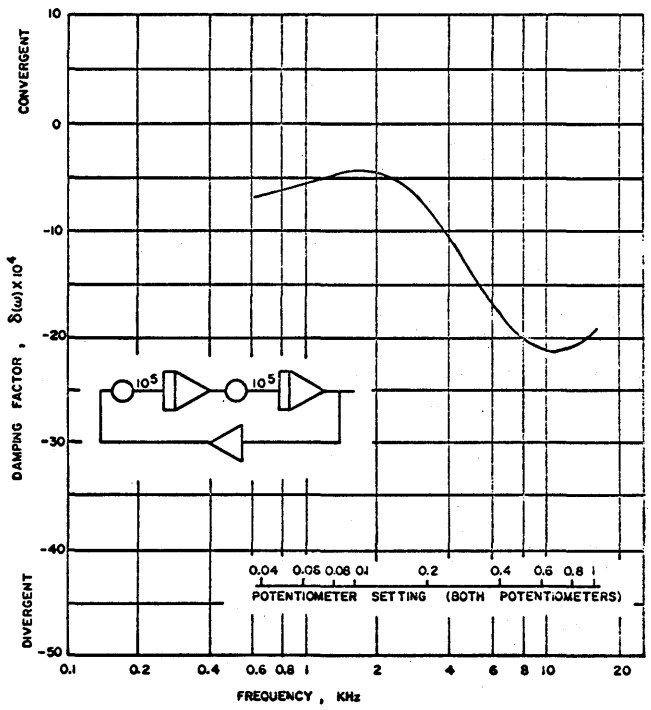


Figure 8—Circle-test performance (Ref. 8) indicates integrator and phase-inverter dynamic errors within 0.1 percent at 10 kHz.

quires only a single patch connection, (2) digital system logic levels can also control capacitor selection, (3) free fixed-delay (100 ns) one-shot multivibrators associated with the shift registers are available, (4) front-panel pushbuttons with digital-system-logic-level outputs are available, (5) an overload logic output common to integrator/track-hold and summing amplifiers, (6) three switch-selected SCAN FORWARD and REVERSE SCAN rates. In addition, trunk, interrupt, and control terminations enable LOCUST to interface directly with a digital computer. For the same reason, INITIAL RESET, COMPUTE, and SINGLE RUN modes can be controlled by signals on the digital patchbay.

Digital control functions are divided among a *master timer*, an *auxiliary timer*, and a *subroutine counter* (Figure 9b). Modular design permits one to build the master timer first and add the other functions as needed.

Clock and repetition-rate selection

A frequency-dividing counter chain receives 4 MHz pulses from a crystal-controlled clock and delivers timing pulses whose frequencies are 4 MHz, 2 MHz, 1 MHz, 500 kHz, 100 kHz, 50 kHz, and 10 kHz for control and display purposes. From these, the *repetition-rate selector switch* selects a clock pulse train (Cl) which has exactly 1,000 times the desired computer repetition rate $f_R = 1/T_R$ = 2,000, 1,000, 500, 250, 100, 50, and 10 computer runs per second (Figure 9b). All subsequent analog-hybrid-computer timing is performed in terms of these Cl pulses (1,000 per computer run), whose duration is called *1 millirun*. This permits the repetition-rate selector to automatically change the time scale of all timing and counting operations. The repetition rate selector can also change integrator capacitors through relays to provide completely automatic time scale changes; this automatic capacitor selection can be overridden with patched logic connections.

The master timer

The master timer (C_1 counter, Figure 9b) fed by Cl, is a three-decade decimal counter with *dual* thumbwheel-switch preset outputs designed to perform the following timing functions:

1. It counts down by 1,000 to mark the start of periodic COMPUTE periods (t = 0, Figure 10a).
2. It produces accurate timing markers at 10 and 100 times the computer repetition rate.
3. Thumbwheel decade switches select counter outputs at t = T and t = t_1 milliruns after the start of each COMPUTE period in steps of 1 millirun.

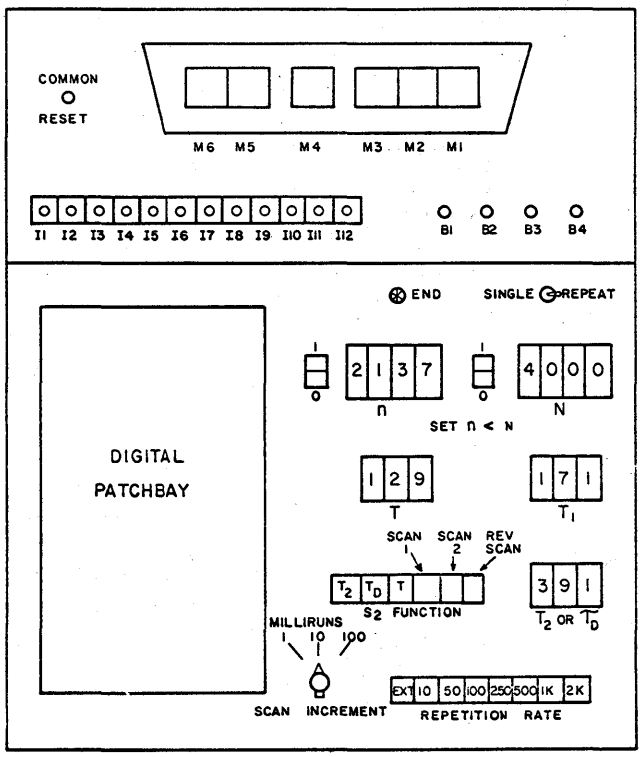
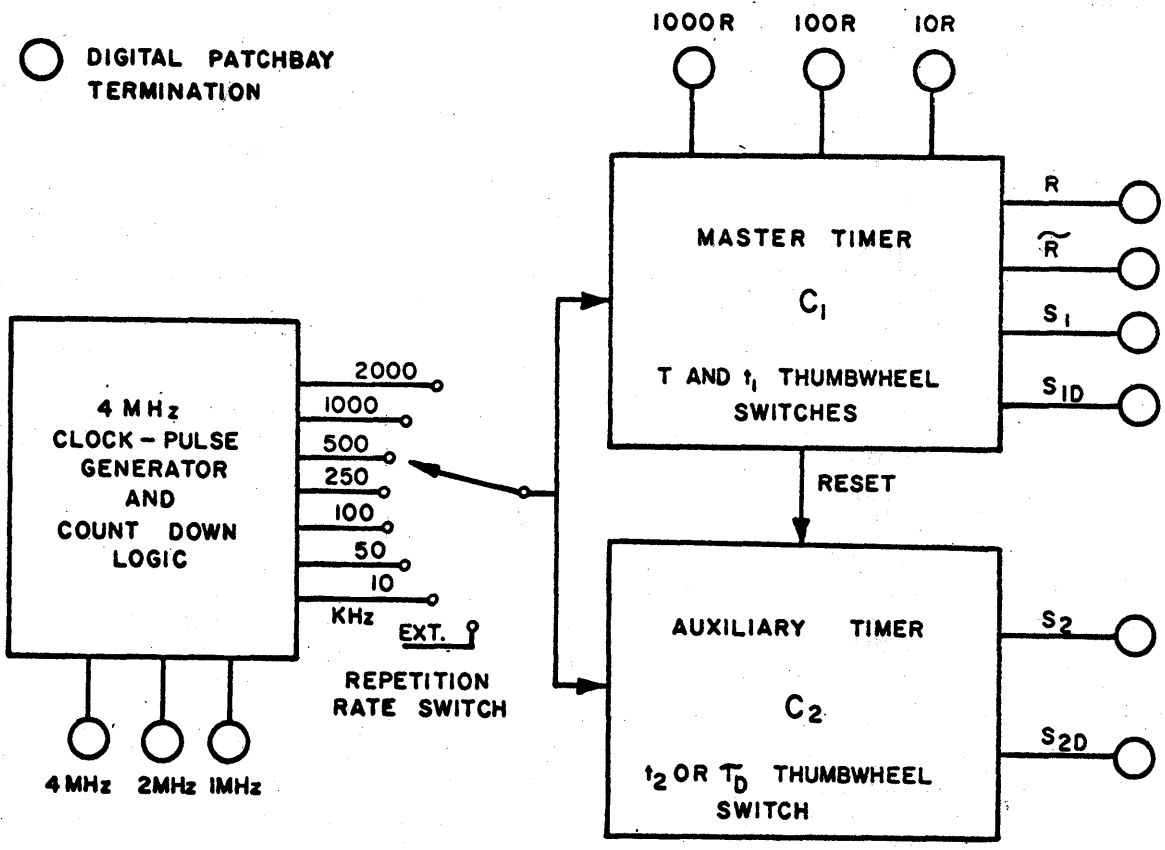


Figure 9—LOCUST digital module panel (a)

In normal repetitive operation (Figure 10a), the integrator control pulse R is equal to logical 1 at t = 0 and a logical 0 at t = T thus producing periodic pulses so that COMPUTE periods of length T alternate with RESET periods of length $T_R - T$. Note that one can independently select T_R (with the *repetition rate selector*) and T (with the thumbwheel decade switches). The output at $t = t_1$ milliruns feeds a timing logic block to produce periodic sampling pulses S_1 and also delayed sampling pulse S_{1D} of length $T_S = T_R/10$ for special track-told-pair operations.[8] A track-hold circuit controlled by S_1 will periodically TRACK for T_S milliruns and then switch into HOLD at $t = t_1$ milliruns. S_{1D} switches T_S milliruns later than S_1 for memory pair or memory triplet (when used with R) operation.

All timing pulses, RESET pulses and TRACK-HOLD control pulses are available on the digital patchbay for flexible control and timing of individual integrators, switches, and digital operations. In normal repetitive hybrid-computer operation integrators are controlled by R, and the track-hold circuits are controlled by S_1 for digital readout of sample values $X(t_1)$ at the accurately set computer time t_1. Different patching



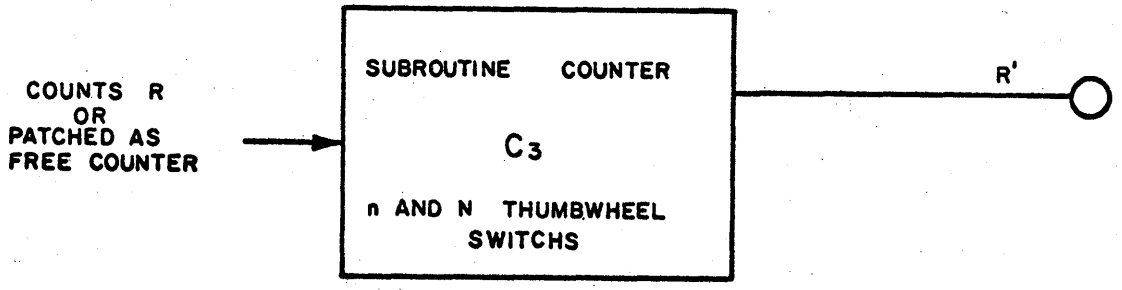


Figure 9— block diagram (b)

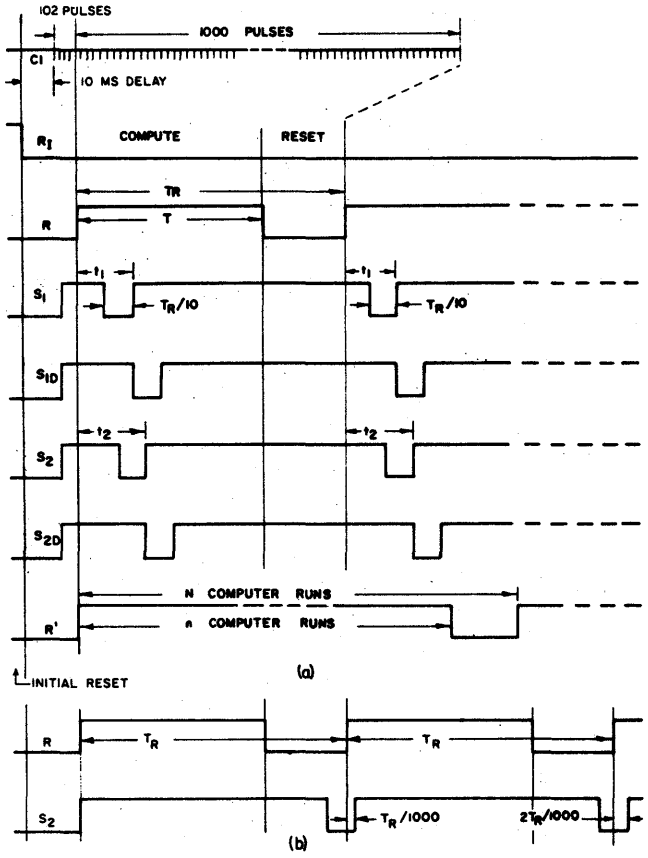


Figure 10—(a) Timing and control signals for normal repetitive operation and (b) control signals for SCAN 1 operation

connections can employ R, S_1, and S_{1D} to produce very flexible memory and timing control.

A patchable master timer-reset input allows one to reset C_1 and permits finer control of the computer-run period than is possible by the normal coarse repetition-rate selection. For instance, one can use S_1 to reset C_1 at t_1; this will reset integrators for 102 milliruns and start a new COMPUTE period of duration t_1 milliruns. Many other possibilities exist. One may also patch to an external clock. One can also interrupt the C_1 pulses with a digital-computer control signal. It is also possible to control the length of individual computer runs with patched comparator logic to conserve time in long computations.

The auxiliary timer

The auxiliary timer (C_2 counter) is another three-decade counter with a thumbwheel decade-switch preset output. The S_2 *function swtich* controls the operation of the C_2 counter and its associated logic. For the different S_2 *function switch* settings, the thumbwheel switch, "t_2 or τ_D", functions as follows (refer to Figures 9 and 10).

1. *T_2 position*—The trailing edge of S_2 (TRACK to HOLD) occurs t_2 milliruns after the start of each COMPUTE period.
2. *τ_D position*—The trailing edge of S_2 occurs at $t_1 + \tau_D$ milliruns after the start of each COMPUTE period. *Thus the "t_2 or τ_D" thumbwheel switch directly sets a delay interval of length τ_D between the S_1 and S_2 sampling pulses.* When S_1 and S_2 are used to control track-hold circuits, the sample pairs $X(t_1)$, $Y(t_1 + \tau_D)$ are produced, e.g., for correlation and prediction studies.
3. *T position*—The trailing edge of S_2 now occurs at $T + t_2$ milliruns after the start of each COMPUTE period. *S_2 and S_{2D} then serve for readout during the RESET period, as S_1 and S_{1D} serve during the COMPUTE period.* This is useful for "alternating" differential analyzer runs using integrator groups controlled by R and $\bar{R}$.
4. *SCAN 1 position*—C_2 recycles after 1,000 + M milliruns where M is 1, 10 or 100 millruns selected by the *scan increment switch*. On the first run, the trailing edge of S_2 occurs at t_2 milliruns; on the second run S_2 occurs at $t_2 + M$ milliruns; on the third at $t_2 + 2M$ milliruns, etc. (Figure 10b). *Track-hold circuits controlled by S_2 and S_{2D} will then "scan" periodic computer runs for readout into slow recorders, printers, or digital computers.* With the *repetition rate switch* set at 100 and M set to 1, for instance, a complete solution scan requires 10 seconds.
5. *SCAN 2 position*—The subroutine counter (C_3 counter) permits the scanning S_2 to step forward by M milliruns only ofter a preset number, N, computer runs or N other events. This feature is used for *automatic computation of random-process statistics (correlation functions, delay errors) over samples of N computer runs.*
6. *REVERSE SCAN position*—On the first run, the trailing edge of S_2 occurs at t_2 milliruns; on the second run S_2 ends at $t_2 - M$ milliruns and continues to scan *backwards* in steps of M milliruns

The *SCAN* modes are useful for slow recording of repetitive solutions (X vs. t, or Y vs. X), but also for automatic parameter changing (new values of a repetitive-analog-computer solution are used in successive computer runs,) for multiple-solution oscilloscope displays and for solution checks with slow computers. The REVERSE SCAN mode is useful for computing convolution integrals, for backward integration (e.g., in boundary-value problems), for modified-adjoint-system techniques, and for controlling delay-line-memory read/write cycles. The SCAN 2 mode is, as noted earlier, intended for automatic computation of statistics over N computer runs.

Subroutine counter

The subroutine counter C_3 (Figure 9) is a four-decade counter with *dual* thumbwheel switch preset outputs and is patched to count computer runs, comparator output steps or other events. C_3 produces preset counter outputs after n and N events ($0<N<20{,}000$). This counter output R′ is used to terminate and/or start subroutine sequences. The *repeat switch* (see Figure 9a) permits C_3 to be reset after N events and to recycle the entire sequence. In particular, one can control fast integrators with R and slow integrators with R′ (two-time-scale operation), thus producing "nested" iterative subroutines.

Starting and external control

Before computation, one depresses the INITIAL RESET button momentarily or holds the COMPUTE button down to produce the following conditions:

1. The pushbutton control-logic establishes the *initial reset* mode (R_1 = "1").
2. The subroutine counter C_3 is reset (R′ = "0").
3. The master timer C_1 and the auxiliary timer C_2 are reset; output logic is set to produce R = "0" and the correct initial values of the control pulses S_1, S_{1D}, S_2, and S_{2D} ("0" TRACK, Figure 10a).

It follows that all integrators, memory pairs, and devices controlled by R, R′, S_1, S_{1D}, S_2, and S_{2D} are now reset to suitable initial conditions (RESET or TRACK) ready for computation.

This state is maintained until one releases (or depresses and releases) the COMPUTE button. Then a delay of 10 ms occurs (determined by a one-shot multivibrator in the pushbutton-logic-control circuit). Then S_1, S_{1D}, S_2, and S_{2C} are set to "1" (HOLD). Next C_1 runs through 102 C_l pulses and then R and R′ are set to "1" (COMPUTE) thus beginning the first COMPUTE period (Figure 10a).

The SINGLE RUN button produces a single computer run without resetting C_1, C_2, or C_3, so that one can check the progress of iterative subroutines computer run by computer run.

External control inputs are available on the digital patchbay and perform the following functions:

1. C_l RESET—permits finer control of the repetition rate.
2. C_l INTERRUPT—permits interruption of C_l pulses into C_1 and C_2.
3. EXTERNAL INITIAL RESET—permits logic, perhaps from a digital computer, to switch the computer into INITIAL RESET.
4. EXTERNAL COMPUTE—permits logic to start the computer.
5. EXTERNAL SINGLE RUN—External logic can initiate a single computer run.

These logic functions are intended mainly for control of iterative analog/hybrid computation by an associated digital computer.

Special computing elements

To make maximum use of the LOCUST computer system, a number of special digital and hybrid computing elements are needed. Noise generators, digitally controlled analog switches (D/A switches and D/A "noise drivers" or reference switches), multiplying digital-to-analog converters (MDACS), analog comparators, shift registers and modulo-2 adders permit computer implementation of random-process studies, sampled-data systems and various optimization schemes. Multiplying digital-to-analog converters interface with a small digital computer (CPDP–9) for computation of statistics, function generation, and parameter adjustment.

Analog comparator

The LOCUST and APE II [11] analog comparators supply a digital output whose logic levels ("1" or "0") depend upon the sign of an analog input or on the sign of the sum of two analog imputs. In modern iterative differential analyzer applications, comparators are used in two different ways. The first mode of application involves the changing of a computer program *between* computer iterations, and the second requires the changing of the program *during* the computation period. The requirement for speed (rise-time plus delay) is not critical in the first case, because the comparator is driven by a track-hold circuit whose output is essentially constant during the voltage comparison. But comparators which must detect sign changes *during* the computation period must be extremely fast, because at iteration rates of 2000 runs per second, voltage change rates of up to $20\pi f$ volts per second can occur in LOCUST. To remain within 0.2 percent comparator-timing error computation, full-scale sinusoids would have to be held below 3 kHz for 100 nanosecond conversion delays.

The LOCUST/APE II comparator utilizes the Fairchild μA710C integrated-circuit comparator (Figure 11).[2] The input circuit is designed with very low impedance to reduce offset-current errors and is diode-protected to handle 10-V analog computing voltages. The output circuit is a MECL gate, which acts as a buffer and logic-level changer. Free patched logic serves for comparator latching[8] and strobing if desired.

Table 1d summarizes the performance of the integrated-circuit analog comparator.

TABLE 1—LOCUST specifications (at 25° C)

(a) D-c Amplifier

Open-loop Gain	120	db
Unity Gain Bandwidth	30	MHz
Rated Output		
Voltage	±10	V
Current	±30	mA
Max. Frequency for Rated Output	5	MHz
Input Offset		
Voltage	Adj. to zero	mV
Current	Adj. to zero	nA
Input Offset Drift		
Voltage	±2.5	μV/degC
Bias Current	±50	pA/degC
Input Noise	15	μVrms
Open-loop Output impedance		
at 100 Hz	100	ohms
at 10 kHz	50	ohms
Unity-gain Inverter (no load)		
Freq. Response — 3 dB	10	MHz
Error at 10 kHz	0.1	%
Max. Capacitive Load	0.001	μF
Quiescent Current	20	mA

(b) LOCUS I Integrator/Track-hold

TRACK–HOLD		
Gain Accuracy at d-c	±0.1	%
Small Signal Freq. Response–3 dB (0.01 μ F holding capacitor)	2	MHz
Max. Dynamic Error in TRACK at 10 kHz (no load)	0.1	%
Rated Output		
Voltage	±10	V
Current	±20	mA
Max. Freq. for Rated Output	40	kHz
Output Impedance (at 100 Hz)	0.1	ohms
Output Voltage Drift		
In HOLD (for 10 ms and 0.01 μF holding capacitor)		
nominal	±0.2	mV
vs. temperature	±0.05	mV/degC
In TRACK		
vs. temperature	±100	μV/degC
Aperture Time (time to switch into HOLD or COMPUTE)	40	ns
Max. Difference Between Aperture Times of Different Integrators	10	ns
Acquisition Time (settling to 0.1% for 10–V input step)	10	μs
SWITCHED INTEGRATOR		
Voltage Gain	10^5, 10^4, 10^3, 10^2, 10, 1	sec^{-1}
Gain Accuracy	±0.1	%
Integrator Input Feedthrough in RESET (for ±10–V, 100 Hz, 1 k Ω input resistor, and 0.01 μF capacitor)	±1	mV
Rated Output		
Voltage	±10	V
Current	±30	mA
Output Impedance (at 100 Hz)	0.1	ohms

TABLE 1—Continued

Output Voltage Drift in COMPUTE (for 10 ms, 10 k Ω input resistor, 0.01 μF capacitor)	±0.3	mV/degC
Switching Times		
Compute	40	ns
Reset (to within 0.1% of initial condition input)	10	μs

(c) Multiplier/Divider

Input Signal Range	±10	V
Rated Output		
Voltage	±10	V
Current	±20	mA
Output Impedance	10	ohms
Static Accuracy		
Multiplication	±0.15	%
Division	±1.5/Y	%
Dynamic Error (at 10kHz)	±0.5	%

(d) Analog Comparator

Input Signal Range	±10	V
Input Impedance	750	ohms
Output Levels (Complementary)	MECL logic levels	
D-c (static) Switching Accuracy	Adj. to ±1	mV
A-c (dynamic) Switching Speed		
Rise Time	20	ns
Delay*	80	ns
Hysteresis	±10	mV
Drift of Threshold with Temperature	100	μV/degC

*Measured with 0.1 peak 500 kHz sine wave input; delay is less than 40 ns with fast rise time inputs. With very slow inputs, additional delay will result from the time required for the input to pass through the 2-mV hysteresis interval.

The comparators are housed in analog-module cans and plugged directly into the rear of the analog patchbay.

Digital-to-analog switch and logic-controlled relay

The LOCUST digital-to-analog switches can be thought of as one-bit digital-to-analog-multipliers, which are used to switch analog variables ON and OFF, and to switch between two analog variables (SPDT action) in response to digital commands. The switches are housed in analog-module cans and plug directly into the rear of the patchbay.

The LOCUST D/A switches are dual-transistor shunt switches (Figure 12) similar to those used in ASTRAC II. Two sets of switches are digitally driven

in pairs to implement a SPDT switch. Switching times are 400 ns turn OFF and 600 ns turn ON.

In addition to electronic switches, logic-controlled reed relay switches are included in each D/A switch module for use in setting up initial conditions for recursive computations in the INITIAL RESET mode, or for switching analog variables in the "slow" mode of operation.

Diode bridge and diode pair

Four diode-bridges and six diode-pairs are available on the analog patchpanel for use in types of nonlinear analog setups such as precision limiters and dead-space circuits, maximum-value detectors, absolute-value circuits etc..[8]

Shift register and modulo-2 adder

The LOCUST logic patchpanel contains twenty-four shift register stages, divided into four groups of six stages each (Figure 13.) Each group has a common reset, initial-condition-setting one-shot multivibrator and shift inputs. Each output is patched to the next to connect any number of stages as a shift register.

Shift registers are used in digital sequence generators, ring counters, switch-tail counters, pseudo-random noise generators, and digital-delay networks. An individual stage can be used as a digital sample-hold and the one-shot multivibrator is often useful in other digital circuits.

Six modulo-2 adders (exclusive-OR circuits) are available for use in pseudo-random noise generator circuits for digital correlation and, of course, for performing the exclusive-OR operation. By inverting the output with a gate, one has a 1-bit digital comparator or a 1-bit multiplier.

Sampling-type binary noise generator and noise driver

Requirements on a noise generator for analog computation are rather stringent with respect to the stability of its statistics (mean, mean square, and spectral density), and these are often difficult to control. The

Figure 11—LOCUST analog comparators employ the inexpensive μA710C integrated circuit with a diode-protected low-impedance input circuit for low drift. Note the decoupling filters

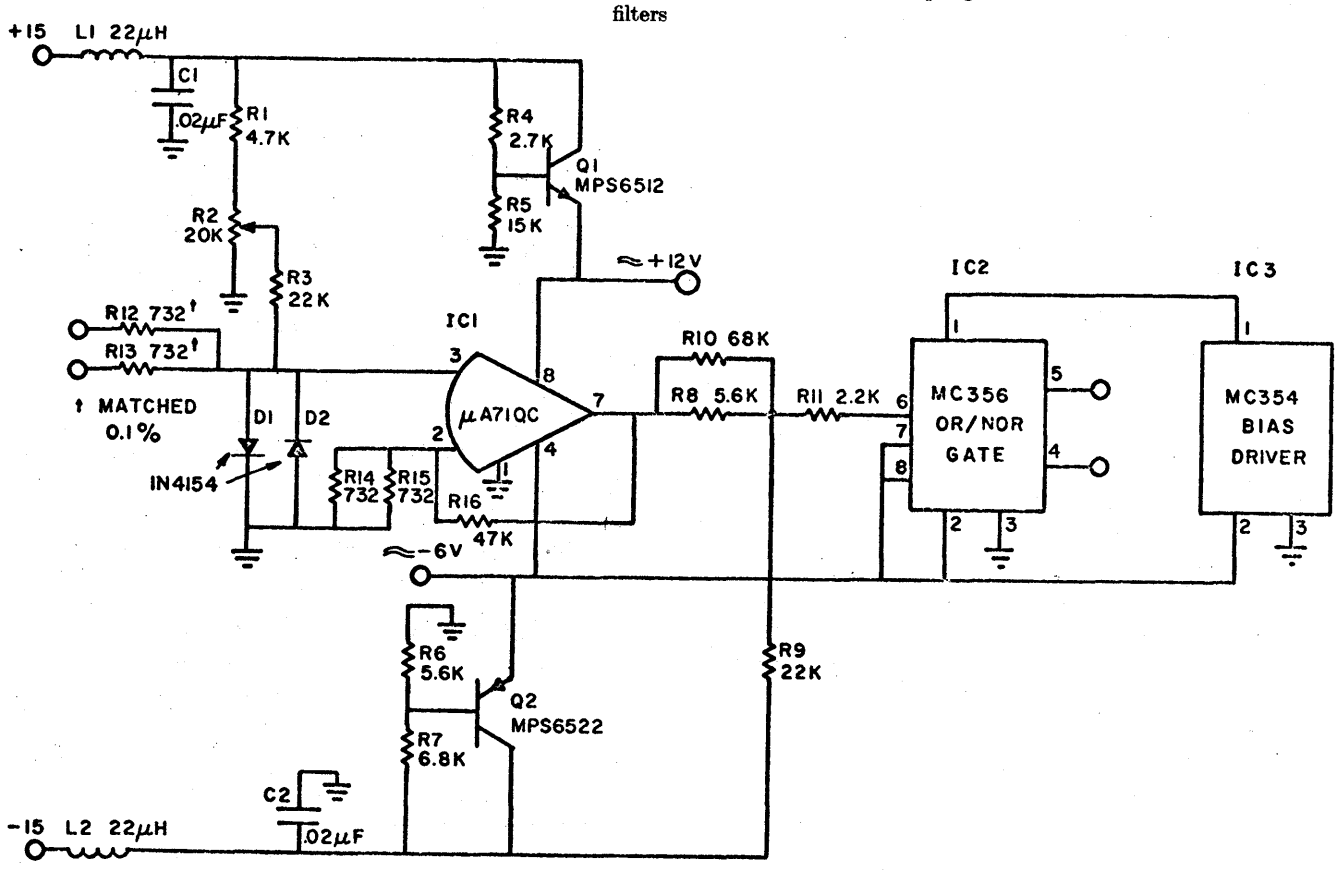


Figure 12—LOCUST D/A switch is a dual transistor shunt switch. The second switching transistor is inverted for low offset

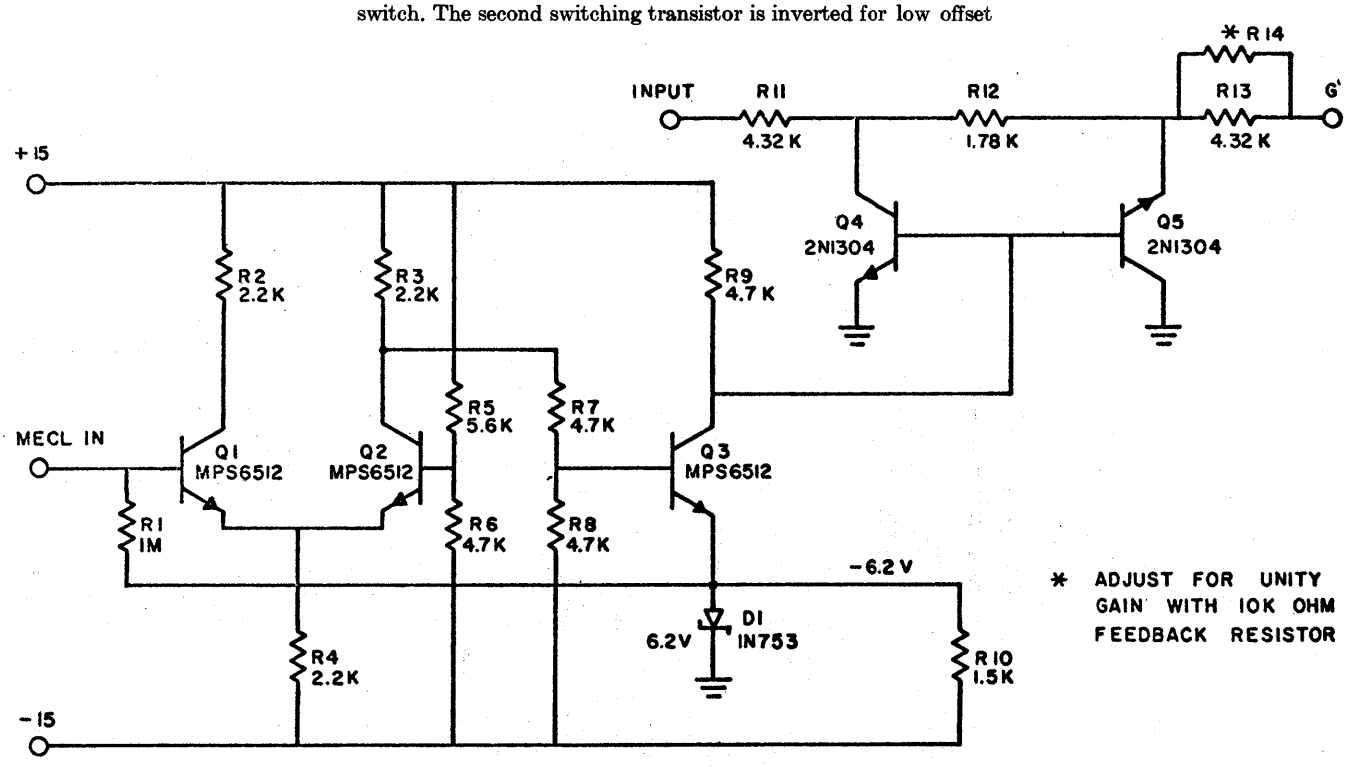
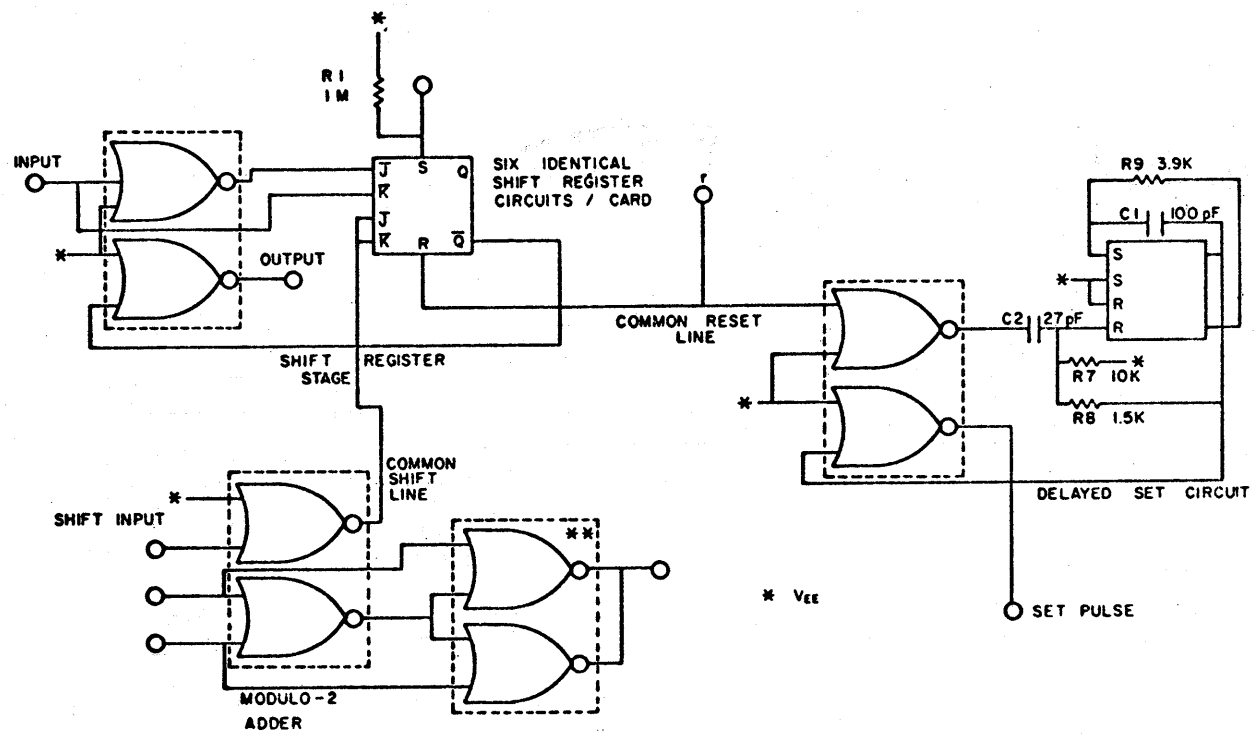


Figure 13—One of the four 6-stage shift registers designed for especially simple patching of sequence generators and coun-



LOCUST binary noise generator is of the type described by Kohne et al[7,12] and is an improved version of the one used in ASTRAC II. The noise generator (Figure 14) provides a clocked binary waveform whose statistics are essentially independent of the fluctuations in the statistics of the noise source and the drift of the Schmitt-trigger level. The key to the stability lies in the fact that flip-flop F0 has a probability of ½ of being either a "1" or a "0," regardless of input probability variations, when the flip-flop is sampled periodically with a sufficiently large intervening number of state changes. One has, of course, sacrificed one-half of the spectral content of the original noise source because the flip-flop divides the mean zero-crossing rate by two.

In order to use a clock frequency of 1 MHz, one requires a low-quality noise source having a flat spectrum (uncorrelated noise samples) to beyond 10 MHz so as to ensure a sufficient number of state changes between clock pulses. The noise is derived from a noise Zener diode biased near the breakdown knee of its characteristic curve. The digital logic is implemented with MECL integrated circuits.

By filtering the output of the binary noise source with an R–C filter whose time constant is large compared to the clock period, one may obtain approximately Gaussian noise, which is useful for many simulation experiments.[10]

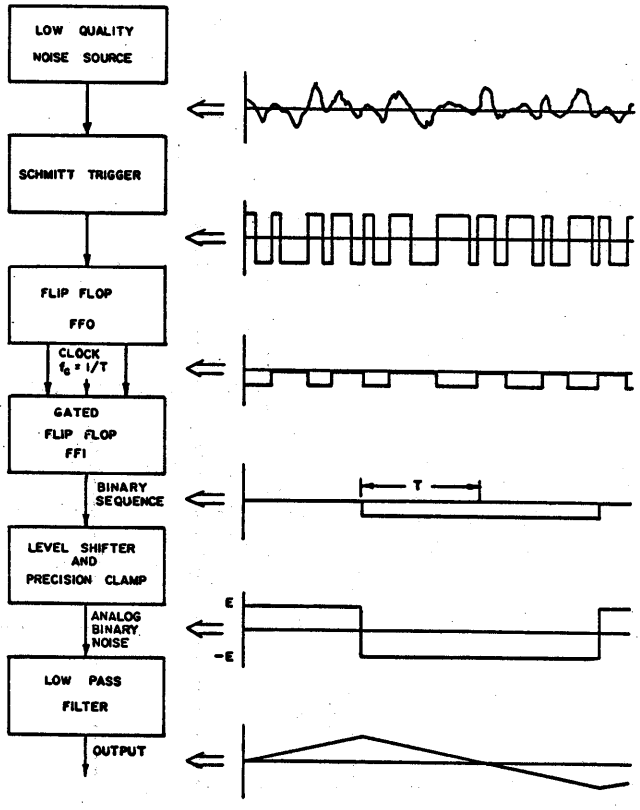


Figure 14—Sampling-type binary noise generator block diagram (a) and circuit (b)

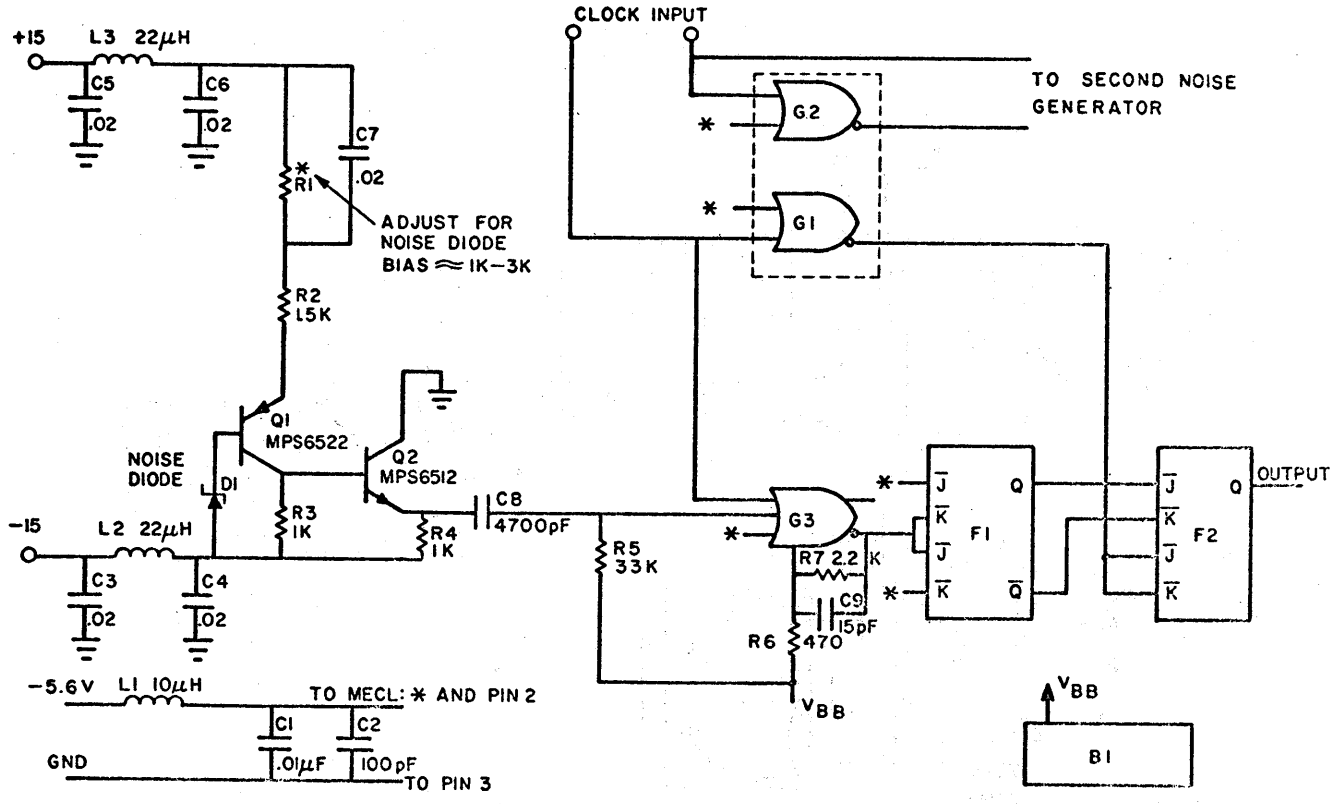


TABLE 2—LOCUST Digital System Logic Cards

1. A front panel *pushbutton logic card* provides computer *initial reset* pulse R_I and a clock interrupt signal to the master timer logic.
2. *A master timer card* provides periodic computer *reset pulses*, a presettable sampling pulse and several other timing pulses.
3. An *auxiliary timer card* provides a presettable *sampling pulse* whose timing relative to the master timer *reset* and *sampling pulses* is selected by a front panel switch; it also provides *scan* modes whereby the sampling pulse is timed to move forward or backward with respect to the computer *reset* pulses one step each computer run or N computer runs.
4. A *subroutine counter card* provides an output pulse every n and N computer runs (or other events) preset by front panel decade switches.
5. Six *free logic cards* each containing 2 J-K type flip-flops, two 3-input OR/NOR gates and one 4-input OR/NOR gate.
6. Four *shift register cards* each with 6 stages and special reset/set logic (which can also be used as a fixed-delay one-shot multivibrator), and a modulo-2 adder all of which can be patched for computation, memory, and pseudo-random sequence generation.
7. A *sampling-type Zener-diode noise generator card* with 2 noise generators.
8. Six BCD *decade counter cards* with associated *in-line readout.*

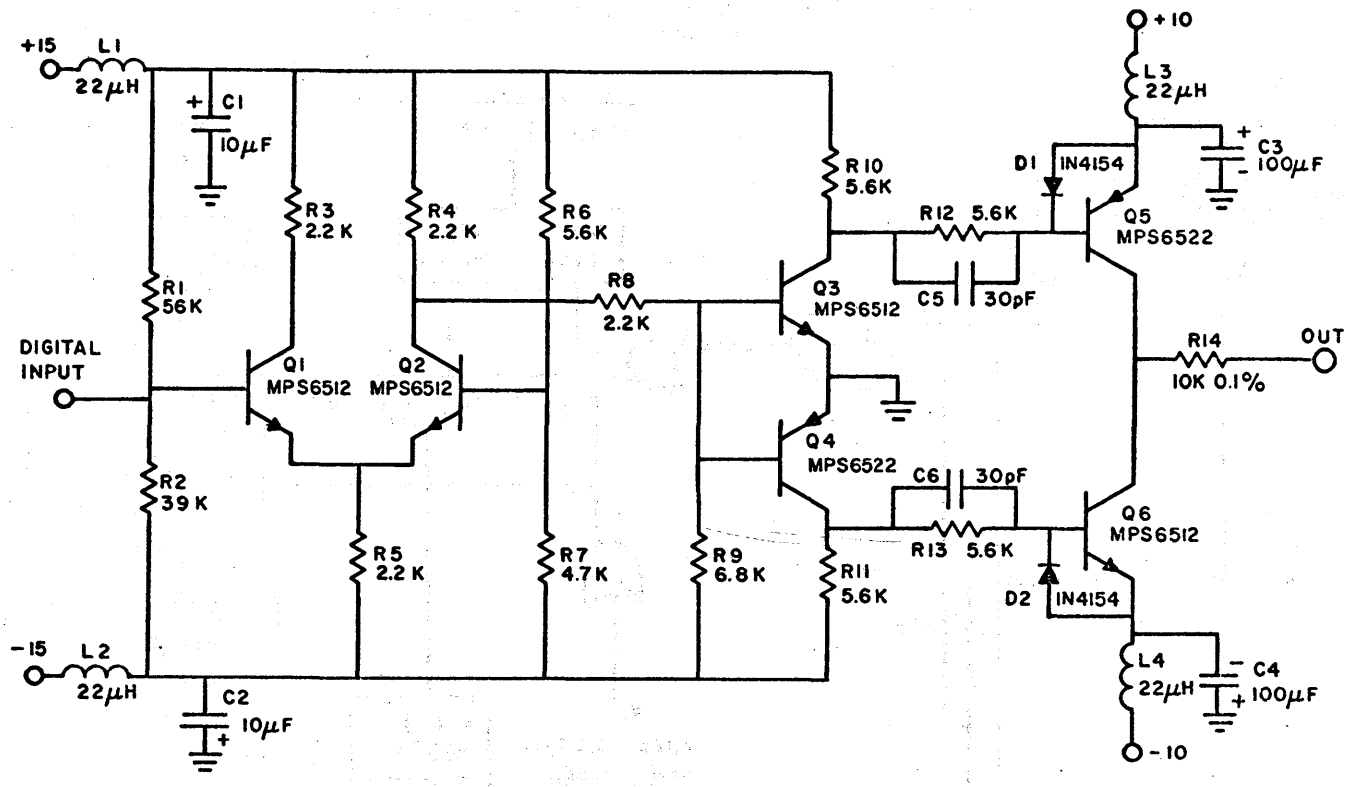
To utilize binary noise on the analog portion of the computer, one must increase the voltage swing and provide accurate output levels. This is accomplished by a driver circuit (Figure 15) which is located in the analog patchbay and switches between the plus and minus 10-volt computer reference voltages under digital control. The output contains a summing resistor which patched to the summing junction of an amplifier; this technique protects the output transistors from an accidental short to ground while the output is being patched.

Multiplying D/A converter and analog-to-digital converters[16]

An interface element between LOCUST and the Analog/Hybrid Computer Laboratory's PDP–9 digital computer will be a 12-bit multiplying D/A converter.

The four LOCUST A/D converters (Figure 16) are of a novel type utilizing a ramp-type voltage-to-time converter implemented with LOCUST integrator/track-hold units patched in the analog patchpanel.[5] The output is an 11-bit 2's complement-code digital word.

Figure 15—"Noise driver" or one-bit D/A converter switches between —10V and–10V under logic control



A conversion begins when sampling signals, patched on the digital patchpanel, switch the switched-integrator from RESET or INTEGRATE and also gates an 8–MHz clock into an 11-bit binary counter which has been preset to 100—0. An output flag signals the end of a conversion which takes at most about 250 μs. While each individual conversion is relatively slow, this inexpensive system requires no multiplexing and becomes very advantageous if many A/D channels are needed.

Digital outputs are at the DEC logic levels (0,—3 volts) for interfacing with the PDP–9 digital computer.

LOCUST/PDP-9 interface

The LOCUST/PDP–9 analog-digital computer interface hardware will consist of the following units in addition to the D/A multipliers and the A/D converters mentioned above:

1. Sense lines (flags) to sense logic states in the analog computer.
2. PDP–9 interrupt lines which can change the digital computer program in response to LOCUST logic signals.
3l A jam-transfer control register from the PDP–9 to set up logic levels in LOCUST for control purposes (analog program changes, etc).
4. Counters preset under control of the PDP–9 to act as C_1, C_2, and C_3, counters so that LOCUST sampling times and sample sizes can be controlled by the PDP–9.

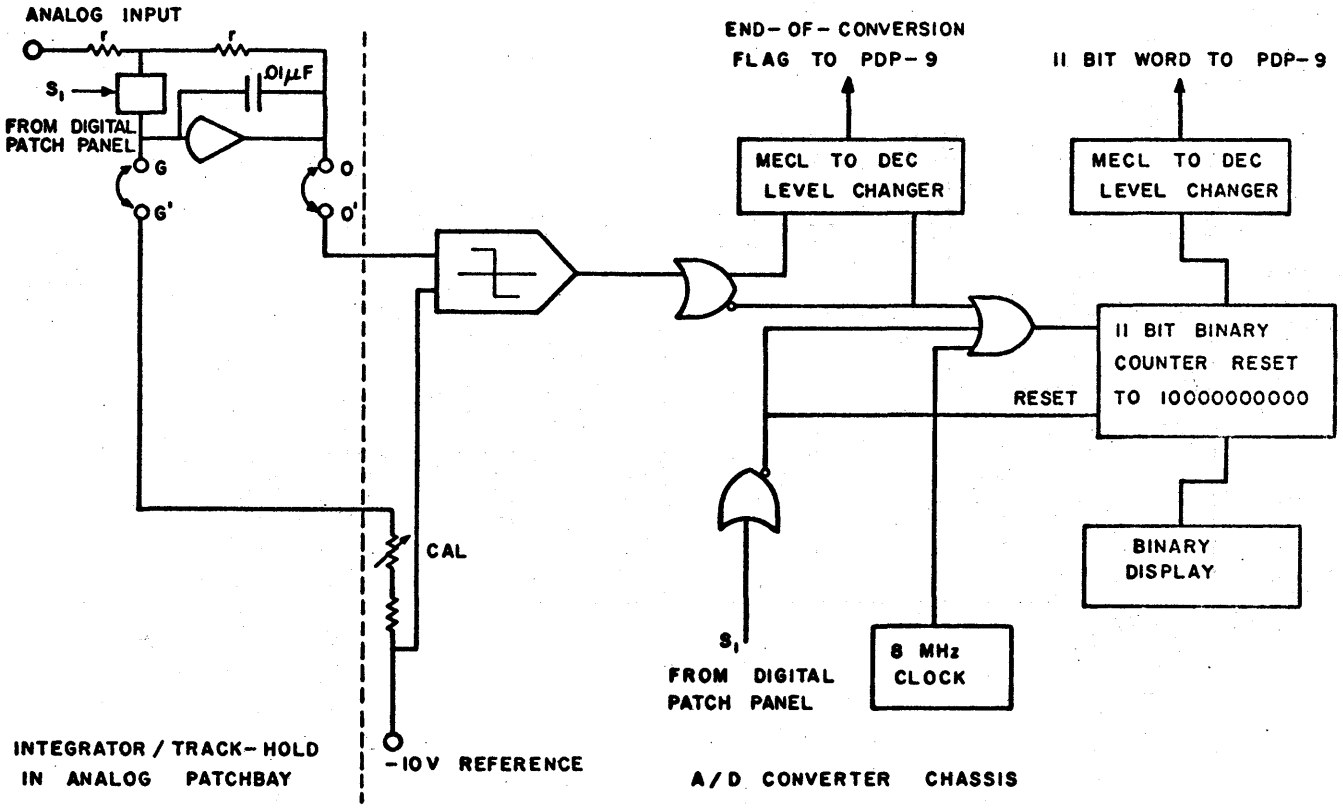
ACKNOWLEDGMENTS

The hybrid computer described in this paper is part of the hybrid analog-digital studies at the University of Arizona directed by Professor Granino A. Korn. The writer is grateful to Professor Korn for his suggestion of the topic and his encouragement and guidance during the development of the new computer.

The writer is grateful to the following individuals and organizations:

The National Science Foundation (Grant GY-379

Figure 16—Each LOCUST analog-to-digital converter comprises only simple ramp-comparator/timing circuits patched to an ordinary LOCUST integrator, which serves both as an input sample-hold and a ramp generating integrator. 11-bit conversion takes about 250 μsec, but no multiplexing (time sharing) is needed



and Institutional Grant to the University of Arizona);
The National Aeronautics and Space Administration (Grant NsG-646);
Drs. David L. Patrick and Edward N. Wise, the University of Arizona Coordinators of Research (for allocating institutional grant funds);
Dr. George W. Howard, Director, The University of Arizona, Engineering Experiment Station; and Dr. Howard S. Coleman, Dean of Engineering, and Dr. Roy H. Mattson, Head of Electrical Engineering Department, for their contribution of University funds and facilities.

REFERENCES

1 B K CONANT
Modern digital control for a small iterative analog computer
The APE II Digital Module
Annales AICA pp 19–23 January 1968

2 B K CONANT P IDE G A KORN W J LILLIS J V WAIT
A simple integrated-circuit comparator for ±10V hybrid computer systems
ACL Memo 130 Department of Electrical Engineering University of Arizona 1967

3 H R ECKES and G A KORN
Digital program control for iterative differential analyzers
Simulation pp 33–41 February 1964

4 H R ECKES
Design test and application of a high-speed iterative differentials analyzer
PhD Dissertation Department of Electrical Engineering University of Arizona 1967

5 J R GOLTZ
A new low-cost analog to digital converter for hybrid computer
Special Course Project Report Department of Electrical Engineering University of Arizona 1968

6 H HANDLER
High-speed Monte Carlo technique for hybrid-computer Solution of Partial Differential Equations
PhD Dissertation Department of Electrical Engineering University of Arizona 1967

7 H KOHNE W D LITTLE A C SOUDAK
An economical multichannel noise sourse
Simulation pp 303–307 November 1965

8 G A KORN T M KORN
Electronic analog and hybrid computers
McGraw-Hill Book Company New York 1964

9 G A KORN
Progress of analog-hybrid computation
Proc IEEE pp 1835–1849 December 1966

10 ———.
Random-process simulation and measurements,
McGraw-Hill Book Company, New York 1966b

11 ———.
APE II—A modern analog/hybrid computer for laboratory instruction simulation
pp 97–104 February 1967

12 W D LITTLE
Hybrid computer solutions of partial differential equations by the Monto Carlo method
PhD Dissertation University of British Columbia British Columbia Canada 1965

13 R A MAYBACH
The solution of optimal control problems on an iterative hybrid computer
PhD Dissertation Department of Electrical Engineering University of Arizona 1966

14 J R NAYLOR
A new high performance computer D-C amplifier
M S Thesis Department of Electrical Engineering University of Arizona 1967

15 E P O'GRADY
Correlation method for computing sensitivity functions on a high-speed iterative analog computer
IEEE Tronsactions on Electronic Computers
pp 140–146 April 1967

16 ———.
Design test and application of a hybrid computer Interface
PhD Dissertation Department of Electrical Engineering University of Arizona 1968

17 C P PRACHT
A new digital attenuator system for hybrid computers
MS Thesis Department of Electrical Engineering University of Arizona 1967

18 L SCHICK
A simplified dual-slope digital voltmeter
MS Thesis Department of Electrical Engineering University of Arizona 1968

19 R H WHIGHAM
A fast 10-volt quarter-square multipiler
Simulation pp 114–120 August 1965a

20 ———.
ASTRACII plug-in diode function generator
ACL Memo 101 Department of Electrical Engineering University of Arizona 1965b

A general method for programming synchronous logic in analog computation

by E. G. GILBERT and
R. A. MORAN

University of Michigan
Ann Arbor, Michigan

INTRODUCTION

Modern analog computers are complex machines having many subsystems which contribute to their problem solving capability. Some of these subsystems may be peculiar to specialized computing tasks such as interfacing of the analog computer with a digital computer or the automated setup of problem coefficients. Whatever the configuration of the computer system, two subsystems are always included: the analog subsystem, which includes all those computing elements which process analog signals, and the logic subsystem, which includes all those elements which process binary control signals. Typically the analog elements are programmed to solve differential equations and hence provide a simulation of a dynamical system. The logic elements may represent the simulation of control hardware in the dynamical system (e.g., reaction jet control logic) or they may provide flexible general purpose computer control. Examples of this later function include: automated computer scaling, tabulation of statistical data in random process studies, automatic plotting of families of response curves, control of high speed A to D and D to A conversion operations, implementation of parameter optimization algorithms, adjustment of terminal conditions in two point boundary value problems, etc. While systematic procedures for programming the analog subsystem are well known, logic subsystem programming techniques are often intuitive and poorly organized. This is particularly true in the programming of logic elements for general purpose computer control. For those situations where the required control functions can be described by a computational flow diagram this paper offers a straightforward programming technique.

While the equipment compliment in a logic subsystem varies from one manufacturer to another, typical elements which are generally included are: flip-flops, gates, counters, shift registers, timers, etc. These elements are interconnected by means of a patchboard system to implement the logic program.

The recent trend toward the use of synchronous circuitry in these logic systems is the result of many factors: the avoidance of programming difficulties such as racing, the elimination of noise sensitive transition triggered devices, the more effective use of integrated circuits, and the simpler intermeshing of groups of logic to perform complex functions. To some, the programming of synchronous logic elements may seem more complex than the programming of asynchronous elements. Actually, the sequential nature of operations in a synchronous system lends itself more readily to systematic programming methods. This paper develops one such method. The method has the advantage that a one-to-one correspondence is established between the flow diagram describing the desired operations and the elements in the hardware implementation. Witsenhausen[1] has also described a general programming procedure. It is felt that the present approach is simpler and more direct. It should be emphasized that there is nothing very deep or profound about the proposed method. For example it is not claimed that element utilization is as effective as might be achieved by elaborate reduction techniques. However, it is well suited to present day analog com-

puter structure and computational requirements. Moreover, the interconnection of logic elements is determined almost immediately from the computational flow diagram and program trouble shooting is straightforward.

An example problem

To provide a motivation for developing a general method consider the following problem. A dynamic system is simulated in which the values of two parameters K_1 and K_2 are provided to the system. A run is made and the value of the error E is evaluated during the run. At the end of the run a cost index J is available. By employing some arbitrary search algorithm a new set of values for K_1 and K_2 is chosen and another run is made. The process continues until the optimum values of K_1 and K_2 are found. Figure 1 depicts such a problem. By listing a sequence of events a logic flow diagram can be drawn to define the search algorithm.

Figure 2 is an example of such a logic flow diagram. The figure shows a strategy for the solution of the two-parameter optimization problem depicted in Figure 1. Operation A initializes K_1^* and K_2^* and J^*. To ease the description a very simple search algorithm has been chosen. See Figure 3. Starting from the current best parameter values K_1^* and K_2^*, trial values of K_1 and K_2 are generated. First $K_1 = K_1^* - \epsilon$ and $K_2 = K_2^* - \epsilon$ is tried. If the cost J corresponding to K_1 and K_2 is less than the cost J^* corresponding to K_1^* and K_2^*, then K_1^*, K_2^*, and J^* are updated to the values K_1, K_2, and J, i.e., a new set of current values is obtained. If the cost J fails to be improved a new trial $K_1 = K_1^* - \epsilon$ and $K_2 = K_2^* + \epsilon$ is generated (the perturbation vector is rotated by 90°). If, after four trials no improvement is obtained the search is terminated. Operation B operates the interval timer once, and cycles the analog modes through INITIAL CONDITION, OPERATE, and HOLD. Branch F routes the Point In Program (PIP) to

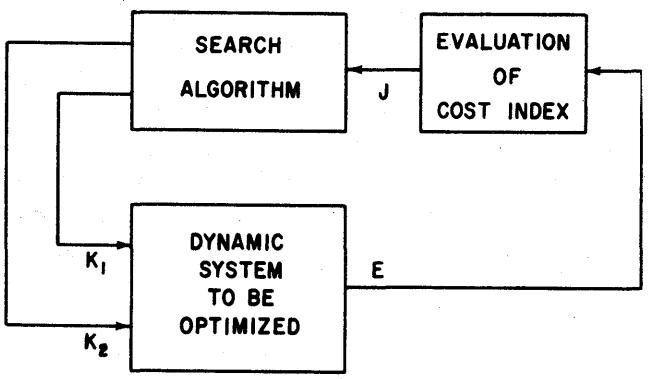


FIGURE 1—Arrangement for (two) parameter opitmization of dynamic system

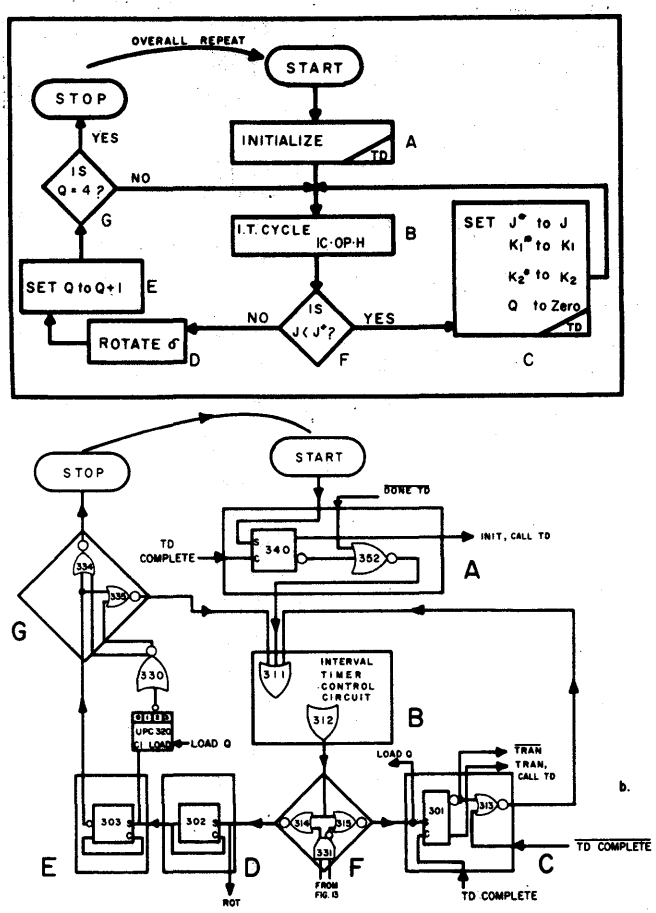


FIGURE 2—Example problem

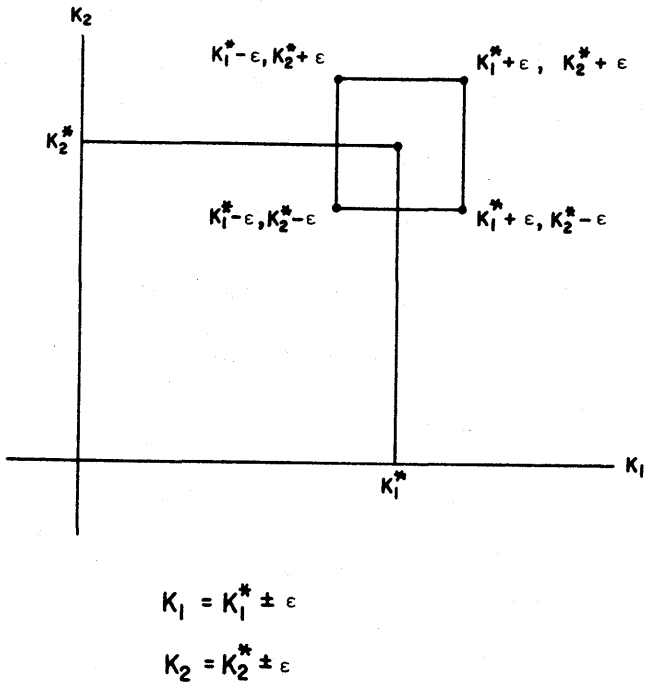


FIGURE 3—Search algorithm

Operation C if there has been an improvement. Operation C stores the improved value of K_1, K_2, and J and sets Q, the number of consecutive failures to improve, to zero. The PIP then returns to operation B. If there is no improvement, branch F routes the PIP to operation D. Operation D rotates the search perturbation in the K_1–K_2 plane. Operation E increments the Failure to Improve Count, Q by one. Branch G routes the PIP back to operation B, unless there have been 4 consecutive failures, whereupon the PIP exits to STOP.

The bottom of the figure is a direct one-to-one patching diagram which will mechanize the logic flow diagram. It was prepared using the following method.

The general programming method

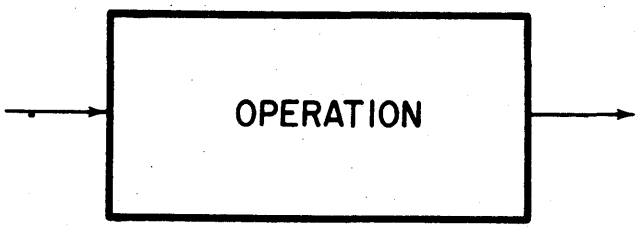
The first step is the preparation of a flow diagram. The flow diagram considered here consist of an interconnection of only two types of blocks, *operation* and *branch*. These are indicated schematically in Figure 4.

An operation is a command to accomplish one or more tasks. In the case of multiple tasks within the same operation, it must be permissible to commence all tasks at the same instant. Operations are cascaded in the Flow Diagram as determined by the sequence in which the computing tasks must be performed. Figure 2 demonstrates how the operation and branch blocks are interconnected to describe a specific computational process (parameter optimization). This figure will be described more completely in the next section.

It should be emphasized that in such a Flow Diagram only one operation is performed at any given time. Even if there are operations which command the performance of several tasks, it is evident that there is a Point In Program (PIP) which designates which particular operation is being performed at any given time. The PIP propagates through a sequence of operations. Logical decisions correspond to routing the PIP through *branches*. Operations and branches are mechanized as follows.

A controller is patched for each operation as shown in Figure 5. The PIP arrives as a one-clock-period pulse called *begin*. Upon receiving a begin, the controller issues a *call* to the task (or tasks) associated with the operation. Upon completion of the task (or tasks), the controller issues a *done* pulse.

The begin pulse for each controller is the done



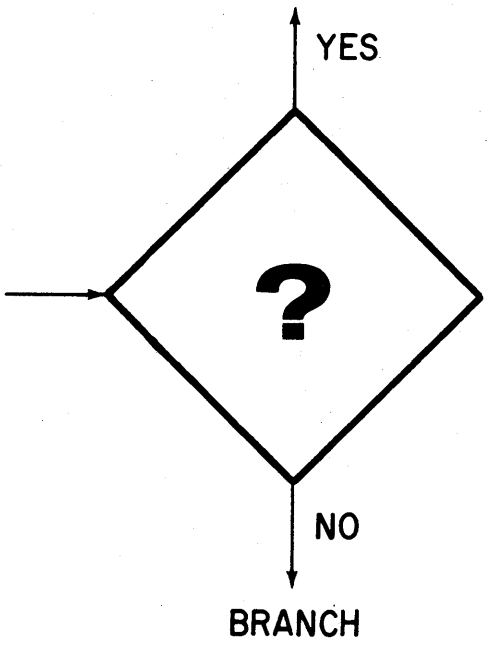


FIGURE 4—Types of blocks in computational flow diagram

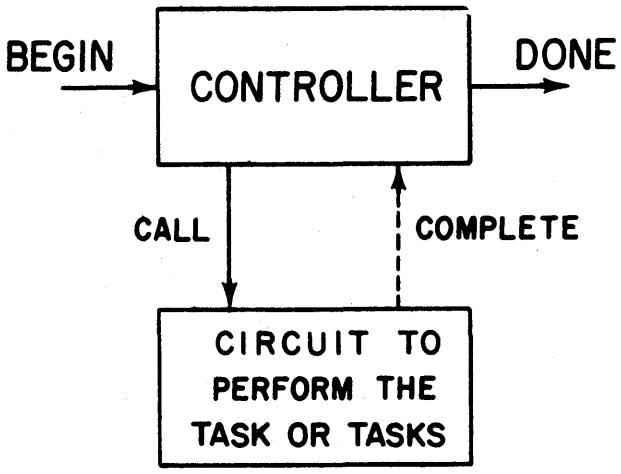


FIGURE 5—Mechanization of operation

pulse from another controller. Thus, the controllers propagate the PIP pulses and are patched in a one-to-one correspondence with the operations in the Flow Diagram. Tasks which can be performed in one clock period are called by a one-clock-period call signal and the controller issues its done pulse at the next clock period. Tasks which require two or more clock periods use a controller which depends upon receiving a *complete* signal from the circuit performing the task(s). This complete signal allows the controller to issue the done pulse.

Figure 6 shows the one-clock-period controller. It is a self-clearing flip-flop. Note that the call is issued by the begin. This insures that the task called will finish before the leading edge of the done pulse. Thus, the done pulse is issued when the task(s) is completed. The PIP is defined for timing purposes as dwelling in operation A during the time that the operation A controller flip-flop is set.

Figure 7 shows the controller for tasks requiring more than one clock period for completion.

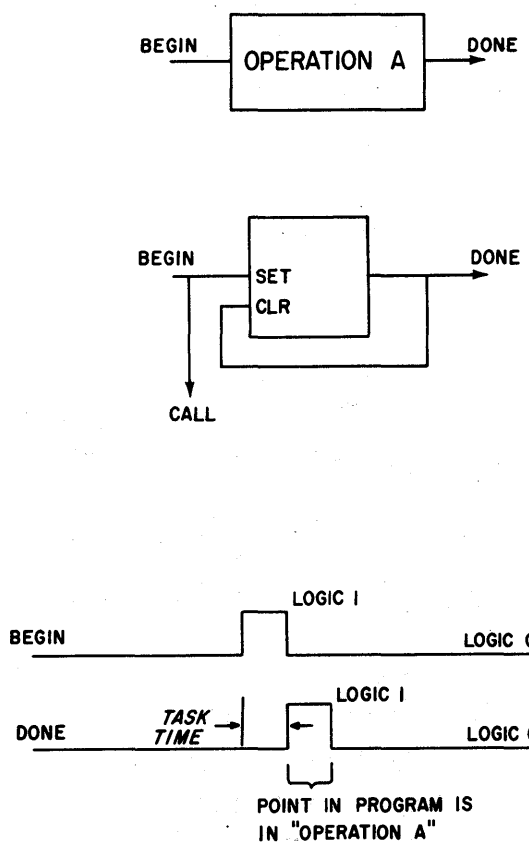


FIGURE 6—Controller for a task requiring one clock period

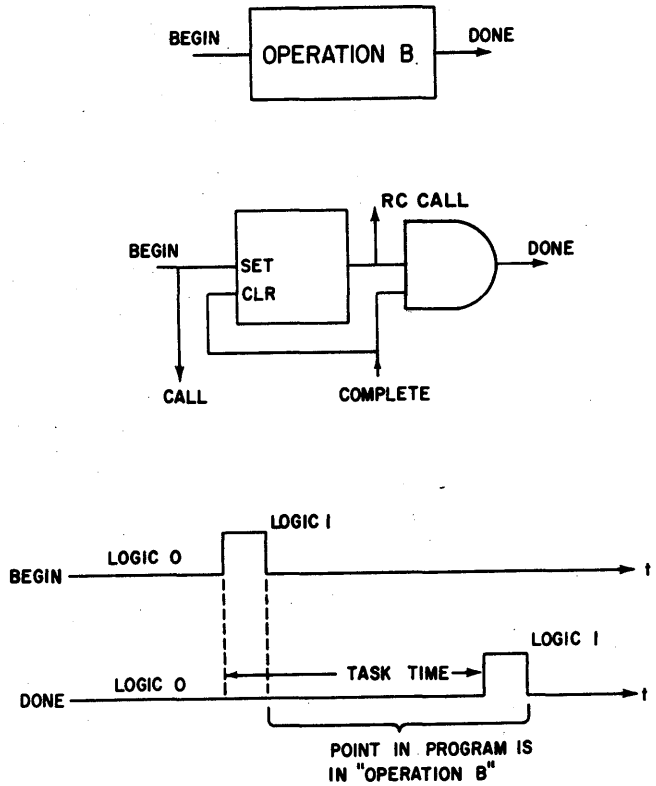


FIGURE 7—Controller for a task requiring more than one clock

This controller can issue two types of call signals. One, similar to that just discussed, lasts for one clock period and is actually the begin signal. The other, an RC call (Requires Complete), is logic 1 during the entire time that the PIP is in operation B. The RC call can be used to operate integrator control lines and perform other similar tasks which require an RC control. Notice that the controller of Figure 6 is a special case of Figure 7 in which the task time is one clock period, and an AND gate is conserved.

Figure 8a shows how the use of an OR gate allows more than one operation (i.e., controller) to call the same task or set of tasks. Thus, the hardware required to perform these tasks need be mechanized only once. If an operation has two or more tasks requiring different task times, an AND gate is needed to obtain the *complete* signal. See Figure 8b.

Figure 9 shows the circuit for a *branch*. The signal A is the answer to the question of the branch. The question is usually expressed as a statement of fact and A is a logic 1 if the statement of fact is true. Notice that the signal A is utilized only when the PIP is at the branch. The

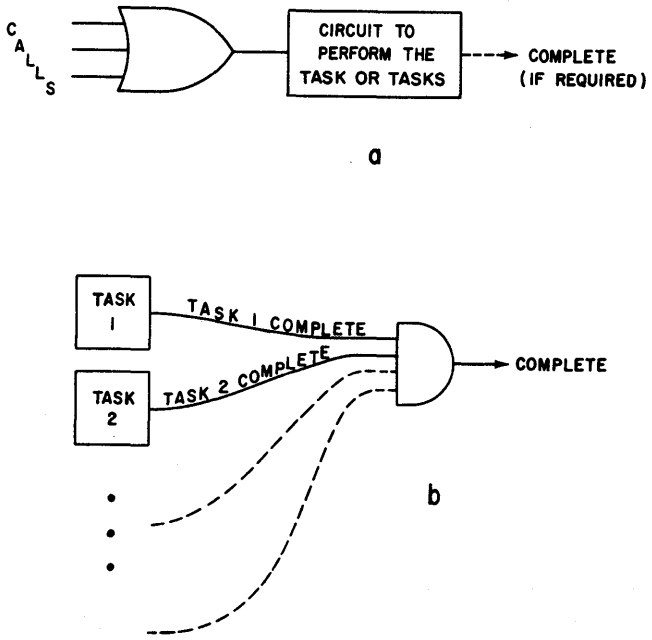


FIGURE 8—A) Circuit which allows a task to be called by more than one operation
B) Circuit for generating complete when two or more tasks requiring different task times occur in a single operation

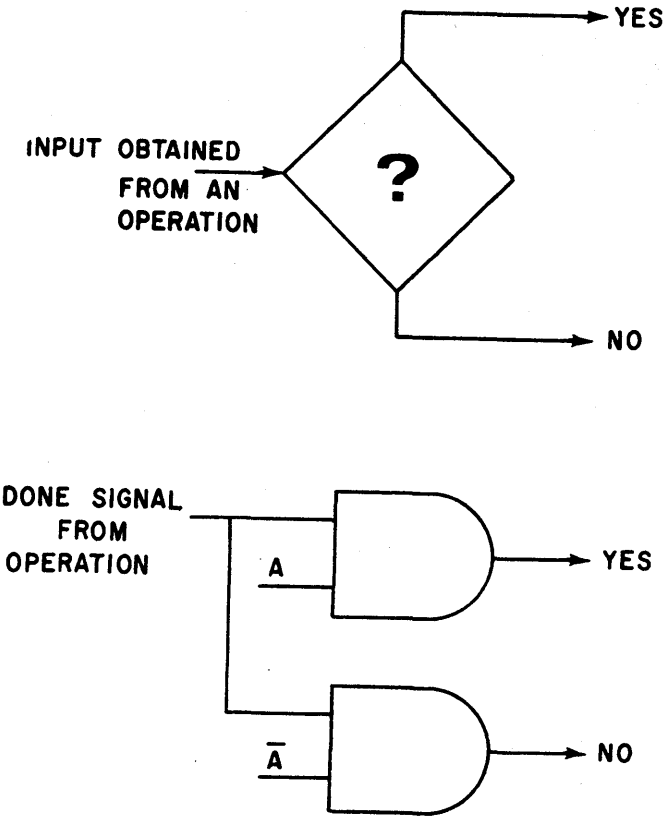


FIGURE 9—Circuit for a branch

branch in the flow diagram routes the PIP; i.e., the circuit routes a done pulse. This routing takes place in a time period corresponding to the gate propagation time and, therefore, is considered to be instantaneous.

In interconnecting the operation and branch blocks to form a complete flow diagram, it sometimes happens that a given operation may be entered from several sources, e.g., operation B in Figure 2. In this case an OR gate must be used to combine all the inputs to the operation in question.

Examples

Consider first some examples of circuits for performing tasks. Figure 10 shows typical circuits which can perform tasks in one clock period. Figures 11 and 12 show circuits which perform tasks requiring more than one clock period.

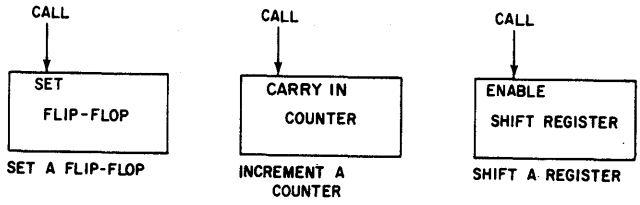
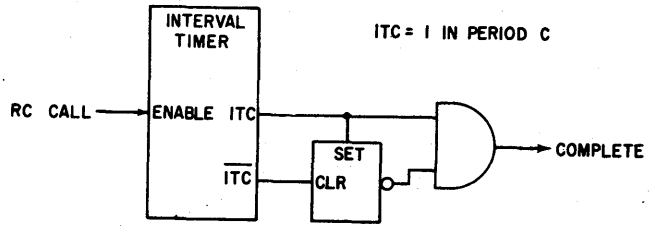


FIGURE 10—Examples of tasks requiring one clock period



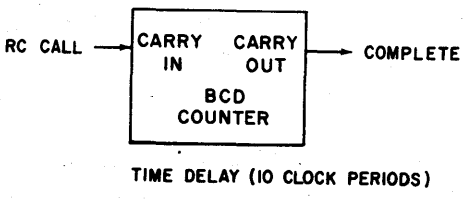


FIGURE 11—Examples of tasks requiring more than one clock period

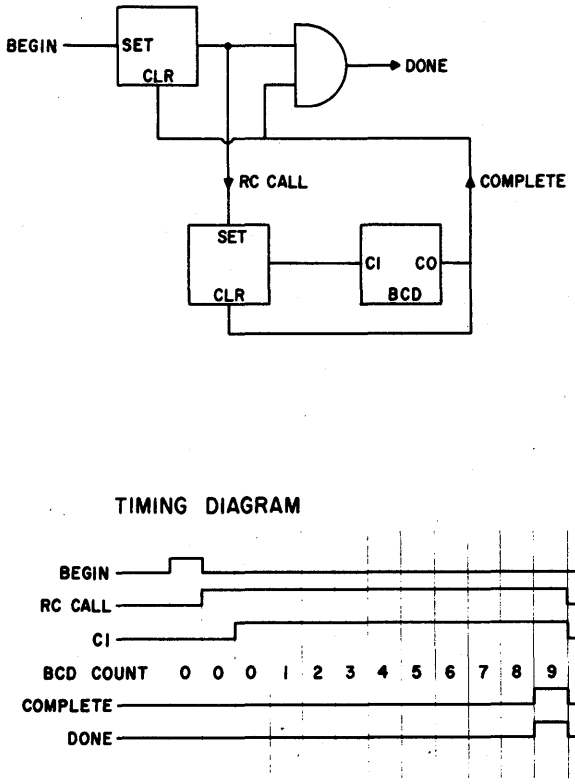


FIGURE 12—Eleven-clock-period delay

The Interval Timer Control Circuit performs the task of cycling an interval timer through three states. The interval timer has three time intervals: A, B, C. It is arranged by initialization to start at the beginning of period C. When called, the Flip-Flop sets, the timer counts through C to A to B and, upon reaching C again, generates a COMPLETE pulse and stops counting. The time delay utilizes a one decade BCD counter which is initialized to a count of zero. When the CI (carry in) signal is logic one for nine clock periods, the count becomes nine and a carry out is generated. Utilized with a controller of the type shown in Figure 7, a ten-clock-period task time is generated.

Figure 12 shows a controller and a ten-clock-period delay circuit arranged to provide an eleven-clock-period delay. Notice that the complete signal has the characteristic desired of the done pulse and could be used for the done pulse, thus eliminating the AND gate.

Program debugging with the method is quite simple. Each controller is patched together with the circuits that it controls, and it is tested independently by activating it with a one-clock period begin pulse. A check is then made that all tasks called by the controller are performed and that a one-clock-period done pulse is generated. This would normally be accomplished by means of the single step mode and inspection of state indicator lamps. If each operation and branch circuit is tested thus as it is patched, then all that remains is to carefully connect the circuits together.

Referring to Figure 2 it can be observed that the logic flow diagram at the top of the figure has been patched using the method just described.

The following is a detailed description of the remaining circuits shown in Figure 13. This specific example was prepared for use on an Applied Dynamics AD/4 computer and was run successfully on a number of problems.

Remaining circuits

Starting at the top of the figure, gate 333 OR's the calls to the time delay circuit from operations A and C (flip-flop 340 and 301 respectively). Note that these calls are RC type calls and remain a logic 1 until a complete signal is received by the controller. Thus, when gate 333 becomes a logic 1, it stays a logic 1. And when the next V 1 second pulse is received by gate 353 (some time between 1 and 999 milliseconds later) flip-flop 360 will set. Exactly one second later flip-flop 361 will set. As soon as flip-flop 361 sets, both flip-flops are commanded to clear and, through gate 353, both flip-flops are enabled. Thus, upon the next clock pulse both flip-flops will clear and the done

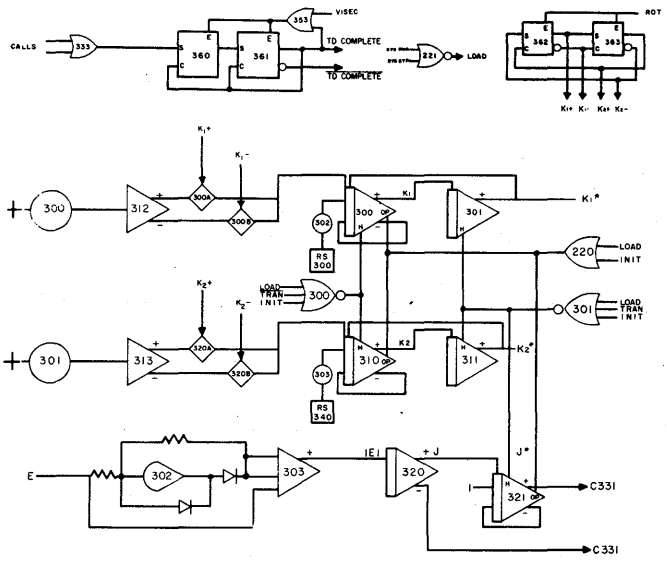


FIGURE 13—Remaining circuits

TD signal is terminated. Note that the done TD signal is self-clearing and occurs sometime between one and two seconds after it is called. This time is not precise; however, the purpose of the time delay is to provide sufficient time for the integrators to initialize, and there is no exact time required for this task. Also, note that the circuit is time-scalable since it is enabled by a V type signal. (On the AD-4 the V signals speed up according to the time scale selected for the computer X1, X10, X100, X1000). Gate 221 generates the signal LOAD which is a logic 1 when the computer logic mode is LOAD. This signal is used to initialize the over-all problem. Flip-flops 362 and 363 comprise a two-bit cross-coupled ring counter. The circuit is self-initalizing and provides 1 of 4 possible control outputs to control K_1 and K_2. Note that K_1 is either plus or minus and that K_2 is either plus or minus.

Upon receiving the call ROT, the circuit changes state. Thus, the four possible combinations of plus and minus K_1 and plus and minus K_2 are achieved sequentially by successive calls to rotate. The circuit is a minimum hardware realization for a four state device. Pot 300 adjusts the value of epsion, the step size for parameter K_1. Amplifier 312 is a bi-polar summing amplifier, only one output of which can be switched by the mutually exclusive D to A switch 300 which provides the input to the bucket brigade circuit of integrators 300 and 301. Reference switch 300 driving pot 302 provides full scale initialization for parameter K_1. The circuit beginning with pot 301 is a similar circuit for parameter K_2. K_1 and K_2 are parameters which are sent to the dynamic system under study. An error signal is continuously received from the dynamic system under study. This signal E, the input to absolute value circuit, amplifiers, 302 and 303, results in the absolute value of E being integrated by integrator 320 to generate the cost index J. J drives the track store integrator 321 which stores the value J*. Outputs of $-J$ and $+J^*$ are compared at comparator 331 to determine if the current value of J is better than the previous stored value J*. Mode control is as follows: Integrators 300, 310 and 321 assume an initial value if mode controlled to the operate mode. To do this the operate control line must be a logic 1 and the hold control line must be a logic 0. There are two times at which one wishes to do this. The first is any time the computer logic mode control is LOAD. This means the operator has not yet started the problem. The second time is any time the call INIT is issued by controller A. When the computer logic mode is LOAD, NOR gate 300 will insure a 0 command to the HOLD input of integrators 300 and 310 and NOR gate 301 will insure a 0 command to the HOLD input of integrator 321 as well as integrators 301 and 311. Also, OR gate 220 provides a logic 1 to the operate control line of integrators 300, 310, and 321. Since each of these integrators has an additional resistor feed-back, they each assume a value equal to the regular integrand input. This is arbitrary for integrator 300 and 310 as determined by pots 302 and 303. Thus, the initial values of K_1 and K_2 can be placed anywhere in the K_1, K_2 plane. The initial value of J* is set to unity and thus provides a large value of initial cost index so that successive runs have something on which to improve. The call TRAN which occurs during operation C is a call to transfer the present values of K_1, K_2, and J to update the values of K_1^*, K_2^*, and J*. Through NOR gate 301 this drops the HOLD command from integrators 301 and 311, and these integrators then respond to their patched input K_1 and K_2. At the same time NOR gate 300 is a logic 1 holding the values K_1 and K_2. When TRAN transitions back to logic 0, the stored values are again held, and K_1 and K_2 become the sum of K_1^* and epsilon and K_2^* and epsilon respectively.

REFERENCES

1 H S WITSENHAUSEN
Hybrid techniques applied to optimization problems
AFIPS Proceedings Spring Joint Computer Conference 1962
page 377

Computer experiments in the global circulation of the earth's atmosphere

by AKIRA KASAHARA

National Center for Atmospheric Research
Boulder, Colorado

INTRODUCTION

The motions of the atmosphere are governed by physical laws which are expressed in the form of the equations of hydrodynamics and thermodynamics. These equations are, in principle, well known, but some of the physical processes in the atmosphere, such as small-scale turbulence and convection, cloud formation, and precipitation mechanisms, are still difficult to express quantitatively.

Basic atmospheric equations

Let us write the basic system of atmospheric equations as follows:

$$\frac{\partial u}{\partial t} = -|V\cdot\nabla u - w\frac{\partial u}{\partial z} + fv - f'w - \frac{1}{\rho}\frac{\partial p}{\partial x} + F_x, \quad (1)$$

$$\frac{\partial v}{\partial t} = -|V\cdot\nabla v - w\frac{\partial v}{\partial z} - fu - \frac{1}{\rho}\frac{\partial p}{\partial y} + F_y, \quad (2)$$

$$\frac{\partial w}{\partial t} = -|V\cdot\nabla w - w\frac{\partial w}{\partial z} + f'u - g - \frac{1}{\rho}\frac{\partial p}{\partial z} + F_z, \quad (3)$$

$$\frac{\partial \rho}{\partial t} = -\nabla\cdot(\rho|V) - \frac{\partial(\rho w)}{\partial z}, \quad (4)$$

$$\frac{\partial s}{\partial t} = -|V\cdot\nabla s - w\frac{\partial s}{\partial z} + \frac{Q}{T}, \quad (5)$$

$$s = C_v \ln p - C_p \ln \rho, \quad (6)$$

$$T = \frac{p}{\rho R}, \quad (7)$$

where

$$|V\cdot\nabla = u\frac{\partial}{\partial x} + v\frac{\partial}{\partial y}$$

In these equations, x, y, and z are the three Cartesian space coordinates, which are directed eastward, northward, and upward, respectively. The time coordinate is denoted by t. The velocity components in the x, y, and z coordinates, respectively, are denoted by u, v, and w. The horizontal velocity $|V$ denotes ui + vj, where i and j are unit vectors in the x and y coordinates, respectively. ρ represents density, p pressure, s specific entropy, and T temperature. $f = 2\Omega \sin\varphi$ and $f' = 2\Omega\cos\varphi$, where Ω denotes the angular speed of the earth's rotation, and φ denotes latitude. F_x, F_y, and F_z are the three components of the frictional force per unit mass. Q is the rate of heating per unit mass. C_p and C_v are the specific heat at constant pressure and constant volume, and R is the gas constant of dry air equal to $C_p - C_v$. g denotes the acceleration due to gravity. For simplicity the equations for predicting the water vapor field and describing the condesation process have been omitted.

In the system of equations (1) through (7), x, y, z, and t are called the *coordinate* variables. u, v, w, ρ, and s are called *prognostic* variables because there are equations for the time rate-of-change of these variables. p and T are called *diagnostic* variables because they are computed from Eqs. (6) and (7), which do not include time derivatives.

If F_x, F_y, F_z, and Q were expressed in terms of u, v, w, p, ρ, s, and T, these equations, together with the proper boundary conditions, would be complete in the sense that there are the same number of equations as of unknowns.

The usual method of solving such a system of nonlinear partial differential equations is to replace the partial derivatives by finite differences in both space and time so that the unknowns, u, v, w, ρ, p, s, and T, are defined only at the points of a four-dimensional lattice in space and time. If the variables u, v, w, ρ, and and s are known at $t = t_0$ at all grid points in space, the

right-hand sides of Eqs. (1) through (5) may be computed by appropriate space differences. The resulting point values for the local time derivatives of u, v, w, ρ, and s may then be multiplied by a suitable time increment, Δt, and added to the values of u, v, w, ρ, and s at $t = t_0$ in order to define them at the time $t_0 + \Delta t$. Repetition of this process will, in principle, yield a forecast for any later time.[1-4]

It turns out, however, that a constraint on the value of Δt exists when these equations are solved by explicit difference schemes. This constraint is known as a *computational stability condition*, and was first discussed by Courant, Friedrichs, and Lewy[5] in 1928. Roughly speaking, the time increment, Δt, must satisfy the condition

$$C_m \frac{\Delta t}{\Delta s} < 1 \tag{8}$$

where Δs represents one of the space increments, Δx, Δy, or Δz, and C_m denotes the maximum magnitude of the characteristic velocity of the system. (If implicit difference schemes are used, such a constraint may not appear. However, implicit schemes are usually not economical to use.)

The maximum characteristic speed in this system is the speed of sound, 300 m/sec. If we choose 300 km as a representative horizontal grid distance, we find that Δt is less than 1000 sec, or 15 min. However, sound waves in this system propagate not only in the horizontal direction but also in the vertical. If we choose 3 km as a representative vertical grid distance, we find that Δt must be less than 10 sec! To use such a small time step for numerical integration of a system designed to predict weather patterns for a period of ten days or so is impractical, even with modern high-speed computers. We must, therefore, consider modifications to the basic system of the atmospheric equations in order to eliminate sound waves from the system.

Modifications of basic atmospheric equations

The *hydrostatic approximation* is defined by

$$\frac{\partial p}{\partial z} = -\rho g. \tag{9}$$

If we substitute Eq. (9) into (3), we find that sound waves are filtered out, but that gravity waves undergo little change (Monin and Obukhov[6]). The maximum speed of gravity waves can be as large as that of sound waves, but the fact that gravity waves propagate only horizontally in the hydrostatic system relaxes the severe computational stability condition mentioned earlier.

The use of the hydrostatic approximation based on scale analysis can also be justified for large- to medium-scale motions in the atmosphere (as defined later). It should be borne in mind that w is no longer a prognostic variable since the equation for the time rate of change of w does not exist.

Because of the hydrostatic equation (9), one can no longer calculate $\partial\rho/\partial t$ and $\partial s/\partial t$ (or $\partial p/\partial t$) independently from Eqs. (4) and (5) [or from Eq. (5) together with (6)]. This constraint leads to a diagnostic equation for w. In other words, w is so calculated that the computations of $\partial\rho/\partial t$ from (4) and $\partial s/\partial t$ from (5) satisfy the hydrostatic condition (9) all the time. This important constraint was first noted in 1922 by Lewis F. Richardson[7] in his pioneering work, *Weather Prediction by Numerical Process*. Therefore, this diagnostic equation for w is now called the *Richardson equation*. The Eulerian hydrodynamic equations modified by the assumption of hydrostatic equilibrium are called, in meteorology, the *primitive equations*.

Richardson actually tried numerical integrations of the primitive equations using observed meteorological data available to him at that time. It is no surprise that his forecast was unsuccessful. In retrospect, explaining why Richardson's forecast failed at that time seems equivalent to describing the history of numerical weather prediction. Several major technological and theoretical breakthroughs were necessary before today's successful weather forecasts could be produced. The real value of Richardson's work is that he laid down a sound scientific scheme for weather prediction by assembling some of the diverse knowledge about various physical processes which was available in his day.

For a long time after what Richardson called his "glaring error," no one attempted to follow his objective approach to weather forecasting. I would like to quote a famous sentence from Richardson's book: "Perhaps some day in the dim future it will be possible to advance the computations faster than the weather advances and at a cost less than the savings to mankind due to the information gained. But that is a dream." (The reader may refer to an interesting review of Richardson's book by Platzman.[8])

In the mid-1940s, John von Neumann, of the Institute of Advanced Studies at Princeton, began to build an electronic computer, primarily for the purpose of weather prediction. At the Institute, a meteorological research group was established to attack the problem of numerical weather prediction through a step-by-step investigation of models designed to approximate more and more closely the real properties of the atmosphere. In 1950, results of the first successful numerical prediction were published by Charney, Fjørtoft, and von Neu-

mann.[9] Their atmospheric model was not based upon the primitive equations, but was rather a *quasi-geostrophic model*, in which even gravity waves are filtered out. I shall explain the quasi-geostrophic model later.

Scales of atmospheric motion

So far, I have not been specific about the scale of atmospheric motions with which we are concerned. As a matter of fact, in the days of Richardson, meteorologists did not have a clear idea about the scale of weather systems and the principal physical processes involved in a particular scale of motions. Perhaps this is why Richardson was reluctant to make even minor approximations and why he considered practically all the physical processes involved, including the effects of condensation nuclei, evaporation from soil and transpiration from plants, even for a one-day forecast. Recent successes in numerical weather prediction have depended upon recognition of the different scales of motion which govern weather systems, and upon simplification of the Navier-Stokes hydrodynamic equations by mathematical approximations in order to describe motions of a particular scale. Table I shows the scales of motion associated with different atmospheric phenomena.

Table I- Scales of motion associated with typical atmospheric phenomena

Classification	Scale (km)	Typical Phenomena
large scale	10,000	very long waves
	5,000	cyclones and anticyclones
medium scale	1,000	frontal cyclones
	500	tropical cyclones
mesoscale	100	local severe storms, squall lines
	50	hailstorms, thunderstorms
small scale	10	cumulonimbi
	5	cumuli
	1	tornados, waterspouts
microscale	0.5	dust devils, thermals
	0.1	
	0.05	transport of the heat, momentum,
	0.01	and water vapor

Motions of a scale greater than a few thousand kilometers are known as large-scale motions. Figure 1 shows a pressure map of the Northern Hemisphere for a particular winter day. The lines are isobars drawn at 5mb intervals at a height of 6 km. The symbols H and L denotes the high and low pressure areas, called *cyclones* (or *troughs*) and *anticyclones* (or *ridges*), depending upon

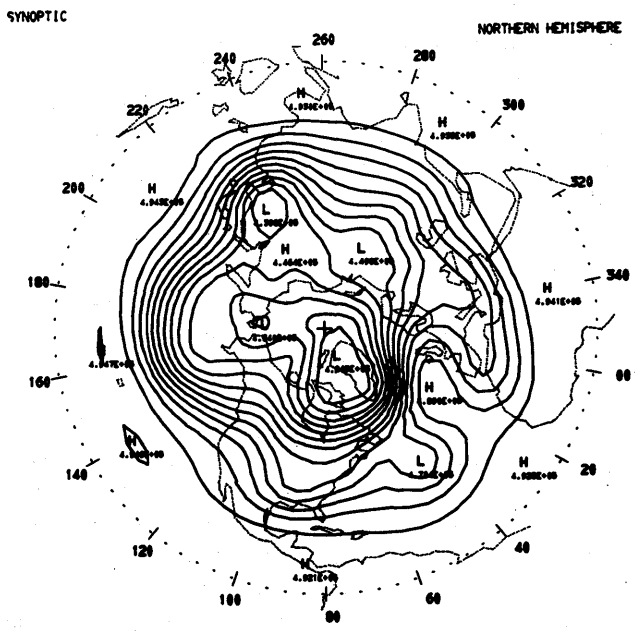


FIGURE 1—Northern Hemisphere map of pressure at 6 km. Contour interval is 5 mb

the configuration of the patterns. For this scale of motion, the relation between the pressure field and the field of motion is given, to a first approximation, by the *geostrophic wind* law. This law states that the air moves parallel to the isobars, clockwise around a high pressure area and counterclockwise around a low pressure area in the Northern Hemisphere, and in the opposite direction in the Southern Hemisphere. The *pressure force* is, of course, directed at right angles to the isobars, from higher to lower pressure. In the mid-troposphere, friction may be neglected, and the pressure force is ordinarily balanced by the *Coriolis force* (the deflecting force which results from the earth's rotation and which acts at right angles to the motion).

Since the pressure generally decreases from the equator to the North Pole, the wind generally blows from the west to the east. It is called *westerly* when the flow is from west to east and *easterly* when the flow is from east to west. Figure 1 shows the prevailing mid-latitude westerlies meandering around the earth. The portion of the zonal flow where the intensity is maximum is called the *jet stream*.

Large-scale meandering motions, of the type seen in Figure 1, are called *long waves*. They were discovered from the upper air analyses during World War II. The work of J. Bjerknes and C. Rossby led to the important conclusions that (1) long waves are closely related to weather systems, and (2) long waves are characterized by a state of quasi-balance between the horizontal wind and pressure fields. The latter observation gave Charney[10] and Eliassen[11] a basis for deriving the concept

of a *quasi-geostrophic approximation* to formulate a prediction model which was later used successfully by the Princeton meteorological group. This was the beginning of a new era, not only for weather forecasting, but also for meteorology in general.

In 1955, the National Meteorological Center, U.S. Weather Bureau, began to issue numerical forecasts on an operational basis by means of an electronic computer. Until recently, their forecasting model has been of the quasi-geostrophic type mentioned earlier.[12,13]

There were two major problems which dynamic meteorologists began to worry about: (1) Although numerical forecasts of the mid-tropospheric flow patterns were superior to the subjective forecasts by skilled forecasters, this was not true of the numerical forecasts of surface flow patterns. (2) Very long waves tended to move too rapidly westward in the numerical forecasts unless the magnitude of a certain parameter in the fore-models was increased beyond an acceptable range.[14]

Figure 2 shows a sea-level pressure map of the Northern Hemisphere for the same date as Figure 1. The lines are isobars, drawn at 5 mb intervals. Compared with the flow pattern at 6 km of Figure 1, the sea-level pressure map shows the presence of smaller scale motions. As shown in Table I, motions of a scale on the order of 1000 km are called medium-scale motions. Typical phenomena are *tropical cyclones* in the tropics (Figure 3) and *frontal cyclones* in middle to high latitudes.

Figure 4 shows a schematic model of frontal cyclones (after J. Bjerknes and H. Solberg[15]). Frontal cyclones

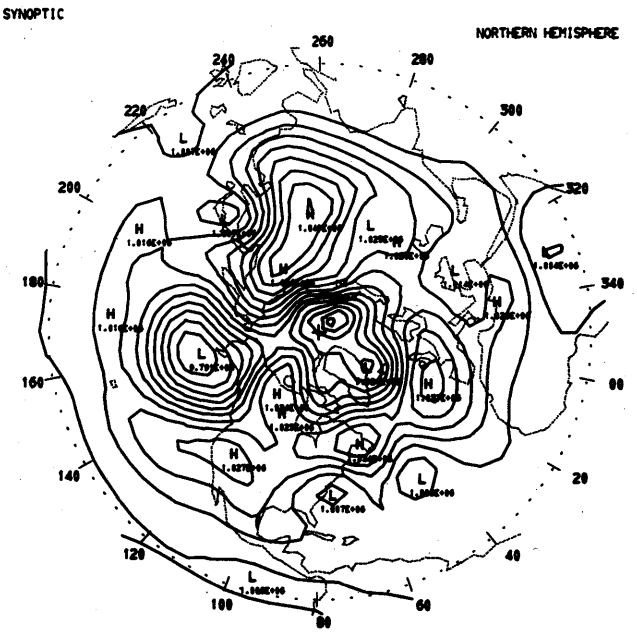


FIGURE 2—Northern Hemisphere sea-level pressure map. Contour interval is 5 mb

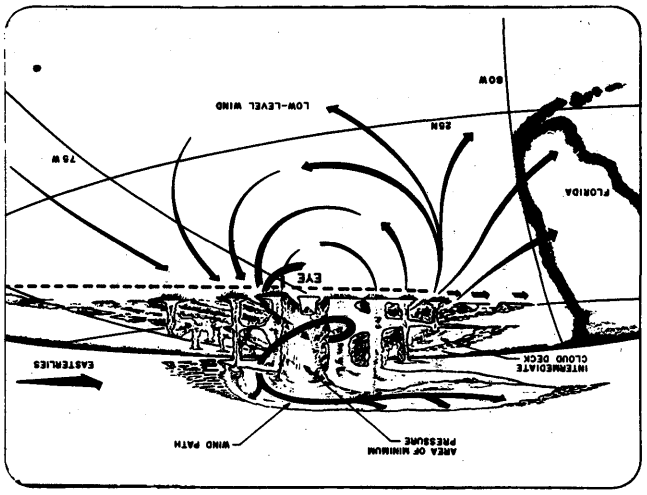


FIGURE 3—Schematic drawing of a hurricane over the Atlantic Ocean

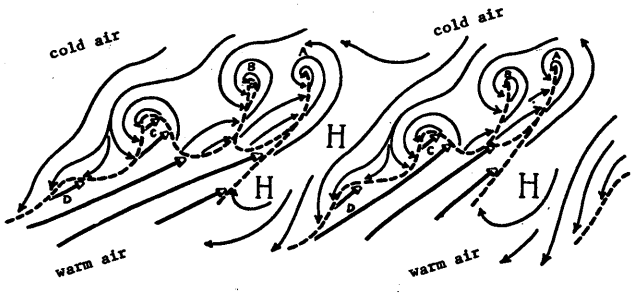


FIGURE 4—Schematic drawing of the development of frontall cyclones (after J. Bjerknes and H. Solberg)

form on fronts separating a cold air mass to the north from a warm air mass to the south. Figure 5 shows diagrammatically the relation between fronts and cyclones at sea level and the long waves aloft (after Palmén[16]).

For motions of medium scale and smaller, the deviation of wind from a geostrophic balance gradually becomes important. One cause for this deviation is surface friction. Since these motions are dominant in the lower part of the troposphere, they are influenced by conditions at the earth's surface. When a current of air moves over an uneven surface, the motion usually becomes irregular or turbulent. Because of surface friction, the kinetic energy of the flow is dissipated by eddy actions. Not only does the surface of the earth act as a sink of kinetic energy, but it also plays a role as an energy source. When cold air flows over a warm ocean, heat is transported from the ocean to the atmosphere by the turbulent actions of conduction and convection.

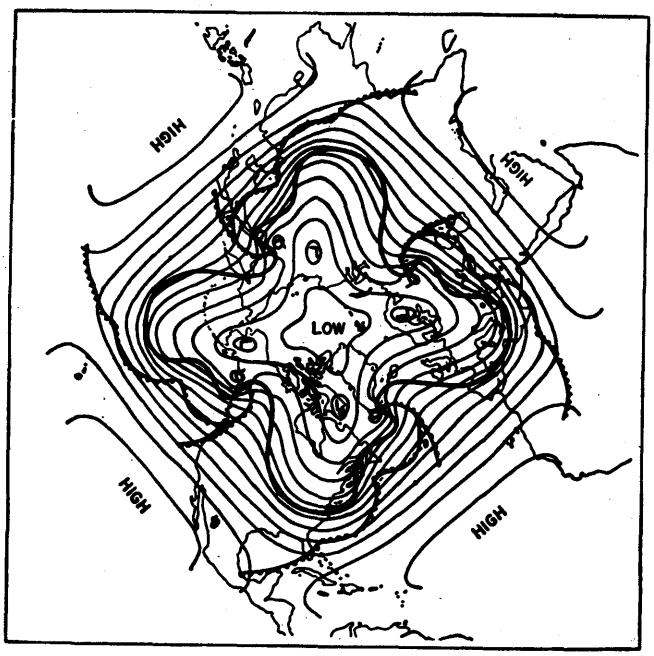


FIGURE 5—Schematic relationship between fronts and long waves (after Palmen)

Another important heat source for motions of medium scale and smaller is the latent heat released due to the condensation of water vapor. This heat source is particularly important for the development of tropical cyclones, hurricanes, and typhoons. *Local severe storms*, such as *hailstorms* and *thunderstorms*, are called mesoscale motions; and the latent heat released in gigantic cumulonimbi is a major energy source for such systems. It is clear that we must incorporate various heat sources and energy sinks in a forecasting model of flow patterns near the surface of the earth.

Coming back to the other difficulty, namely why the propagation of very long waves is overpredicted by quasi-geostrophic models, it was later found that there are two types of geostrophic motion in the earth's atmosphere.[17] As noted earlier, motions on the order of 5000 km can ordinarily be in a state of geostrophic balance. This is also true in the case of *very long waves*; but because of the large scale, variation of the Coriolis parameter with latitude becomes very important. The upshot was that quasi-geostrophic models were inaccurate in describing the motions of very long waves and that the spherical geometry of the earth must be taken into account. This led to the conclusion that the primitive equations on spherical coordinates must be used in predicting the global circulation system days in advance. Thus, we have returned to the position of L. F. Richardson a half century ago.

General circulation models

With the advent of high-speed electronic computers and the development of numerical weather prediction techniques, dynamic meteorologists have begun looking into the fundamental problem of the general circulation of the earth's atmosphere. Phillips[18] made the first successful attempt to simulate large-scale motions through a step-by-step integration of quasi-geostrophic equations. Despite many simplifications employed in setting up the model, the experiment demonstrated that certain gross properties of atmospheric motion, including the energy-transformation processes (which will be discussed later), could be numerically simulated by the model. Later Smagorinsky[19] performed a similar experiment using the primitive equations.

Following these earlier attempts, more elaborate types of general circulation experiments using the primitive equations were designed and carried out.[20–23] In 1964, when Kasahara and Washington decided to work on a numerical experiment of the general circulation of the atmosphere at the National Center for Atmospheric Research (NCAR), we made a survey of the characteristics of various general circulation models thus far reported, and we found that height as the vertical coordinate (rather than pressure, as customarily done) would be convenient in formulating a prediction model similar to Richardson's.[7] Besides the different vertical coordinate, the NCAR model includes various physical processes in the atmosphere which have been formulated somewhat differently from those incorporated in other general circulation models.[24]

Let us now consider the basic physical processes involved in the mechanism of the general circulation. All energy of atmospheric motion is ultimately derived from incoming solar radiation. If directly reflected radiation is ignored, the atmosphere receives from the sun 2.0 cal cm^{-2} min^{-1}, which is known as the *solar constant*. This gives a mean flux of solar energy perpendicular to the earth's surface of about 0.5 cal cm^{-2} min^{-1}. (The factor 4 is the ratio of surface area to cross-section for a sphere.) Of this, approximately 35 percent is reflected from the atmosphere (including clouds) and portions of the earth's surface. This percentage of energy reflected back to space is called the *albedo* of the earth as a whole. Approximately 20 percent is absorbed in the atmosphere, and 45 percent is absorbed at the surface. Since the albedo of the eath is 0.35, an average of 0.65×0.5 or about 0.3 cal cm^{-2} min^{-1} is available for direct or indirect heating of the earth and the atmosphere.[25]

Since the mean temperature of the atmosphere and the earth do not change appreciably from one year to the next, the amount of energy received by the earth

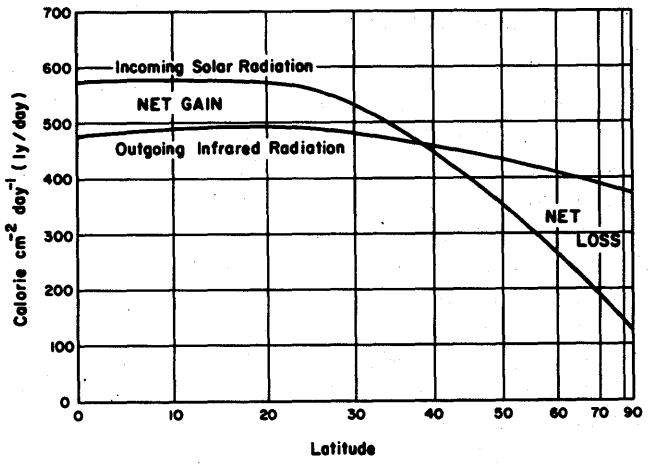


FIGURE 6—Balance between incoming solar radiation and outgoing infrared radiation from the atmosphere—earth system

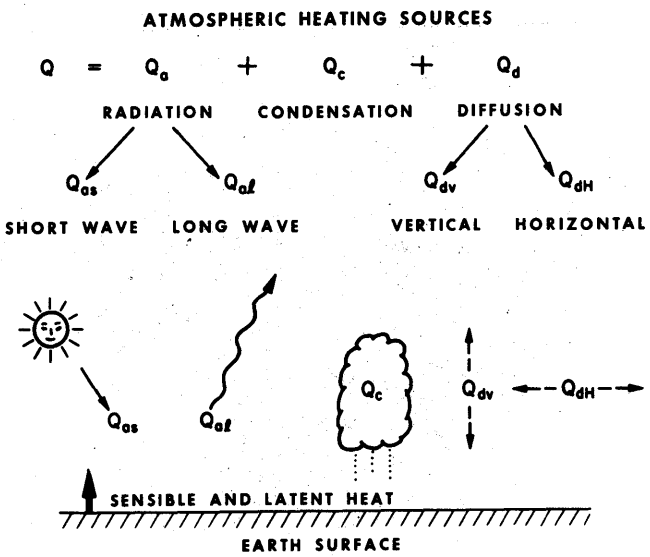


FIGURE 7—Various sources of heating in the atmosphere

must be sent back to outer space. This ultimate return of energy to space is in the form of low-temperature terrestrial radiation. Figure 6 shows the two curves. One represents the annual mean of solar radiation absorbed by the earth and the atmosphere, and the other the outgoing terrestrial radiation from the atmosphere, both as functions of latitude.[26,27]

At low latitudes the earth-atmosphere system gains more heat energy per unit area by the absorption of short wave radiation from the sun than it loses to space by the emission of long-wave radiation; the reverse happens at high latitudes. The net radiation gained by the earth-atmosphere system at the equator is, for example, about 100 ly/day,* while the net loss at the North Pole is about 260 ly/day. At a latitude of 38°, the incoming and outgoing radiation are balanced.

It is clear that the surplus heat energy in the tropics must somehow be carried to the poles. Otherwise, tropical regions would become steadily hotter and polar regions steadily colder. About ten percent of the surplus heat is transported poleward by ocean currents. The remainder is transported by the atmospheric circulation, including both the large-scale eddy motions (which are cyclones and anticyclones) and the mean meridional circulations.

Since atmospheric motions are generated by non-uniform heating of the air, a physical prescription is needed for the heating rate, Q, in the thermodynamic equation (5). In general, the following three processes (expressed as rates) are important in the atmosphere (Figure 9): heating from radiational sources, denoted by Q_a; the release of latent heat by condensation, denoted by Q_c; and heating/cooling due to eddy diffusion, denoted by Q_d.

The rate of heating due to radiational sources, Q_a, may be divided into two parts: one is the rate of heating due to absorption of solar insolation in the atmosphere, Q_{as}, and the other is the rate of heating/cooling due to infrared radiation, $Q_{a\ell}$. These terms can be evaluated by integrating the transfer equations, taking into account the distributions of water vapor, carbon dioxide, and ozone, the major absorption gases in the atmosphere.

However, the amounts of Q_{as} and $Q_{a\ell}$ are considerably altered by the presence of clouds in the atmosphere. The evaluation of Q_c, the rate of released latent heat of condensation, is, of course, directly dependent on the prediction of cloud formation. Thus, the prediction of clouds is one of the important aspects of a general circulation model.

The moisture field of the atmosphere is continually replenished by evaporation from the earth's surface. Since heat is required to evaporate water, and heat is released by condensation of water vapor, the evaporation process transports latent heat from the earth's surface to the atmosphere. Similarly, sensible heat is transported from the surface to the atmosphere, and vice versa, by the actions of small-scale turbulence and convection.

The direction of energy flow, in the form of latent heat and sensible heat, is determined primarily by the difference between the temperature of the earth's surface and the air temperature directly above it. Thus, it is necessary to calculate the temperature of the earth's surface.

The temperature of the earth's surface is determined

*ly is an abbreviation for langley, a unit equal to one gram calorie per square centimeter.

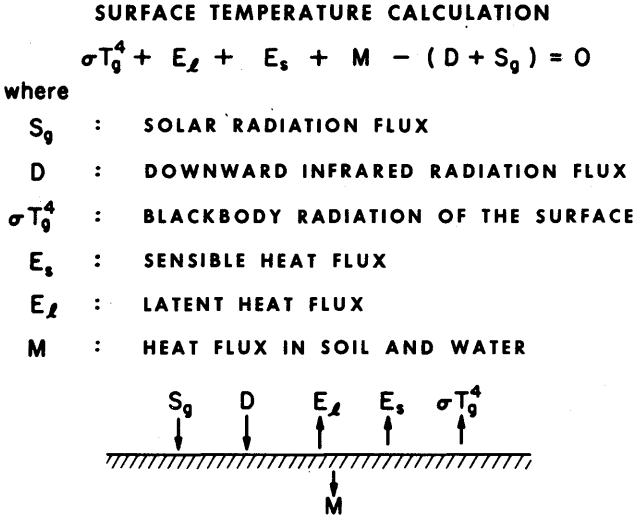


FIGURE 8—Balance requirement of heat flux at the earth's surface for calculation of surface temperature

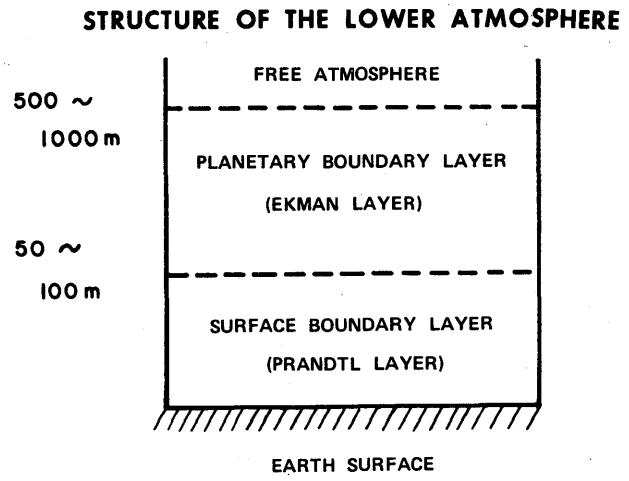


FIGURE 9—Schematic structure of the lower atmosphere (after Sutton)

by the balance of energy flux at the earth's surface, as shown in Figure 8. The surface of the earth absorbs solar radiation, S_g, and downward infrared radiation, D, from the atmosphere. A portion of the absorbed energy is transported down into the ground by heat conduction in the soils or is carried away by the motion of water in the oceans. This quantity is denoted by M.

Energy is also carried away in the form of sensible heat, E_s, and of latent heat, E_ℓ. The magnitudes of E_s and E_ℓ are dependent on both the thermal and dynamical conditions of the atmosphere and the properties of soil and water.

The remainder of the absorbed energy is re-radiated to space as infrared radiation from the earth's surface. This amount is expressed by σT_g^4, where σ is the Stefan-Boltzmann constant and T_g is the temperature of the earth's surface which we wish to determine.

The evaluation of M requires knowledge of the mechanism of heat and moisture transports in the *lithosphere*—and also of the circulation of water in the *hydrosphere*. Oceanic circulation is important in determining M as well as E_s. This is why there is a growing awareness of the role of the ocean's circulation in meteorology.

Perhaps equally important to meteorology is the role of soil. Knowledge of soil physics, particularly the transport processes of heat and moisture, will be essential when we try to explain, for example, the formation of deserts in connection with the long-term effects of the atmospheric circulation upon the earth's surface.

It is clear that what happens in the first hundred meters above the earth's surface is very important in accounting for the sources and sinks of energy for the atmosphere. *Micrometeorology* is a branch of meteorology dealing with various phenomena in the earth's boundary layer. Figure 9 shows a schematic structure of the lower atmosphere (after Sutton[28]).

In the surface boundary layer, which extends up to 50-100 m above the surface, shearing stress of wind and heat flux are both constant with height, and the structure of the wind is primarily determined by the characteristics of the surface and the vertical gradient of temperature. This region is referred to as the *Prandtl* layer.

In the planetary boundary layer, which extends approximately 500-1000 m above the ground, the structure of the wind is influenced not only by the pressure gradient, density stratification, and surface friction, but also by the earth's rotation. This region is referred to as the *Ekman* layer.

After the heating rate, Q, and the frictional terms, F_x and F_y, are suitably expressed in terms of dependent variables, the primitive equations can be integrated from prescribed initial conditions under well-posed boundary conditions.

Computational aspects of six-layer NCAR model

Now I would like to discuss some computational aspects of our problem, specifically the six-layer version of the NCAR general circulation model. The model is global. Horizontal grid points are placed at 5 degree longitude and latitude intersections. However, the grid system is staggered in time, as shown in Figure 10. The cross and dot points are at different time levels of one time increment, Δt.

Figure 11 shows a vertical cross-section of the model.

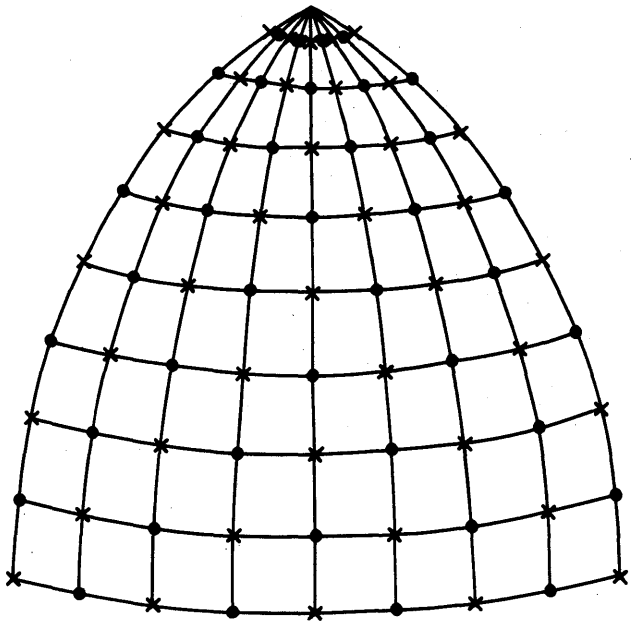


FIGURE 10—Horizontal grid system of a general circulation model

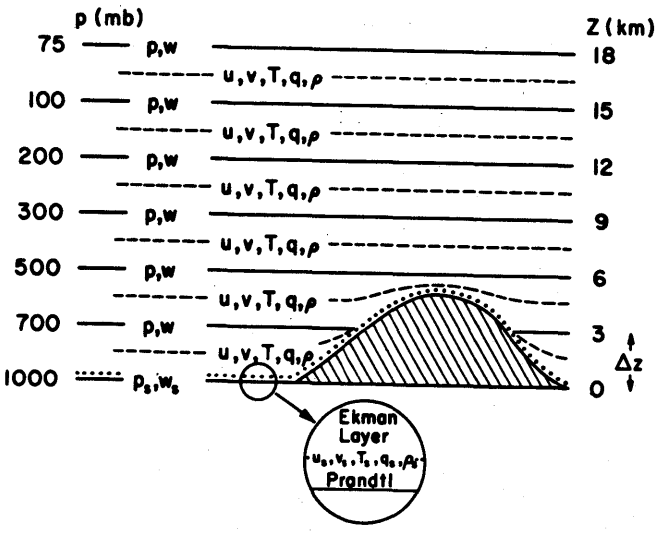


FIGURE 11—Six-layer NCAR general circulation model with orography

The height increment, Δz, is chosen to be 3 km. The top is located at $z = z_T$ ($= 6\ \Delta z$), which corresponds approximately to the 75 mb pressure level. It is assumed that vertical motion vanishes at the top in order to conserve the mass of the model atmosphere. The lower boundary of the model is a curved surface which takes into account the earth's orography (incicated by shading).

Surface pressure, p_s, is evaluated along the mountain surface. Pressure, p, and vertical velocity, w, are placed at z = 3, 6, 9, 12, and 18 km; and wind components, u and v, temperature, T, density, p, and the mixing ratio of water vapor, q, are placed at the intermediate levels.

The effects of both the surface boundary layer and the planetary boundary layer are included in the model, as shown in the inset of Figure 11. The variables u_s, v_s, T_s, q_s, and p_s, which correspond to variables in the surface boundary, are evaluated from the lower boundary conditions.

In the model, the earth's surface is divided into three parts: oceanic regions, continents, and any region where the surface temperature is below the freezing point (0°C), be it land or ocean. The third region is, therefore, considered to be covered by either ice or snow.

The ocean surface temperature is prescribed from the distribution of observed climatological mean temperatures for the appropriate season of the year. The surface temperature of the land masses is computed by the method previously described. Over ice covered regions, an upper limit of 0°C is assumed, with the implication that any excess heat flux goes into the latent form.[21]

One unique feature of the atmospheric equations is that the time rate-of-change of the variables is about an order of magnitude less than the rest of the terms in the equations. For example, the acceleration of air in large-scale motions is about one-tenth the magnitude of the Coriolis or the horizontal pressure gradient force. This fact is in striking contrast to other hydrodynamic problems, particularly shock wave problems in which large accelerations are involved. Thus, in predicting large-scale motions, extra care must be taken to compute individual terms invloving space derivatives, since the time tendency is evaluated as the sum of individual terms in the equations.

Another special requirement for the finite difference prediction equations is that the model be run long enough to study long-term fluctuations of weather systems. To produce a 100 day forecast with a time step of 6 min requires over 20,000 iterations. This is why strongly dissipative difference schemes are not preferable to use for numerical weather prediction. Of course, any practical finite difference scheme must be both stable and accurate. For long-term numerical integration of the atmospheric equations, it is, therefore, necessary to design finite difference schemes which conserve certain properties of the original equations, such as momentum and energy.

Even though a finite difference equation is stable in the sense of the *von Neumann condition*,[29] there is no guarantee of stability if such a scheme is applied to nonlinear equations. As an example, consider the very

simple hyperbolic nonlinear equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0 .$$

When the centered scheme is applied in order to approximate both the time and space derivatives, the finite difference analog of the above equation may become unstable, even under the usual stability condition:

$$\max |u| \frac{\Delta t}{\Delta x} < 1 .$$

Since this instability typically appears in nonlinear equations, it is called *nonlinear instability.*[29,30] However, a similar kind of instability may occur even in linear equations if the coefficients are not constant. Phillips[30] showed that instability arises because the grid system cannot resolve wavelengths shorter than two grid intervals. When such wavelengths are formed by the nonlinear interaction of longer waves, the grid system incorrectly interprets them as long waves.

This type of instability is inherent in all meteorological equations, and many means to eliminate this difficulty have been proposed. Our general circulation model is based on a scheme which was originally proposed by Lax and Wendroff[32] and later modified by Richtmyer[31] who called it the *two-step Lax-Wendroff scheme.* The scheme consists of a combination of the first step, called "diffusing," and the second step, called "leapfrog." The diffusing step is strongly dissipative, but the leapfrog step is neutral in stable conditions. Because of the dissipative nature of the first step, the combined steps provide effective smoothing for shorter waves and suppression of nonlinear instability. We found, however, that the two-step method damps the solution too severely. In order to eliminate this difficulty, we apply the diffusing step less often, so that short waves are not over-damped. By experimenting, we have found it sufficient to apply the diffusing step only after every 45 steps of leapfrog.

In order to give an idea of the magnitude of computation involved in our general circulation experiments, let us consider the amount of data which must be stored in the computer. In each horizontal plane, there are $72 \times 36 = 2592$ data points for a 5 degree longitude and latitude mesh. There are seven levels, including the surface, so that the total number of data points in three dimensional space is $2592 \times 7 = 18144$. Since the model is staggered in horizontal space, only half of this number (9072) is needed at one time. There are four prognostic variables, u, v, p, and q, which are needed at two consecutive time levels. Here q denotes the mixing ratio of water vapor in the atmosphere. There are three diagnostic variables, w, ρ, and T. We need 15 to 30 extra fields of heating, friction, and similar terms for diagnostic purposes, in addition to such physical parameters as cloudiness and precipitation, depending on the complexity of the model. Thus, a total of more than two million pieces of information plus about 12,000 arithmetic and logic computer instructions must be stored in the computer memory. Of course, there is no need to store all of these data in the high-speed core memory all the time during a computation. Since computation is done in a sequential manner, only a portion of the data is needed at one time for the calculations.

At NCAR, we use a Control Data 6600 computer which has only 65,000 words of core memory. The grid point data are stored in six magnetic drums, each of which contains about one-half million words. Data are constantly read from the drums to the core and transferred from the core to the drums simultaneously with the arithmetic computations. About 150 floating operations are required at each mesh point, and it takes about 25 sec for one time step. Since a time step of 6 min is used, it takes about 90 min of computing time to produce a one-day forecast.

Because a large amount of data are involved, there is a serious problem in processing the output. The situation is in contrast to, for example, an eigenvalue calculation in which the output may be a single number after hours and hours of computation. Although it is possible to print out a large amount of numbers in a short time, it is practically impossible to digest the results and evaluate the experiment without resort to graphic analyses of numbers to display the patterns. At NCAR, the output data processing is aided by a cathode ray tube (CRT) plotter called the Data Display 80 (dd80). Using this equipment, grid point data are analyzed and graphic contour lines are then superimposed on a suitable map projection.

Even though the output is analyzed in the form of weather maps, it is still difficult to comprehend the evolution of the flow patterns in a short time. Here the idea of motion pictures is very useful. The making of computer-generated movies is relatively new. Welch,[33] for example, describes a computer movie of numerical simulation of water waves from a breaking dam. Washington, O'Lear, Takamine, and Robertson[34] discuss the techniques of making black-and-white and color computer movies of the output of the NCAR general circulation experiments. We have found the use of color very effective in illustrating simultaneously the patterns of different variables for comparsion. Color film is processed from a black-and-white original film by a multi-exposure method using filters of different

colors. Eventually, we hope to have an output device to demonstrate three-dimensional patterns. We will then be able to increase greatly the information content of a single graphical display.

Two-layer NCAR *general circulation experiment*

Now I would like to talk about the result of one experiment of our two-layer general circulation calculations. This experiment simulated flow patterns of January (Northern Hemisphere winter). The season is determined by the declination of the sun. In addition, we prescribed a climatological mean distribution of sea surface temperature for January. The calculation started from an isothermal atmosphere of 250°K. This particular value is chosen as the equilibrium temperature of the earth-atmosphere system, which is evaluated from the balance requirement of incoming solar radiation with outgoing infrared radiation. This value is lower than the average for the earth's surface, but is close to the average temperature of the atmosphere itself.

Let us now look at the color computer movie which illustrates the evolution of flow patterns. The upper figure shows the surface pressure distribution, and the lower figure shows the temperature distribution at 3 km. The blue lines delineate continents, the yellow contours are isobars at 5 mb intervals, and the red lines are isotherms at 5°C intervals.

As I explained earlier, at low latitudes the earth-atmosphere system gains more energy by the absorption of short-wave radiation from the sun than it loses to space by the emission of long-wave radiation; the reverse happens at high latitudes. Because of this radiational imbalance, the north-south temperature gradients develop. The circulation pattern in the early phase of the development is of large-scale monsoons. Large highs develop over the North American continent and over Siberia, and lows form over the warm North Atlantic and to some extent over the North Pacific Figure 12a, b). Meanwhile, the warming in the low latitudes slows down, but the cooling in the higher latitudes, particularly near the North Pole, continues. Since the map is a cylindrical equal-spaced projection, the patterns near the poles are greatly exaggerated.

These large monsoonal circulations are too inefficient to transport heat northward to make up the radiational imbalance. As a result, the north-south temperature gradients kept increasing. After approximately 15 days, the north-south temperature gradients became so large that the large-scale monsoonal circulation broke down into eddy motions. These eddies are nothing but cyclones and anticyclones, the familiar systems that appear on weather maps (Figure 13a, b).

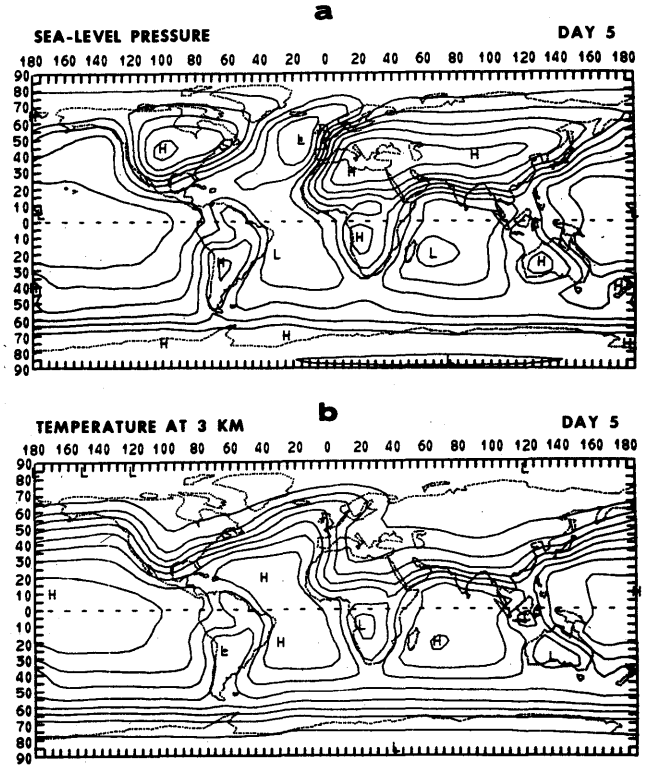


FIGURE 12—Computed distributions of (a) sea-level pressure, and (b) temperature at 3 km height after 5 days. H denotes loca maxima and L denotes local minima in pressure and temperature distributions

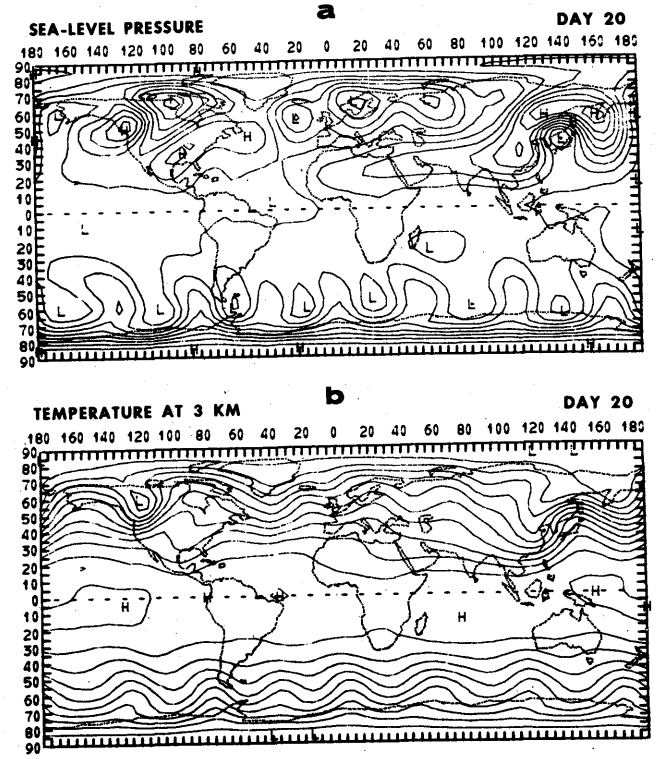


FIGURE 13—Same as Figure 12, but after 20 days

The breakdown of the monsoonal circulations into eddy motions is known in meteorology as *baroclinic instability*, and these eddies are called *baroclinic waves*. (The nature of baroclinic instability of long waves was first theoretically analyzed by Charney[35] and Eady.[36])

Mechanism of general circulation

Having noted some of the principal features of the evolution of global circulation, let us see to what extent those features can be explained. Since there is a net warming in low latitudes and a net cooling in higher latitudes in the troposphere, the pressure aloft is, in general, higher in low latitudes and lower in high latitudes. Thus, the presence of westerly flows can be anticipated in the middle latitudes, and the intensity of westerlies must increase with respect to height in the troposphere.

In addition to transporting heat from low latitudes to high latitudes, the meridional circulation and the superimposed eddy motions also transport momentum and moisture poleward. Thus, we can infer that there must be a source of momentum in the lower latitudes and a sink somewhere in the middle or high latitudes. It turns out that the frictional drag of the surface of the earth gives rise to a *source* of zonal mean westerly momentum in those latitudes with *easterly winds*, and a *sink* in those latitudes with *westerly winds*. Therefore, the presence of surface easterlies in the low latitudes can be explained from the point of view of momentum balance. In middle latitudes, surface westerlies play the role of momentum sink.

In reality, the easterlies in the low latitudes extend throughout the troposphere, though the latitudinal width is very much reduced in the upper troposphere. The presence of easterlies in the upper troposphere is difficult to explain at present. In our general circulation experiment simulating January climatology, we were able to simulate surface easterlies in the low latitudes, but upper easterlies were not present. This failure may be due to the fact that our two-layer model lacks an efficient mechanism for transporting easterly momentum upward from the earth's surface. There are many indications that the convective activites of cumulonimbi in lower latitudes provide this mechanism.

Here is an interesting point in connection with the poleward transport of momentum. In the mid-troposphere, there is a belt of easterlies in low latitudes and one of westerlies in middle latitudes. Yet westerly momentum is transported poleward. This means that momentum is transported against the north-south gradient of momentum. It is obvious, then, that the mechanism of momentum transport is not one of eddy diffusion, for we would have to assume a negative coefficient of turbulent viscosity in order to permit the counter-gradient flow.

To summarize the mechanisms of the general circulation of the atmosphere, let us look at Figure 14, which shows schematically the energy transformation. Each of three boxes represents a type of energy. The notation $\bar{I}$ indicates the zonal (east-west) average of internal energy of the atmosphere. Since the kinetic energy is a squared quantity, zonal averaging of the kinetic energy can be divided into zonal kinetic energy, $\bar{K}$, and eddy kinetic energy, K'. The zonal average of the potential energy, $\bar{P}$, is added into the $\bar{K}$ box to form the zonal mechanical energy.

The net energy source, Q, contributes heat to the air columns. As a result of differential heating, cyclones and anticyclones are maintained. Thus, energy flows from the $\bar{I}$ box to the K' box. The mechanism of energy flow is due to *baroclinic instability*. A portion of the eddy kinetic energy is lost through the action of dissipation. The eddy motions then feed momentum into the zonal motion indicated by $\bar{K} + \bar{P}$. This process of energy flow is accomplished by the counter-gradient momentum transport mentioned earlier, and is due to *barotropic stability*. Here the terms "instability" and "stability" are defined according to the responses of the eddy motions to the zonally averaged motions. If eddy motions grow at the expense of the zonally averaged motion, the resulting phenomenon is called "instability." If eddy motions feed their energy into the zonal motion, the resulting phenomenon is called "stability." The importance of the role of barotropic stability in connection with the mechanism of the general circulation was first pointed out by Kuo.[37]

Again there is an energy sink for $\bar{K} + \bar{P}$ due to dissipative actions. There is also energy flow between $\bar{K} + \bar{P}$ and $\bar{I}$. If we consider a water-filled dishpan having uniformly heated sides, we expect radial flow to

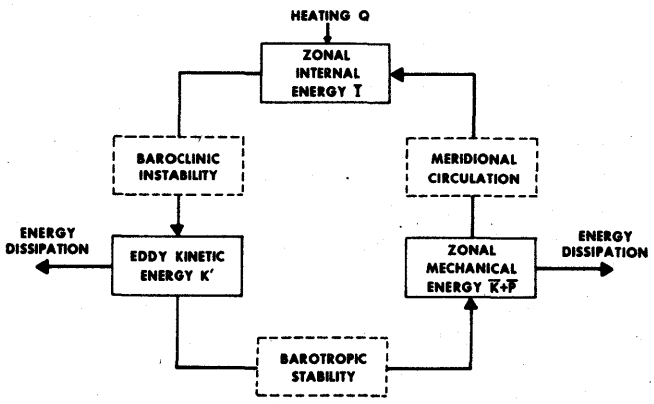


FIGURE 14—Scheme of energy transformation

be generated between the center of the dishpan and its side. If the dishpan is made to rotate, the radial motion of the flow may cause the generation of tangential motion as the result of the deflecting force of the system's rotation. Hadley,[38] using this idea, was the first to explain the trade winds system.[39] If zonal kinetic energy is generated as a result of meridional motion, such a meridional circulation is called *direct circulation*, or *Hadley-type circulation*. On the other hand, if zonal kinetic energy is consumed in order to maintain meridional circulation, such a circulation is called *indirect circulation*.

In middle and high latitudes, eddy meridional transport of kinetic energy seems strong enough so that the energy is even driven back to the $\overline{I}$ box, as first suggested by Starr.[40] However, in tropical latitudes, the conversion of energy from K' to $\overline{K} + \overline{P}$ is rather small, and, therefore, direct circulation converts zonal internal energy into zonal kinetic energy.[41] Many investigations have been carried out to determine the magnitude of energy conversion terms. However, scarcity of upper air data over the tropics and the Southern Hemisphere still prevents us from performing satisfactory diagnostic analyses on the heat, momentum, and energy conversions in the global atmosphere.

CONCLUSION

We have seen that numerical experiments help us to understand many questions about the mechanisms of the general circulation. The degree of understanding may be measured by our ability to simulate, with a model, actual atmospheric motions. A successful model, therefore, must be capable of predicting actual weather changes. Or, put in another way, we judge how good a model is by verifying predicted patterns. For this reason, we have also been working on the use of real data in order to check the forecasting capability of our general circulation model. So far, the results of our forecasts are very encouraging. Incidentally, it is stil difficult to prepare *global* upper air maps due to the paucity of observational data over the oceans and over the Southern Hemisphere in general. This means that it will be a long time until satisfactory short-range weather predictions are achieved because of the lack of accurate initial conditions for forecasting models.

Every year in the Untied States alone, weather hazards, such as hurricanes, severe storms, floods and drought, kill approximately 1300 people and cause property damage in excess of $11 billion.[42] When this problem is considered on a global basis, it is obvious that improved weather prediction can lead to immense social and economic benefits. In the last few years there has been an encouraging development toward eliminating deficiencies in the present international system of global weather observation. The program is called the World Weather Watch (WWW), and it will be carried out during 1972 under the sponsorship of the World Meteorological Organization (WMO). Another important and related plan is called the Global Atmospheric Research Program (GARP), which is the research-oriented partner of WWW. GARP is sponsored jointly by the International Council of Scientific Unions and the International Union of Geodesy and Geophysics. The goal of GARP is to promote a vastly improved understanding of the general circulation of the atmosphere.[43]

One year's operational forecasts at the National Meteorological Center, Weather Bureau, ESSA, have demonstrated that their new six-layer primitive equation model significantly improves the accuracy of short-range forecasts. Their experience with this model, which is as complicated as a general circulation model,[44] clearly indicates that even a short-range forecasting model must take into account detailed physical processes and that only by doing so can an improvement in the accuracy of weather forecasts be expected. It should be borne in mind that such an attempt has become possible only recently through the use of large-capacity and high-speed computers, such as the Control Data 6600 and the IBM 360.

Very recent studies by NCAR staff members and others have demonstrated that the accuracy of numerical weather simulation will increase significantly if a horizontal grid mesh finer than the current 5 degree mesh is used. A 2½ degree mesh will increase computing time by a factor of eight. It is obvious that to run a general circulation model which includes ocean circulation and detailed atmosphere-ground boundary layer calculations with a finer horizontal resolution than that presently used will require a supercomputer 500 times faster than the Control Data 6600.[45] A few "supercomputers" are being developed, the best known of which is the University of Illinois' parallel-processing ILLIAC IV. ILLIAC IV will be available by 1972.

Using upper air data gathered during 1972 by the World Weather Watch, together with a supercomputer, we will soon be able to attack the fundamental problem of an objective long-range weather forecast for two weeks and beyond. However, in order to achieve meaningful long-range forecasts, we will first have to learn more about the mechanism of long-term weather changes, an important, yet very poorly understood, problem. Results of our general circulation experiments have already indicated the importance of the condition of the earth's surface, including dynamic and thermal effects, upon atmospheric motions. We have begun to see the important effects of clouds interacting with atmospheric radiation and the equally important

effects of small-scale turbulence and convection in the atmospheric boundary layer. These problems must be investigated in even greater detail before the various effects can be properly included in general circulation models.

REFERENCES

1 N A PHILLIPS
Numerical weather prediction
In Advances in Computers ed F L Alt Academic Press New York 1960 Vol 1 43–90

2 P D THOMPSON
Numerical weather analysis and prediction
The Macmillian Co New York 1961 170 pp

3 I A KIBEL
An introduction to the hydrodynamic methods of short period weather forecasting
Translated from the original volume published in 1957 Moscow The Macmillian Co New York 1963 383 pp

4 G I MARCHUK
Numerical methods in weather forecasting
In Russian Gidromet Izdat Leningrad 1967 356 pp

5 R COURANT K O FRIEDRICHS H LEWY
Über die partiellen differenzengleichungen der physik
Math Annalen 100 32–74 1928

6 A S MONIN A M OBUKHOV
A note on general classification of motions in a baroclinic atmosphere
Tellus 11 159–162 1959

7 L F RICHARDSON
Weather prediction by numerical process
Cambridge University Press 1922 236 pp Reprinted by Dover Publications New York with an introduction by S Chapman 1965

8 G W PLATZMAN
A retrospective view of Richardson's book on weather prediction
Bull Am Meteorol Soc 48 514–550 1967

9 J G CHARNEY R FJØRTOFT J VON NEUMANN
Numerical integration of the barotropic vorticity equation
Tellus 2 237–254 1950

10 J G CHARNEY
On the scale of atmospheric motions
Geofys Publikasjoner 17 2 17 pp 1948

11 A ELIASSEN
The quasi-static equations of motion with pressure as independent variable
Geofys Publikasjoner 17 3 44 pp 1949

12 P D THOMPSON W L GATES
A test of numerical prediction methods based on the barotropic and two-parameter baroclinic models
J Meterol 13 127–141 1956

13 G P CRESSMAN
A three level model suitable for daily numerical forecasting
Tech Memo No 22 National Meteorological Center Weather Bureau ESSA 1963 22 pp

14 G P CRESSMAN
Barotrophic divergence and very long atmospheric waves
Mon Wea Rev 85 293–297 1958

15 J BJERKNES H SOLBERG
Life cycle of cyclones and the polar front theory of atmospheric circulation
Geofys Publikasjoner 3 1 18 pp 1922

16 E PALMÉN
The aerology of extratropical disturbances
In Compendium of Meteorology ed T F Malone American Meteorological Society 1951 599–620

17 A P BURGER
Scale consideration of planetary motions of the atmosphere
Tellus 10 195–205 1958

18 N A PHILLIPS
The general circulation of the atmosphere: A numerical experiment
Quart J Roy Meteorol Soc 82 123–164 1956

19 J SMAGORINSKY
General circulation experiments with the primitive equations: I The basic experiment
Mon Wea Rev 91 99–164 1963

20 C E LEITH
Numerical simulation of the earth's atmosphere
In Methods in Computational Physics Vol 4 ed B Alder et al Academic Press New York 1965 1–28

21 Y MINTZ
Very long term global integration of the primitive equations of atmospheric motion
In Proc WMO/IUGG Symposium on the Research and Development Aspects of Long Range Forecasting Boulder Colo 1964 WMO Tech Note No 66 1965 141–167

22 S MANABE J SMAGORINSKY R F STRICKLER
Simulated climatology of a general circulation model with a hydrological cycle
Mon Wea Rev 93 769–798 1965

23 J SMAGORINSKY S MANABE J L HOLLOWAY JR
Numerical results from a nine level circulation model of the atmosphere
Mon Wea Rev 93 727–768 1965

24 A KASAHARA W M WASHINGTON
NCAR global general circulation model of the atmosphere
Mon Wea Rev 95 389–402 1967

25 R M GOODY
Atmospheric Radiation Vol 1 Theoretical Basis
Clarendon Press Oxford 1964 436 pp

26 H G HOUGHTON
On the annual heat balance of the northern hemisphere
J Meteorol 11 1–9 1954

27 J LONDON
A study of the atmospheric heat balance
Contract AF 19 122–165 ASTIA No 117227 Department of Meterology and Oceanography New York University 1957 99 pp Final Report

28 O G SUTTON
Micrometeorology
McGraw-Hill Book Co New York 1953 333 pp

29 R D RICHTMYER K W MORTON
Difference methods for initial-value problems
Interscience Publishers New York 1967 405 pp

30 N A PHILLIPS
An example of non-linear computational instability
In The Atmosphere and the Sea in Motion Rossby Memorial volume ed B Bolin Rockefeller Inst Press New York 1959 501–504

31 R D RICHTMYER
A survey of difference methods for non-steady fluid dynamics
Tech Note 63-2 National Center for Atmospheric Research Boulder Colo 1963 25 pp

32 P D LAX B WENDROFF
Systems of conservation laws
Comm Pure Appl Math 13 217–237 1960

33 J E WELCH

Computer simulation of water waves
Datamation 12 41–47 1966

34 W M WASHINGTON B T O'LEAR J TAKAMINE D ROBERTSON
The application of CRT contour analysis to general circulation experiments
To be published in Bull Am Meteorol Soc 1968

35 J G CHARNEY
The dynamics of long waves in a baroclinic westerly current
J Meteorol 4 135–162 1947

36 E T EADY
Long waves and cyclone waves
Tellus 1 33–52 1949

37 H L KUO
Dynamic instability of two-dimensional non-divergent flow in a barotropic atmosphere
J Meteorol 6 105–122 1949

38 G HADLEY
Concerning the cause of the general trade wind
Phil Trans Roy Soc 39 58–62 1735 Reprinted by C Abbe Smithsonian Misc Coll 51 5–7 1910

39 E N LORENZ
The circulation of the atmosphere
Am Scientist 54 402–420 1966

40 V P STARR
An essay on the general circulation of the earth's atmosphere
J Meteorol 5 39–43 1948

41 E PALMÉN
General circulation of the tropics
In Proc Symposium on Tropical Meteorology Rotorua New Zeland Nov 1963 3–30 1964

42 R M WHITE
The world weather program
Bull Am Meteorol Soc 48 81–84 1967

43 W O ROBERTS
The global atmospheric research program
Bull Am Meteorol Soc 48 85–88 1967

44 F G SHUMAN J B HOVERMALE
An operational six-layer primitive equation model
J Appl Meteorol 7 525-547 1968

45 H G KOLSKY
Computer requirements in meteorology
Tech Rept No 38002 IBM Scientific Center Palo Alto Calif 1966 158 pp

A computational proplem encountered in a study of the earth's normal modes

by FREEMAN GILBERT and GEORGE BACKUS

University of California
San Diego, California

There are many scientific disciplines where one is faced with the problem of finding solutions to a system of coupled, ordinary, linear, first-order differential equations

$$\frac{d}{dx} f_i(x) = \sum_{j=1}^{n} C_{ij}(x)\, f_j(x); \; i = 1, \cdots, n \qquad (1)$$

$$a \leq x \leq b$$

given $f_i(a)$. For the normal modes of the Earth, n can be as large as 20, and $2 \leq n \leq 6$ is common (Alterman, Jarosch and Pekeris, 1959). Finding solutions to (1) can be time-consuming and expensive. In this paper we present a few, sometimes justifiable, tricks that we have found useful and economical in solving (1) when C is piecewise continuous.

We suppose that we have M one-step methods of order m, m = 1,..., M, for obtaining approximate solutions to the initial value problem (1). For a given relative truncation error, ϵ, the m^{th} one-step method is associated with a step size, $h_m(\epsilon,x)$, that depends on x because the elements of the coefficient matrix, $C_{ij}(x)$, are functions of x. We assume $h_1(x) < h_2(x) < \ldots < h_m(x)$, and that the computing time for using the m^{th} one-step method increases with m less rapidly than does the step-size.

Let

$$a = x_0 < x_1 < \cdots < x_L = b \qquad (2)$$

We seek to determine $f_i(x_\ell)$; $\ell = 1, \ldots, L$, given $f_i(x_0)$, using the M one-step methods as economically as possible. Suppose we have $f_i(x_\ell)$ and seek $f_i(x_\ell + 1)$. Let $h = x_{\ell+1} - x_\ell$. Choose $\nu \leq M$ and $\mu \leq M$ such that

$$\begin{aligned} h_{\nu-1}(x_\ell) &< h \leq h_\nu(x_\ell) \\ h_{\mu-1}(x_{\ell+1}) &< h \leq h_\mu(x_{\ell+1}) \end{aligned} \qquad (3)$$

Let m be the larger of (ν,μ). Then the m^{th} one-step method can be used to determine $f_i(x_{\ell+1})$ with a relative truncation error less than ϵ. If h is too large to satisfy (3) let $h_M = \min(h_m(x_\ell), h_m(x_{\ell+1}))$ and use the M^{th} one-step method to find $f_i(y_{\ell+1})$; $y_{\ell+1} = x_\ell + h_M$. At this stage let $h = x_{\ell+1} - y_{\ell+1}$ and try again to satisfy (3). The method is summarized in the flow diagram of Figure 1.

Given ϵ, the acceptable relative truncation error, and C(x), the coefficient matrix in (1), how do we estimate $h_m(x)$; m = 1, ..., M; before solving (1)?

Let $f_i'(x) = df_i(x)/dx$. Then, taking the scalar product of (1) with itself,

$$\sum_i f_i'^2 = \sum_i \left(\sum_j C_{ij} f_j \right)^2 \qquad (4)$$

Using the Schwartz in equality

$$\left[\sum f_i'^2 \right]^{\frac{1}{2}} \leq \left[\left(\sum_i \sum_j C_{ij}^2 \right) \sum_k f_k^2 \right]^{\frac{1}{2}} \qquad (5)$$

Then

$$\lambda = \left(\sum_i \sum_j C_{ij}^2 \right)^{\frac{1}{2}} \qquad (6)$$

is an upper bound (probably a crude one) on the growth of f(x). If we accept $\lambda(x)$ as the maximum rate of change of f(x) an m^{th} order one-step method has a relative truncation error ϵ_m

$$\epsilon_m \leq \gamma (\lambda h)^{m+1} e^{\lambda h}/(m+1)! \qquad (7)$$

For most one-step methods, particularly Runge-Kutta methods, γ is an expression too complicated to be of much practical use. Moreover, γ is often near unity and is traditionally disregarded in (7). We assume that λ is large enough to permit us to set $\gamma = 1$ in (7). $\Lambda = \lambda h$.

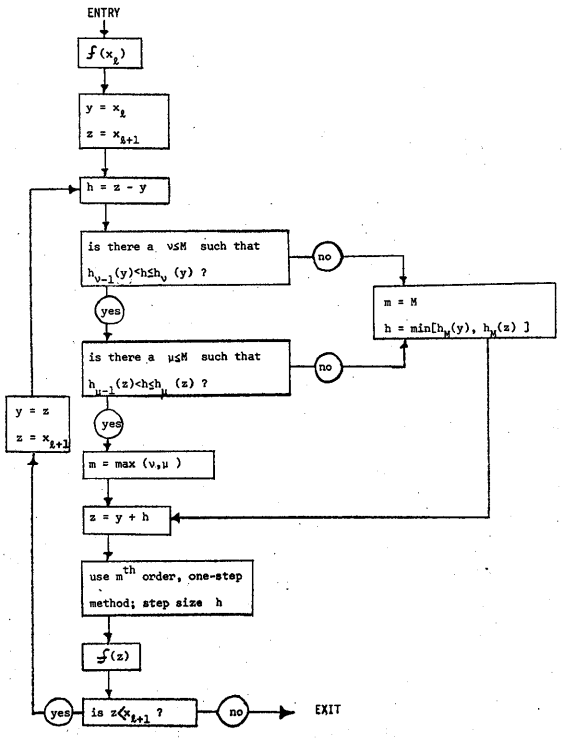


FIGURE 1

If ϵ is given, a value of h that satisfies

$$\Lambda^{m+1} e^{\Lambda}/(m+1)\,! = \epsilon \tag{8}$$

will be a conservative estimate of the step-size. Define

$$\Lambda_m^{(0)} = \left[\left(m+1\right)!\epsilon\right]^{1/m+1}$$

$$\Lambda_m^{(k+1)} = \Lambda_m^{(0)} \exp\left(-\Lambda_m^{(k)}/m+1\right); k = o, 1, \cdots \tag{9}$$

$$\Lambda_m = \lim_{k\to\infty} \Lambda_m^{(k)}$$

For $\epsilon<1$ ($\epsilon<(m+1)^{m+1}/(m+1)!$) the limit in (9) exists and is the solution to (8). A set of values of Λ_m for m = 1,... M can be calculated for a given ϵ. Then the variable step sizes are

$$h_m(x) = \Lambda_m/\lambda(x) \tag{10}$$

A less conservative, but intuitively plausible, upper bound λ can be found as follows. Suppose C does not depend on x for $x_\ell \le x \le x_{\ell+1}$

Let

$$P_{ij} = [\exp(hC)]_{ij};\ h = x_{\ell+1} - x_\ell$$

$$= \delta_{ij} + C_{ij} + \tfrac{1}{2}! \sum_k C_{ik}C_{kj} \tag{11}$$

$$+ \tfrac{1}{3}! \sum_k \sum_\ell C_{ik}C_{k\ell}\, C_{\ell j} + \cdots$$

Then

$$f_i(x_{\ell+1}) = \sum_{j=1}^{n} P_{ij} f_j(x_\ell) \tag{12}$$

(Frazer, Duncan and Collar, 1960, p. 219.) The series (11) can be converted into one of n terms by using the Cayley-Hamilton theorem and the Lagrange-Sylvester interpolation formula. Some examples from seismology are given by Gilbert and Backus (1966). [In (2.10) and (2.11) of the paper cited the integral is the argument of the exponential function, as in (2.12).] Using the finite series representation of P in (12) leads to the equivalent representation

$$f_i(x_{\ell+1}) = \sum_{j=1}^{n} \phi_{ij} \exp(h\lambda_j) \tag{13}$$

where the λj are the eigenvalues of C and ϕ_{ij} are related to the matrix of eigenvectors, e

$$\sum_{j=1}^{n} (C_{ij} - \lambda_k \delta_{ij})\, e_{jk} = 0$$

$$\phi_{ij} = e_{ij}\nu_j$$

$$\sum_{j=1}^{n} \phi_{ij} = \sum_{j=1}^{n} e_{ij}\nu_j = f_i(x_\ell) \tag{14}$$

From (13) it appears that $\lambda = \max |\lambda_k|$ is a plausible upper bound.

When C is not constant (13) suggests that the representation

$$f_i(x) = \sum_{j=1}^{n} \phi_{ij}(x) \exp\int_{x_\ell}^{x} \lambda_j(z)\, dz;\ x_\ell < x < x_{\ell+1} \tag{15}$$

may be a useful one if the $\phi_{ij}(xl)$ are computed from $C_{ij}(x_l)$ and f_i (x_l) according to (14), and if the $\lambda_j(z)$ are

the eigenvalues of C(z). Substituting (15) into (1) yields

$$\sum_{j=1}^{n} \exp\int_{x_\ell}^{x} \lambda_j(z)\,dz \left[\phi'_{ij}(x) + \phi_{ij}(x)\,\lambda_j(x) - \sum_{k=1}^{m} C_{ik}(x)\,\phi_{kj}(x)\right] = 0 \tag{16}$$

A sufficient condition for (16) to be satisfied is

$$\phi'_{ij}(x) = \sum_{k=1}^{n} [C_{ik}(x) - \delta_{ik}\lambda_j(x)]\phi_{kj}(x) \tag{17}$$

For $x = x_\ell$, $\phi'_{ij} = 0$ so, if we assume a power series representation for ϕ_{ij}, it has the form

$$\phi_{ij}(x) = \phi_i'_j'(x_\ell) + \tfrac{1}{2}\,\phi_{ij}(x_\ell)\,(x - x_\ell)^2 + \cdots \tag{18}$$

Therefore it is plausible to assume that ϕ'_{ij} is small compared to $\phi_{ij}\lambda_j$ in (16), and that ϕ_{ij} changes slowly in (15). If so, then $\lambda = \max|\lambda_k(x)|$, $x_\ell < x < x_{\ell+1}$, is a reasonable upper bound. It has been our experience that such a definition of λ works quite well.

To illustrate the method we consider a simple but characteristic example. Let

$$C(x) = \begin{vmatrix} 0 & 1 \\ n^2/x^2 - 1 & -1/x \end{vmatrix}; \quad \begin{aligned} f_1(x) &= J_n(x) \\ f_2(x) &= J'_n(x) \end{aligned} \tag{19}$$

Then

$$\lambda = .5/x + |n^2/x^2 + 1/4x^2 - 1|^{\frac{1}{2}} \tag{20}$$

Given n = 10 and

$$\begin{aligned} f_1(1) &= 2.63061\ 512\ (-10) \\ f_2(1) &= 2.61863\ 505\ (-9) \end{aligned} \tag{21}$$

we seek $f_i(x)$, x = 1.5(.5) 20. We take M = 8 and use the one-step Runge-Kutta methods of [Shanks (1966)]. With $\epsilon = 10^{-5}$ we find $f_i(20) = .18648\ 540$. The correct value is $J_{10}(20) = .18648\ 256$ giving a relative error of 1.524 (—5) which is not much larger than e.

A generalization of (1) is the matrix equation

$$\frac{d}{dx} F_{ij}(x) = \sum_{k=1}^{n} C_{ik}(x)\,F_{kj}(x); \quad i = 1, \cdots, n;\ j = 1, \cdots, \ell \tag{22}$$

There are numerous eigenvalue problems that can be put in the form (22). In many of these problems one or more minors of F are sought. It is not uncommon that the elements of F are large and nearly equal, the result being that some of the minors are calculated with severe rounding error. This difficulty can be overcome by calculating the minors of F directly, as follows:

An n x ℓ matrix ($\ell \geq$ n) has N × L minors of order m $\geq \ell$ where

$$N = \binom{n}{m} = \frac{n!}{m!(n-m)!},\quad L = \binom{\ell}{m}$$

When the minors of order m are arranged in an N × L array in some definite manner the array is called the m^{th} minor matrix of F [Gantmacher's (1959) compound matrix] which we denote by $\mathfrak{F}^{(m)}$.

Let $F\begin{pmatrix} i_1\, i_2 \cdots i_m \\ k_1\, k_2 \cdots k_m \end{pmatrix}$

denote an m^{th} order minor of F

$$F\begin{pmatrix} i_1 i_2 \cdots i_m \\ k_1 k_2 \cdots k_m \end{pmatrix} = \sum_{i_1=1}^{\ell}\sum_{i_2=1}^{\ell}\cdots\sum_{i_m=1}^{\ell} F_{i_1 j_1}, F_{i_2 j_2} \cdots F_{i_m j_m}\, x \tag{23}$$

$$x \in \begin{matrix} j_1 j_2 \cdots j_m \\ k_1 k_2 \cdots k_m \end{matrix}$$

where ϵ is the m^{th} order alternating symbol. For example let n = 4, ℓ = 3, m = 2.

$$F\binom{34}{12} = \sum_{j_1=1}^{3}\sum_{j_2=1}^{3} F_{3j_1} F_{4j_2} \qquad \begin{matrix} j_1 j_2 \\ \in \\ 1\ 2 \end{matrix}$$

$$= F_{31} F_{42}\,\epsilon_{12}^{12} + F_{32} F_{41} \in_{12}^{21}$$

$$= F_{31} F_{42} - F_{32} F_{41} \tag{24}$$

Since F is a solution to (22) we can differentiate (23) and express the result in terms of C.

$$\frac{d}{dx} F\begin{pmatrix} i_1 i_2 \cdots i_m \\ k_1 k_2 \cdots k_m \end{pmatrix} = \sum_{\nu=1}^{n} \left\{ C_{i_1 \nu}\, F\begin{pmatrix} \nu i_2 \cdots i_m \\ k_1 k_2 \cdots k_m \end{pmatrix} + \right.$$

$$+ C_{i_2\nu} F\begin{pmatrix} i_1 \cdots i_m \\ k_1 k_2 \cdots k_m \end{pmatrix}$$

$$+ \cdots + C_{i_m\nu}\begin{pmatrix} i_1 i_2 \cdots \gamma \\ k_1 k_2 \cdots k_m \end{pmatrix}\Bigg\} \quad (25)$$

Thus $\mathfrak{F}^{(m)}$ is a solution of

$$\frac{d}{dx}\mathfrak{F}^{(m)}_{ij}(x) = \sum_{k=1}^{n} \mathcal{C}^{(m)}_{ik}(x)\,\mathfrak{F}^{(m)}_{kj}(x)$$

$$i = 1, \cdots, N; j = 1, \cdots, L \quad (26)$$

where $\mathcal{C}^{(m)}$ is a square coefficient matrix or order N. For example let n = 4, ℓ = 3, n = 2. Then N = 6, L = 3. We arrange the second order minors of F in the 6 × 3 array

$$\mathfrak{F}^{(2)} = \begin{bmatrix} F^{12}_{12} & F^{12}_{13} & F^{12}_{23} \\ F^{13}_{12} & F^{13}_{13} & F^{13}_{23} \\ F^{14}_{12} & F^{14}_{13} & F^{14}_{23} \\ F^{23}_{12} & F^{23}_{13} & F^{23}_{23} \\ F^{24}_{12} & F^{24}_{12} & F^{24}_{23} \\ F^{34}_{12} & F^{34}_{13} & F^{34}_{23} \end{bmatrix} \quad (27)$$

Then $\mathcal{C}^{(2)}$ is the array

$$\mathcal{C}^{(2)} = \begin{bmatrix} C_{11}+C_{22} & C_{23} & C_{24} \\ C_{32} & C_{11}+C_{33} & C_{34} \\ C_{42} & C_{43} & C_{11} \\ -C_{31} & C_{21} & 0 \\ -C_{41} & 0 & C_{21} \\ 0 & -C_{41} & C_{31} \end{bmatrix}$$

$$+ C_{44}\begin{bmatrix} -C_{13} & -C_{14} & 0 \\ C_{12} & 0 & -C_{14} \\ 0 & C_{12} & C_{13} \\ C_{22}+C_{33} & C_{34} & -C_{24} \\ C_{43} & C_{22}+C_{44} & C_{23} \\ -C_{42} & C_{32} & C_{32}+C_{44} \end{bmatrix} \quad (28)$$

Notice that trace $(\mathcal{C}^{(2)})$ = 3 trace (C) in general

$$\text{trace}\,(\mathcal{C}^{(m)}) = \binom{n-1}{m-1}\,\text{trace}\,(C) \quad (29)$$

In the special case m = ℓ = n $\mathfrak{F}^{(n)}$ is a scalar

$$\mathfrak{F}^{(n)}(x) = \det(F(x)) \quad (30)$$

and $\mathcal{C}^{(n)}$ = trace (C). Thus

$$\frac{d}{dx}\det(F) = \text{tr}\,(C)\det(F) \quad (31)$$

and

$$\det(F(x) = \det(F(x_0))\exp\int_{x_0}^{x} \text{tr}\,(C(z))dz \quad (32)$$

a relation known as Jacobi's identity.

For the normal modes of the Earth n = 6 for the equations governing the elastic-gravitational oscillations. The dispersion relation is a third-order minor of F with ℓ = 3. Thus N = 20, L = 1.

Notice that the N eigenvalues of $\mathcal{C}^{(m)}$ are the combinations of the n eigenvalues of C taken m at a time, a result related to Kronecker's theorem (Gantmacher, 1959, 1, p. 75)

As a trivial but revealing example take ℓ = m = n = 2 and

$$F(x) = \begin{bmatrix} \cosh x & \sinh x \\ \sinh x & \cosh x \end{bmatrix} \quad (33)$$

Now $\mathfrak{F}^{(2)}(x) = \det(F(x)) = \cosh^2 x - \sinh^2 x = 1$. For (33)

$$C = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad (34)$$

If we integrate (22) to get F(x) for increasing x then, no matter what precision we use in the calculation, eventually round-off error will cause det (F(x)) to approach

zero. However, from (34) we have trace C = 0 so we know from (26) and (31) that det (F(x)) is a constant.

REFERENCES

Z ALTERMAN H JAROSCH C L PEKERIS
Oscillations of the earth
Proc Roy Soc *A 252* 80–95 1959
R A FRAZER W J DUNCAN A R COLLAR
Elementary matrices
Cambridge London 1960
F R GANTMACHER
Theory of matrices
Translated from Russian by K A Hirsch 2 vols Chelsea New York 1959
F GILBERT G E BACKUS
Propagator matrices in elastic wave and vibration problems
Geophysics *31* 326–332 1966
E B SHANKS
Solutions of differential equations by evaluations of functions
Math Comp 21–38 1966

Computer animation and the fourth dimension

by A. MICHAEL NOLL

Bell Telephone Laboratories, Incorporated
Murray Hill, New Jersey

INTRODUCTION

Man is a creature restricted to a world of three spatial dimensions in which he is reasonably free to move about at will except for the arbitrary territorial boundaries imposed by different nations. Man also lives in another dimension over which he presently has no control other than to watch its continual forward movement in the mechanical and electrical gadgets he has devised to measure this dimension which he calls time. Many people call time the fourth dimension, but because of its many unique qualities I would rather consider time as a special dimension. Therefore, in this paper the fourth dimension is a fourth purely spatial dimension not to be confused with time.

Since we live in a world of three spatial dimensions, we are unable to visualize a fourth spatial dimension perpendicular to our three spatial dimensions. However, the computer is not bothered by problems of human visualization and is able to deal with objects in four-dimensional space as easily as it performs calculations for three-dimensional objects. Only a fourth number is required by the computer to locate a point in four-dimensional space. But if the computer is to present the results of its calculations with four-dimensional objects to man, then the computer must come down from its digital tower and perform the necessary operations so that man can visualize and possibly obtain some intuitive feel for the results. The technique of perspective projection from four dimensions to three dimensions is particularly helpful here.

In a previous paper ("A Computer Technique for Displaying n-Dimensional Hyperobjects," Communications of the ACM, Vol. 10, No. 8, August 1967, pp. 469–473), I described in general the technique of perspective projection of n-dimensional hyperobjects and also the application of this technique to the production of computer-generated three-dimensional movies of rotating four-dimensional hyperobjects, and in particular, the hypercube. The mathematical details of this previous paper will not be repeated here but a physical interpretation of the results will be explored with particular emphasis on the purely artistic effects obtainable with the technique.

A computer technique for displaying four-dimensional hyperobjects

The computer performs its calculations and other operations under the control of a program as outlined in Figure 1. The computer first reads into its memory the four-dimensional coordinates of each point of the hyperobject. Just as three numbers exactly position a point in three-dimensional space, each set of four numbers exactly positions a point in four-dimensional space. These points usually are the end points of straight lines in four-dimensional space and have been previously punched onto standard punched cards.

Rotation

After the four-dimensional object has been converted to straight lines and the four-dimensional coordinates of the end points of these lines have been read into the

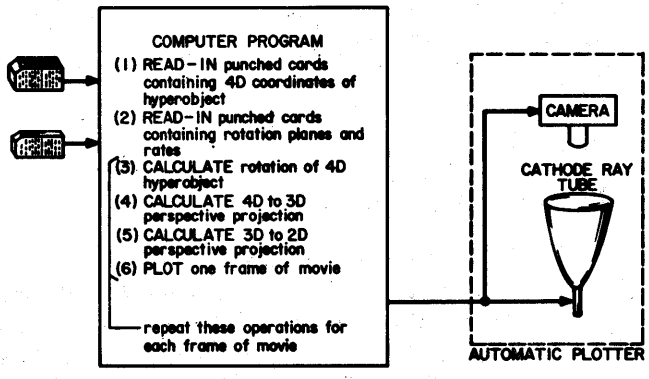


Figure 1—Outline of computer program and technique used to generate movies of rotating four-dimensional hyperobjects

memory of the computer, the object must then be moved in some fashion in the four-dimensional space since a movie of a completely stationary object is quite senseless to most people.

There are many types of motions which can be considered such as shrinking, expanding, translating, or rotating the object in four-dimensional space. I arbitrarily decided to restrict the movie to rotation of the hyperobject in four-dimensional space, but this decision is unfortunately not enough since some thought must be given to exactly what is meant by rotation in a four-dimensional space.

In three dimensions, we usually think of rotation in terms of rotating an object about some specified axis, i.e., the axis of rotation. An alternative approach is to consider the two-dimensional plane that is perpendicular to this axis and define a rotation as a rotation within this two-dimensional plane. If we define a principal rotation to be one which involves a rotation about one of the principal coordinate axes, then the successive application of three rotations about each of these three axes can rotate the three-dimensional object to any specified position. In any single plane perpendicular to one of the principal axes, only two coordinates change as a result of the rotation in that particular plane.

In four dimensions I have decided to extend this three-dimensional concept of rotation. However, in four-dimensional space there are many axes all perpendicular to the same two-dimensional plane so that specifying a rotation about an axis is both intuitively and mathematically meaningless. The solution is to specify the two-dimensional plane in which the rotation is to take place and to completely drop the three-dimensional concept of rotation about an axis. Once again, all of this is somewhat arbitrary since one could define other four-dimensional rotations, but I desired a mathematical definition that was a simple extension of a rotation in three-dimensional space as specified by a two-dimensional plane.

Thus, the computer program next instructs the computer to read in the punched cards specifying the desired two-dimensional rotation planes and the corresponding rates of rotation in these planes as degrees per frame of the movie.

Perspective projection

Since we are restricted to three spatial dimensions, it is impossible for us to visualize a fourth spatial dimension. Of course, these problems do not affect the inhuman computer which purely manipulates numbers in a mathematical fashion according to specified formulas. But, since we should desire to see the results of the rotations of the hyperobject in four-dimensional space, the computer must perform some transformation upon the hyperobject so that we can look at it. In other words, the four-dimensional object has to be projected to three dimensions so that we can see it in our three-dimensional world.

Just as there were many alternatives for various motions, there are many alternatives for the type of projection from four dimensions to three dimensions. One type of projection is parallel projection in which the projection lines from the object to the projection space are all parallel to each other. This is also equivalent to simply eliminating one coordinate in the specification of each four-dimensional point so that one is left with three numbers for each point in the object. However, parallel projection is not usually encountered in our three-dimensional world since our eyes are really a perspective projection system. For this reason, I decided to use perspective projection from four dimensions to three dimensions. The equations for this projection are derived in the previous paper and are a very simple extension of the projection equations for going from three dimensions to two dimensions. The use of perspective projection allows one to retain a three-dimensional feel for what is happening although absolutely no feel for the fourth dimension in terms of rigidity of the four-dimensional object is obtained as will be described later.

Plotting the movie

The three-dimensional projection of the four-dimensional object is finally perspectively projected to two dimensions. These two-dimensional coordinates are used as input to an automatic plotter consisting of a cathode ray tube and a camera positioned to photograph the face of the tube. After each single frame of the movie has been plotted, the computer program instructs the computer to rotate the hyperobject to its next position, to project the hyperobject to three dimensions and then to two dimensions, and finally to plot another frame of the movie on the face of the cathode ray tube. These operations are repeated until the desired number of frames of the movie are obtained.

The hypercube

The four-dimensional hypercube is introduced now because of its importance in understanding the motions that result from rotations in four-dimensional space. The three-dimensional perspective projection of a four-dimensional hypercube looks like a three-dimensional cube within a three-dimensional cube as depicted in Figure 2. Some intuitive insight can be gained by considering the two-dimensional perspective projection of a three-dimensional cube, as depicted in Figure 3, which looks like a square within a square. What is happening here is that the face of the cube closest to the projection

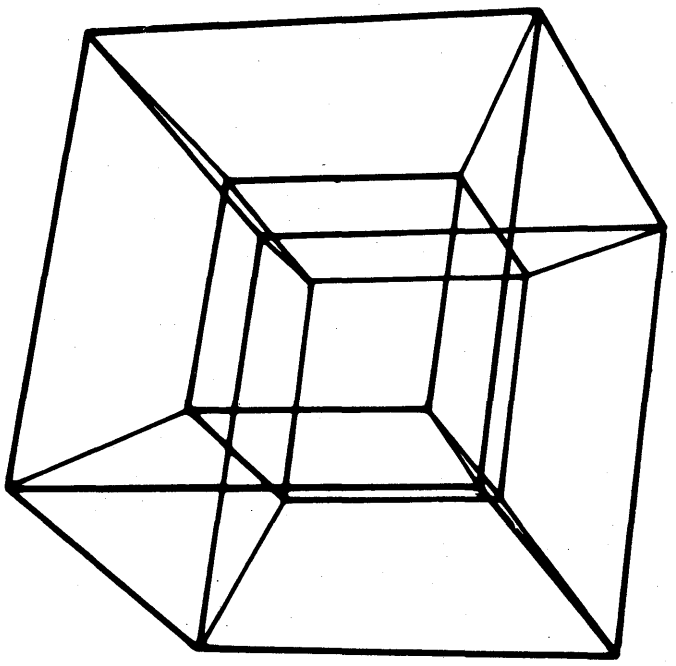


Figure 2—The three-dimensional perspective projection of a four-dimensional hypercube is a cube-within-a-cube with the outer cube corresponding to the face of the hypercube which is closest to the projection space

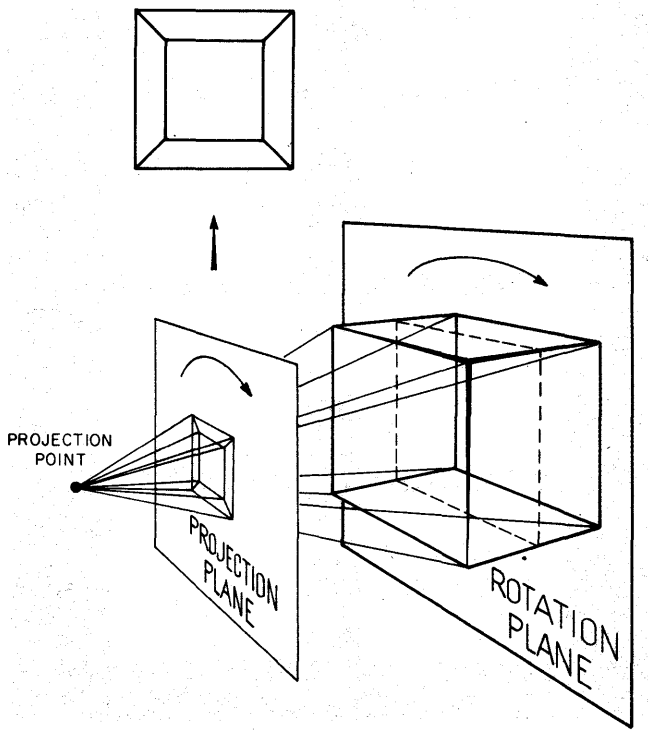


Figure 3—The perspective projection of a three-dimensional cube with one face parallel to the projection plane is a square-within-a-square. If the rotation plane is perpendicular to the axis along which the cube is being viewed, then the square-within-a-square simply turns as a whole

plane appears largest while the face farthest away is smallest. If the closest face is paralled to the projection plane, then its perspective projection is undistorted except for its overall size. Thus, the perspective projection of a three-dimensional cube is actually an undistorted two-dimensional face within an undistorted two-dimensional face. Since the face of a four-dimensional hypercube is a three-dimensional cube, the perspective projection of the hypercube with one face parallel to the projection space is a cube within a cube.

When the hypercube is rotated, some very intriguing distortions occur in the three-dimensional perspective projection depending upon the choice of rotation plane. If the two-dimensional rotation plane is perpendicular to the axis along which the hypercube is being viewed, then the cube-within-a-cube turns as a whole. If the rotation plane is not perpendicular to the viewing axis, then the inner cube enlarges until it becomes the outer cube while the outer cube shrinks until it becomes the inner cube. This turning-inside-out motion is to be expected since as the hypercube turns one face comes closest to the projection plane and therefore appears largest. All of this is similar to the three-dimensional cube that rotates in a plane which is not parallel to the projection plane except that the faces of the three-dimensional cube are squares which distort and change their size as different faces come closer or recede from the projection plane as shown in Figure 4.

The four-dimensional cube is of course completely rigid while it is being rotated, and the turning-inside-out motion is purely a result of the perspective projection from four dimensions to three dimensions. It was hoped that one might be able to obtain some feel or visualization for the rigid four-dimensional hypercube by observing the three-dimensional perspective projection but no such feel could be obtained. On the contrary, only a cube-within-a-cube turning inside out was observed as if its members were made of rubber and could stretch to perform the observed contortions. Thus, this experiment in computer graphics was something of a complete failure in terms of its stated purpose of assisting man to visualize a fourth spatial dimension. However, the technique might yet have merit as a pure display method for observing scientific data in higher than three dimensions such as the results of multi-dimensional factor analyses of experimental data, and research in this direction involving real-time computer graphics is actively being pursued.

Artistic consequences

Most people, including many artists and animators, were awed by the thoroughly fascinating artistic beauty of the cube-within-a-cube that so gracefully turned itself inside out; it was purely incidental to them that the

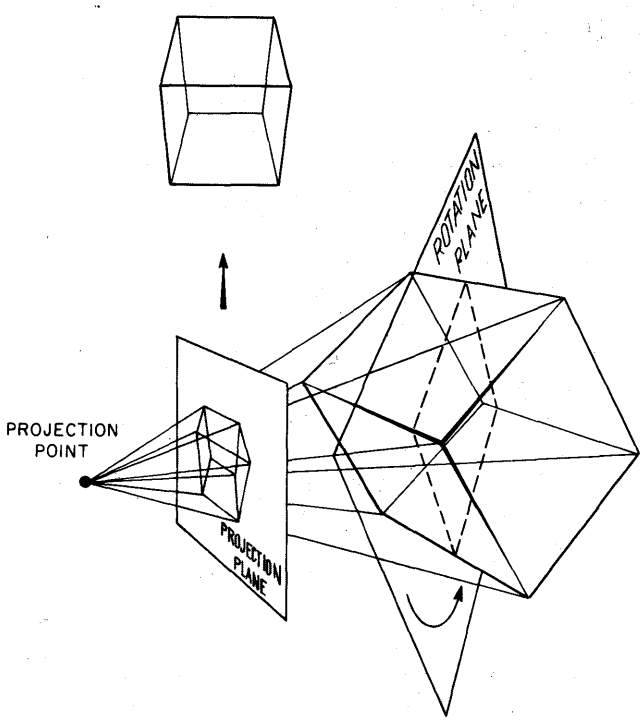


Figure 4—If the rotation plane is not perpendicular to the viewing axis, then the faces appear as distorted squares which change both their size and shape as the cube rotates

movie was produced by a digital computer or that the cube-within-a-cube was the perspective projection of a four-dimensional hypercube. Some further investigation and exploitation of the technique for its artistic consequences alone was quite definitely indicated by all the enthusiasm generated by the movie. The remainder of this paper describes these artistic explorations as far as it is possible to verbalize matters involving aesthetic concepts.

Four-dimensional letters

The computer program for producing the movie of the hypercube was general purpose in the sense that any set of four-dimensional points could be used as input. In my mind, I visualized letters rotating and moving through each other in an extremely graceful yet obviously thoroughly controlled manner similar to the motion of the hypercube. To investigate this effect, I used the four-dimensional coordinates of points specifying the end points of straight lines as input to the program. These straight lines formed individual letters which I could easily in effect place on different two-dimensional planes in the four-dimensional space. Conceptually, this is identical to placing the letters on the two-dimensional planes which make up the hypercube. I chose the letters A, M, and N which form my initials. Some selected frames from the movies corresponding to different configurations of the three letters in four-dimensional space are shown in Figure 5. In the movie, the individual letters appear to float through space or to be hinged together although able to mysteriously distort while rotating as a whole.

"Incredible Machine"

If individual letters could be used in the program, then a series of letters could be strung together to form words on two-dimensional planes in the four-dimensional space. This approach was also experimented with and resulted in words rotating about and through each other in a beautifully fascinating but thoroughly controlled uniform fashion. I used this effect to create the main title for a movie about computer graphics produced for American Telephone and Telegraph Company by Owen-Murphy Productions in New York City.

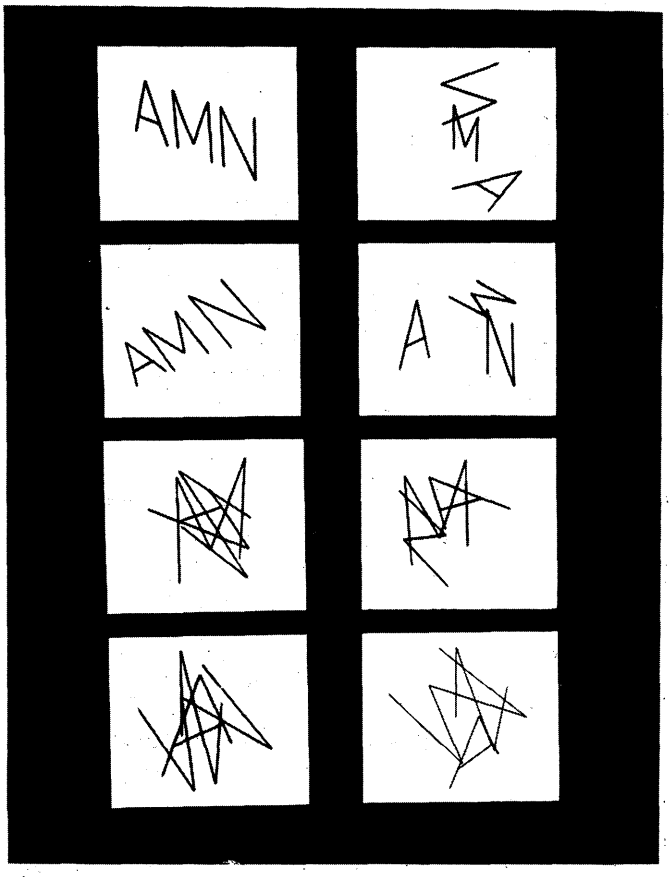


Figure 5—Selected frames from different computer-generated movies of different configurations or placements of the letters A, M, and N on different two-dimensional planes in four-dimensional space

The individual letters of the title of this movie "Incredible Machine" were converted to straight lines connecting points whose two-dimensional coordinates were then obtained by plotting the letters of each word on graph paper. The two words were then in effect placed upon two different two-dimensional planes in four-dimensional space, and the four-dimensional points of the end points of the lines forming the letters were punched into cards for input to the computer. A number of different configurations of the two words in four-dimensional space were then appraised for their artistic merit by my judgment of the effectiveness of the computer-generated movies of the rotating words. A single configuration was finally chosen, and fairly long movie of the rotating two words was produced. From this movie, a single frame was chosen in which the two words both were individually legible and also formed an aesthetically pleasing overall form.

The final title movie opens with a zoom-in on the rotating words. The zoom has a linearly-decreasing acceleration so that transition at the end of the zoom to the rotating portion is smooth. The rotating portion continues until the previously-mentioned frame is reached whereupon all motion ceases for a very short while. This freeze is followed by another zoom-in which was done optically rather than by the computer since this was easier than programming the computer to determine which lines exceeded the picture area. The final title movie was optically superimposed over the opening scene of a man working at a graphical display console. Selected frames from the title movie including the freeze frame are shown in Figure 6.

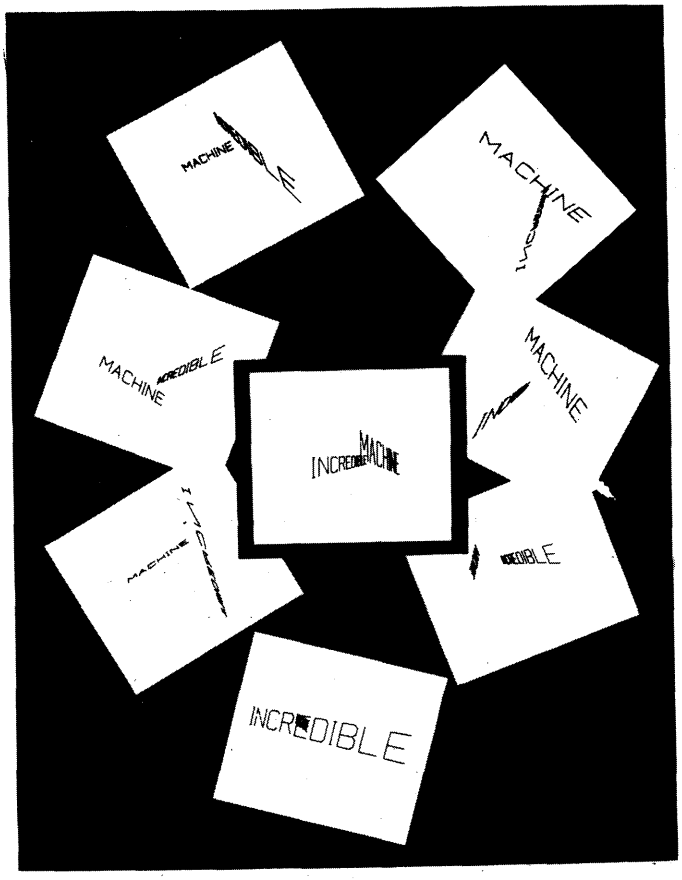


Figure 6—Selected frames from the computer-generated main title of a movie about computer graphics entitled "Incredible Machine"

CONCLUSION

A question arises in some people's minds about the artistic merit of purely scientific techniques which are applied to artistic purposes. I would like to think that the end result, no matter through what medium it was produced, should be judged for its own artistic merit. The fact that these movies were produced by a digital computer performing all sorts of mathematical operations on four-dimensional data should be incidental to the artistic effects thereby achieved. However, I do feel that these techniques involving the new technology, and in particular computers, will only be exploited fully for artistic purposes when the artist who has dedicated his life to artistic explorations learns to use these new tools as new artistic media. I am an engineer and my artistic ideas are somewhat conservative. But, I am quite excited by the prospects for the new artistic effects and beauty which will surely result from creative collaboration between artists and the computer.

Computer displays in the teaching of physics

by JUDAH L. SCHWARTZ and EDWIN F. TAYLOR

Massachusetts Institute of Technology
Cambridge, Massachussets

INTRODUCTION

An egg falls on the floor and breaks. Now run the movie backward: the broken egg reassembles itself and jumps into the air. Everyone laughs; they know that all the king's horses and all the king's men cannot put Humpty Dumpty together again. Even a child recognizes that an egg will not reassemble itself. *Why* won't it do so? The answer is deep, involving concepts of order, entropy, irreversibility. The intuition is simple and natural. The precise formulations should be embedded in a developed intuition.

Higher education, particularly in the sciences, tends to start with the formal, the symbolic, the abstract. Usually it is only after the equations and concepts are mastered that one is able to develop an intuition for the subject. And the teacher usually provides no help in the intuition-building process. Even in laboratory and demonstration experiments a long chain of inference lies between the observations and the physical questions at issue.

Our scientific education is largely backward, i.e., formalism first, intuition later (if at all). By teaching in this order we repel the majority of people who do not feel comfortable starting with equations. We also make it difficult for the professionals, since we offer them little assistance in developing the intuitive grasp of a subject from which new ideas spring.

The display computer provides a powerful means for modeling phenomena of nature. It can help the student visualize some effects more directly than through equations. Moverover, there are realms of physics in which one cannot hope to have a direct experience with the senses. For example, the high speeds of special relativity and the sub-microscopic scale of atomic physics closes both to direct visual observation. This paper deals with some attempts to present as visually and directly as possible some of the fundamental results of special relativity and quantum physics using computer displays.

Relativity

Rocket travel

Figure 1 shows a display tube face containing a view of a country road as might be seen from the front windshield of a "rocket car." Lining the road are telephone poles with crosspieces making the poles L-shaped. Because of computer memory limitations, only three poles are displayed on each side of the road. The operator controls the forward or backward acceleration of the rocket (the "proper acceleration") down the road by giving light pen commands. Rocket velocity is limited only by the laws of relativity and can approach the velocity of light. However, the visual presentation is *scaled*, so that relativistic effects are easily visible. This scaling can be interpreted in either one of two alternative ways: (1) the road is normal width, and the speed of light is sixty miles per hour, or (2) the speed of light has its normal value, while the telephone pole height and the road width are comparable to the dimensions of the planet Jupiter. Either interpretation applies to the one display.

By manipulating the display, an operator can become familiar with the following effects due to time delays for light propagation (effects 1 and 2 below) and due to the kinematics of spacetime (effects 3 thru 5 below).

1. *Bending Poles.*

As poles move toward the observer ("forward motion down the road") the poles appear to be bent backward (Figure 2), with the tops farther back down the road than the middles. This is due to the fact that the top of a pole is farther from the observer's eye than the middle of the pole. Therefore it takes light longer to reach the observer's eye from the pole top than from the pole middle. Therefore the observer sees the more distant top of the approaching pole where it was at an earlier time (i.e., farther away from him) than the observed position of the middle portion of the pole.

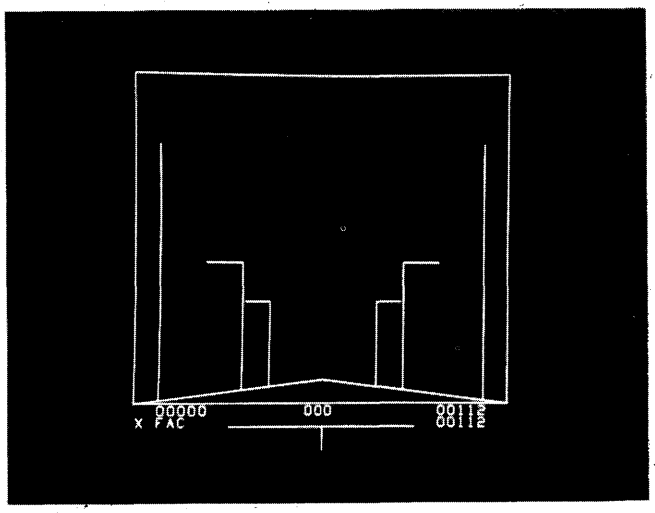


Figure 1—Perspective view of "country road" as seen through front windshield of rocket ship at rest. Three L-shaped telephone poles are shown on either side of the road. *Numbers displayed below picture:* center number is rocket speed v/c along road as a fraction of the speed of light; on the right, lower number is reading of clock carried in cockpit, upper number is time read off adjacent road clock; number at left is count of telephone poles passed. Light pen controls: operator sets forward or backward acceleration on horizontal line at bottom of picture; letters at left: X means "start again"; F means "freeze"; C means "coast without acceleration"; A means "go back to accelerating."

2. *Terrell rotation effect.*

The cross pieces of the poles appear to rotate about the pole as axis as the poles approach the observer (Figure 2). This effect is a rather-more-complicated result than (1) of the relative time delay of signals from different parts of the pole. The effect was analyzed by Terrell[1] in 1959, more than fifty years after

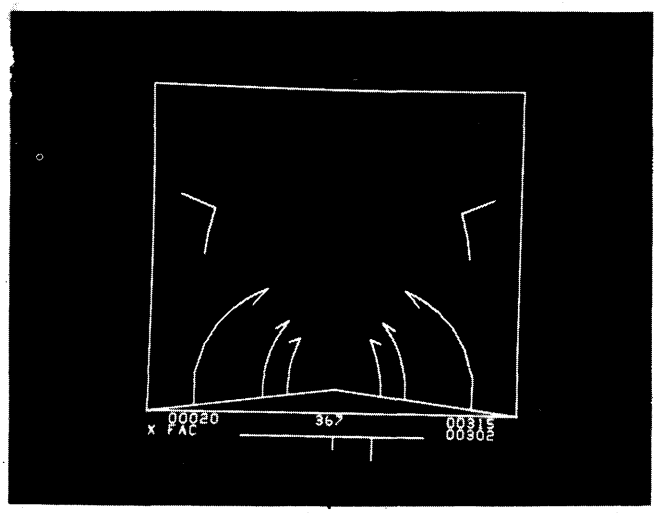


FIGURE 2—Rocket moves down the road with speed v/c = 0.367. Acceleration is positive. Poles appear to bend backward down the road; crosspieces show Terrell Rotation Effect.

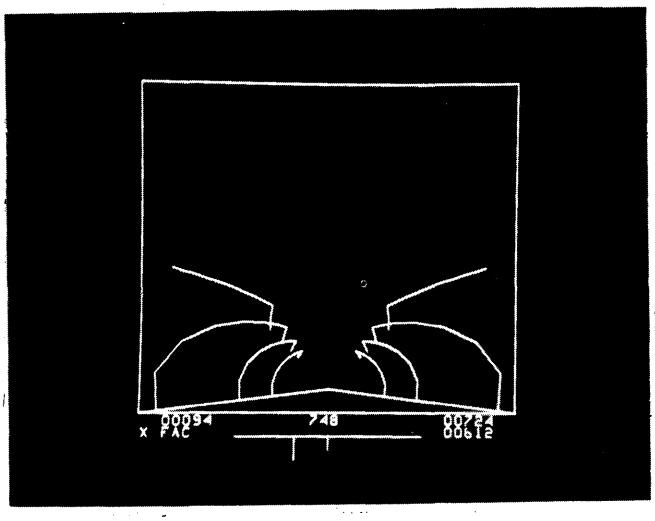


FIGURE 3—Rocket moves at velocity almost three-quarters of the speed of light. Acceleration is negative. Note different readings on rocket clock (lower number on right) and passing road clocks (upper number on right)—the "time dilation effect."

relativity was propounded.[2] This effect results automatically from the transformation equations and was not specially programmed.

3. *Time dilation.*

The lower number of the pair on the right of Figure 3 is the time measured in the cockpit of the rocket car, in arbitrary units (about three units per second). The number above is the time, in the same units, read off a series of road clocks, synchronized in the road frame, as each in turn passes by the rocket. The rocket clock always shows a lower cumulative reading than the adjacent road clock. By taking a round trip down the road and back while watching the pole counter at the left of

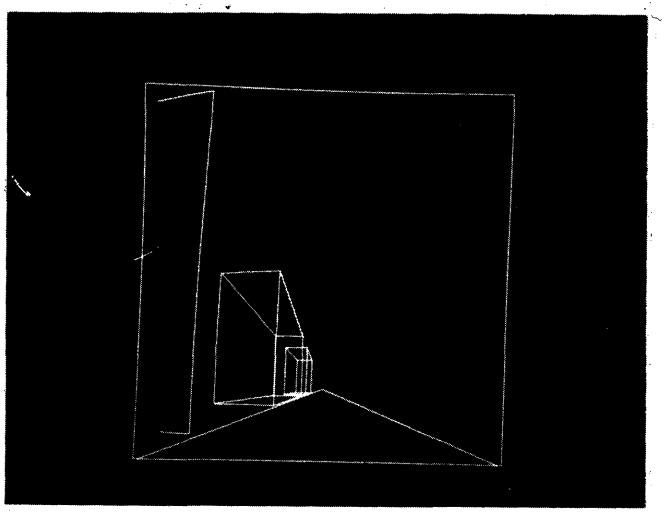


FIGURE 4—Display of cubes along one side of road; rocket at rest.

the figure, the operator can see the two clock readings to demonstrate the "twin paradox" according to which a traveling twin who moves at high velocity is younger on his return than his stay-at-home-brother.

4. *Non-addition of velocities.*

No matter how high a rocket acceleration is set by the operator, the speed of the vehicle (expressed by the central numeral as a decimal fraction v/c of the speed of light) cannot exceed that of light. Instead, the speed with respect to the road is seen to increase more and more slowly as the velocity v/c approaches unity. This is true even though the acceleration of the rocket maintains its constant value ("constant proper acceleration"). In a real rocket, the passengers would continue to be pressed against the backs of their seats with a constant pressure—a pressure not reproducible with a computer display! One way to describe this effect is to say that velocities do not add: the second-by-second increase of rocket velocity in the local frame is not equal to the incremental change of velocity of the rocket in the road frame.

5. *Surprising changes in aberration angle.*

When the rocket is first accelerated at a high rate from rest *forward* along the road, the poles appear to move initially *away* from the observer instead of toward him as one expects from everyday experience. This reresult was unanticipated and was initially perplexing to those who developed the display. It is explained in terms of aberration angle: At high velocity, objects are observed in directions different from those at which they would be seen if the observer were at rest. If the acceleration is great enough, the velocity change modifies the aberration angle to a nearby pole faster thandisplacement down the road can displace the pole in the opposite direction.

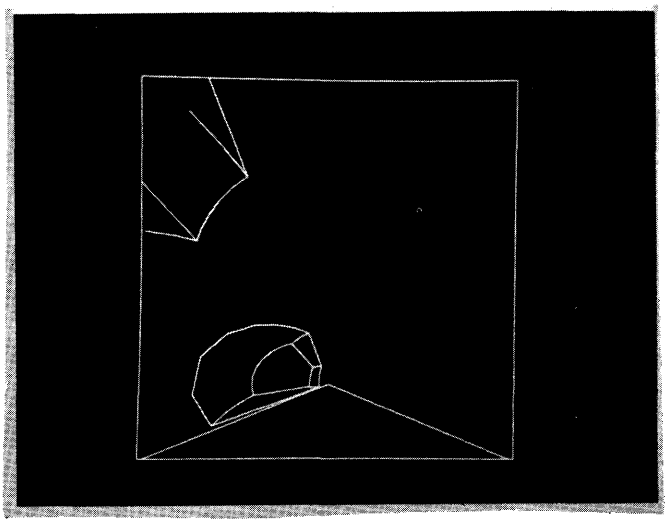


FIGURE 5—View of cubes at high rocket velocity. Leading face of distant cube is turned around, so we look at its back or inner side.

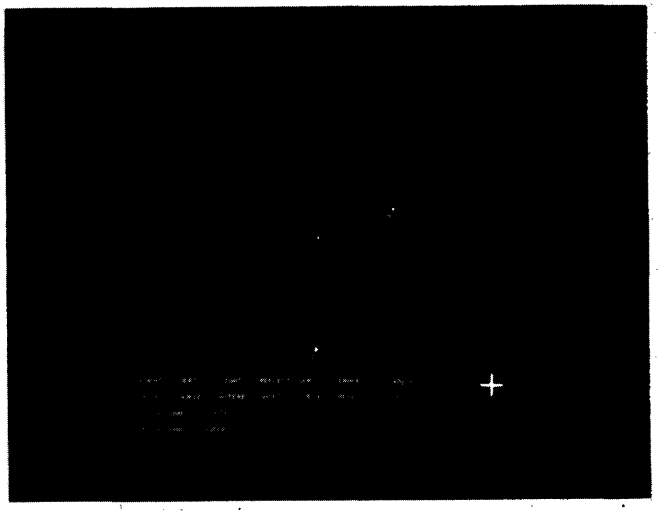


FIGURE 6—Spacetime diagram display. Horizontal axis represents a single space dimension; vertical axis represents time dimension. Point A shows location of event in spacetime (coordinates of A displayed on lower right in identical units, such as meters). To place a point-event, tracking cross on right is moved by light pen to desired location and button EVENT touched to create labeled point. This diagram is drawn for the "original" reference frame, so speed v/c = 0 on lower left.

Figures 4 and 5 show a similar display with cubes on one side of the road instead of telephone poles on both sides. The cube display is destined for film loop use and not as an interactive program.

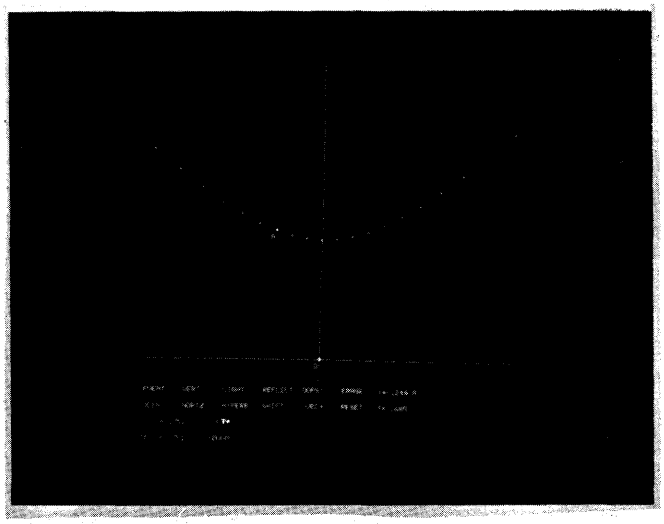


FIGURE 7—Spacetime diagram locating event A of preceding figure for rocket observer moving at speed v/c = 0.751 relative to "original" frame. Transformation is accomplished by holding light button + on symbol − LT+. Note altered space and time coordinates as observed in rocket frame. "Invariant hyperbola," along which event A slides during transformation, has been added using HYPERB button.

We must not leave these relativity pictures without acknowledging that they are deficient in at least two respects: at high velocities the *colors* and *intensities* of observed objects would vary rapidly as they approach and pass the observer. These changes in color and intensity are not reproduced in the display programs presented above.

Spacetime diagram

A more analytic manipulation of the results of special relativity is presented in the spacetime diagram display (Figure 6). This display demonstrates how the x (space) and t (time) coordinates of an event as observed in one frame of reference are related to the corresponding x′ and t′ coordinates of the same event are observed in a second frame of reference in high uniform motion with respect to the first frame. The horizontal axis in Figure 6 is the x-axis. The vertical axis is the t (time) axis expressed in units of length by plotting ct. The location of a point in the x-t plane (such as point A) gives the space and time coordinates of that event (see coordinate reading for point A in lower right portion of diagram). Coordinates of events in all frames are measured with respect to some common agreed-upon "reference event" (or "starting gun") taken by definition to occur at the zero of time and space (event 0 at the origin of Figures 5 and 7).

Figure 7 shows the same event A and its coordinates as measured with respect to a rocket reference frame moving at a speed v/c = 0.751 along the x spatial direction with respect to the original reference frame (value v/c recorded automatically in lower left corner of display in Figure 7). The transformation of the coordinate

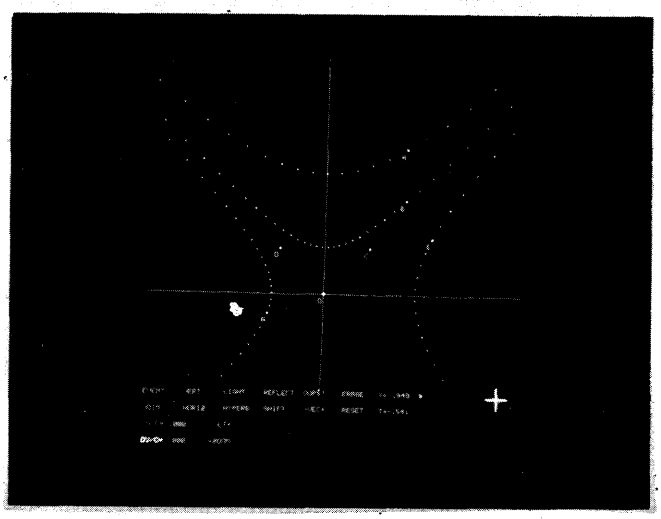


FIGURE 8—A field of point-events, showing some of the constructions available on the display. Light cone through origin event 0 divides spacetime into three kinds of regions with respect to this origin event.

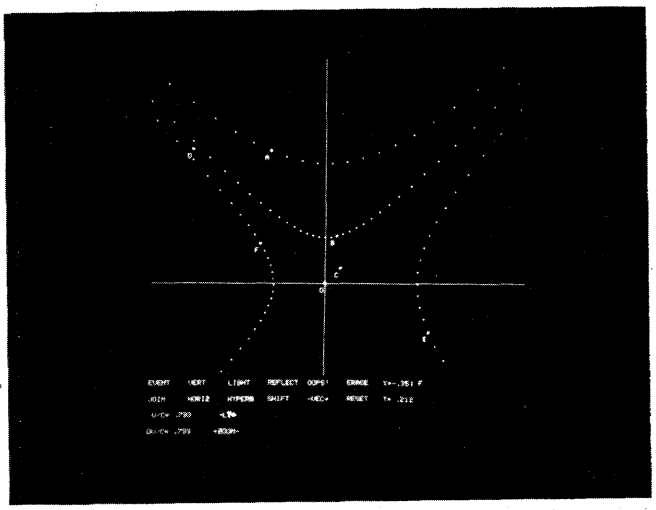


FIGURE 9—Diagram of Figure 8 transformed to rocket frame. These two figures, taken together, illustrate the Doppler shift, time dilation, relativity of simultaneity, relative synchronization of clocks, and invariance of the spacetime interval.

plot is accomplished by touching with the light pen the "Lorentz transformation button" (LT+) on the display. The relative velocity between frames increases continually as long as the "button" is touched with the light pen. The reverse transformation back to coordinates in the original reference frame is accomplished by pushing the *minus* sign in the symbol "— LT+ ".

Event-points are placed on the display by using the light pen to move the tracking cross to the desired place on the screen and then touching the "EVENT" button. Up to 16 points can be displayed on the screen simultaneously (Figure 8). Each new point is automatically labeled with a letter in alphabetical order. Coordinates of any single event are displayed when the cooresponding display point is touched by the light pen. When a transformation is made from one frame to another, the entire field of points is simultaneously transformed, each to its new coordinate position (Figure 9).

Other light buttons are used to erase selectively (ERASE), to clean the slate (RESET), to contract and expand the scale of the display (ZOOM), to join events with lines (JOIN), to draw through any event: an invariant hyperbola (HYPERB) or crossed straight lines at 45 degrees from horizontal (LIGHT) or a horizontal line (HORIZ) or a verticle line (VERT). The respective space and time coordinates of two events can be added to create a third event (ADD).

Using these commands, one can investigate a wide variety of the properties of spacetime including:

the world lines of particles and light flashes—their paths through spacetime

invariance of the interval $(ct)^2$—x^2 that separates two events

the light cone as a partition in spacetime (invariance of the speed of light.)
regions of spacetime: timelike, lightlike and spacelike relations between events
clocks using light pulses
Doppler shift
time dilation
Lorentz contraction
non-additivity of velocities

The components of momentum and the relativistic energy of a particle transform from one frame to another in the same way that the space coordinates and time of an event transform. This means that a point on the spacetime diagram can be used to represent the x-momentum (horizontal dimension) and energy (vertical dimension) of a particle. Using this interpretation of the diagram, the operator can analyze one-dimensional collisions between particles. In particular the REFLECT button reflects all points about the verticle axis. This can be used to represent the exchange of x-momentum of two particles in an elastic collision as observed in the center-of-momentum frame.

The spacetime diagram is especially versatile in that the student can both manipulate it to get a qualitative feel for relativistic phenomena and also read numbers from the figure accurate to three places. In this way he can pass easily between the analytic and intuitive aspects of learning relativity.

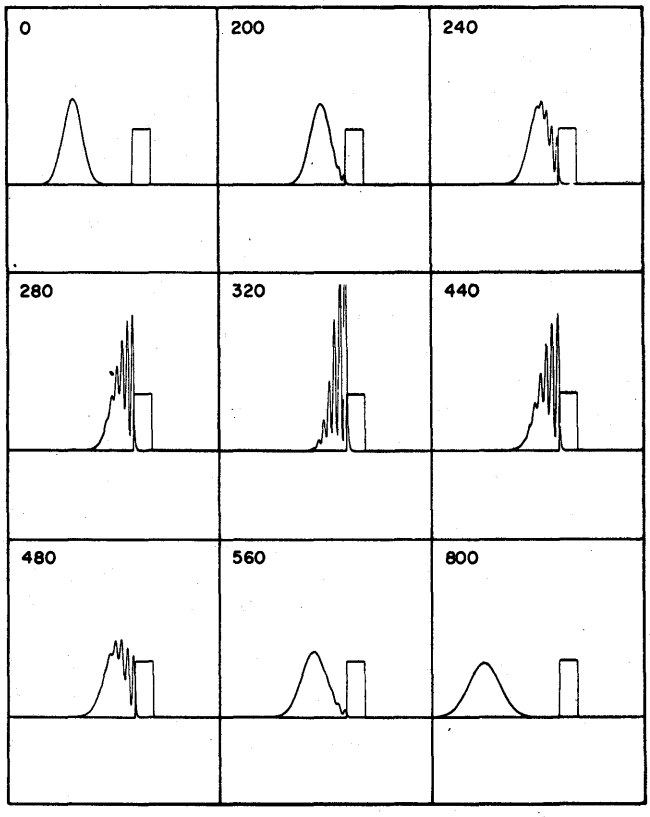


FIGURE 10—Gaussian wave packet scattering from a square barrier. The average energy is *one half* the barrier height. Numbers denote the time of each configuration in arbitrary units.

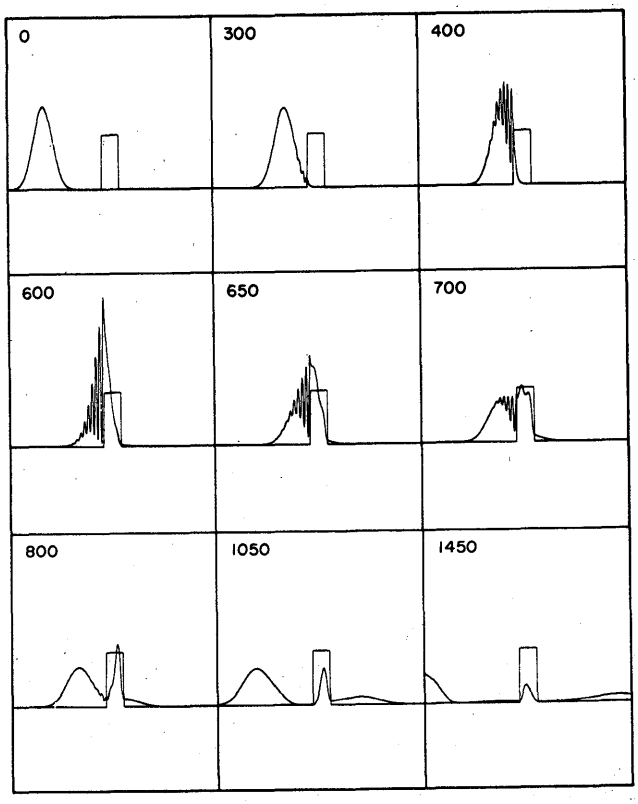


FIGURE 11—Gaussian wave packet scattering from a square barrier. The average energy is *equal to* the barrier height. Numbers denote the time of each configuration in arbitrary units.

Quantum physics

One-dimensional scattering

In a quantum mechanical description of nature it is impossible in principle to describe the instantaneous position of a moving particle with complete precision. The dynamical theory provides a probability distribution as a function of position that evolves in time.* This is to be understood in terms of the real world as the cumulative result of a succession of a large number of independent identical experiments, each one of which by itself is unpredictable. One can detect both classical

*The equation to be solved is the time dependent Schroedinger equation. In general we solve this partial differential equation in position space for the complex function $\psi(x,t)$ at each time t for somewhere between 1000 and 2000 values of x. Except where otherwise indicated the absolute square of this function is the plotted curve in the figures. Interested readers may find the computational techniques described in Am. Jour. of Phys. *35*, 177 (1967).

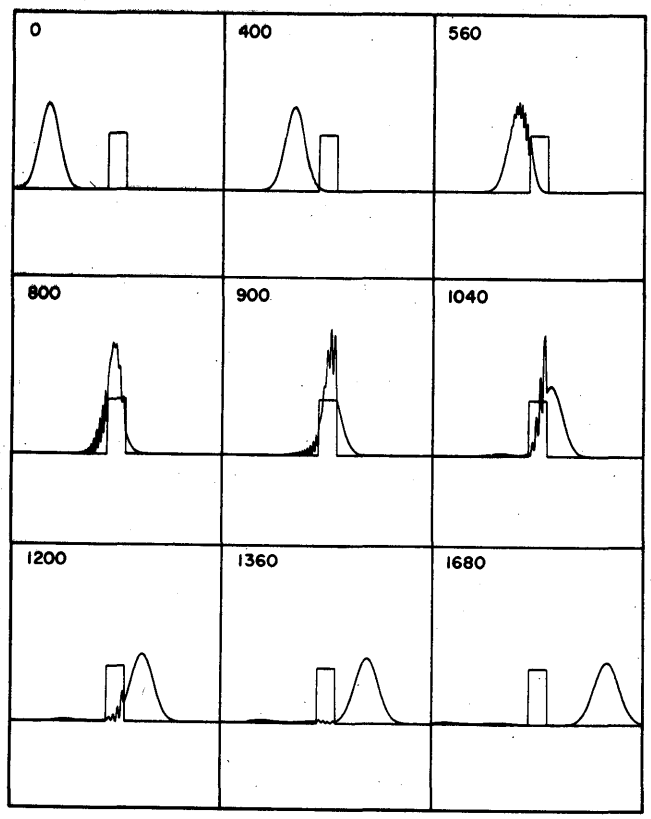


FIGURE 12—Gaussian wave packet scattering from a square barrier. The average energy is *twice* the barrier height. Numbers denote the time of each configuration in arbitrary units.

and non-classical patterns of behavior in these situations. For example in the first scene of the film entitled "Barriers (see Figure 10), one observes the probability distribution for a particle that is approaching a barrier with an energy that is not sufficient to allow it to be transmitted. During the most violent part of the scattering the probability distribution penetrates the region of the barrier. This corresponds to a finite probability at that time of finding the particle in a region that is inaccessible to it in a classical description of nature. On the other hand, toward the close of the first scene one sees the wave packet reflected from the region of the barrier in much the same way that a classical particle would bounce off a wall. The viewer will note, however, that the probability distribution has widened substantially since the beginning of the process. This, too, is strictly quantum effect.

The second scene in this film (see Figure 11) shows the quantum description of a particle approaching a barrier with energy just equal to the barrier height. In this case one would expect on the basis of classical physics that the particle would enter the region of the potential and lose kinetic energy until finally all its kinetic energy is expended in mounting the barrier, where it would then find itself in a region of no force with all its kinetic energy gone and, therefore, would remain in the region of the barrier forever. The film shows the quantum mechanical analog of this special case. Sure enough, a probability packet remains in the region of the barrier, at least for a long time. In addition, however, one observes transmitted and reflected packets representing the chances for transmission and reflection when many experiments are carried out. This "splitting up of the incident packet" due to the fundamentally statistical character of the quantum picture is foreign to classical mechnics.

In the final scene (see Figure 12) the particle has more than enough energy to overcome the barrier, but we find, nonetheless, a small but significant probability that the particle will be reflected.

Thus far we have displayed the time development of probability in *space*—the so-called "configuration space representation" of events. An equivalent method for describing an encounter of a particle with a barrier is the "momentum space representation"—that is, in terms of the momentum distribution of the probability packet. Quantum mechanics shows that the momentum space representation is every bit as good as the configuration space representation; each one carries all the information obtainable about the probable results of experiment.**

Most students have their first exposure to quantum mechanics in configuration space and, as a result, obtain a somewhat lopsided view of quantum mechanics. The next film, entitled "Momentum Space" portrays a collision as described by quantum mechanics in configuration space *and* in momentum space, in order to contrast the two and to see how the same physical event manifests itself in different representations. The first scene depicts the evolution in time of the position probability distribution of a particle as it approaches a region in which it will feel an attractive force (a potential well). After the "event" we observe a substantial probability of reflection. The second scene in this film depicts the evolution in time of the momentum probability distribution. Momenta corresponding to motion to the right in position space are here plotted to the right of the center line and momenta corresponding to leftward motion in position space are plotted to the left.†

**The computer displays filmed here present squared magnitudes of momentum and configuration functions. Information about complex phase (necessary to demonstrate the complete equivalence of momentum and configuration representations) is thus masked.

†The function here plotted is the absolute square of the Fourier transform of the solution to the dependent Schroedinger equation in position space. Only the recent availability of the Cooley-Tukey algorithm for Fourier Transforms as a standard library subroutine makes this computation a feasible one.

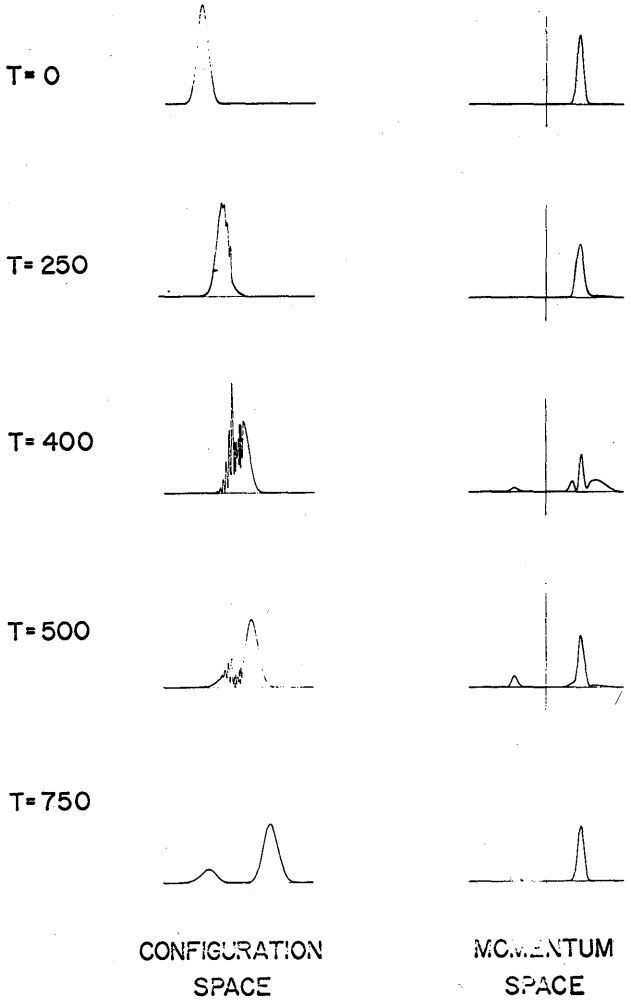


FIGURE 13—The probability distributions in configuration space and momentum space at corresponding instants of time for the scattering of a Gaussian wave packet from a square well. The particle energy is equal to one half the magnitude of the well depth. The vertical scales are arbitrary. See A. Goldberg, H. M. Schey and J. L. Schwartz, American Journal of Physics, *36*, 454, (1968).

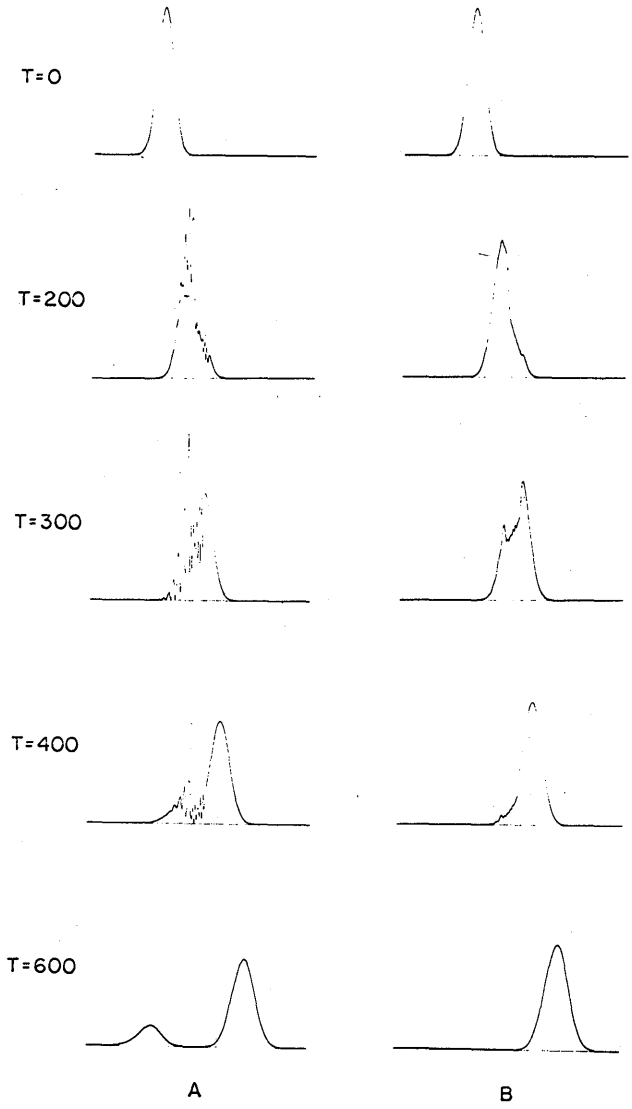


FIGURE 14—The position probability density at selected times for scattering from a square well and a Wood-Saxon well. See A. Goldberg, H. M. Schey, J. L. Schwartz, American Journal of Physics, *35*, 777, (1967).

As long as the particle travels freely toward the region of the force the distribution of momenta remains unaltered, an aspect of quantum theory that Newton would have been content with. As the particle begins to feel the effect of the force, the distribution of momenta undergoes severe distortions, due in part to the increased kinetic energy (and therefore increased momentum) as the packet passes over the potential well. After a short while the momentum distribution begins to develop a peak centered about some average leftward momentum, the momentum of the reflected packet. When the interaction between packet and well is completed, the momentum distribution again remains constant while the probability packets in configuration space sail majestically offstage. Figure 13 shows these two descriptions of the collision at five instants of time.

The barriers encountered in real quantum collisions are atoms and nuclei. These potential barriers and wells are not square in shape but have a potential that changes smoothly with position. In this respect the quantum displays shown thus far do not correspond to reality. Happily, the computer makes its calculations using a point-by-point specification of the potential. Therefore interaction with a potential with "rounded corners" is no more difficult to program and compute than an interaction with a square potential. The result of rounding the corners of the barrier is a dramatic reduction in the probability of reflection from the well. In analogy to electromagnetic wave phenomena, one may say a more gradual "change in impedance" at the sur-

face of the well results in less reflection of the incident "wave."

The effects of softening the barrier walls is presented in the next film, entitled "Edge Effects." The results of progressive rounding of the corners is shown in three stages. Notice that during interaction the structure of the probability is much smoother for the rounded barrier than for the square one, also that the probability for reflection is less for the rounded barrier. Figure 14 contrasts the scattering from a square well with the scattering from a well which has a diffuse surface.

CONCLUSION

It is not very difficult to propose additional aspects of modern science that are difficult to visualize and for which this kind of presentation would be of some benefit. In fact, under development now at the Computer Film Project of the MIT Education Research Center are efforts in physical chemistry, electrodynamics, and molecular biology. Each of these, we have reason to believe, will enhance the understanding of students as well as broaden the range of tools available to the instructor.

Some of the displays discussed here have been used informally with students and professors. Others have recently been made available commercially as film loops As yet the use has not been widespread enough to allow us to draw firm conclusions about their effectiveness in developing intuition about relativity and quantum physics. What little evidence we have leads us to want to continue developing these materials. Students appreciate the opportunity to focus on the phenomena without being burdened by the constant onslaught of mathematical symbols. Some of our most gratifying moments occur when professional physicists of the highest caliber respond to one of these films with wonder and delight and indicate that, although they have grown up with the field, they have never seen the behavior of the system revealed this clearly.

REFERENCES

1 J TERRELL
Physical Review 116 1041 1959

Art, computers and mathematics

by CHARLES CSURI and JAMES SHAFFER

The Ohio State University
Columbus, Ohio

The computer

The computer is having an implosive effect upon the way we deal with a variety of problems. As an extension of man's senses, computer technology can provide an exciting new potential for the creation of art. The frontiers of knowledge in computer research suggest a new approach to problem solving in the arts. With a computer the artist can now deal with different variables in his decision making process than with conventional methods. For example, it is possible to put into the memory of the computer a color representation of a landscape. This landscape can be simulated on a graphic console. Then with computer programs which implement mathematical functions, the artist can watch the effects of wind velocity, temperature and factors which involve the amount of daylight upon his landscape. He can also observe data which are generally unavailable such as the effects of molecular structure, weight, mass and time upon the landscape. In his decision making process, the artist can rely on non-visual cues as well as visual cues. He can modify many more parameters in the total landscape environment to create a work of art than by conventional methods.

The artist and the modern environment

The frontiers of knowledge in computer research offer a glimpse into the future role of the artist. At M.I.T. and Stanford University considerable research is in progress which attempts to deal with artificial intelligence programs. Some researchers suggest that once we provide computer programs with sufficiently good learning techniques, these will improve to the point where they will become more intelligent than humans. Suppose we have a machine which has stored in it, a knowledge of art history, theories of philosophy and aesthetics, in fact, the intellectual history of man. Every known technique about painting, sculpture and the computer graphics will also be stored in the computer, not to mention an ability to make judgments more logically than man. What happens to questions about art when there is a dialogue between man and such a computer program? What becomes the problem? Who is the artist? What are the implications for man? It is both terrifying and exciting at the same time.

The emergence of new forms and media in contemporary art indicates the artist's deep involvement in twentieth century technology. Robert Rauschenberg creates forms which combine objects such as automobile doors and heating ducts with sophisticated electronic devices that create sounds and smells. Artists Tom Wesselman and George Segal utilize plastics and neon lights as well as radios and TV sets. Kinetic art has produced a vast array of objects which move or vibrate. Current artists have used practically every product in our society to make art. In the tradition of the craftsman, these artists seem to be more concerned about materials and technical processes than any underlying scientific concepts which produced these products.

The computer which handles fantastic amounts of data for processing brings the artist close to the scientist. Both can now use the same disciplines and knowledge in different ways. For the first time, the artist is in a position to deal directly with the basic scientific concepts of the twentieth century. He can now enter the world of the scientist and examine those laws which describe a physical reality. The artist can enter

a microuniverse of science and alter parameters to create a different kind of artistic world. In a highly systematic and disciplined manner he can deal with fantasy and imagination. One example of the use of a scientific concept for artistic purposes is the well-known Lorentz transformation. It is a theory of special relativity which is a scientific explanation of the apparent distortion of a form as it approaches the speed of light. It would be interesting to see what happens graphically to a drawing of a turtle or a hummingbird, which can be used as input to the computer, as it approaches the speed of light. The artist may be interested in the absurdity of such an idea and it may give him a different kind of form. He may enjoy the contradiction of a turtle traveling near the speed of light.

The artist need not necessarily stop at the parameters defined by a transformation in relativity. He can arbitrarily declare that objects will move at a speed which is five times that of light (provided the mathematical equations do not degenerate). In fact, he can, with the computer, take a broad variety of well-known equations which describe our physical universe and change the parameters. He can create his own personal fiction.

It is quite apparent that the computer artist will have contact with the scientist. The computer and the mathematical disciplines required to solve problems, artistic and scientific, make for a common ground. Those fields will be brought closely together, and both will benefit by the dialogue made possible through computer science.

The artist, when involved in the creative process, feels free to deal with experience in any terms which can express his conception. He is not restricted to the rules required of the scientist to express a reality. In a sense, one might say that he takes the many parameters of ordinary experience and changes them to express his imagination. He creates a new universe. Since his purpose is to make art, the artist is not bound by the laws which account for the physical world. On the other hand, the scientist is also interested in realities. He explains the behavior of physical phenomena and usually verifies it in mathematical terms. The famous mathematician and writer Jacob Bronowski summarizes the similarities and the differences between artist and scientist in the following statement:

> "The creative act is alike in art and in science; but it cannot be identical in the two; there must be a difference as well as a likeness. For example, the artist in his creation surely has open to him a dimension of freedom which is closed to the scientist. I have insisted that the scientist does not merely record the facts, but he must conform to the facts. The sanction of truth is an exact boundary which encloses him, in a way which it does not constrain the poet or the painter. . . ."*

Mathematics and the arts

Some artists throughout history have been intrigued with the possibility of making use of mathematics in their art. The renaissance idea of virtual space (the imitation of 3-D space) was generated by mathematical formulae for linear perspective. Modern artists such as Paul Klee, Moholy Nagy, Naum Gabo, and Antoine Pevsner have used simple mathematical systems to analyze and develop forms. In the past, mathematics has been given limited application as a tool for the discovery of aesthetic form because the techniques employed were slow and extremely time-consuming. Traditional ways of solving problems of measurement and plotting were too awkward and mechanical for artistic application. As a consequence, the artist's concept of structure was limited by what he was able to design or draw by hand. If an artist was able to understand or use the traditional methods of mathematical analysis, he was quickly discouraged because of the many repetitive steps in the computation of a problem. What he could visualize was limited by traditional methods.

An artist can now make use of complex mathematical functions. With the advent of computers the artist has at his disposal the computational power to apply many mathematical transformations to a variety of images. By implementing mathematical functions through computer programs, and generating the results on a mechanical plotter or CRT display, the artist can pursue an orderly in-depth inquiry of visual form. He can examine closely related or widely divergent functions. The results can be evaluated artistically against the background of the mathematical functions involved. Above all, there are new form possibilities that can be generated by the computer. Artists are faced with the prospect

*From J. Bronowski *Science and Human Values*, 1956.

of a new medium to enable the exploration of visual ideas.

The interest here is in the use of mathematical functions to modify form in a variety of ways. There is no intention to define a particular mathematical theory of art, but simply to say that mathematics is a useful means to create many kinds of art. One artist may prefer to work with realistic images while another one may prefer abstract images. An artist may select visual information in such a way as to communicate qualities of surrealism or expressionism, and even a tragic or lyrical view of reality. Questions about artistic content involve artistic decisions. Mathematics per se does not make the art. One brings artistic criteria to the object and makes value judgments about it. This criteria must deal with the nature of aesthetic experience and those aspects which make it art. Mathematics and the computer only provide us with another means to make artistic inquiry.

In a broad sense certain modern educational theories, and the way in which mathematical strategies can be applied, are also relevant. Currently there is considerable discussion about "psychological sets" in learning. This concept of sets suggests that the artist is often the victim of his own "set producing tendencies." At times, he has great difficulty in breaking down his own biases in order to solve an artistic problem in a creative way. Because of his biases, the artist usually gets only slight variations on a basic structural theme. A mathematical orientation toward visual problem solving can enable the artist both to break down his biases and to express another range of solutions. Essentially this would depend upon the types of strategies which the artist employed to effect his original data. To say this another way, the computer's capabilities, if exploited by the artist, would be made to yield a greater number of alternatives more rapidily and efficiently. Within a short period of time a computer-cathode ray tube-plotter or graphic console capability can give hundreds of variations of a form.

It is conceivable that any given transformation can be of use in computer art. Special consideration seems appropriate for transformations with properties which enable an artist to visualize the approximate result of the transformation and to transformations which are easily adapted to machine computation.

Mathematical functions are found in almost every branch of mathematics and they can produce an unlimited number of variations for the discrete set of coordinate points associated with a line drawing. Successive applications of the same or different functions can produce interesting results. In the example of SINE CURVE MAN there was a successive application of a sine curve function in which the amplitude was increased each time. Numerous possibilities exist if one uses a multiplicity of functions to achieve a final transformation. Some of the applications include a sequence of functions in which many functions are used. The example PLUS SIGNS, includes functions which deal with orientation, size and distribution. The motion picture HUMMINGBIRD, which is in time, utilizes several functions simultaneously, each varying independently as in the "three bird scramble" sequence. Each bird was on a different mathematically defined path, while their size was changed sinusoidally—each at its own frequency. Also each image moved from an abstract picture to a realistic bird at an independent and non-uniform rate.

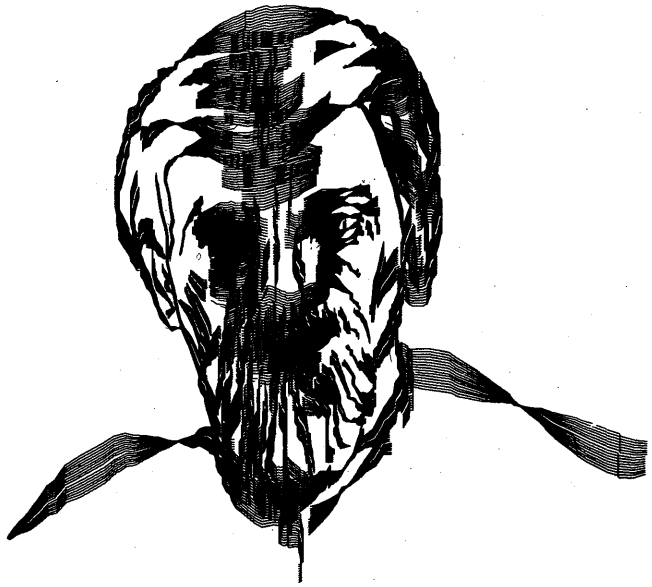


Figure 1—SINE CURVE MAN, 1967. A digitized line drawing of a man was used as the input figure to a computer program which applied a mathematical function. The X value remained constant and a sine curve function was placed upon the Y value. Given the X and Y coordinates for each point, the figure was plotted by the computer from $X' = X$, $Y' = Y + C^* \text{SIN}(X)$ where C is increased for each successive image.

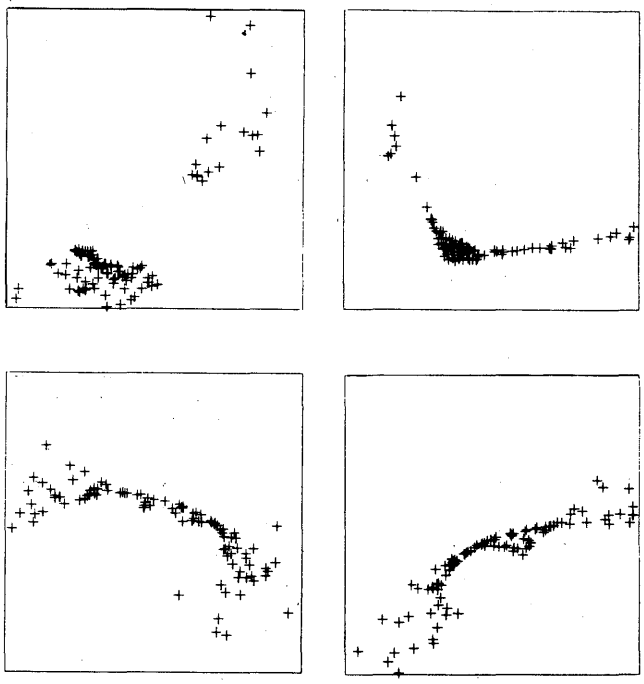
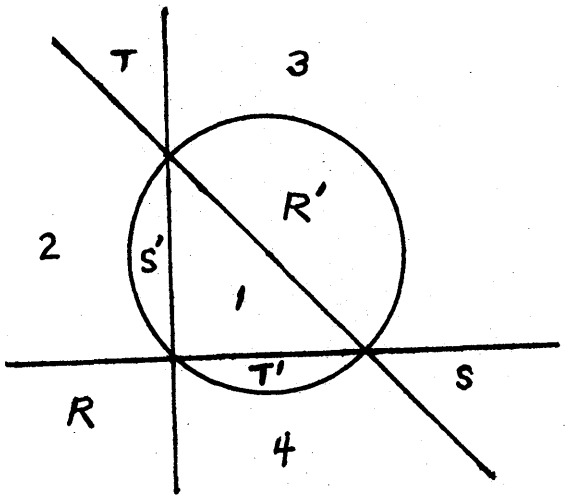


Figure 2—In figure 2 the equations $x' = y^* (x + y - 1)/(x^2 + y^2 - x - y)$ $y' = x^* (x + y - 1)/(x^2 + y^2 - x - y)$ cause the lettered primed and unprimed areas to interchange while the numbered areas map into themselves. To produce the accompanying figures rectangular areas were defined in which random coordinates were generated. After applying the transformation equations those coordinates which fell inside a second rectangle were accepted as the location for a fly or plus sign.



Applicable types of mathematical transformations

Transformations can be classified in many ways. One approach is to group transformations according to properties such as: those which preserve size and shape; those which preserve lines; those which preserve angles of intersection; those which preserve continuity; and so forth. If transformations are given by equations for coordinates of image points, the transformations can be classified by the form of the equations. Perhaps one might even try to develop a mathematical theory of art, and include a heirarchy of functions based upon ideas about periodicity, harmony, permutation, ratio, progression, and so forth. This would seem to presuppose a theory of artistic structure and one may find many problems in such a mathematical theory of art. There is no clear cut criteria on which to make a judgment about classification. What is important, however, is that the artist have a broad range of functions and computer programs available to him.

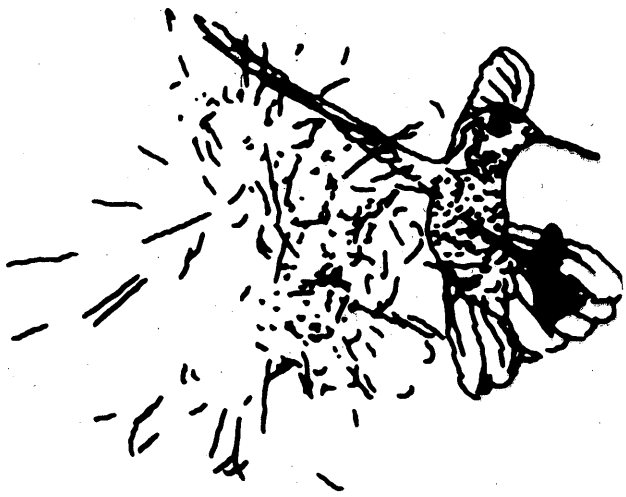


Figure 3—Computer film HUMMINGBIRD, 1967. One still frame from a computer animated movie.

It is quite true that an artist can probably imitate any image the computer can generate. Some images would be difficult, but not impossible. However, the question is not one of hand skills versus machine skills. The problem is one of conception, wherein the mathematical transformations made possible by the computer present a new dimension to art.

Research

During the past two years we have produced several hundred pictures by computer. Our work has involved color as well as black and white. We have developed many programs which make use of functions in trigonometry, coordinate sys-

tems, conformal mapping, n-dimensional geometry and randomness. In addition we have developed several programs which deal with special problems of graphic display by computer.

The technique we use is as follows: (1) An artistic drawing is made with line segments of points of the subject matter to be used. (2) The drawing is digitized line by line with the resultant coordinates punched into cards. (3) Decisions are made about the type of form modification and the mathematical steps required to accomplish it. (4) The mathematical algorithm is programmed for a computer which then generates the plotter commands. (5) At this point another decision is made about the color and line width for the transformation. (6) The transformed image is plotted on a CalComp 563 plotter.

Using a variation of the photo-nylon screen technique we have developed a method to transfer the plotter output to canvas and plastic sheets. This allows us to use a permanent non-fading paint to represent the image.

We have also completed a ten minute computer animated motion picture entitled HUMMINGBIRD.* The subject was a line drawing of a hummingbird for which a sequence of movements appropriate to the bird were outlined. Over 30,000 images comprising some 25 motion sequences were generated by the computer. For these, selected sequences were used for the film. A microfilm plotter recorded the images directly on film.

To facilitate control over the motion of some sequences the programs were written to read all the controlling parameters from cards, one card for each frame. Curve fit or other data generating programs were used to punch the parameter decks. We also built a windowing option into our plot subroutine.

Our most recent project is sculpture using a 3-axis, continuous path, numerically controlled milling machine. Mathematically generated surfaces such as the Bessel function were our first works. We have developed our own supporting

*Awarded a prize at the 4th International Experimental Film Competition, Brussels, Belgium, 1967.

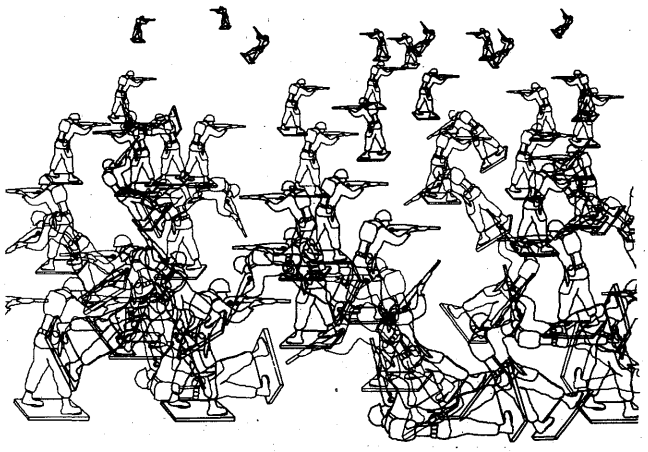


Figure 4—RANDOM WAR, 1967. A drawing was made of one toy soldier and this became the data deck. A computer program which generates random numbers is called a pseudo-random number generator. Such a program determined the distribution and position of 400 soldiers on the battlefield. One side is called the "Red" and the other one the "Black", and the names of real people were given to the program. Another computer program assigned military ranks and army serial numbers at random. The random number generator decided the following information and the computer made this picture with the casualty list. (1) Dead (2) Wounded (3) Missing (4) Survivors (5) One Hero for each side (6)Medals for Valor (7) Good Conduct (8) Efficiency Medals.

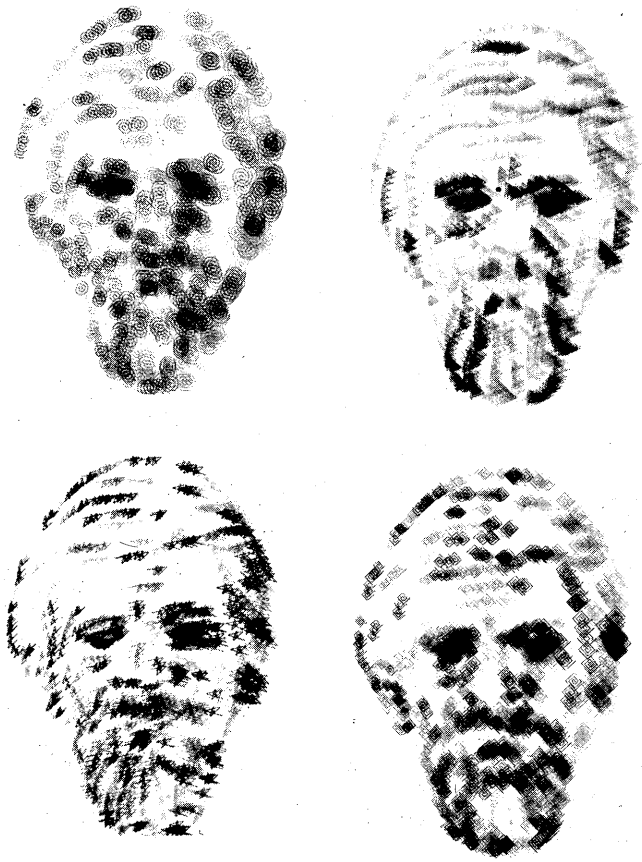


Figure 5—RANDOM COLOR DISTRIBUTION, 1967. A realistic line drawing of an old man was used as the data deck. A line drawing was transformed mathematically into a shaded image. A spiral, rectangle, triangle and star were the character symbols used to replace each line segment. The position of the four colors was determined by the random number generator. The size of each character symbol is a function of distance from a reference point outside the picture space.

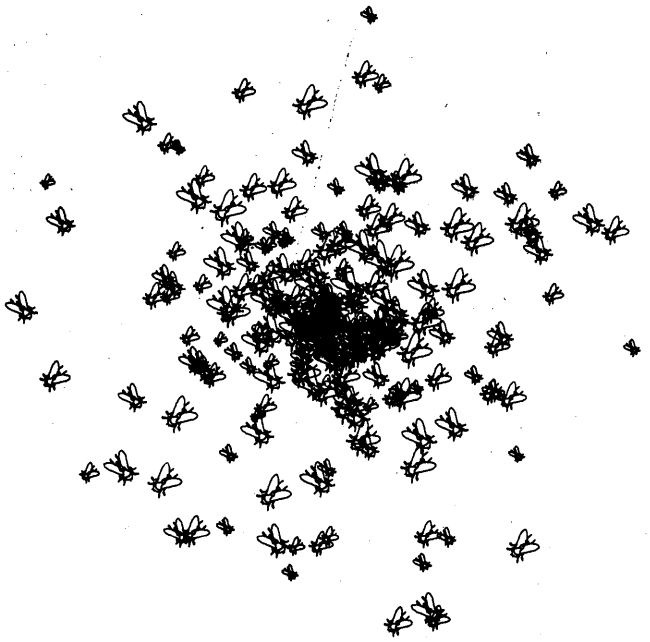
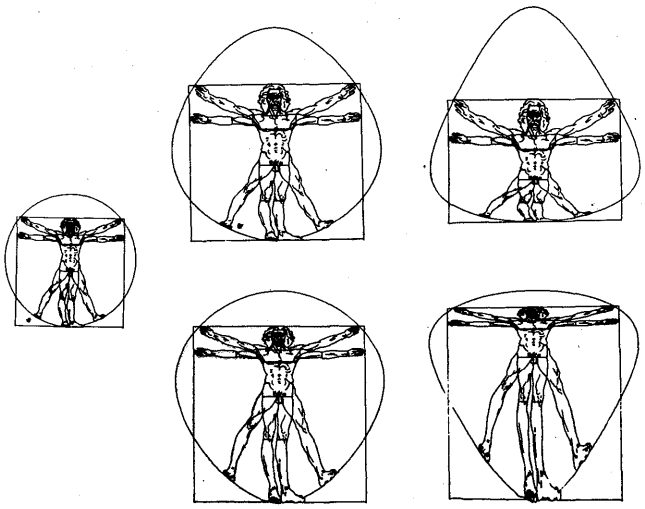


Figure 6—FLIES IN A CIRCLE, 1966. A computer program generates random numbers which determine the distribution of a specific number of flies in a series of 1″ concentric rings. Within predetermined limits the random number generator also decides the orientation and the size of each fly.

routines rather than using one of the special languages such as APT. The rigid program—part correspondence as in APT was not suited for our purposes. This problem lead us to develop a method which provides a more flexible input capability for general artistic use.

ACKNOWLEDGMENTS

Mr. Sam Cardman for his mathematical analysis and programming for numerically controlled milling and HUMMINGBIRD. Professor Leslie Miller for his ideas and concepts dealing with projective transformations. California Computer Products Company for the use of their 835 microfilm plotter. A special note of thanks to Mr. Arthur Francis and the Cincinnati Milling Machine Company for guidance and the use of their equipment for numerically controlled milling. The Ohio State University for its generous support of our research efforts.



After Leonardo's HUMAN PROPORTIONS

Figure 7—The program finds Y_{min} and Y_{max} (or X_{min} and X_{max}) for the input image. Then from λ_{min} and λ_{max}, read as input parameters, λ_i can be calculated for each Y_i. The mapping function is $Y'_i = K^* \tan(\lambda_i)$ where K is a constant calculated to yield a final figure at the desired height.

Figure 8—CIRCLE INVERSION, 1967. The input image is scaled to fit inside a circle of unit radius. Working in polar coordinates the transformed image is $\theta' = \theta$, $r' = 1/r$. A point at the unit circle's center transforms to a point at infinity while a point on the circle's circumference is invariant.

CAMP—computer assisted movie production*

by J. CITRON

IBM Corporation
Los Angeles, California

and

JOHN H. WHITNEY

Pacific Palisades, California

INTRODUCTION

In designing a language for computer assisted production of animated film sequences, a serious attempt has been made to avoid dependence upon the user's knowledge of mathematics, geometry, and programming logic. Such a user can be expected to have the sense and sensitivity of an artist in manipulating given geometric figures, so our first objective is to provide a general way to construct a wide variety of figures and then to provide manipulative functions for their spatial and temporal evolution. While the mathematical and logical program necessary to perform this processing may be complex, the language seen by the user must afford control over all the technical flexibility available in the program, but from the user's non-technical standpoint.

The approach we have taken is to minimize the number of types of statements in the language—basically there are three—and introduce numerous fundamental concepts which may be learned by experience with the aid of an interactive program.

Various approaches to hardware are possible, and so the language only depends on the ability of the computer to address a display device. If a cathode ray tube is used, a camera should also be attached under full control of the computer.

Our implementation consists of a program written in the GRAF language[1] to run on the IBM 360 with a 2250 display unit equipped with a program function keyboard. Mr. D. Bottles of the IBM Los Angeles Scientific Center has constructed a control box which connects to the function keyboard and a camera constructed by Mr. Whitney. This device allows the light circuits on the keyboard to activate the camera controls under computer direction to photograph images on the 2250 and advance the film frame, and also uses the key circuits as feedback sensors to advise the computer of the camera's status with regard to these controls.

Technical background

The list of figures which one would like to be able to generate is practically endless. Thus, a direct geometrical approach suggests an infinity of subroutines which can be singled out according to the figure classification requested. An alternate viewpoint would be to consider any desired patterns to consist of curve segments of limited size (they must fit on a film frame) and then attack the problem of creating generalized curve segments. This decreases the required number of figure generation routines tremendously and, in fact, a single routine can be devised in different ways which will prove quite adequate for constructing just about any desired pattern within the given hardware limitations.

The curve equation and its parameters

Behind the scenes, so to speak, of our program lies a single mathematical function embedded in an algorithm with a number of parametric controls. This function may be expressed as the polar equation:

$$R = A(\sin B\theta)^P.$$

Two points are specified in a 2-dimensional Cartesian

*Development was carried out at the Health Sciences Computing Facility of the Medical School, University of California at Los Angeles.

frame by four parameters—(X,Y) and (U,V)— to define the lower left-hand and upper right-hand corners of the computed picture. Theoretically, assignment of values to A, B, and P then produces the curve $R(\theta)$ which may or may not lie within the frame defined above. However, because of the nature of current display hardware, the values taken on by the variable θ must be enumerated in a discontinuous fashion so that if the discrete points $(R,\theta)_i$ are to be connected, they will be joined by straight lines. Fortunately, this provides many advantages and few, if any, handicaps since the eye cannot detect the discontinuous nature of the curvature of a segment whose points lie reasonably close together. This introduces the need for four more parameters. S and T are the first and last values taken on by θ. N is the number of intervals into which the range T – S will be divided. The connectivity of the points marking the intervals is given by the parameter, J. If J equals 0, only points are displayed while the internal ordering of the points is the same as for J equals 1. For J not equal to 0, θ runs from S to T in N steps but ordered by skipping J points ahead and cycling in the given range. If J and N have a common submultiple, fewer than N points are used.

The use of the parameters A and B is clear from the equation, but the exponent, P, requires further clarification. If P were restricted to integer values (which it is not), the odd and even integers would govern two different regimes of (R,θ). Even values for P would cause all R values to have the same sign as A, regardless of the sign of $(\sin B\theta)$, whereas odd P values would give R the sign of the product of the signs of A and $(\sin B\theta)^P$. Further ,negative values for P destroy the main attribute of the sine function—its boundedness! Our algorithm makes available both the odd and even integer regimes in a continuous way for all real values of P (not just integers) and avoids the negative P problem by using the sign of P to specify the regime. Negative P assigns the sign of A sin $B\theta$ to A $|\sin B\theta|\ ^{|P|}$ while positive P mimics the even integers. P set to zero causes R to equal A for all values of θ.

The algorithm uses two additional parameters for remapping the $(R,\theta)_i$ points to $(R,\emptyset)_i$. A rotation, Q, is added to all θ values and the interval containing the rotated points (S + Q to T + Q) is stretched (or shrunk) linearly by a factor, W, so that the points run from S + Q to WT + Q in units of J(TW – S)/N. If W is equal to zero, the program uses the original interval (S + Q to T + Q).

Now, given the point (R,ϕ), one would expect its Cartesian coordinates to be expressed by (x,y) where:

$$x = R\cos\phi$$

$$y = R\sin\phi.$$

However, we have added three additional pairs of parameters to allow individual x and/or y translations and scalings:

$$x = C + ER\ (\cos\phi)^H$$

$$y = D + FR\ (\sin\phi)^Z$$

where the signs of H and Z are handled like the sign of P. Finally, a rotation with respect to the center of the film frame is specified by the parameter O.

In all then, there are 20 parameters which may be used to define a single curve. We refer to each set of 20 parameters as a curve so that a number of curves may make up a single figure. Default values are predefined for the parameters so that only "distorted" parameters need to be specified for any curve.

Options connecting curves

Beside the ability to control multiple exposures, there are three options which provide a manifold of techniques for "combining" curves. We refer to these as the add, link, and concatenate options and use the symbols +, —, and | to represent them in equations. Further, we label the polar coordinates of the computed points with a curve index, c, and a point ordering index, i, which runs from 1 to n(c). Thus $(R(^2_4), \phi(^2_4))$ represents the fourth point in the second curve. We are now in a position to describe the curve combining options in a precise way.

If curves (1) and (2) are added in that order

$$\text{i. e. } (1) + (2)$$

the computed points have coordinates

$$(R(^1_i) + R(^2_i), \phi(^2_i))$$

while

(2) + (1) results in plotting the points:

$$(R(^1_i) + R(^2_i), \phi(^1_i)).$$

Thus we see that the add option is not commutative but results in using the angular coordinates of the last curve specified. Of course, if the two curves have the same number of points and the same range of θ and identical point ordering (that is identical values for N, J, S, T, and W) as well as the same frame scaling parameters, this operation is equivalent to a direct algebraic addition of the two curves and their apparent order is immaterial.

Linking two curves causes points with the same

ordering index to be connected by straight lines. Thus (1) – (2) results in a pattern of lines connecting points $(R(^1_i), \phi\ (^1_i))$ to points $(R(^2_i), \phi\ (^2_i))$ with connections between points having identical i indices. One of the many uses of this option is to connect two curves which are identical in all respects except for position on the frame and possibly size thus providing 3-D "wire frame" perspectives.

Concatenation of curves orders their points in a sequential manner for treatment as a single curve in subsequent add or link operations. That is, (1) | (2) effectively produces a single curve whose point indices run

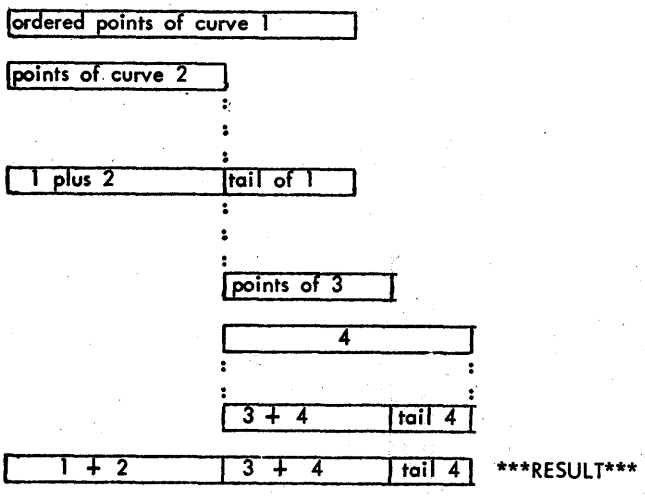
$$(^1_1), (^1_2), \ldots, (^1_{n(1)}), (^2_1), (^2_2), \ldots, (^2_{n(2)}),$$

giving a total of n(1) + n(2) ordered points.

Two special counters affect the option algebra strongly. One marks the point at which the last concatenation started and the other marks the beginning of the previous link. Add is carried out starting at the position of the concatenation counter so that

$$1 + 2 \mid 3 + 4 \text{ implies } (1 + 2)|(3 + 4).$$

Concatenate uses the last point of the last curve as a starting point so that, in the above example, the final result depends on the relative lengths of the point strings 1 and 2. If there are fewer points in curve 1 than in curve 2 (or if the number of points is equal in both), the figures contain all computed points. But if curve 1 has more points than curve 2, there is a resultant loss of some points. This is most clearly seen graphically:



Note that the points of curve 1 which overlapped in the add operation with curve 2 are simply lost in the result. This could be avoided if desired by specifying 2 + 1 | 3 + 4.

Link is always carried out from the beginning of the previous link, providing there was one. Thus 1 – 2 + 3 | 4 – 5 is expressed logically as 1 – [(2 + 3) | 4]–5 where the three curves in the brackets are treated as a single string of points for the two link operations.

It should now be clear that the three options provide much flexibility over and above that afforded by the 20 parameters per curve.

Temporal control

So far, our definition of a curve seems fairly straightforward because of the assumed static nature of the parameters. However, two different systems for control of the time rate of change of each parameter are available in the program algorithm. Thus a curve can be an extremely complex function of time and may appear on a film frame to be identical to some other curve at one or a few separate instants of time. This ability requires that at least one and possibly even all twenty parameters be specified by more than just a single numerical value. Each parameter now consists of a list of up to 16 items, and the interpretation of these list entries depends on which temporal control system is in effect.

For the type 1 rate controls, the list contains three kinds of entries—values, rates, and relative times—grouped in that order. The use of relative times allows a curve to evolve in a given manner during the time spanning any number of frames. Such evolution may occur in separate sequences or overlapped in the same sequence.

To explain the rate entries, we'll call V_1 the initial value of the parameter at time t_1 and V_2 the value to be reached at time t_2. The value V at any intermediate time T is computed according to the rate entry in the list. If the rate is zero, the parameter goes from V_1 to V_2 linearly in time—that is:

$$V = \frac{T - t_1}{t_2 - t_1}(V_2 - V_1) + V_1$$

If the rate is a positive number, r, an acceleration is made so that the value of the parameter at any time other than t_1 or t_2 is less than the linearly computed value. The equation used is:

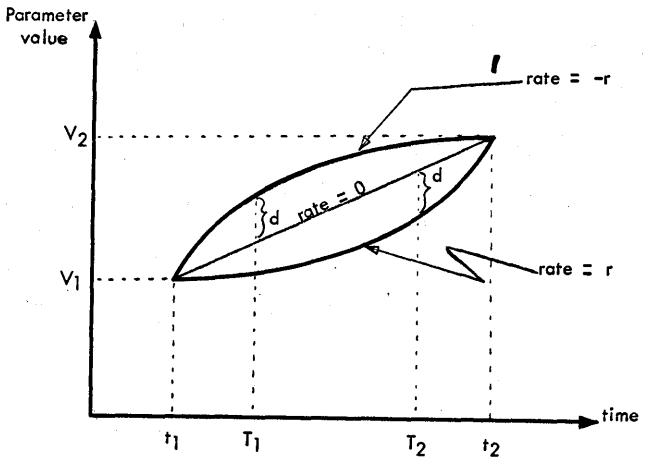
$$V = \left[\frac{T - t_1}{t_2 - t_1}\right]^{1+r}(V_2 - V_1) + V_1.$$

To obtain a deceleration, a negative rate is used. In this case, in order to provide temporal symmetry with the accelerating rate path, the required equation turns out to be:

$$V = \left[1 - \left[\frac{t_2 - T}{t_2 - t_1}\right]^{1+|r|}\right](V_2 - V_1) + V_1.$$

The symmetry mentioned is most easily seen geometri-

cally:



$T_1 - t_1 = t_2 - T_2$ in the figure. At T_1, the difference between the linear value and the decelerating value is d. At T_2, the accelerating path is the same distance from the linear one. Thus, temporal segments can be pieced together so as to match velocities as well as values at their connecting point.

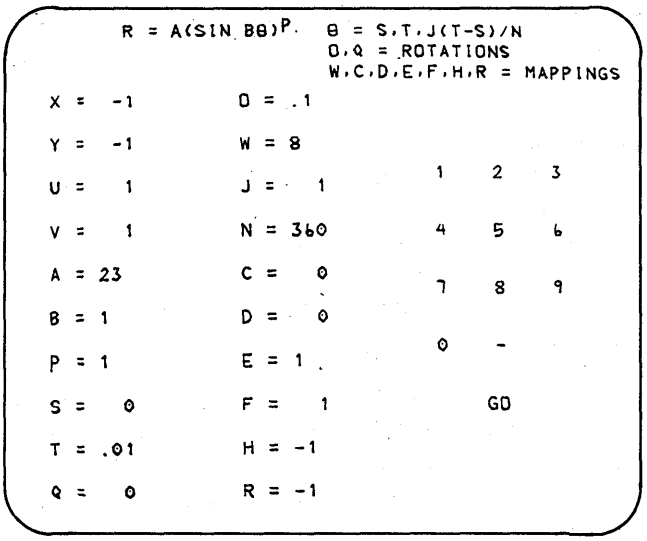
The type 2 temporal controls are quite different in appearance. Actually, one can set up the same motions in either system, but it may well prove very tedious to acquire certain results by one method and relatively simple by the other. Under type 2 control, a parameter obeys the equation:

$$V = (V_m - V_0)(\sin T)^p + V_0$$

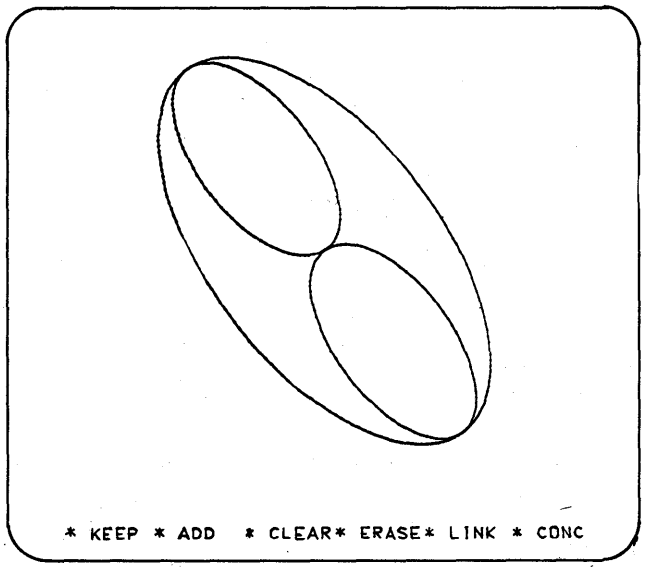
where T runs from T_i to T_f and the sign of the exponent p is used just as for P, H, and Z to affect the shape of the oscillation. For this kind of control then, the list of items characterizing each parameter must contain values, phases, exponential rates, and relative times.

Learning the fundamentals

Static use of the parameters and options is learned with the assistance of a separate program. This code displays on the 2250 a list of the twenty parameters equated to their default values. Along with the parameters there are also displayed a numeric "keyboard" and the word "GO." Touching a parameter with the light pen causes its value to be erased. Then as "keyboard" entries are touched, those particular digits appear consecutively setting up the new value for the parameter. When "GO" is touched, the entire menu is erased, and the curve specified by the parameters appears along with a list of return options. Besides the add, link, and concatenate options, there are "ERASE," "CLEAR," and "KEEP." Touching the erase option erases the curve and returns the parameter menu with all the previously set values still intact. Touching "CLEAR" removes the curve from the display and also from the computer's memory. The default values again appear on the menu. "KEEP" functions in the same way as "ERASE" except that, after setting up a new curve and touching "GO", that new curve is displayed along with the kept ones.



Parameter menu—Default values are right justified in Fortran 13 format. Values entered with the light pen build from left to right, digit by digit. In the paper, the parameter, R, has been called Z to avoid confusion with R in the basic equation.



A typical figure display showing the return options

This program can be used to develop an intuitive feeling for the parameters and options as well as to study seriously the analytic geometry of the fundamental equation. It is also most useful to the experienced user for determining experimentally how to construct any desired figures and then transcribing that construction knowledge directly into the animation program language. The animation program itself may also be used in a "dry run" mode for testing and debugging time sequences.

The language

As soon as one feels he has some grasp of the options and a few of the parameters, he is ready to learn the format of the data cards which comprise the language for the animation program. The three types of cards are referred to as identification statements, parameter statements, and frame statements. For one time sequence, a curve is specified by one identification statement and as many parameter statements as needed. Up to ten curves can be defined in one time sequence in the present program. After all curve specification cards, the frame statements are entered for the sequence. Other sequences may follow these directly.

The identification statement

An identification card contains the leters "ID" in columns 1 and 2. From that point on, the field positions are not rigidly defined except for two rules:

1) from one to four spaces may separate fields
2) anything following a 5-space gap will be treated as a comment

The first numeric field is a curve identification number from 1 to 10. The remaining fields are optional. As many as will fit on a card may be used to establish parameters which are to remain constant for the entire sequence at a value different than the default value. Typical identification statements are:

```
ID 1
ID  2  P=0  T=1.75  W=179.651
ID  3  N=251  C=-.5  F=.75  H=-3
                              Z=-2.566
```

Parameter statements

Each parameter that is to vary with time must be specified on a single card. Column 1 contans the symbol representing that parameter—

X, Y, U, V, A, B, P, S, T, J, N, W, Q, O, C, D, E, F, H, or Z.

If column 2 is blank, the remainder of the card is interpreted according to the type 1 temporal control system described earlier. If any non-blank character appears in column 2, a type 2 list is assumed to follow. The same two rules regarding spaces between fields hold for the remainder of the card as was the case for identification statements.

For the type 1 parameter card, the field beginning in column 3, 4, or 5 gives the initial value of the parameter. If the rest of the card is blank, the parameter is held constant at the initial value for the entire sequence. If a slash follows the initial value, a decimal fraction representing a relative time may follow the slash. The effect here is to hold the parameter constant for that fraction of the total time of the sequence. If no slash appears, the next field contains another value toward which the parameter will approach by a type 1 rate. This rate occupies the following field and is in turn followed by a decimal fraction representing the relative time in the sequence over which the parameter must make the transition between the two values. The last group of three fields—value, rate, time—forms a typical unit of type 1 parameter card entries. Following the initial value of the parameter (the first numeric field), up to five such groups may be given. The last group (and there might only be one) leaves the time field blank implying that that group carries the parameter up to the last frame of the sequence. Any fractional time field may be followed by a slash which effectively replaces the value and rate of the next group. This causes the last value reached at the last time (before the slash) to be held constant until the new time (the field following the slash). Typical examples would be:

```
B  1  7  0  .25 /  .5  3  1.5
O  0  2 -1 .333 /  .5  4  1  .667/
C .5 / .5  -.5 0
```

The parameter B starts at the value 1.0 and varies at a constant rate to the value 7.0 which is reached at the quarter point of the whole sequence. The value of 7 is then retained as a constant until the half way point in time. An accelerating change (rate = 1.5) is then made to the value 3.0 which occurs on the last frame. The parameter O (and all other angular parameters) has its value given in units of pi (3.1415927) radians. In the example above, O begins with the value 0 and changes to the value 2 (360 degrees) at a decelerating rate of −1 in one third the total time. It retains this

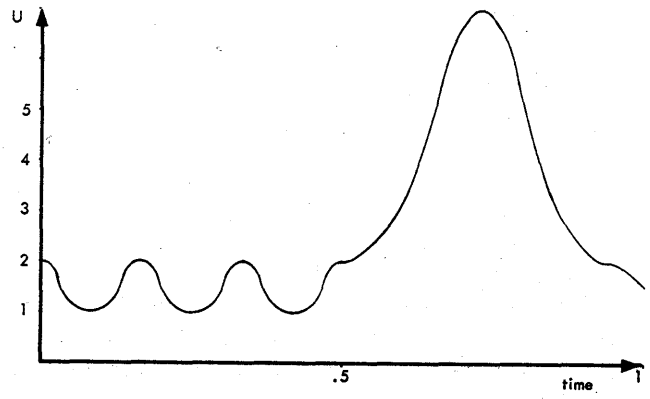
value until one half the sequence is over and then goes through another 360 degrees (to the value 4) at an acceleration of 1 up to the 2/3 point. The slash followed by blanks then makes it stay constant to the end. C begins with the value .5 and stays that size for the first half of the sequence. It then progresses to a value of −.5 at a constant rate (rate = 0) reaching that value on the last frame.

For the type 2 time controls, recall that the equation for the parameter was written as

$$V = (V_m - V_0)(\sin T)^p + V_0$$

where T runs from T_i to T_f.

On the parameter card, the field following the parameter itself (and the non-blank entry signifying type 2 control) contains the median value, V_0, for the first oscillating segment. The next field is the peak value, V_m. This is followed in order by the initial and final phases, and T_i, T_f in units of pi, the exponent, p, and the fraction of the total time, t, at which this segment ends if less than the total time for the entire sequence. Three groups of (V_m, T_i T_f, p, t) may appear on one card (with t omitted for the final group implying the value 1 for the relative time). For groups after the first, V_0 is is taken as the last value of the parameter actually calculated in the previous segment. Thus it is easy to obtain oscillating behavior of various kinds as well as continuous piecing together of up to three different oscillations for a single time sequence. An example would be the following card on which U is specified to oscillate as shown in the sketch:



U/ 2 1 0 3 2 .5 7 0 1.1 −2

Note that the first exponent, 2 in the fifth numeric field, confines the oscillation to one side of the median value and gives a zero slope at the median. The second exponent, −2 in the last field, causes an inflection point at the new value median (which happens to be the same as the old one).

Frame statements

Following all identification and parameter cards for a sequence are the frame cards. These must be blank in the first two columns. The first two fields on each frame card contain the initial and final frame numbers for the time sequence defined by that card. The remainder of each card contains curve identification numbers and options in alternating fields with a maximum of five curves and four options per card, and up to twenty cards can describe one "conglomerate" sequence. For example, the cards

1 1440 1 / 2 + 3

100 300 1 / 2 + 3

200 1300 4 L 1, 2 + 3 GO

define three overlapping sequences. From frame 1 to frame 1440, the curve whose identification number is 1 evolves as specified by its parameter cards. Simultaneously, curves 2 and 3 are combined by the add option. The slash causes curve 1 to be displayed separately on the 2250. The camera photographs it, the picture is erased, but no frame advance occurs. The combined curve, 2 + 3, is computed and displayed. This picture is photographed on the same frame of film and the 2250 is agained erased. The program then checks to see if any other frame cards require an exposure for the current frame number. If such is not the case, the frame is advanced. When frame 100 is reached, the same sequence starts over as specified by the second frame card. Notice that this sequence occurs much more rapidly while the same curves are still evolving in the same relative way but in the absolute time span fixed by the first card. When 200 is reached, a third sequence begins in conjunction with the two already in progress. The third card states that curve number 4 is to be linked ("L") to curves 1 and 2 + 3 which are concatenated (comma implies concatenation on the frame cards). The "GO" informs the computer that this is the last frame card for the defined curves. This entire set of cards may be followed immediately by another set of identification, parameter, and frame cards for successive sequences.

Other controls

Two data cards which the user may wish to alter contain the default parameter values and the ex-

posure time for the camera. These can be changed at the beginning of any sequence.

Before a sequence begins, a pair of function key lights are turned on and the computer waits for a depression of one of the two corresponding keys. The key selected determines whether or not the camera will be used for the sequence. If it is not to be used, the figures are displayed as rapidly as they can be computed. If the camera is to be operative, a curve is computed and displayed, the camera is activated, the curve is erased, the next curve is computed and displayed, and so on.

The camera in use was specially made for this project. Some features were included that are not usually associated with microfilm recording systems. First, the film registration meets quality standards of the motion picture industry. Excellent frame to frame steadiness has been achieved. Second, also in conformance with motion picture procedures, the camera film magazines accepts standard 1000 ft. core wound rolls of 35 mm film.

Other features of this camera are typical of microfilm systems in general. The film movement is independent of the shutter action. To advance the film one frame at a time, a Slo-Syn stepping moter is used with an attendant translator power supply for variable speed. The shutter is activated by a Ledex rotary solenoid. Switching, interface connections, and power supplies are contained in a box mounted on the camera's own floor pedestal. The interface connections to the computer include feedback circuits to detect shutter open or closed positions and to confirm film advance so that computer control is fully automatic and well supported with failsafe and sequence error detection.

Exposure tests established that an ASA 25 slow panchromatic type 5220 film was ample for good exposure density. With the camera in place before the console, an f/1.9 Baltar high resolution lens is set and locked for a fixed field so that final positioning of the camera is simply a matter of bringing a focus tape into contact with the surface of the 2250 CRT and sighting through a precision gun-sight type range finder. The camera is positioned into fine adjusted alignment with the aid of a computer generated test pattern.

The program turns on light 1 as a signal to open the camera shutter. On responding, the camera effectively depresses key 1. This is followed by an exposure timing loop in the program and subsequently light 2 is illuminated ordering the shutter to close. A successful response to this command is signalled by the apparent depression of key 2 by the camera. Programmed logic decides whether or not more information is to be displayed for this same frame of film. When the film frame is to be advanced, light 3 goes on and satisfactory response by the camera initiates the corresponding key depression.

With a working knowledge of this language, an artist can compose choreographic movements of simple or complex visual patterns and exercise control over such elements in ways never before possible. With additional darkroom techniques, he can make still further use of technological developments to add color and even sound as well as indulge in the purely human process of editing. For example, the filmed product of our program is subjected to certain optical procedures. The 35 mm black and white negative from the camera is processed normally and printed on high contrast stock yielding an image that consists of clear lines on a dense black field. This film is threaded into the projector side of an optical printer with a Bell and Howell movement and a 16 mm camera. The optical printer has several special features. The optical axis of the system is vertical with the camera looking down into the projector. The projector itself is mounted on a compound mill table. Thus additional translations and rotations of a mechanical nature may be superimposed, and the camera may be moved along the axis so as to provide for an additional scaling factor of from .1 to 10. A stepping switch circuit and preset frame counter allow a wide range of skip frame ratios to expand editing capabilities in the time domain.

At this point, we would like to present some slides showing curves drawn with the static program, blackand white moving pictures taken directly from the 2250, and color movies with sound added which were produced using some of the optical printing methods described above.

REFERENCES

1 A HURWITZ J CITRON J YEATON
GRAF Graphic additions to FORTRAN
AFIPS Conference Proceedings 1967 SJCC

What good is a baby?

by NELS WINKLESS and PAUL HONORE

Communications Contact, Inc.
Mountain View, California

Some of us think that Computer Graphics is the wave of the future and that our electronic gadgets will become standard tools for commerical artists. This is not likely to happen soon or suddenly. Both buyers and sellers of computer graphics must change in significant ways before our prophecy can be fullfilled. The changes will not occur automatically.

Let us approach the utilization of computer graphics with the thought that however hard we expect it to be it will turn out to be harder. This business is not for those who tire easily because we *expect* the whole undertaking to be depressingly difficult.

To begin with, people in the computer business always find it difficult to explain to their relatives and neighbors just what they do.

Every once in a while someone rudely asks us what we computer guys actually do with all the money we take from him in the form of taxes and phone bills. He doesn't necessarily object to our having the money, but he would like to have some feel for what is done with it.

When we explain that computers are great for tasks like matrix inversion and the solution of polynomial equations he admits that he is impressed but he'd like to have something in lay terms—and never mind telling him about accounting and record-keeping. He knows about that because he has been corresponding with a computer at the book club of which he is a member and the impact of computers in that area is already painfully familiar to him.

At this point we not only tell him what we do, we show him.

We make pictures with computers, see? (Figure 1)

He looks at our so-called pictures and says that since he is neither an art critic nor a masochist he doesn't have the proper background to judge our work . . . and by the way, how much would it cost to have a guy who can draw make a real picture of something?

There is a nagging feeling after one of these encounters that all is not right with the world. If we actually had to do something useful to justify our place in society, those of us who deal with computer graphics would be very hard pressed. If we wish to improve our standing instead of just defending our present position, we face uphill work.

After all, how do we measure the usefulness of something in our society? We ask if somebody will pay money or effort to have whatever it is.

Will people pay for computer generated pictures?

Who? Why?

It's no fair counting other people in the computer business although this trade increasingly uses pictorial output. Computer people, like normal human beings, are especially designed to make use of graphic information.

According to somebody who counted: "38% of all

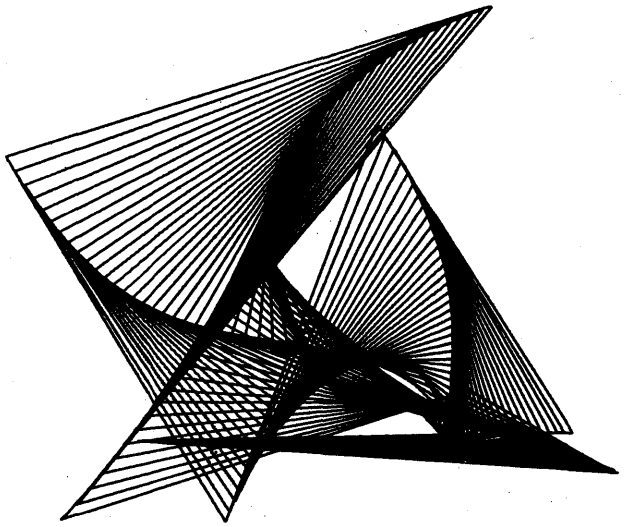


FIGURE 1

fibers entering or leaving the central nervous system are in the optic nerve." It is estimated that as much as 75% of information entering the brain is from the eyes.[1]

As Scientific American readers know, there is a great deal of evidence to suggest that information processing is done in the eye before data are passed to the brain. A recent estimate indicates that there are about a hundred million sensors in the retina and only five million channels to the brain from the retina. This twenty to one sort of data reduction is apparently not accomplished simply by using every fifth input on a sampling scheme.

For example, to recall McCullough's famous work at MIT, if a frog's eye is isolated from its brain in an experiment that measures output from the eye in response to visual stimuli it turns out that the eye is very selective. If something that looks like a fly is moved past the eye at about one frog's tongue distance, the eye puts out a substantial jolt in response. The eye will not respond similarly to a list of numbers describing a fly.

Even our Indo-European languages equate the words "see" and "understand." "I see, Ya vizhu, je vois." Seeing something is tantamount to understanding it.

We often plot numbers on a graph so we can *see* what they mean. Programmers often surrender efficency and their native neatness to make their typewriters print out verv bad graphs because even a bad graph provides more meaning usually than a tidy list of numbers.

Some processes are too difficult to visualize mentallv and we must turn our numbers into motion pictures to grasp the ideas. We've been involved with Sandia's ACCEL circuit card design program[3] and it's interesting to recall that the first motion picture sequences showing the placement and routing processes were not made to demonstrate the working program after its completion. Thev were used to let the programmers see what the programs were actuallv doing all along the way.

It has been suggested that prime numbers might be plotted graphically in such fashion that we could grasp the visual pattern of their occurrence and learn to predict their place in the number sequence.

Of course the computer business can use computer graphics, but what percentage of our society is in the computer business? A kid who sets out to be an insurance salesman can sell a few policies to his uncles and cousins but sooner or later he has to go out and tackle hostile strangers to justify the draw his company pays him. We manage to draw a good many million dollars from society and it seems we must figure out how to sell our work to hostile strangers.

A small number of people have been triyng to make a commercial venture of computer graphics. The survivors of the effort report vivid impressions of hostile strangers.

When we say "computer graphics" we tend to put the accent on "computer" and let the "graphics" take care of itself. Actually, hostile strangers are only moderately interested in computers. To them a computer is sort of like a head on a pole—nothing they'd deliberately seek out, but interesting as a novelty.

On the other hand our prospects do know a lot about commercial picture making. They are keenly aware of the mechanics and economics of their work. They want to know what sort of thing we can draw, how fast, for what price.

We can draw anything. (Figure. 2)

The actual drawing is done very rapidly and a thousand variations on a given piece of work can be produced with the speed of summer lightning, but the programming and especially the input of data are complex and tedious procedures. Further, the making of pictures is a creative undertaking subject to irrational and unavoidable disagreements. We are unable to predict with confidence the time required to do the work *and* live through all the arguments over taste.

Similarly we cannot predict the price. Any number we choose is really a blind stab.

We have tried to get around these sordid realities by deciding that we are actually fine artists—as opposed to commercial artists. Fine art speaks for itself. Commercial art has to be good for something.

Like everyone else in this field we were equipped with a huge collection of randomly derived geometric forms. They are quite interesting the first few hundred

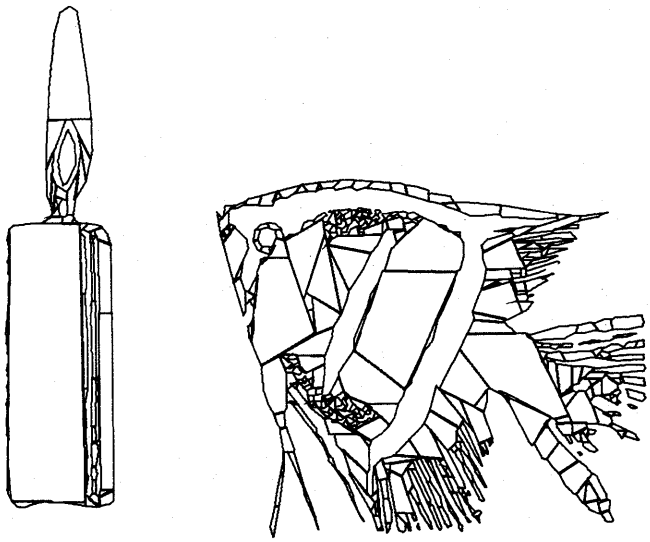


FIGURE 2

times through. It isn't that we were so crazy about geometric forms, but the computer likes to draw them and it jolly well won't draw anything else without a large investment in time and money.

With op art at the height of its popularity we looked around for somebody who appeared to like randomly derived geometric forms. The big time art people who run the museums were interested in the material we offered and asked what artist had done the work. When we explained that no one who could quite be described as an artist had done the work, only a programmer who set some policies for the computer, there was a sense of coolness in the air.

It was regretful that an artist had not infused our work with artistic value. They named some artists who might lend grace to our poor efforts.

The recognized artists we talked to were hearty, pleasant chaps who were not fools. They could tell that we harbored a cynical attitude toward that sort of art of which op is a sub class. They didn't care to be used by us. We did not wish to use them.

We might whip up more enthusiasm if we thought there were money in fine art, but how many non-objective artists is the world willing to support? When we consult standard statistical references like the World Almanac we do not find any figures dealing with fine art, no indication of the annual yearly business of art galleries, or the number of people who list their profession as "fine artist."

On the other hand, there are figures dealing with advertising expenditures, with television programming, with various sorts of things that depend upon the creation of pictures. One learns, for example that the annual billing of an ad agency like Leo Burnett Company is approximately $250 million. That is $37 million more than a published estimate of the annual budget of Sandia Corporation.[4] Burnett is not the largest agency. It's only about fifth. Sandia is certainly the largest atomic bomb manufacturer in this country.

Presumably one can find out about the economics of fine art but the lack of publicity attending the numbers does not suggest that we can win the hearts and the cash of our countrymen by automating Piet Mondrian . . . even in three dimensions.

Fine artists will have an opportunity to use our instruments and their creative help will be stimulating and provocative. On the other hand we have not been able to break down any barriers by acting as if our randomly derived geometric forms are anything but interesting exercises. If they speak for themselves, they speak an unknown tongue.

We really have little to show, collectively, in this field. There is some variety and there are some spectacularly beautiful things. Some of the fine figure work at Boeing and some of Whitney's handsome work come to mind. The trouble is that we have a little man who couldn't draw better, faster, and cheaper.

On that basis it is very difficult to approach people who normally make good pictures that fit their purposes well. In that case we are inviting them to take a fling on research and development with our talent and their money. We can weave dreams for them, but we don't know how to perform to their specifications. Indeed, they can't imagine how to specify anything for us.

Obviously, we can use our techniques in the long run for things like package design.

A designer could feed the basic elements of a new package into the computer and get back five hundred variations on a basic theme overnight. He could paste up a few selected treatments right from the hardcopy and deliver a virtually complete job to his client along with a bill.

Many such commercial applications will be practical once we have appropriate software, I/O hardware, and the right attitude.

Meanwhile, there is one area in which we are just barely able to do something of value. We have in fact sold computer animation to producers of television commercials.

We started some years ago with jittery sequences of geometric nonsense and bit by bit—literally—we have worked up to smoothly flowing, shimmering, full color sequences of geometric nonsense.

Television network executives look at it and say: "Whatever would we do with anything like that?"

Hollywood producers say: "You show me an animated character emoting and I'll get you millions of dollars." By then, who needs it?

When we showed black and white material to ad agencies they said; "It certainly would be better in color."

When we show them color footage they say: "It certainly is hard to judge the basic art with all that distracting color."

One producer was putting together a series of psychedelic backgrounds for use behind musical groups on television.

He likes our colorful geometric nonsense and offered to pay a dollar a foot for specially prepared material. That's sixteen frames per 35mm foot. A three minute sequence would pay us $270. If that footage cost ten cents a frame, total, including postage for sending the bill, we'd lose $162 on the deal. Still, he was a live customer offering genuine cash money. It happens that he is just as happy with out-of-focus pictures of splattering ink as he is with our expensively wrought material. So are most people. Besides, ten cents a frame is

not enough by a factor of twenty or fifty or eighty, depending.

We are left searching for wealthy customers who want to buy very carefully controlled, dramatically complex animation effects.

There is one such market.

When we worked in the television commercial business a typical one minute spot cost seven or eight thousand dollars to produce—about five dollars a frame. That seemed like rather expensive production even in Hollywood a few years ago.

Old friends in the business tell us now that a typical full blown spot for a national sponsor costs eighteen thousand or so dollars and it is not unusual to spend thirty or forty thousand dollars—twenty dollars a frame. That's enough to pay for some careful computer work . . . not too much, because most of that money is carefully earmarked. Still, it beats trying to swindle an artist out of his fellowship money.

People argue the point, but we feel that the television commercial is the great American Art Form—native grown, governed by stringent disciplines of time and purpose—subject to fast feedback from a responsive audience of a hundred million people. There is more creative innovation in television commercial design, in commercial music, in the deft handling of language than there is in films that are designed for entertainment. We may quarrel about specific pieces of work, but notice that your kids will run to watch a commercial embedded in a program they have been ignoring.

Our highly stylized computer animation is acceptable and useful in television commercials.

Since we grasp the economics and the mechanics of both film making and computer animation we assumed that we could head off most of the problems and interpret adequately between the film people and the computer people in doing commercial work on contract.

We learned a lot that was not intuitively obvious. One thing was not a surprise. The man in the middle gets a lashing from both sides. The surprise was the number of subjects that bring on trouble.

There is a difference of viewpoints that makes it hard for either side to tolerate the obstinacy and irrationality of the other. Since we in the computer business are trying to sell something to the film people, it seems that *we* must flex, not they—at least in the short run.

Let us consider some conventions in the motion picture business and brood a bit about the ways in which our computer material varies from those conventions.

A professional motion picture is usually composed (Figure. 3) of rectangular images 3 × 4 in aspect ratio

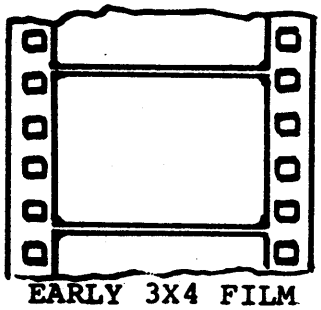


FIGURE 3

arranged at regular intervals along a strip of sprocketed 35mm film. Sixty years of working with international standards have established these specifications rather firmly.

Computers put out pictures by two standard means, through an oscilloscope tube, thence to either film or sensitized paper that is processed to produce hard copy, or through a stylus that draws an image directly on paper.

Since oscilloscope faces tend to be round, efficient practice suggests that the image area utilized on the scope face should be the largest practical square. The typical field is 1024 by 1024 points. This is not only practical, but 2^{10} is a number of great power among mathematicians. The product of this mystical rationale is a square picture.

A chap at Eastman Kodak did a study almost forty years ago in which he measured 250 paintings by fifty famous artists. These were pictures that people were willing to pay good money for and to hang on their walls.

The plot of width to height of these pictures looked like this: (Figure 4). Square pictures were very scarce.

Another researcher wrote during the time when sound movies were just coming into use:

"We purposefully use the expression 'rebellion' because for the first time in motion picture history a number of exhibitors and at least one of the greatest exhibiting and distributing organizations in America have taken matters into their own hands and have reduced the height of the projector aperture. They have considered it essential to maintain the rectangular form of the screen even at the risk of cutting off parts of the performers heads or some detail of the lower part of the general scheme of composition. Such procedure is rebellion and what is more important, it is seemingly justifiable."[6]

(Figure 5) The situation was this: the 3 × 4 aspect ratio had been established early in the movie game because people liked it that way. When sound came in

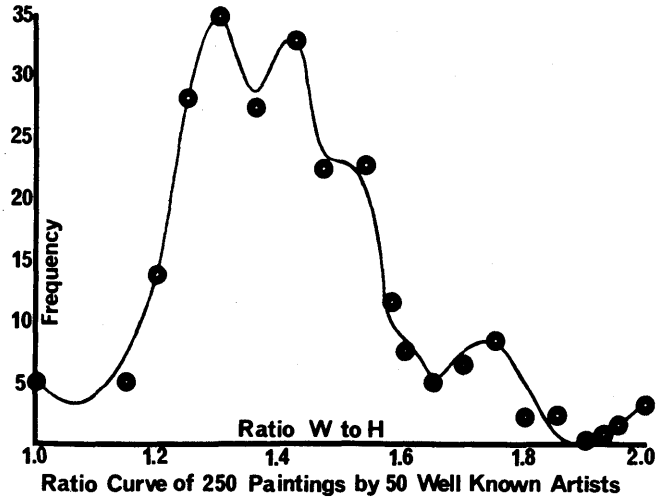


FIGURE 4

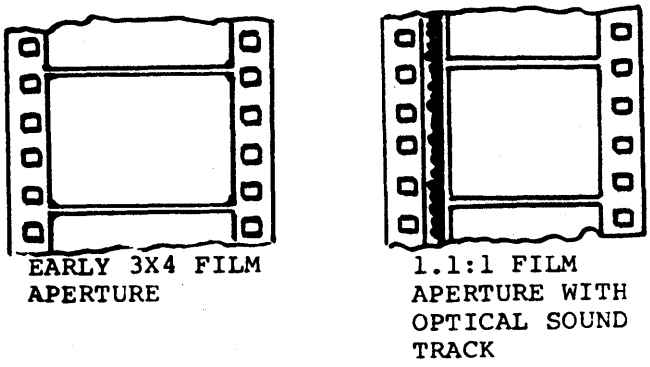


FIGURE 5

it was clear that the track had to be put somewhere so it was printed inside the sprocket holes in what had been picture area and the picture area was chopped off to make room, leaving a square image. It was efficient and practical and people hated it.

(Figure 6) The present so-called Academy Aperture was a compromise arranged by the Motion Picture Academy to meet this problem. It is inefficient and reduces by an appreciable fraction the possible resolution of the picture on the screen, but people like it.

At least thirty years after this drama when we first needed some computer generated footage, the messenger brought back from the data center a roll of randomly spaced square images filling from side to side a piece of non-sprocketed 16 mm film. Run that through your projector! We had to shoot more nearly standard film footage from hard copy of this stuff.

This brings up another quaint custom of the computer business. It seems a shame to make an issue of the fact that this industry at the very forefront of

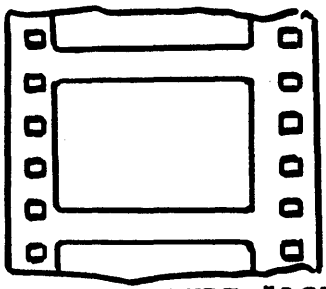


FIGURE 6

technology is using antiquated methods, but the fact is that scrolls have been largely out of use in civilized society for some hundreds of years. Computers tend to print out scrolls.

Machines may handle scrolls comfortably but people don't. Further, machine operators aren't accustomed to really long runs of hard copy. During our two thousand frame runs the machines gradually ran out of developer and the last few hundred frames were decorated with orange-brown streaks that wove strangely through the movies.

No matter, we can now put the images on regular movie film but it would never occur to a producer to specify sprocketed film and it does not necessarily occur to computer people that it matters.

We use the scrolls only in short test runs of some material and calm handling of the paper solves all problems. Still, there is little calm in the motion picture business and the customer always winds up with crushed and torn artwork somehow. It seems a pity.

Don't accept the simple thought that standard cameras taking pictures off the scope solve all the problems. There's still that square picture to contend with and the computer readout equipment is in general completely inflexible. The camera is locked into an electronics rack over the tube and it stares fixedly at the square image. If you're like us you don't own the equipment and you're in no position to modify what they have at the data center.

(Figure 7) Suppose you want the camera to see only that portion of the square that makes a 3 × 4 image and you can program so that all the action takes place inside that rectangle.

You call the man with the readout machine and say: "Can I position the camera so that each standard rectangular frame of motion picture film is filled from side to side with the scope picture?"

"Yes" he says, "the image is normally 17 × 17mm

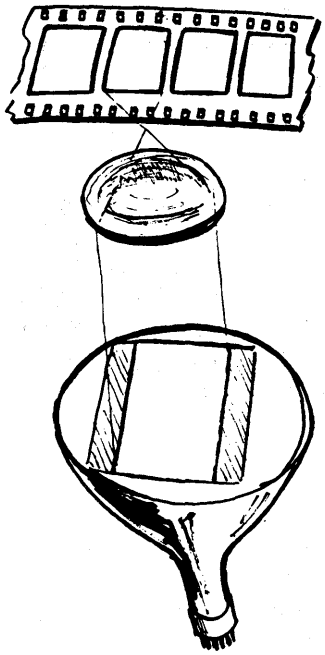


FIGURE 7

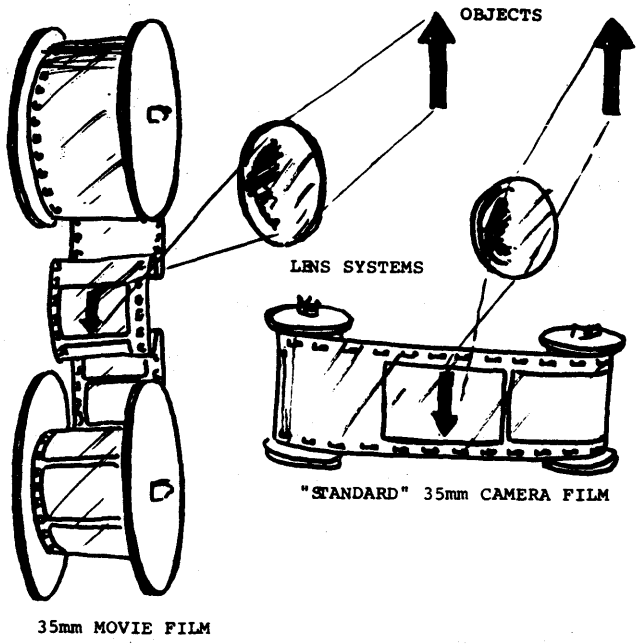


FIGURE 8

but it can be expanded to fill the frame from side to side."

"Good!"

"We can expand it to 19mm," says he.

Since a normal frame is about 25mm wide you can't fill it with 19mm of picture and you point this out.

(Figure 8) After an airplane ride he didn't want you to find out that his idea of side to side is your idea of top to bottom. He thinks of the film as running horizontally as it does in a 35mm still camera.

Crying relieves your feelings, but it doesn't modify the hardware for you.

You carve your 3 × 4 area out of the available square and photgraph it. This gives you a small rectangle of picture the proper shape somewhere in the normal film field.

The producer who is paying you will have to blow the image up to fill the frame. He resists the thought and expense and you tell him about 2^{10} and how mathematicians are fond of it. He tells you to move the camera and you re-live the whole scene at the center.

Of course, the producer has to go to the optical house with his film anyway because you are supplying him with three separate strips of high contrast black and white microfilm. (Figure. 9)

These three strips must be printed in register, each with a different color filter, to produce a final color negative. It's just like color separation work in printing. Dreary technical concerns prevent production of

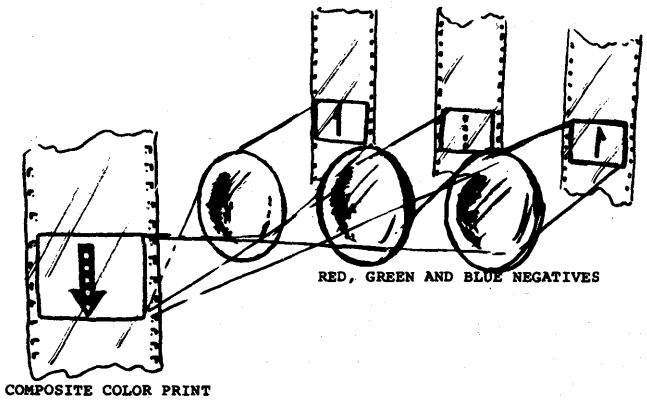


FIGURE 9

fully saturated color pictures right from the scope onto color stock.

The producer is also horrified to hear that the picture we produce for him is centered on the film. The Academy aperture to which he must convert this stuff is not centered on the film. It is off to one side, leaving room for the sound track the way it is supposed to.

Not only that, the producer must rely on the computer people to generate designs that allow each of the three strips to be approximately uniform in density. That is to say: very fine lines on the negative can stand to have a lot of light pumped through them for reproduction on prints. Too little light will leave weak, broken lines with poor color saturation. On the other

hand, large areas of solid color will bloom on the print if too much light is used in printing. If a given frame contains both fine lines and large areas the printer cannot balance for either and the product is bad.

Further, if the producer does not want the colors to blend at the points where lines intersect, he must make traveling mattes—a process too complex to cover here, but fussy in the extreme and especially difficult with multidudes of fine lines.

If our irregular film product is to be made useful to the buyer, we must anticipate a great many little problems. Some unexpected things crop up.

Normal motion picture film is numbered along the edge so that the original film can be matched foot for foot with prints made from it simply by matching numbers.

The microfilm stock normally used in computer systems is not edgenumbered. Nobody thinks about this until the editor bursts into tears after days of working with footage he cannot make head nor tail of.

We've been talking about purely mechanical matters. Consider creative problems.

A computer can do things a human animator cannot do. The computer can move each of ten thousand lines from frame to frame without fatigue. The human animator gags at the thought.

He not only can't do it practically. He can't imagine why he would want to. It is utterly outside his experience. If he designs action he does understand he wastes the capabilities of the computer. Anything he really understands can be done better and cheaper by hand.

He can't figure out what to do with hundreds of units of animation. He can't imagine what such stuff would look like, what to ask for. He does get a warm glow when we tell him that the computer will cheerfully draw action that is tough for human animators. It will move active figures toward and away from the camera and maintain the sort of perspective variations that occur when real objects are photographed

We're not talking about regular forms that appear to move near or far simply by changing size. We're talking about the kind of distortion you get with a wide angle lens. (Figure 10) Such a lens enlarges closer parts of the figure more than distant parts. It is easy for an animator to show a figure walking left to right across the field at a given distance. It is much harder to make that figure look right when it walks toward the camera. The computer can handle that sort of thing well by formula.

The animator senses the power of the tool we offer when he thinks about this sort of thing, but he still doesn't know how to use it.

He's not about to learn for a given commercial job.

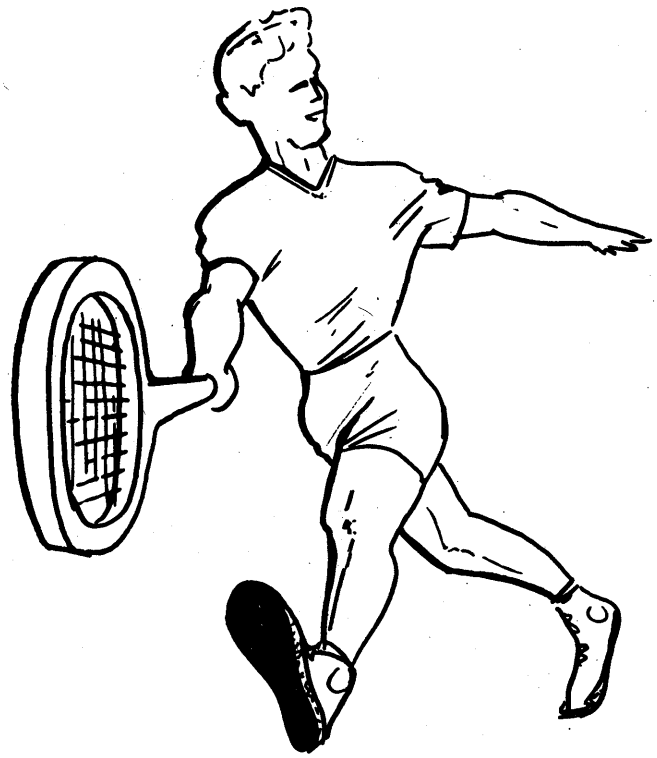


FIGURE 10

The producer suggests that he send his animation director out to work with us at the computer so he can see what actually happens to the rough ideas they eventually pull together.

He is disappointed when we explain that the computer is a smashingly uninteresting box that lives in a room of its own behind an iron fence somewhere. Even if we do have access to the computer it does nothing more dramatic than blow hot air on your shoe while it hums and soaks up money.

The computer output is invisible, just a lot of magnetic signals on tape that must be transported to the printout system somewhere else.

Fine, he'll have his man watch the printout machine. No such luck, the thing is sealed tight and the outside of the box doesn't have the visual appeal of a Brownie Reflex.

He can see some test pictures, stills drawn on scrolls, that show him what the computer thinks it is supposed to be drawing, but he can't see test runs of animation unless he wants to pay a fortune for the privilege.

Normally his animators draw the material roughly in pencil and photograph that directly in what is logically known as a pencil test. It gives him a feel for the action and allows correction at an inexpensive level.

It's all or nothing with the computer. Onlv faith and

hard liquor allow the customer to hold his peace while he waits to see the finished product.

In spite of all, the struggle has been worthwhile in practice. The effects are so striking, especially the effects of bugs, that customers do come back for more.

The point that seizes our attention is really that the computer is completely wasted if it is used to produce material that is already well understood and is under full control. We have a brand new medium and none of us quite knows what to do with it. This is not without historical parallel in the computer field or in the motion picture field.

Each of us has sometime been through a discussion with people who are convinced that a computer is simply the equivalent of ten thousand men with adding machines. They grant the idea is impractical but they feel we are begging the question when we insist that computers are really more than that. Well, maybe, but we were not able to predict from our knowledge of people with adding machines many of the things we now do with computers. Our viewpoint has changed with experience.

With respect to graphics, we suspect that the computer is more than the equivalent of ten thousand men at drawing boards. We must assume that the future holds surprises for us.

When motion pictures were first developed—if you'll pardon the pun—the mere technical fact of being able to record and reproduce moving images was stunning. It struck people that they could record great actors in great plays and carry the film off to places where people had never seen fine drama. Culture could be taken to the people.

Indeed, they put cameras down in the audience and recorded plays for the enrichment of mankind.

It turned out that motion pictures were no good for what they had in mind. If you plop a camera down and let it watch a play you find that the camera doesn't see a play the way a human being does. It is non-selective. When a human being watches what the camera has recorded he is bored to tears because he cannot escape the camera's viewpoint.

Every motion picture technique we now take for granted was at one time a startling innovation.

Everybody knows full well that one image following another on film follows in time as well as in position on the film. Still, if you see a burning house in one scene, then action at a firehouse, then a picture of a woman picking up a baby in a smoky room, you assume that these pieces of action are related and may be simultaneous in time. How do you know?

The first motion picture to include edited action of this sort was very creative in the eyes of the people who saw it. Although roughly similar action in stage plays was common the effect was not quite the same. The camera could be moved and it was a surprise that story continuity could be maintained—even enhanced—in spite of abrupt changes in viewpoint. Motion picture technology was wasted if it was used only to record material that people already understood and could comfortably control.

Motion picture techniques developed rapidly because the technology was rather simple. Anybody could operate a camera and even process his own film with a bit of experience. Of course there was a big hassle about standards. Further, stage directors and actors did not all readily adapt to the limitations and new opportunities inherent in making movies. A whole new class of creative craftsmen had to grow up in the film business.

Now here we are with a new medium—computer generated pictures. Our technology is not simple. Our equipment is not designed so that anybody can handle it and innovate in the new medium. Even timesharing systems with light pens and memory drums and CRT displays allow laymen to do only such crude work at great expense that we do not see these systems as the key to general progress. Outsiders with fresh attitudes cannot easily join our club.

We are doing something about the input bottleneck. We're starting to beat the problem of teaching the computer what to draw. Till now the mere input chore was more costly than the programming and computer cost. It isn't that the programming is easy. It's just that after years of learning what to do to make things move the way we want, the big task in any given production job is to ram the picture data into the miserable machine. We are improving that and it will help a lot.

Does it matter to us how the computer graphics revolution will occur if we are so confident that it will—apart from commercial considerations?

Well, there's that nagging question about what we have been doing with all that money society spends on our recommendation. Can we produce anything people want to buy?

So far, nobody can tell what we have to offer and as long as we speak gibberish and put out our work in forms nobody can use, hardly anybody will trouble himself to find out what we have to offer. Their enthusiasm for paying us may evaporate.

There is the happy possibility that a whole new class of creative craftsmen will grow up among us. If we actually follow standard practices in the graphics field and speak something resembling English or Russian or Chinese and we develop hardware that anybody can use without five years of intensive training—then the hostile strangers may actually buy things from us, whatever those things turn out to be.

There's an old story about a man who presented a

new idea and was asked what it was good for. He floundered for a while and said: "I don't know, but what good is a baby?"

If we help our computer graphics baby to grow up we'll find out what it's good for.

REFERENCES

1 F R SIAS JR
The eye as a coding mechanism
Medical Electronic News

2 G BIERNSON
A feedback-control model of human vision
IEEE Proceedings June 1966

3 C J FISK D L CASKEY L E WEST
ACCEL Automated circuit card etching layout
IEEE Proceedings November 1967

4 *Article re Sandia Corporation*
Business Week March 1968

5 L A JONES
Rectangular proportions in pictorial composition
SMPTE Journal #14 Jan-Jun 1930

6 A S HOWELL J A DUBRAY
Some practical aspects and recommendations on wide film standards SMPTE
Journal #14 Jan-Jun 1930

A computer animation movie language for educational motion pictures*

by D. D. WEINER and S. E. ANDERSON

Syracuse University
Syracuse, New York

INTRODUCTION

The value of motion picture films as an effective teaching aid has been long established. In recent years, preliminary efforts have demonstrated the feasibility of producing animated films by means of a digital computer in conjunction with a microfilm recorder. A block diagram of the computer animation movie process is indicated in Figure 1. A programming language is used to translate the motion picture script into a computer program. The program, usually in the form of a deck of cards, is processed by the digital computer which generates a magnetic tape output for input to the microfilm recorder. The recorder, containing a cathode ray tube and motion picture camera, produces an exposed roll of film that is then developed to yield the finished motion picture.

Advantages of computer animation

Computer animation has many advantages to people having educational needs. Some of these are briefly indicated below:

1. Computer animation makes it possible for individuals with many different backgrounds to make motion pictures. The movie language, if carefully designed, can readily be learned and used by personnel who have no particular technical background in computers and motion picture production. In addition, because of the large number of computer centers distributed throughout the nation, the computer animation capability can be available to potential film producers irrespective of their geographical location.

2. Computer animation is a relatively economical technique for producing motion pictures. Of course, the cost depends upon the complexity of the images being generated. Past experience with simple line drawing animation indicates a typical cost of approximately \$.10 per frame. In contrast, conventional motion picture techniques average around \$1.40 per frame. Thus, it should be possible to extend the use of the animation technique into areas of education and training where formerly it was precluded because of cost.

3. Computer animation allows the film maker to generate revisions with minimum expense and effort. By changing the proper parameter in only a few instructions of the program, it is possible to produce several versions of a film sequence having different motions, spacings, scale, object dimensions, and picture complexity. Thus, at relatively small cost, the film maker can experiment with different arrangements to obtain certain desired effects so as to improve the illustration and explanation of the subject matter of his film.

4. Computer techniques can produce animated film much more rapidly than conventional methods. When both digital computer and microfilm recorder have been available at the same site, it has been possible to begin and finish a five minute film clip within one week's time. In addition, successive animated films require less time and effort because subroutines developed for a particular film can then be used in future films.

Basic movie language developed at Syracuse University

The capability for graphic digital computer output has been available for many years. However, where movies are concerned, thousands of frames of film are usually required. For example, a 5 minute movie running at 24 frames per second consists of 7200 frames. It is

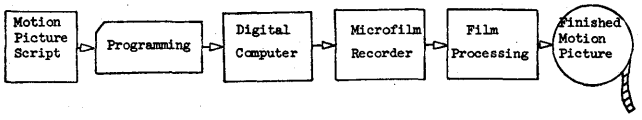


FIGURE 1

*The above work was sponsored by the United States Air Force under the Contract No. AF 30(602)-4283.

impractical to program each frame as a separate entity. What is needed is a language that is capable of generating tens, or even hundreds, of frames by means of just a few instructions. Such a language has been developed at Syracuse University.

The language consists of the four programs:

1) CALD (Computer Aided Line Drawings)
2) CAPER (Computer Aided Perspectives)
3) CAMP (Computer Aided Motion Pictures)
4) CAMPER (Computer Aided Movie Perspectives).

CALD and CAPER are written for output on the Cal Comp 565 mechanical plotter while CAMP and CAMPER are to be used with the Stromberg Carlson 4020 microfilm recorder. A capability for output on both the mechanical plotter and microfilm recorder is needed because of the unavailability of a conveniently located microfilm recorder. When producing computer animated motion pictures, several attempts are usually required before satisfactory results are achieved. In addition to debugging the program, the animator often experiments with various picture parameters so as to best obtain a certain effect. Because no microfilm recorder is located within the Syracuse area, the magnetic tapes are sent for processing to the nearest installation at the Polytechnic Institute of Brooklyn located approximately 300 miles away. In order to eliminate long delays during the debugging and experimental phase of the movie, provision is made for graphic output on the University's mechanical plotter. It is intended that movies be debugged by plotting selected frames using CALD and CAPER. Since the names of the commands in CAMP and CAMPER are identical to those of CALD and CAPER, little additional work is required to convert from the mechanical plotter to the microfilm recorder. The option exists for drawing pictures in either a two- or three-dimensional representation. CALD and CAMP are programs for generating and manipulating basic line drawings and simple geometric figures in two dimensions. CAPER and CAMPER are programs for creating, manipulating, and plotting three-dimensional figures onto a two-dimensional plane in true perspective.

All four programs have been written in FORTRAN IV. It is recognized that other programming languages are more versatile and efficient. However, the use of such languages is limited to a relatively small number of computing centers. If computer animation is to develop rapidly into a useful, effective, and widely available educational tool, it is desirable to create a common library of subroutines that will be available to as many computer installations as possible. Clearly, FORTRAN IV is ideally suited for this purpose. Unavoidably, either assembly or machine language is needed to provide the input codes for both the plotter and the microfilm recorder. (FORTRAN IV is unable to manipulate data on a bit level.) Fortunately, this is required for only a very small number of commands.

The four programs consist of a series of subroutines. Each subroutine is called by one data card. The program user need not know FORTRAN. He need only learn the movie language (i.e., the name and function of each command and the purpose of the parameters associated with each command). The names of the subroutines and associated parameters have been chosen to be as simple and descriptive as possible. Should the user be familiar with FORTRAN, it is possible for him to enlarge the original set of subroutines.

A list processing technique is used for the storage of pictures. Each new figure is stored consecutively at the end of a variable length list, called a stack. Each figure within the stack is called an array. Figures are referred to by specifying their respective stack and array numbers. Stacks and arrays can be altered either in total or in part. Hence, it is possible to individually alter entire frames, figures within frames, or points within figures. Figure 2 gives an example in which the arrays of two separate stacks are used to store the basic figures generating the dog and the letters comprising the text. Should it be desirable to wag both the dog's tail and ears, it is only necessary to modify the fourth and seventh array of the first stack. Thus, constant backgrounds and non-varying portions of a frame need not be computed more than once during an animated sequence.

Basic figures are provided for in the movie language. For example, Figure 3 illustrates the command used to generate an arrow. S and ARR denote the stack and ar-

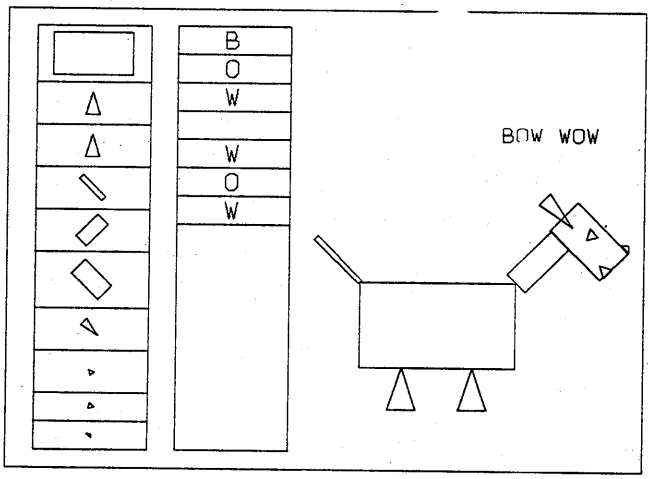


FIGURE 2

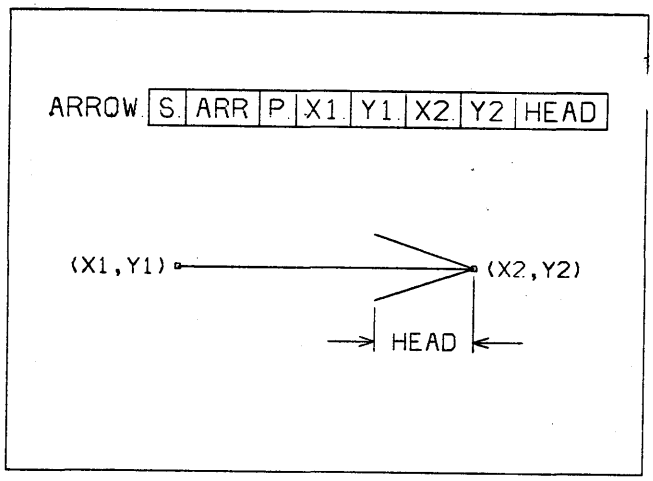


FIGURE 3

ray numbers in which the arrow is to be stored. The parameter P equals 0, 1, 2, or 3 depending upon whether the shaft is to be solid or dashed and the head open or closed. The arrow is directed from the coordinates (X1, Y1) to (X2, Y2) and the size of the head is specified by the parameter HEAD. Subroutines for generating circles, rectangles, triangles, crosses, grids, clocks, and sine waves are also available. Some electronic circuit symbols are included.

In addition, an alphanumeric set of 49 characters is provided. They have been created using straight line segments and may be manipulated in size, position, and orientation so as to give a large degree of flexibility in creating text. When lines of text are desired, a single command can be used that automatically spaces up to 20 characters. If more than 20 characters are required, the command can be used consecutively to concatenate the strings of text. A sample title used in a movie explaining the movie language is shown in Figure 4.

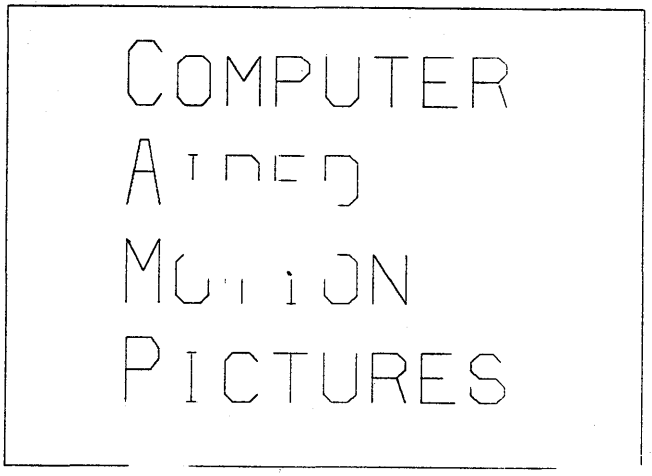


FIGURE 4

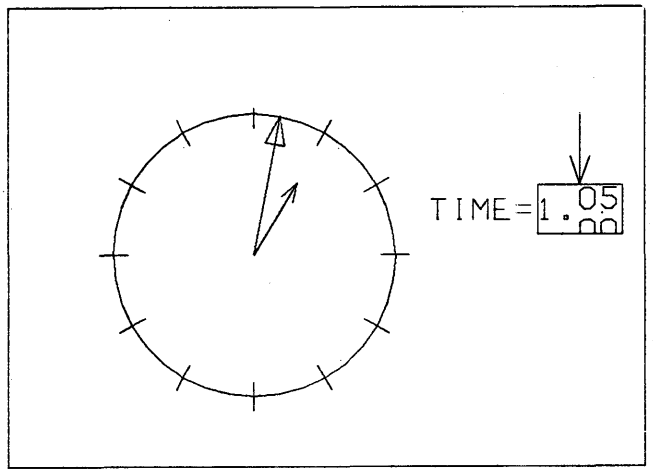


FIGURE 5

Figure 4 also illustrates the masking routine. By specifying the boundary of a rectangular area, it is possible to draw only those lines and portions of lines that fall outside of the specified area. As an application of the masking routine, Figure 4 is taken from a "dissolve" sequence in which the rectangular area is made successively larger until the entire title has been masked leaving a blank frame of film. In this way, an interesting scene transition is made. Windowing, the inverse of masking, is also part of the movie language. With this routine only those lines and portions of lines that fall inside a specified rectangular area are drawn. In Figure 5 the windowing routine is used to simulate an odometer. During the animation sequence, numerals move vertically through the window indicating current time on the clock.

Commands are provided for the translation, rotation, and magnification of objects. The rotation command ro-

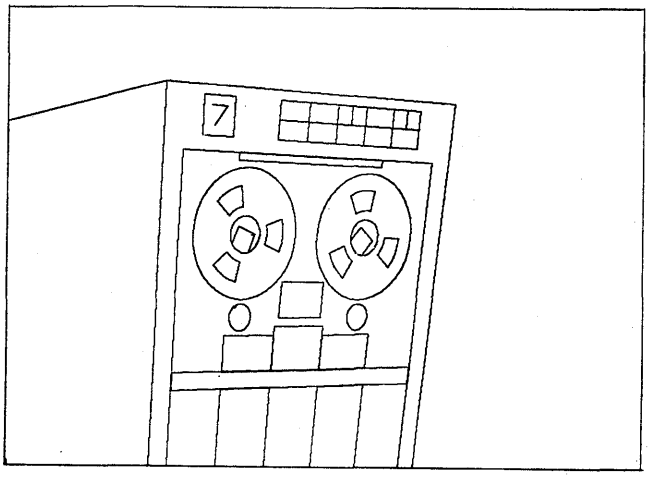


FIGURE 6

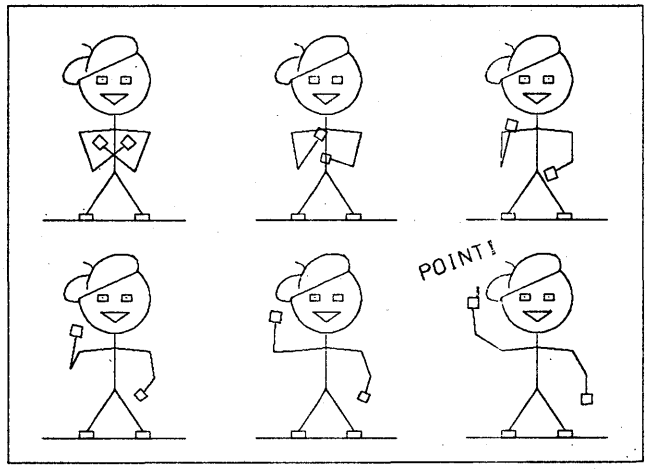


FIGURE 7

tates an object about any desired center point by a specified angular increment. The magnification command expands or contracts all points of an object as measured from any specified reference point. Figure 6 illustrates the simultaneous use of magnification and windowing to properly truncate an object as it is blown up beyond the limits of the screen. Another command, helpful in animating sequences, is the MOVE command which moves an object in equal increments from an initial to final position. This is demonstrated in Figure 7 where the six animated figures are produced by means of only two MOVE commands.

Finally, three commands (SAVE, ESAVE, and REPET) are included for use when the same scene is to be repeated many times. A great deal of computer time is saved since the scene is retained in SC–4020 coded form.

Figures 2–5 and Figure 7 are taken from a ten minute computer animated movie explaining the CAMP movie language. This movie will be shown as part of the conference presentation. Additional film clips will be included as time permits.

REFERENCES

1 K C KNOWLTON
Computer-produced movies
Science Vol 150 Nov 26 pp 1116–1120

2 A M NOLL
Computer-generated three-dimensional movies
Computers and Automation Nov 1965 pp 20–23

3 E E ZAJAC
Computer animation a new scientific and educational tool
Journal of the SMPTE Vol 74 Nov 1965 pp 1006–1008

Design highlights of CABAL– A compiler-compiler*

by RICHARD K. DOVE

Carnegie-Mellon University
Pittsburgh, Pennsylvania

INTRODUCTION

The advantages offered by compilers over machine or assembly coding are well known to all of us. The personal satisfaction and minimal execution time of optimally perfect code rarely outweigh compiler language implementations which are implemented and debugged quicker and cheaper. The returns from readability and machine independence further unbalance the scales.

In an effort to reduce the amount of required machine coding to a minimum, compiler-compilers are beginning to attract attention. Although the systems programmer must continue to struggle with operating systems, compiling has been around long enough that formalisms and common techniques are being forged into various compiler writing languages. The current literature reflects the youthfulness of the field. New concepts and solutions to new problems have affected compiler-compiler design in a predictable way: the primary concern has been to demonstrate a working system. Only after we relegate these new problems to an innocuous position and comfortably understand the new concepts involved do we turn our attention to useable systems as opposed to workable systems.

The current state of the art is surveyed byFeldman and Gries in the February 1968 Communications of the ACM.[1] Along with compiler-compilers, he includes discussions and evaluations of automatically produced recognizers, syntax-directed symbol processors, meta-assemblers and extendible compilers, as well as an introduction to the necessary terminology.

Before discussing the CABAL design some information drawn from the Feldman-Gries survey of compiler-compilers will supply some context. Generally speaking, when a compiler for some language is needed, its syntax and semantics are each expressed in appropriate meta-languages which are fed into syntax and semantics loaders. Each loader builds a table, one controlling recognition and parsing and the other containing a description of the meaning of statements. These two tables, a single pushdown stack, code generation routines, and common translator facilities form a table-driven translator. Each element in the stack contains a syntactic construct and its associated semantics. When a construct is recognized the semantic information in the stack and the semantic table direct the translator to take some action, usually code generation or compile time memory for things such as declarations.

Currently, the field is characterized by three efforts: FSL,[2] TGS,[3,4] and CC.[5] FSL consists of two parts: Floyd-Evans[6,7] productions provide a picture language for syntax recognition and a series of semantic routines linked to certain productions direct the semantic processing when constructs are recognized. Code generation is accomplished through a code bracket facility which allows one to specify arithmetic and control coding using semantic information stored in the pushdown stack. The effort here was mainly toward developing a machine independent formal semantic language.

Unlike FSL, TGS consists of a single language compiled into an interpretive code. It contains a pattern matching facility similar to FSL's productions, but the matching range is more flexible in that it can be used on a number of stacks as well as various properties other than identity of symbols. Code generation first goes through an intermediate phase whose output is then processed by a code selection phase producing output for later assembly. Although TGS has more flexibility than FSL, it requires more detailed information from the compiler-writer.

CC, the compiler-compiler project at Manchester University, uses a top down syntax analysis which produces a complete syntax tree later used by the semantic phase. As it builds a tree top down analysis

*This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-146).

does a considerable amount of backup; consequently, the recognition phase is slow. The syntax phase uses a BNF-like construct for recognition. Non-terminal symbols are handled by format statements which in turn call format routines to do semantic processing and code generation. An open ended design allows new format statements to be defined at compile time; thus, CC can produce extendible compilers.

CABAL

The highlights of CABAL are presented in this section following a brief discussion of the design goals. The design criteria were formulated by an evaluation of CABAL's ultimate environment considered on three levels: physical system, user, and maintenance. Additionally, the fact that these will change demands an open ended design which can grow as the environment changes.

The physical system environment contains both hardware and software appropriate to a rapidly expanding computer utility. This means a configuration consisting of various types of remote terminals, a multiple number of control processors of unlike design, and operating systems capable of time sharing, batch processing, and possible parallel processing.

Maintenance must be considered for compilers produced by CABAL as well as for CABAL itself. With a useable compiler-compiler we may well expect more compilers than would otherwise be the case. Anything which could help to reduce the amount of time for maintenance personnel to learn and maintain systems would be welcomed.

The CABAL user environment will include the usual systems staff interested in implementing permanent systems. However, there will also be researchers working on compiler and compiler-compiler techniques, languages, and large programming systems; graduate students whose theses involve special purpose languages to implement their main concern; and enough non-systems oriented talent looking for means to express itself that lowering the amount of pain incurred to implement a language system will increase the amount of talent used.

Considering the environmental factors and some well known rules of thumb has led to the following list of design criteria:

1. applicability of compiling language translators;
2. wide range of producible translator types;
3. flexibility in a dynamic environment;
4. modularity;
5. power and terseness for the professional systems programmer;
6. transparency for the non-professional user;
7. readability;
8. machine independence;
9. reliability;
10. compatability with environment.

Basic structure, appearance, and control flow

CABAL is an ALGOL-like language and in fact has ALGOL's block structure complete with BEGIN-END pairs and identifier scope. In addition, the internal structure of a block is divided into a declaration part followed by a statement sequence part. Among other things, the declaration part includes declarations for two programming structures; co-routines and ALGOL procedures. A co-routine declaration is syntactically similar to an ALGOL procedure declaration and will be discussed shortly.

At a lower level, modular concepts are applied to the structure of the various language constructs. Concepts with similar meanings exhibit similar structure. For example, the syntax for accessing an element within a multiple element data structure uses square brackets around the indexing information. An extension of this concept demanded square brackets around the indexing information associated with accessing bit fields within elements. Thus, A [3] . [4] requests bit number 4 of element number 3 in data structure A.

Applying modular structural concepts again resulted in declarations resembling those of PL/I rather than ALGOL's. Whereas ALGOL has individual declarations for each kind of typed storage structure, PL/I has a single DECLARE construct which is parameterized. Extensions can easily be accommodated without adding a new declaration concept.

Modular structure concepts and simplicity considerations were responsible for unitizing five data structures into a single concept. An examination of data structure declarations will finish the structural concepts. CABAL has only one data structure. However, it has the capability to function as a scalar, array, threaded list, stack, or queue. Trees and plexes can be built from threaded lists. The syntax is:

```
<declaration> :: = DECLARE <declaration list>

<declaration list> :: = <name list>
                                    (<structure>)
  |<declaration list>, <name list>
                                    (<structure>)

<structure> :: = <nature> <element linkage>

<nature> :: = NUMBER, <bound>
                                  <number optio>n
  |LOGIC, <bound>
```

|STRING, <bound>

|NAME, <bound>

<bound> :: = <number expression> |?|? <number expression>

<number option> :: = , FIXED | <empty>

<element linkage> ::= , [<bound pair list>] <pop linkage>

| <empty>

<pop linkage> ::= , LIFO | , FIFO | , RANDOM | <empty>

<bound pair list> ::= <bound> : <bound>

| <bound pair list> , <bound> : <bound>

<name list> ::= <name> | <name list> , <name>

<name> ::= <letter> | <name> <letter> | <name> <digit>

<empty> ::= (the null string of characters)

<number expression> ::= (expression of type number)

Examples follow for scalars A and B, array C, stack D, and list E:

```
DECLARE A,B (NUMBER,6),
  C(LOGIC, 4, [0:10,0:10]),
  D(STRING, 10, [1:?15],LIFO),
  E(STRING,?,[1:?],RANDOM);
```

Thus, if there is no element linkage a scalar is signified, element linkage without pop linkage results in an array, and the presence of pop linkage signifies how elements will be pushed and popped for stacks, queues, and lists. FIFO signifies first in first out; LIFO, last in first out; and RANDOM, arbitrary inserting and removing.

The expressions following the type indicate how many significant figures associated with the type in question are to be retained. Limits on these will be set by the implementation. A ? signifies an unknown number and results in dynamic allocation. An expression following a ? indicates the user's guess and allows the compiler to take advantage of this information for efficient structuring.

Co-routines

Conway[8] introduced co-routines in 1963 in a paper discussing their use in implementing a COBOL compiler. Since then co-routines have been recognized as a natural implementation for pass concepts in multiple-pass compilers. For this reason, as well as their application to parallel processing, they are included in CABAL.

A co-routine is a programming structure similar to a procedure. However, unlike a procedure which begins processing at its first instruction each time it is called, a co-routine resumes processing following the point where it last gave up control. A second difference is that co-routines do not exhibit the master-slave relationship between a procedure call and the called procedure. A co-routine receiving control is not required to return to the caller but may consider itself at the highest control level; being free to pass control anywhere without creating a chain of linked returns. Thus a co-routine is characterized by an incremental mode of processing and an autonomous mode of access.

Syntactically co-routines are identical to procedures with two exceptions: their declaration specifies CO-ROUTINE rather than PROCEDURE and they may contain a COMMON declaration which has no meaning in a procedure. The COMMON declaration is simply a statement which is executed every time control enters the co-routine in concern. Immediately after execution of the COMMON statement control passes to the internal point which last gave up control.

Co-routines retain their autonomosity when receiving control from GOTO statements. However, they may also receive control in the standard procedure call method. In this latter situation they are required to return control just as a procedure would.

Recursive operation necessitates a different approach than procedures due to their incremental processing nature. Rather than having only the deepest nested recursion available to the system, as is the case with procedures, instances of co-routines are duplicated and each is actively a part of the system.

Allocating an instance of a co-routine may be done one of two ways. For example, if C is the name of a declared co-routine and N is a variable of type NAME, then N←FORM(C) will assign to N the form of C, allowing N to be used as if it had been declared in place of C; and N←COPY(C) will assign to N both the form of C and its current status, again N can be used in place of C. The current status provides current local variable values and the next point of entry associated with the co-routine when the COPY function is invoked.

Reductions and related concepts

A reduction statement provided for syntax recognition offers a convenient pattern matching facility for stack elements. An example will clarify:

L: |'WHILE', EXP|→STA|WHCODE;RETURN|

The sequence of events is as if an IF-THEN statement were executed. If the top two stack entries match the left parts, EXP and WHILE respectively, the statement sequence WHCODE;RETURN is executed, otherwise control passes to the statement following the reduction statement. The right part, STA, is used only if a match occurs; in which case the left part elements are replaced by STA, a single element. This replacement occurs either when the statement sequence completely gives up control or a manipulation statement is encountered in the statement sequence, whichever occurs first. A manipulation statement is simply a * which requests the stack be manipulated now.

Elements in the left and right parts are of two kinds: literals and meta-characters. The literals, signified by quotes, cause straightforward comparisons between the character string enclosed in quotes and the character string associated with the corresponding stack position. Meta-characters represent a group of literals and are matched if the character string associated with the corresponding stack position is identical to any of the group.

CABAL associates a meta-character with its member literals through a define declaration. An example covering unary and binary numerical operators follows:

```
DEFINE UNOP = "−" | "+,"
  BNOP = UNOP |"/"|"*"|"↑"|"o"|"ϵ";
```

Additionally, meta-characters can be defined for non-printing characters like end-of record, end-of-file, carriage return, tab, space, backspace, etc. by using the numerical equivalent of the character in question as a definiens:

```
DEFINE EOC = 64.
```

The matter of the sigma function, a meta-character which always matches, is taken care of by the absence of either a meta-character or literal, when one is clearly called for by the comma placement. Alternatively, for those who demand a printable sigma function, a meta-character may be used whose definens is empty.

Actually reductions are not restricted to operating on stacks, any data structure will do. However, trying to match a two element reduction to a scalar is undefined as is any match which is larger than the declared size of the coupled data structure. A multiple element data structure with ? number of elements is permissible though.

Should ambiguous cases arise such as:

|A,B,A|→A,B|RETURN|

where the meta-character A in the right part could be either of the two in the left, a subscript is allowed in the right part signifying the left part element number.

|A,B,A|→A(1),B|RETURN|

Counting is from right to left.

Couple statements are used to dynamically associate reductions to a specified data structure. Reductions are coupled to the structure specified in the last executed couple statement at the same or higher block level.

There is a system supplied routine which will produce a unique integer value for every unique character string supplied to it. The inverse function is also available. Normally, reductions will expect these unique integers to be stack entries rather than character strings as a considerable time and space saving can be realized. However, should a particular parsing job consist of short strings only, it might be profitable to bypass the string translation. Communicating the nature of stack contents is accomplished by the stack type, either NUMBER or STRING, associated with the stack specified by the appropriate couple statement. An example of a couple statement follows:

```
COUPLE(SYNXSTK);
```

Reductions match only a certain field within NUMBER stack elements extending from digit one to X where X will be defined by the implementation. Thus, the user may declare a wider or multiple dimensioned stack and gain parallel storage if he so desires. Stacks of type STRING will have the entire first element of each stack row matched by the reductions and consequently the only way to gain parallel storage here is with multiple dimensioning.

The left two sections of a reduction statement act as a pattern recognition and generation device for syntax parsing and bear little resemblance to the rest of CABAL even though reduction control is similar to IF-THEN statements. The third section, however, exhibits the full range of CABAL as its content is syntactically defined as a statement sequence, with the exception that another reduction statement may not appear unless imbedded within a BEGIN-END pair.

This structure, along with the placement freedom of reductions allows diverse ways to organize a language translator. Notably, the two ways most usually desired: a complete set of productions following one right after another with their associated semantic routines also grouped in one sequential mass disjoint from the reductions; and alternations of syntax and semantics statements with each syntactic mechanism containing or followed by its associated semantic routines.

T. E. Cheatham[9] has noted (p. 65) that—"While there exist reasonable elegant schemes for 'automatically' doing syntactic analysis (and even much of code synthesis), the handling of declarations is generally messy with any but the simplest of languages. For this reason (among others) it will prove highly useful in any general purpose compiling system to have the ability to do arithmetic and relationals—i.e., the 'action language' should contain at least the rudiments of a good algebraic language."

Rather than just containing "rudiments," CABAL has all the power of ALGOL as well as some significant extensions, notably co-routines, stacks, field and bit level addressing within storage elements, and the reduction statement which is not limited to syntax parsing alone.

Expressions and types:

Expressions have four types as do data structures. However, typing in CABAL does not follow the usual rules. An expression type is determined purely by the operators appearing in it unless it has only a single operand, in which case it takes on the operands type. Thus, a string variable could be numerically added to a logic variable and would produce a value of type NUMBER. The main reason for typing variables is so that information within them may be partially accessed. For example: <variable> . [<field designator>] is a means to address a portion of the variable. If the construction was A. [2:4] we would be addressing the second, third, and fourth digits within element A where digit is defined according to the type of A. Digit definitions are as follows: NUMBER-decimal digit, STRING-alphanumeric character, LOGIC-one bit, NAME-one entire name.

The unary and binary operators are classified according to the expression type they produce.

<unary numerical operator> ::= $-|+|\downarrow$

<binary numerical operator> ::= $-|+|/|*|\uparrow|o|\epsilon$

<unary logical operator> ::= $\neg$

<binary logical operator> ::= $>|<|\geq|\leq|=|\neq|\wedge|\vee$

<binary string operator> ::= &

Some of the unfamiliar operators are & : concatenation, o:mod, ϵ: is a substring of, and $\downarrow$: truncation. The substring operator provides a value of 0 if the left string is not contained in the right string and a value corresponding to the digit position of the left string's first character match in the right string if it is contained.

Name expressions never contain operators as their function is to convey pointers. For example, a name variable N might be assigned the name of a label, N←NAME(LABELX), and then appear in a subsequent GOTO statement. Name variables may be used in place of any CABAL name provided they have been assigned either the name, copy, or form of the item they are representing. Assigning N[X]←A allows N[X] to be used in place of A as N[X] has a pointer to A. However, if the value of A is changed then the value of N[X] has also been changed. To circumvent this, N[X]←COPY(A) will duplicate A and assign a pointer of the copy to N[X]. Additionally, [NX]←FORM(A) will allow N[X] to be referenced as if it had been declared the same way A had. The name variable concept is particularly useful in building up threaded lists or trees.

Statements:

There are four statements specifically supplied for semantics processing: assignment, conditional, iterative, and push-pop. The assignment statement is straight forward:

<assignment statement> ::=
<variable> ← <expression>

<expression> ::=
<name expression> | <number expression>
| <logic expression> | <string expression>

The replacement operator does not affect type as the value of the expression is stored in the variable with no regard to type differences. If the length of the expression value cannot fit in the variable then the replacement is undefined.

The IF-THEN conditional is standard ALGOL and needs no further explanation. The iterative statement is as follows:

<iterative statement> ::=
<iterative condition> DO

<iterative condition> ::=
WHILE <logic expression>
| FOR <variable> ← <number expression>
<terminal condition>

<terminal condition> ::= TO <number
expression> BY <number expression>
| TO <number expression>
WHILE <logic expression>
| WHILE <logic expression>

Although the syntax has been shortened, the semantics here are similar to ALGOL for statements.

The push-pop statement is used to push and pop elements associated with LIFO, FIFO, and RANDOM linked storage.

<push-pop statement> ::= <stack operator> <name>
| <stack operator> <group reference>

<stack operator> ::= Δ | <stack operator> Δ
| ∇ | <stack operator> ∇

<group reference> ::= <name> [<group definition>]

<group definition> ::= <index range list> <number expression list>

<index range list> ::= <empty> |*, | <index range list>*,

To exemplify the range of manipulation this buys consider the following structure:

DECLARE MST (LOGIC, 32, [1: X,1 :Y, 1:Z], RANDOM);

This gives us a master symbol table similar to the one which must be declared by the user at a block level sufficiently high to encompass all code bracket statements. Z is the maximum element number in the list and corresponds to the maximum number of identifiers the users compiler will handle; Y specifies the number of 32 bit storage elements needed per identifier; and X specifies the number of remembered nested declarations any one identifier may have. By asking for RANDOM linkage we can push and pop any group of elements in the list at any time. The result of pushing element [*,*,2] is to have its address now by [*,*,3] ; [*,2,2] goes to [*,3,2] ; and [2,2,2] goes to [3,2,2]. Popping any one of these, though, removes it entirely from the list and changes group [*,*,3] to [*,*,2] and so on.

Consequently, using this as our master symbol table allows us to push and pop information concerning every declaration of a given identifier name as well as only that information associated with its declaration at a specific block level.

The master symbol table (MST) serves as a communication link between the semantics processing and code generation. As will be discussed in the code generation section, storage allocation and other necessary MST information is entered into the MST by the code generators. During semantics processing this information as well as additional data the semantics routines may store in the MST is available for use.

Code generation

Co de generation is accomplished through the use of code statements and item statements. A code statement simply encloses in code brackets constructs which for the most part compose the CABAL language as a whole. Item statements are used to provide parameters in conjunction with the indicated code from a code statement. Thus, the translator writer merely translates his input stream into valid CABAL and encloses it in code brackets.

There are a few restrictions and extensions which may appear in code brackets. With the exception of having to put out complete statements, structure is non-existent. Declarations and statements may be interspersed at will. A declaration in code brackets causes the generator to push the MST at the appropriate place and store the necessary information. Subsequent usage of declared items in statements causes retrieval of MST data in order to produce code.

The output from code brackets is a stream of items which are interpreted by the generator into code. Code is produced whenever a sufficient amount of information comes through the code brackets. Thus, one statement may be executed such as CODE(#1), producing no code until the construct is sufficiently completed by, perhaps,

CODE(←#1+#2).

All names within code brackets which must be identified have syntax =#<integer>. This relates them to parameters in the most recently executed item statement. For example:

ITEM(SMT[4],SMT[1],#A,#B);
CODE(FOR#1←#4 TO #3 BY #2 DO);

Here we have the parameters in the item statement implicitly numbered starting with 1 at the left. The code statement designates which parameter it wants by signifying the number. The # in the item statement is used to indicate that a parameter represents a constant rather than a declared name.

When the generator receives a parameter representing a declared name, it looks in the MST to get the appropriate storage address and related information. When a parameter name has more than one declaration, the top one is always used.

Should it be necessary to pass a name parameter to the generator which has not yet been declared, like a label name for instance, the CHAIN function is useful. This function need only be used once per name as it continues to chain all instances of the name until the ASSIGN function is invoked. ASSIGN is used after the appropriate declaration has been made and retraces the chained list inserting appropriate addresses and completing any unfinished coding.

There are times when a compiler written in CABAL wishes to transfer some of the data collected during compilation to the compiled program. This is facilitated quite easily as:

```
ITEM(RUNNAME,#TABLE);
CODE(#1←#2;);
```

This transfers the data stored in TABLE, a compile-time data structure, to a run-time data structure of identical dimension called RUNNAME which has been previously declared in a CODE statement.

Control can be passed to compiled code by generating a GOTO <label name> NOW; where <label name> is some pre-declared label in the generated code. Returning control to the point following the code statement which gave up control will happen if execution runs into a pre-generated RETURN NOW. Execution of HALT by either the compiler written in CABAL or the translated code produced by the compiler will give control to the operating system.

CONCLUSION

Looking now with an eye for comparing the language description with the design criteria, there are a number of points worth mentioning. For applicability to compiling-compilers there are included co-routines for natural phase and pass separation, reductions for syntax parsing, full algebraic power for semantics processing, structures and operators for string manipulation, system supplied routines for handling strings comfortably, a master symbol table easily accessible by both code generation system routines and user written semantic routines, and a code generation facility that couples the easy and nonchalant use of a high level language with optimal system routines capable of streamlining the resultant code to a degree determined by how much time is deemed worthwhile.

Producible translators include interpreters, which make use of the output facility for compile time data and code as well as reduction statements; conversational compilers, aided by ease of control flow between environment, compiler, and generated code; and multiple language systems using co-routine and reduction stack coupling. Compatibility is maintained with a dynamic environment through machine independence and an open ended design ready to accept extensions. In this respect a macro facility is anticipated and seems reasonably easy to implement.

Modularity is provided not only by the language structure and its program and data structures, but also at a lower level its statement and declaration syntax is modular to a degree that makes extension trivial. Flexibility, besides benefitting from points already mentioned, is further assured by an absolute minimum of communication linkages and interdependencies within and among the language system subdivisions and the system environment.

There is full algebraic power as well as the ability to reference all system variables and even drop down to an assembly language sequence. Readability and transparent learning is assured from the ALGOL nature as well as keeping the concepts to a minimum and using a high level language for code generation. The syntax was specially geared for terseness to such an extent that FOR-STEP-UNTIL-DO was superceded by FOR-TO-BY-DO.

Finally, reliability is facilitated through an easily understood and well-partitioned language system with no major hidden subdivisions. To the author's knowledge, this is the first high level language to provide recursive co-routines, a structure well suited for compiler writing. A generalized data structure declaration providing scalars, arrays, lists, stacks, and queues as well as operators for these structures are conveniently imbedded in any compiler produced by CABAL. This last point is in answer to the Feldman-Gries[9] statement (p. 28): "One central question in any TWS [translator writing system] is the choice of data structures built-in at both translation and execution time . . . essentially nothing has been done to provide built-in structure operators for execution time." The code generation facility provides the answer to this problem by using CABAL as an output language, providing storage, operation, and control transfer through a high level language.

Optional control over compiler storage efficcy isent provided in the data structure declaration. The intein is that the compiler writer tell CABAL as much as he knows about storage requirements; receiving the most efficient storage layout for the information he provides. Code generation also has optional features governing the efficiency of compiled code. A simple declarative means allows the compiler writer to request varying degrees of code optimization; depending on how much compile time he is willing to sacrifice for shorter run time.

Until CABAL has been used on a few systems, an attempt to name its shortcomings would be premature. One possible disadvantage might be the time taken for code generation. This, however, will be a function of the completeness of each CODE statement and consequently will be under some control by the compiler writer.

Although compiler-compiler techniques are by no means near to being a closed issue, there is enough foundation material currently available that a little bit of polish can produce a palatable system. I think

CABAL demonstrates that programming languages aimed at complex problems not fully formalized or understood need not wait for absolute insight before user problems are considered. In fact, making the usage of such a system as painless as possible will do much for increasing the number of users and thereby quicken the time when language translation is yesterday's problem.

ACKNOWLEDGMENTS

The CABAL design project involved a number of people over its three years duration and is indebted to each of them. Jan Fierst, the initial project leader, and Mary Shaw did much of the initial design. Other contributers include James Eve, Edward McCreight, David Adams, and Bernard Lansac. Working papers written during the design process by these project members are available as computer center reports and serve as evolutionary documentation of the design effort.

REFERENCES

1 J FELDMAN D GRIES
Translator writing systems
Comm ACM Vol 11 No 2 Feb 1968 pp 77–113

2 J A FELDMAN
A formal semantics for computer oriented languages
Computation Center Carnegie-Mellon University Pittsburgh Penna 1964

3 J PLASKOW S SCHUMAN
The TRANGEN system on the M460 computer
AFCRL–66–516 July 1966

4 T E CHEATHAM
The TGS-II translator-generator system
Proc IFIP Congress New York 1965 pp 592–593

5 R A BROOKER D MORRIS J S ROHL
Compiler-compiler facilities in atlas autocode
cumput J9 1967 pp 350–352

6 R W FLOYD
A descriptive language for symbol manipulation
J ACM 8 Oct 1961 pp 579–584

7 A EVANS
An ALGOL 60 compiler, nanual review in automatic programming
Vol 4 1964 pp 87–124

8 M E CONWAY
Design of a separable transmition diagram compiler
Comm ACM July 1963 pp 396–408

9 T E CHEATHAM
Notes on compiling techniques
Computer Associates Inc Wakefield Massachusetts 1965

Checkout test language: An interpretive language designed for aerospace checkout tasks

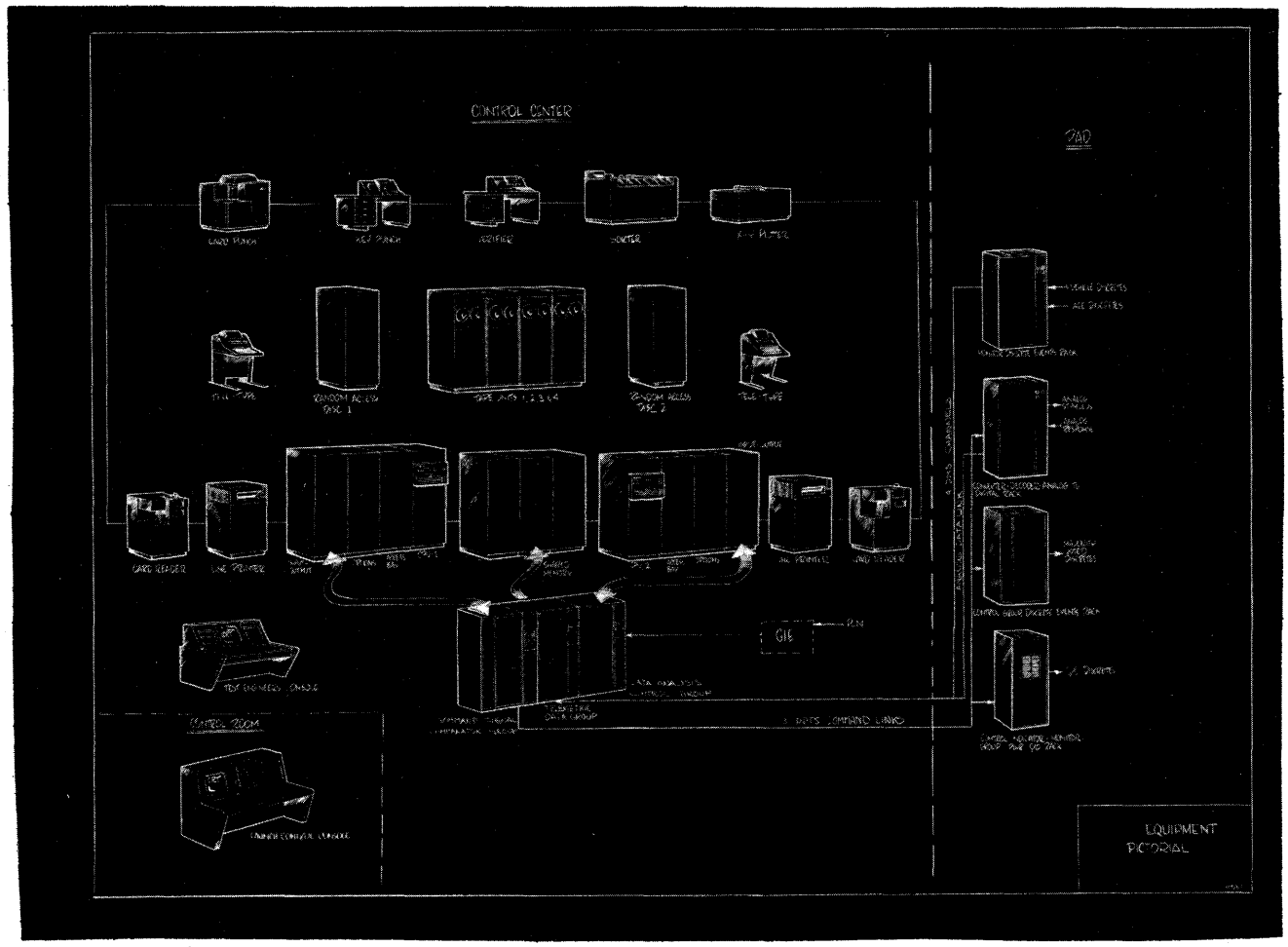
by GENE S. METSKER

Martin Marietta Corporation
Denver Colorado

The topic of this paper is the CHECKOUT TEST LANGUAGE (CTL), which is a new interpretive test-oriented language designed in near-English form. However, before discussing the language itself, a brief description of the system will be given so the language/software/hardware relationship can be shown.

The center of the system is two general purpose computers (reference Figure 1). The two central processor units (CPU's) can operate independent-

FIGURE 1

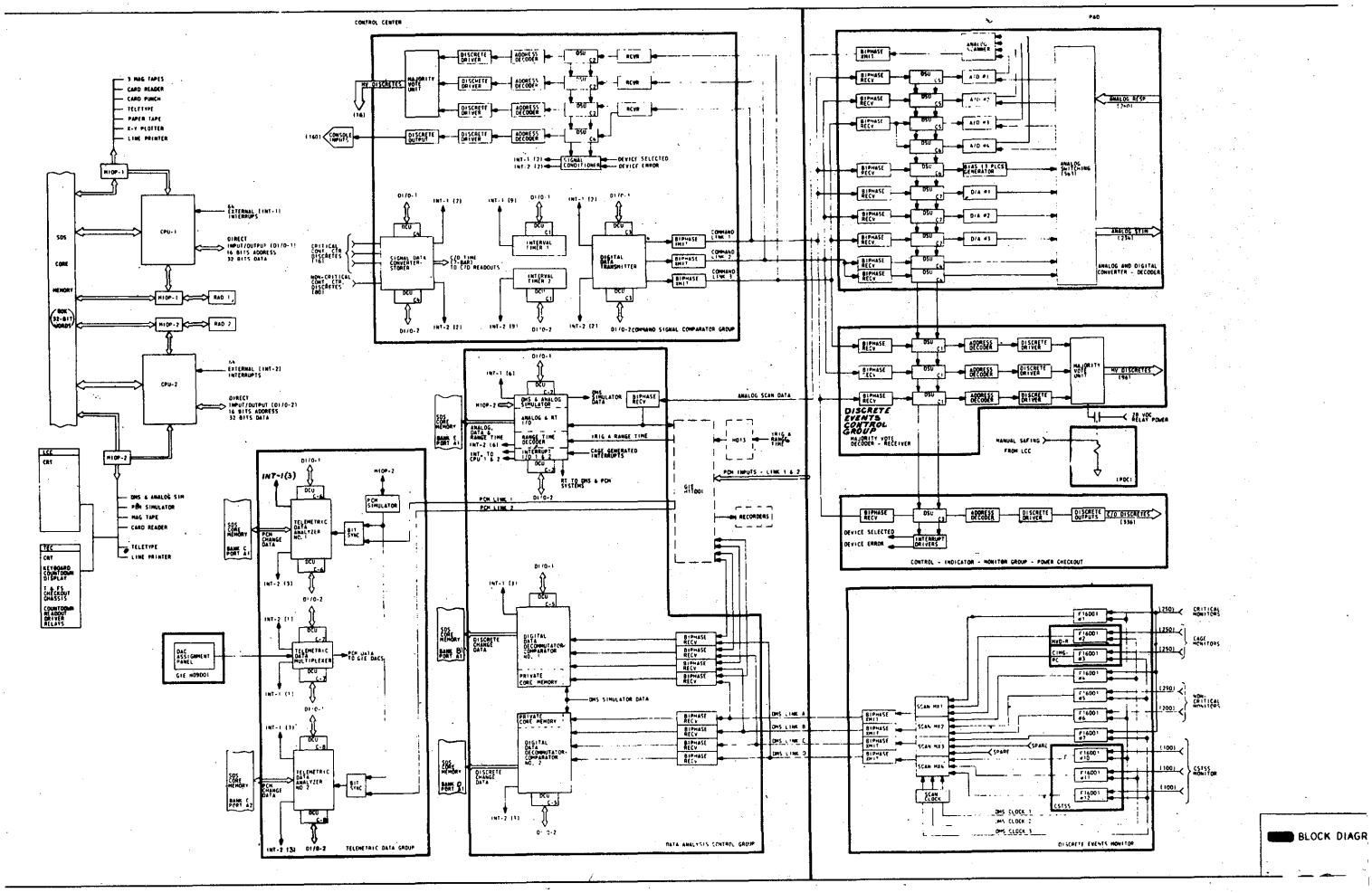


ly or in a dual redudant mode during critical operating periods. The two CPU's share 80k of core memory contained in five 16k banks. The computer word is 32 bits, and memory is addressable and alterable by byte, halfword, word and doubleword. Numerous peripherals interface with each CPU. Associated with CPU I are three magnetic tape units, one 1000 LPM line printer, one card reader, one card punch, one teletype, one X-Y plotter, one paper tape reader, and one 24 M BYTE rapid access disc. Associated with CPU 2 are 2 CRT displays, one magnetic tape unit, one 1000 LPM line printer, one card reader, one teletype and one 24 M BYTE rapid access disc.

Output control functions from the CPU's, located in the control center (reference Figure 2) are routed through the command I/O, verified for correctness, and transmitted to the remote equipment over one or more of the three 300kc biphase transmission links. The instructions are received, checked for correct parity, decoded and routed to the correct device. Located at the remote installation are three analog stimulus generators, three bias generators, four programmable A/D converters, a 600 point analog switch matrix, 320 single-input discrete switches, 96 majority voted discretes, and Data Monitor System (DMS) scanners capable of monitoring on a real–time basis, 3000 discrete inputs.

The analog data are returned from the remote location to the control center over a single 250kc biphase transmission line. The four A/D output are multiplexed onto this line. The discrete data are returned to the control center via four 250kc biphase transmission lines. Each transmission line carries a maximum of 750 channels of discrete data, thereby establishing a scan cycle for up to 3000 channels of 3 milliseconds. The analog data

FIGURE 2



are routed through the analog I/O at the control center to memory bank C. The CPU then compares the data value against stored limits after appropriate manipulation of the data. The DMS data to the DMS I/O is compared against stored "success criteria" which is continually updated as a function of the sequence, events or time. When the criteria is not satisfied, one or both of the CPU's is notified of the anomaly, and suitable error sequences are initiated.

Additionally, the PCM data is routed to the PCM I/O from the Ground Instrumentation System (GIE) in the form of two serial data links. The analog and bi-level PCM data are not used by CAGE as hold criteria, but all error data will be printed and are available as backup data to the analog and discrete system.

Having briefly described the checkout system, it is relevant to now examine the need for a special purpose test language.

The pattern has become well established that any entrant into the aerospace ground equipment field is going to have a computer as an integral part of the system. The computer may play only a passive monitoring role, or it may be fully active in the control loop, but it will be there.

This is a desirable evaluation in checkout technology. Computers can play a very useful role in the checkout field due to their ability to analyze, sort, and report on massive quantities of data in near real time. If they are also used in the control loop, computers can take alternate or corrective action to a dynamically changing situation by selection of the proper alternate path among a myriad of possibilities, while conventional test equipment is normally limited to a single preselected "abort" procedure, or worse, reliance on the human to take proper action.

However, to realize the potential of a computerized checkout system, the combined intelligence of many experienced engineers and scientists must be entered into the computer before it is turned loose in an operational environment. It is the purpose of this paper to examine how this transfer of intelligence takes place, and how steps can be taken to make this process almost painless for the human and the computer too!

It is unfortunate but indisputable that most experienced engineers who must define the checkout and control tasks are not experienced programmers, and most computer programmers are not experienced test engineers. A language such as FORTRAN enables a man to easily master the use of a computer for general purpose business use or standard mathematical operations, but do not even begin to satisfy the aerospace requirements which require unique hardware and software manipulations.

When Martin Marietta's checkout system was in the conceptual phase, one of the prime areas of emphasis was in the development of a language that was near-English in form but which contained all the tools by which the test engineer could express commands and criteria completely, simply and unambiguously. Additionally, it was considered imperative that the format be readily understandable to all interested observers such as the customer, the user and the reviewer. Thus, the checkout TEST LANGUAGE (CTL) was developed. This paper will describe the salient features of this language, and will examine some of the trade-off's that must be made when designing such a language.

The basic objective of the CTL is to provide a vocabulary of terms which can be easily translated into machine instructions. The vocabulary must provide the flexibility that the test writer needs to set up and evaluate his test. However, most checkout and control systems usually have potentially hazardous conditions that they can cause, or must react to, and because of this it is often desirable to place certain hardware or software safeguards or restrictions on what the man is allowed to do. When designing the CTL it was decided to provide a great deal of flexibility in the basic language structure, and make extensive use of diagnostics during the off-line translation and on-line execution of a sequence in order to protect against illegal or hazardous operations. Thus, the man is not restricted in his use of the language, but the test article is protected against inadvertent human errors.

A second object of the CTL is to reduce the writing burden of the test engineer by providing sequence preparation aids. These aids enable the writer to abbreviate his tests in such a form that the translator (the software programs that convert test language statements into machine usable form, can expand the statements into a fully defined list which can be easily understood and reviewed by all interested parties. Most typical sequences written in test language undergo about

a 20:1 expansion ratio as they are passed through the translator. Also, each CTL statement requires about four computer words after translation. Thus, the total reduction in the physical writing task from an assembly language program to a CTL program is about 80:1. This means that in addition to writing in a near-English form using common everyday verbage, the test writer is required to specify only 1.25% as much as he would if he were writing in machine language.

Before describing the CTL itself, it is relevant to now summarize the importance of an interpretive language.

1) Assembly language programmers who do not understand test techniques and requirements are not involved in generating sequences.
2) Test engineers who do understand vehicle test requirements can prepare the sequences in a near-English format.
3) The customer and contractor quality control personnel can easily understand and validate the sequences.
4) Any addition, deletion or update to any sequence can be accomplished by an engineer by simply stating the change in test language.
5) Field modifications to translated sequences are easily accomplished.
6) The writing burden has been reduced almost a hnudredfold.
7) On-line use of the CTL enables generation and execution of tests on-site with near zero delay.

The CTL consists of 20 basic elements or verbs and their associated modifiers.

The various test elements may be classified according to their function and how they relate to the on-line programs. Many of the elements correspond to element subroutines, while others are provided by the translator to simplify preparation of test sequences in the Test Language.

The elements which correspond to internal programs may be roughly divided into four classes:

1) Discrete Signal Observation and Control
   APPLY
   RESET
   CHECK/DISCRETE
2) Analog Signal Observation and Control
   CONNECT
   STIMULATE
   MEASURE
   CHECK/ANALOG
3) Test Flow Control
   BEGIN
   END
   TIME
   SET
   DEFER
   CONTINUE
4) Special
   DISPLAY
   SAVE
   RESTORE

Several "Elements" and translator facilities are provided as tools for the test engineer. All of these are provided to eliminate non-productive repetition of recurrent test sequences, and detailed descriptions of frequently used groups of data.

Features designed to facilitate test sequence preparation include:

1) SEQUENCE
2) REPEAT
3) REP/Test
4) SYSTEM/TEST

The first of these, SEQUENCE, is actually an element used in on-line control. It facilitates the use of established library test sequences within any other sequence. The other three are facilities of the translator; i.e., the translator inserts the repetitious data or elements into the control file using the REP/TEST, REPEAT, OR SYSTEM/TEST elements as a mode.

Before further elaboration of the elements themselves, it is useful at this time to demonstrate the organizational structure of the test sequences.

The highest level of organization in the test language is the test sequence. The test sequence is composed of a logical grouping of lower level blocks, which as a whole, accomplish the specified task. The terms describing the levels of organization are unique to this language (reference Figures 3 and 4).

ELEMENT—The test element is the basic building block of the test language. Each element serves a specific purpose such as commanding a relay closure, establishing a stimulus, etc. Each element has a set of modifiers asso-

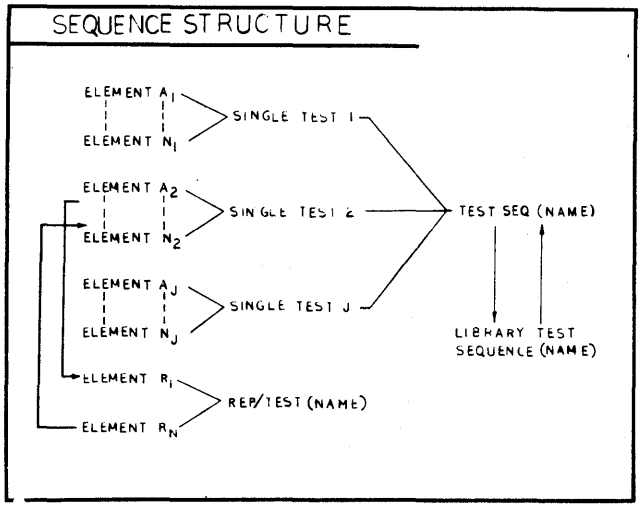


FIGURE 3

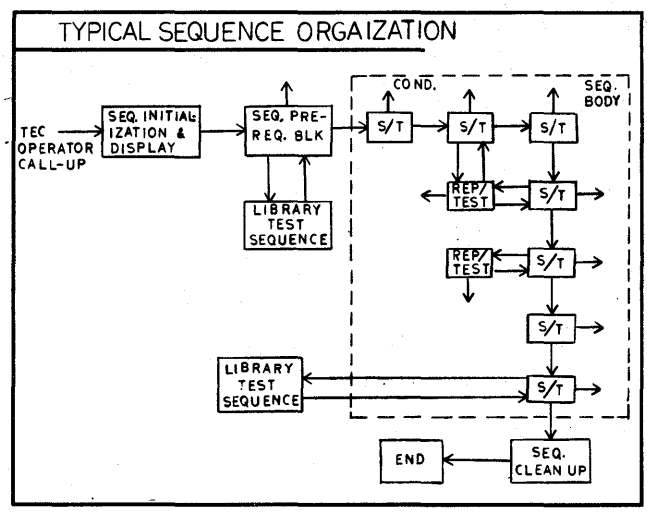


FIGURE 4

ciated with it which serve to amplify the element to specify uniquely the required operation. There is *no* test language level below the test element and its associated modifiers.

SINGLE TEST—Test elements grouped together to satisfy some unique control, response or display function make up a single test. The number of test elements within a single test may vary from 1 to several dozen. Each single test will be assigned a unique number.

REPETITIVE TEST—Many operations are repeated frequently with only slight parameter variations. The repetitive test allows the user to establish the order and number of the test elements required, and simply by varying the numeric value assigned to the parameter list and calling out the unique repetitive test name, repeat the test without restating each test element. A repetitive test cannot stand alone, and must be defined in any sequence that uses it. It may contain as many test elements as required, but will *not* contain additional single test numbers. The repetitive test name is 5 alphanumeric characters, and the parameter list can contain up to 25 variables.

LIBRARY TEST SEQUENCE—A library test sequence is a specialized version of a test sequence. The library test sequence category will be used whenever a function (such as apply vehicle power) is used in many other test sequences. It must contain all required prerequisites and control and response statements required to satisfy the function the library test sequence is designed for. A library test sequence can be called for singularly from the Test Engineer's Console (TEC). It may also be called by any currently executed sequence. The library test sequence will have a number of the same form as a test sequence except it will be uniquely identified as a library test sequence. The library test sequence may be called by any test sequence as long as it has been defined and translated previously.

TEST SEQUENCE—A test sequence is a collection of single tests, repetitive tests and library tests arranged in a logical order to accomplish a specific testing or control function. Each test sequence can be called for from the TEC; any test sequence may call other test sequences, which may in turn call other test sequences. The Go and No/Go logical paths through a test sequence shall be defined by properly linking the single tests, repetitive tests and library tests together in an unambiguous manner (reference Appendix A).

Having demonstrated the organizational structure of test sequences, the basic language elements that are used to build these sequences follow:

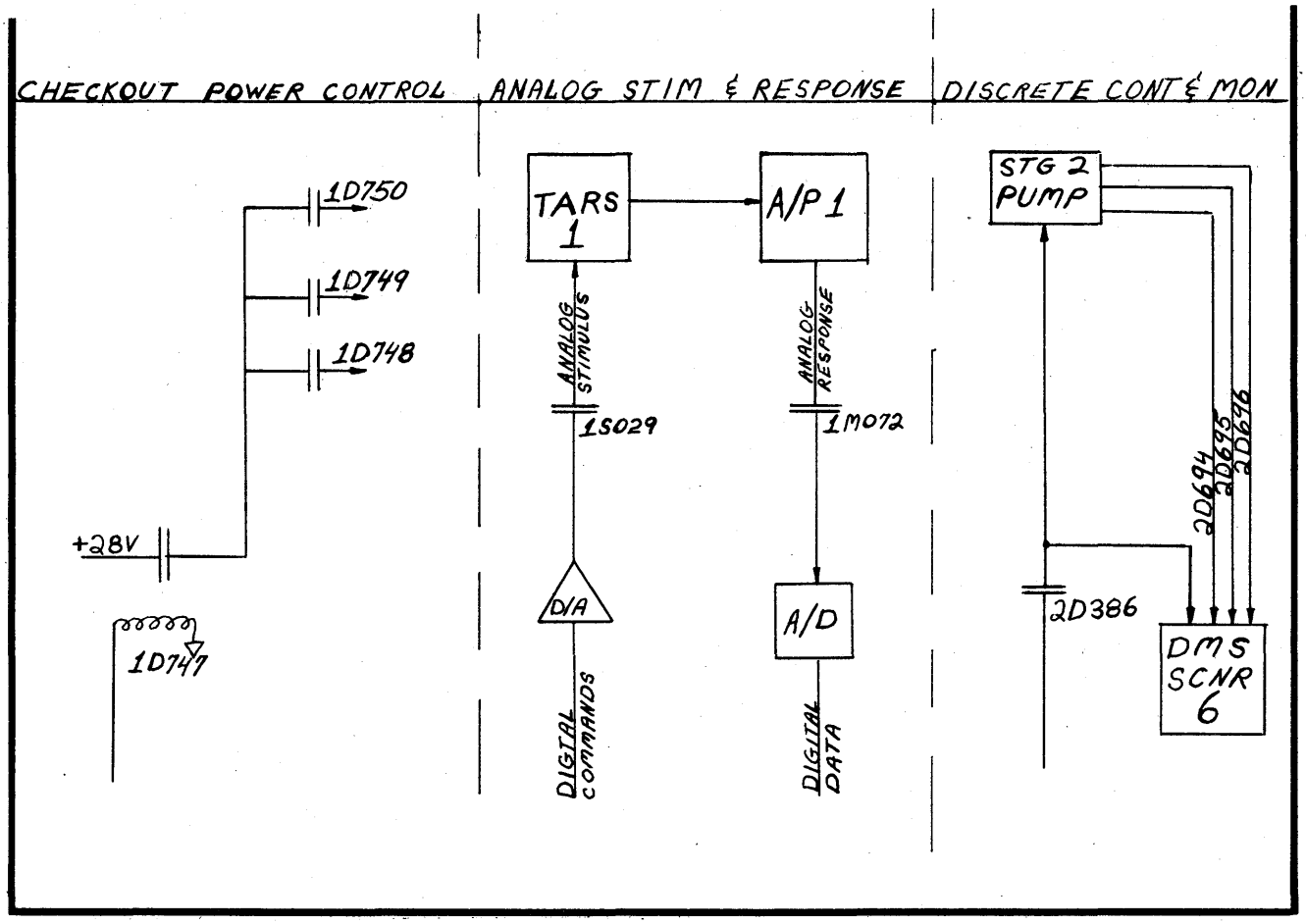
OFF–LINE LANGUAGE

ELEMENT	PURPOSE
1. BEGIN	Used to transfer sequence control or unconditionally branch.
2. END	Used to designate the logical end of a sequence to the interpreter. Also serves as a key to return to higher level sequence.
3. SEQUENCE	Used to call any supporting or library sequence. Control is returned when the sequence is completed.
4. REP/TEST	Used to call a repeat test (RTNNN) which defines a pattern of elements and modifiers with variable parameters.
5. APPLY	Used to apply majority voted or single ended discretes.
6. CONNECT	Used to connect analog stimulus, measurement and internal switching.
7. STIMULATE	Used to apply DC voltage and current stimuli, set up the bias generators, and generate .lcps to 20 cps sinusoids.
8. MEASURE	Used to set up the A/D's as to mode, range, filter, sample, rate, filter constant, etc.
9. CHECK/ANALOG	Used to specify type of analog computation, limits, time, etc.
10. CHECK/DISC	Used to check the status of a discrete channel, either instantaneously or within some time window.
11. SET/DMS	Used to establish continuous monitor criteria for DMS channels or to put them in a "don't care" state.
12. SET/PCM	Same as SET/DMS, except used with PCM channels.
13. DEFER/TIME	Defers execution of a string of elements until the specified time elapses.
14. DEFER/KEY	Defers execution of a string of elements until the specified key event occurs.
15. CONTINUE	Used with the defer's to continue normal execution.
16. SYSTEM/TEST	Like REP/TEST, except it is valid only at system level (not sequence peculiar).
17. TIME	Used to set monitor timers, specify time delays, and start, stop or key to countdown time.
18. RESET	Resets: Discretes, D/A's, B/G's, A/D's, analog switches and timers.
19. REPEAT	Used as a handwriting tool to repeat elements on a single test level.
20. DISPLAY	Used to control displays of milestones, text, data and no-go's. Also used to cause a sequence halt.
21. SAVE	Used to save the status of the system at any given point.
22. RESTORE	Used to restore the system back to any given SAVE point.

APPENDIX A

In order to clearly demonstrate what a test sequence looks like in test language, a short example will be given below. This example sequence involves turning on certain checkout power, issuing some discretes, setting some discrete monitors and then verifying the response of a channel of an analog autopilot. Comment cards (those with a "C" in Column 1, are used to clarify the test steps).

```
*SEQUENCE APTEST 'TEST OF PITCH TARS CHANNEL'
*SEQUENCE ELEMENTS
C DISPLAY THE DESCRIPTION OF THE SEQUENCE ON THE CRT AND THE PRINTER
     10.0D/T,      'A/P TEST,TARS PITCH CHANNEL,STG 2'
C       APPLY CHECKOUT POWER,CHECK THAT THE POWER COMES ON PROPERLY
C AND ESTABLISH CONTINUOUS MONITORS ON THE POWER
     20.0SET/DMS,OPEN,1D747
      ANDSET/DMS,OPEN,1D748
      ANDSET/DMS,OPEN,1D749
      ANDSET/DMS,OPEN,1D750
         APPLY,1D747
         CK/D,1D747,ON,15MS,20.1
         APPLY,1D748
      ANDAPPLY,1D749
      ANDAPPLY,1D750
         CK/D,1D748,ON,15MS,20.1
      ANDCK/D,1D749,ON,15MS,20.1
      ANDCK/D,1D750,ON,15MS,20.1
         SET/DMS,ON,1D747,CLOSE,ERON,PAUSE
      ANDSET/DMS,ON,1D748,CLOSE,ERON,PAUSE
      ANDSET/DMS,ON,1D749,CLOSE,ERON,PAUSE
      ANDSET/DMS,ON,1D750,CLOSE,ERON,PAUSE
         BEGIN 30.0
C ESTABLISH A CONDITIONAL (NO-GO) TEST THAT WILL BE BRANCHED TO
C IF ANY OF THE CHECKS (CK/D) IN 20.0 FAIL
C THIS TEST WILL DISPLAY THE NO-GO'S, STOP EXECUTION AND RESET THE DISCRETES
     20.1D/NO-GO,DISC,STOP
         RESET,1D747
      ANDRESET,1D748
      ANDRESET,1D749
      ANDRESET,1D750
C TURN ON STG 2 HYDRAULIC SYSTEM,ESTABLISH MONITORS FOR HYDRAULIC TEMP
C PRESSURE AND RESVR LVL AND SET UP A RUN TIME MONITOR
     30.0SET/DMS,OPEN,2D386
      ANDSET/DMS,OPEN,2D694
      ANDSET/DMS,OPEN,2D695
      ANDSET/DMS,OPEN,2D696
         TIME,T1,5MIN,SEQUENCE HYDOFF
         APPLY,2D386
         CK/D,2D386,ON,15MS,30.1
         TIME,DELAY,10SEC
         CK/D,ON,2D694,30.1
      ANDCK/D,ON,2D695,30.1
      ANDCK/D,ON,2D696,30.1
         BEGIN 40.0
     30.1D/NO-GO,DISC
         RESET,2D386
         D/P,'ST 2 HYDRAULICS NOT APPLIED, OMIT TESTS 40 TO 120'
         BEGIN 130.0
C STIMULATE THE PITCH TARS CHANNEL,LOOK AT THE 1-2 A/P OUTPUT WITH AN
C A/D CONVERTER,AVG THE OUTPUT OVER 1 SEC, AND COMPARE AGAINST LIMITS
     40.0CONNECT,1SC29
         CONNECT,1MC72
```


CHECKOUT POWER CONTROL
ANALOG STIM & RESPONSE
DISCRETE CONT & MON
1D750
1D749
1D748
+28V
1D747
TARS 1
A/P 1
ANALOG STIMULUS
1S029
ANALOG RESPONSE
1M072
D/A
A/D
DIGITAL COMMANDS
DIGITAL DATA
STG 2 PUMP
2D694
2D695
2D696
2D386
DMS SCNR 6

```
    STIMULATE,D/A1,+26.5MA,RNG4
    MEASURE,A/D1,RNG2,V,LIM(1.15,1.35)V,200MS,40.1
    CK/A,A/D1,AVG,1SEC,LIM(1.2,1.3)V,40.2
    RESET,D/A1
    RESET,A/D1
    RESET,1SC29
    RESET,1M072
    BEGIN 50.0
40.1D/N0=G0,ANAL0G,ST0P
    RESET,ANAL0G
    BEGIN 50.0
40.2D/T,'LIMIT CHK PASSED,AVERAGING TEST WAS N0=G0'
    D/N0=G0,ANAL0G,ST0P
    RESET,ANAL0G
50.0          AND S0 0N AND S0 F0RTH
```

ELEMENT / MODIFIER	BEGIN	END	SEQUENCE	REP/TEST	APPLY	CONNECT	STIMULATE	MEASURE	CK/ANALOG	CK/DISC	SET/(DMS)(PCM)	DEFER/(KEY)(TIME)	CONTINUE	SYSTEM/TEST	TIME	RESET	REPEAT	DISPLAY	SAVE	RESTORE
	TEST NO.	NONE	SEQ. NAME	RT NAME	CAGE NO.	CAGE NO.	D/A NC.	A/D NO.	A/D NO.	CAGE NO.	CAGE NO.	CAGE NO.	NONE	ST NAME	TIMER NC.	D/A	REP. NC'S	TEXT	TEMP	TEMP
				"P" LIST			B/G NO.	MODE	INTER -VAL	GROUP NO.	GROUP NO.	TIME VALUE		"P" LIST	INTER -VAL	B/G		MILE-STONE		
							SIN	RNG	LIM-ITS	ON/ OFF	ON/ OFF	ON/ OFF			DELAY	A/D		DATA		
							FREQ	SAM-PLE	COMP-UTA-TION	N/M	OPEN/ CLOSE	DEF. ELE-MENTS			C/D VALUE	ALL		STA-TUS		
							LEVEL	LIM-ITS	RE-TAIN	DEC. TIME	ERON/ EROF				C/D START	ANAL. SW.NO		PAUSE		
							RNG	INTER -VAL	ADD, SUB DIV	COND. TEST	ALT. AC-TION				C/D STOP	ANA-LOG		NC-GO		
							DURA-TION	SMO-OTH	DATA1 DATA9		NULL, AMB, OP.LIM				PAUSE /SEQ.	TIMER NO.		PRINT		
								SPS	COND. TEST							C/D		LCC		
								POINTS								DISC SW.NO		CAGE NO.		
								COND TEST								DISC-NO.		DE-VICE NO.		
								DATA RATE										AMB		

CTL ELEMENT AND MODIFIERS SUMMARY

An efficient system for user extendible languages

by MALCOLM C. NEWEY

University of Colorado
Boulder, Colorado

INTRODUCTION

The recent survey by Feldman and Gries on Translator Writing Systems[1] devotes space to discussion of "extendible" compilers but fails to explain fully the motivation. The reasons for an extendible high level language (and hence for an extendible compiler) are as follows.

i) It is not feasible to foresee all the programming techniques which have not been developed at the time of language definition, and which might deserve a place in a language.

ii) It is not practical to incorporate all the data structures, operations and statement forms currently known into one colossal language since this language will be barely teachable, and implementation may be uneconomical or maybe impossible.

Extendible compilers have, of course, not been available and it has been shrugged off as a fact of life that most programmers constantly come across problems to which the language available is really not suited. The normal "solutions" of this difficulty are both unattractive. The first way out is to abandon the general purpose language for assembly code or change to some special purpose language (which may require a compiler being written for it). This line of attack has given rise to a multitude of special purpose languages such as string processors, list processors, etc. The other way out of the problem is to resign oneself to tedious, inefficient, complicated code, and is illustrated by the way many ALGOL 60 programmers incorporated **complex** variables into their programs.

i) represent each complex quantity by a 2 element array.

ii) operations between complex quantities are performed by procedures and so z: = (u + v)* (u — v)/(u * v) is computed as follows:—

```
add complex (a, u, v);
subtract complex (b, u, v);
multiply complex (c, a, b);
divide complex (a, c, u);
divide complex (z, a, v);
```

This awkward dilemma can be avoided completely if we have a language with very strong basic structure (such as ALGOL) and a facility for extension by the user of the language. The problem with this scheme is that an extendible compiler must be devised. This is a problem of automating something that compiler writers are old hands at; that is, amending the compiler to recognize extensions to the base language.

There are two semi-automatic schemes for extendible compilers that deserve a mention:

i) *Syntax Directed Compiler Approach*

The Syntax Directed Compiler[2] is a compiling system accepting first the translation rules of a language and then a source program in that language, to produce a target program according to the rules supplied. The system has great merit where researchers are experimenting with various source languages. The system could be made to process an extended source language by altering the translation rules appropriately.

ii) *The Compiler-Compiler Approach*

This genus of translator writing system (usually associated with Brooker, Morris, et al.[5]) accepts a definition of a phrase structure language including the meaning of each of the phrases (in terms of the target language) and produces a compiler. As with the Syntax Directed compiler the description of the source language can be altered to accommodate the extended language.

This paper describes an approach to this subject which is claimed superior to each of these possible attacks and also superior to the approaches described by Cheatham,[3] Galler, Perlis.[4]

It must also be noted that although ALGOL 60 is used as a basis for discussion (as did Galler and Perlis,[4] the notions can be extended to extendible compilers for other high level, powerful, general purpose languages such as ALGOL 68 and PL/1. Its use is justified by the fact that ALGOL 60 is more familiar.

Nature of useful extensions

It must be realized that if one has an automatic means of extending a compiler then there is precisely the same problem (as encountered with languages such as ALGOL) that led to the need for extendible languages. That is, it is probably not possible to be simple to learn and operate, readily implementable and completely general. This difficulty is offset, as will be indicated in this section, by the fact that there are a small number of discrete areas which contain most extensions. The difficulty is also offset by the fact that there are often many ways to extend a language but only certain of these can be said to be in "harmony" with the language, and it appears that these harmonious extensions are the most powerful and most easily describable. To illustrate this last point, consider a typical example of a feature to be added to ALGOL: string processing as used in SNOBOL.[6]

1) It is quite likely that someone wanting both string processing and arithmetic power could take the main SNOBOL statement (which is very powerful) and incorporate it in ALGOL. This would be syntactically recognizable provided each such statement was flagged by some symbol such as asterisk.

2) Alternatively one could revise the type **string** or ALGOL 60, add a concatenation operator, **tie** and introduce a statement of a form such **as**

search < string > **for** < string exprn > :=
<str exprn> **else** <Roman>

Comparing these two alternatives we note the following points in favor of the second which is, of course, in harmony with ALGOL 60.

i) Because **string** is added in the second case with the same status as **real** or **Boolean**, both string functions and strings as procedure parameters are possible.

ii) In the first case there will be confusion and contradiction of labelling conventions.

iii) Because the second implies **string** arrays, indirect addressing of strings is made simpler.

iv) Implementation of the first scheme is tantamount to including most of a SNOBOL compiler inside an ALGOL 60 compiler which is of course nasty for implementation.

The above example does emphasize that some methods of including a concept into ALGOL are in harmony and some are not. Furthermore, it illustrates that harmonious extensions make use of and work in with the various concepts of ALGOL that are well founded (the concepts of type, statement, label, variable, etc.).

It is with this principle in mind that the following basic methods of extending ALGOL-like languages are proposed.

Provision for additional data types

This was the technique that was used in the above example for incorporation of string manipulation and which made a very attractive extension. In fact it is probably the most powerful device possible since it opens large numbers of non-numeric fields to the scope of Algorithmic languages in a very simple manner. An example of the type **complex** was mentioned in the introduction and it may readily be seen that introduction of a type **complex** with the same status as **real, integer,** and **Boolean** is the only satisfactory way of introducing the concept of complex arithmetic.

A flood of other extensions to ALGOL 60 come to mind by means of this device of providing additional types. It is the author's opinion that the bulk of the shortcomings of ALGOL 60 that people feel at various times could be removed if they could define additional types. The following examples illustrate the wide application for this technique.

i) The type **double real** would enable double precision hardware, presently available on most current computers, to be utilized by a programmer,

ii) the type **half real** would sometimes be useful when space is at a premium rather than accuracy.

iii) the type **polynomial** would make polynomial manipulation programs (Hamblin,[7] Collins [8] etc.) quite redundant since the inclusion of the routines

into the framework of ALGOL allows the more comfortable programming speech of the more sophisticated language.

iv) The types **note** or **chord**, etc., could be a great boon to the researcher in music by making music text manipulation available in the ALGOL framework.

The implications of additional types

The extensibility of ALGOL-like languages in the direction indicated in the previous section is obviously very powerful but in order that its full value may be realized the following additional types of extension must accompany any provision of additional types.

i) *Additional significance for existing operators.* If the concept of subtraction, for example, is applicable to the extra data type then the significance of the operator "minus" must be defined for operation on quantities of the new type.

ii) *Additional Transfer Functions*

Consider the case where the extra type is **complex.** Just as it is necessary for an inbuilt rule to be available for conversion of any **integer** quantity to the equivalent **real** quantity, so it is necessary that the transformation from *real* to **complex** be defined. The combination of these two transfer functions should automatically, then, be available for conversion from **integer** to **complex,** There are actually two classes of transfer functions which I shall label "generalizing" and "degeneralizing." It is readily seen that *real* may be considered a more general type than *integer* since each integer can be transformed to a *real* quantity without loss of information but most **real** quantities have no **integer** counterpart. Hence it is appropriate that the transfer **integer** $\rightarrow$ **real** is called "generalizing" and the transfer **real** $\rightarrow$ **integer** is termed "degeneralizing."

It is obvious that "generalizing" transfer functions are essential for most new types. Wirth and Hoare [9] express clearly the major argument for using operators and functions instead of degeneralizing transfer functions.

". . . , and he (the programmer) is thereby encouraged to select the alternative which corresponds to his real requirements, rather than rely on a built-in "default" conversion about which he may have only vague or mistaken ideas."

The opposing point of view is that such functions and operators can clutter up a program. I don't propose to add anything to this debate and so will just proceed anticipating either. Therefore we should allow extendibility in the direction of both generalizing and degeneralizing transfer functions.

iii) *Additional Operators*

It is readily seen that if entirely new data types were introduced then new ways of combining quantities of those types could also be required. For instance if the new type were **complex** then monadic operators **ip** and **rp** suggest themselves for extracting **real** and **imaginary** parts of a complex quantity. Similarly, if the new data type were polynomial then a diadic operator **deriv** would be useful for taking the derivative of a polynomial with respect to some variable.

iv) *Additional Forms of Constant*

For every type in a language there must be some format, for a primary of an expression, which can be used to specify constants of that type. The types of ALGOL 60 include **real** and **Boolean** and the corresponding constant forms are numeric constants such as 5.2, 123.4 and logical constants **true** and **false.**

Provision for additional statements

This is another very powerful way of grafting onto the framework of any general purpose language additional special purpose features. Because harmonious extensions are ones which make the most of existing parts of the compiler, it is reasonable to specify that the components of the statement are either basic symbols or the various important syntactic units already found in the language. These are the components of existing statements and include < statement> , < variable> , <Boolean expression > , < array > , <label> . The delimeters would be chosen from the standard set of the language or manufactured with some mnemonic significance. The following example of a parallel processing statement illustrates that extra statement forms can be the only convenient solution for some tasks. The construction "**do** < statement > **also** < statement > . . . **also** < statement >"; is almost essential for making use of parallel processing hardware via ALGOL 60.

A language for extensions to ALGOL

The motivation for a compiler expansion

scheme and the directions in which extensibility is required have now been expounded. The extension scheme described in this paper consists of two programs. The main one is the skeleton of a compiler without information about types, operators, etc. The second part of the system is a program which will accept as input the information necessary to describe the extensions desired to the language and use this information to pad out the skeleton compiler to accept and compile any programs written in the new extended language. The information that the extender program requires can be considered to be a program in the language LACE (*L*anguage for *A*LGOL *C*ompiler *E*xtension). A LACE program will be a sequence of statements—each one being a specification of some new addition to ALGOL. There will, of course, be statements associated with each of the directions for extension that were described in PART II.

The following ten LACE statement forms are proposed to allow descriptions of extensions to be specified.

TYPE	T
CONSTANT FORM	CF
MONADIC OPERATOR SIGNIFICANCE	MOS
DIADIC OPERATOR SIGNIFICANCE	DOS
GENERALISING TRANSFER FUNCTION	GTF
DEGENERALISING TRANSFER FUNCTION	DTF
OPERATOR PRIORITY	OP
STATEMENT FORM	SF
BASIC SYMBOL	BS
CONTROL ROUTINE	CR

Each of these statements or functions require parameters and so a convenient form for each statement is its name (either full name or abbreviated) followed by the parameters required. These parameters will be separated by commas. A description of the parameters required follows.

i) *TYPE <name>, <length>*

To add a new type to ALGOL, a name, which is a new basic symbol, must be specified so it can be used in declarations, specification parts etc. The parameter 'length' is required to specify the amount of memory space that is required to contain the value of any quantity of that type. This will normally be a number of bits required but for certain types of data, such as **polynomial**, a variable length field is required for values and so instead of an integer another symbol such as 'V' (for variable length) would be used.

ii) *GTF and DTF <initial type>, <final type>, <code>*

The parameter 'CODE' is the body of code that is required to transform quantities of the initial type to quantities of the final type. The body of code must be surrounded by appropriate symbols. We will use "and".

iii) *MOS <operator>, <T1>, <TR>, <code> and*

DOS <operator>, <T1>, <T2>, <TR>, <code>

In the statements to define new operators T1 and T2 are the names of the types of the arguments of the operator. TR is the name of the type of the result of the operation and the parameter 'CODE' is the body of code necessary to perform the operation.

iv) *Operator priority <operator>, <priority value>*

The priority value for an entirely new operator would be obtained by referring to a list of priorities for use with the system. The compiler requires an ordering of priorities for all operators.

v) *Constant Form <type>, <syntax>, <semantics>*

Each constant form for a particular type in the language will have some syntax associated with it. The form of syntax strings in LACE is discussed later. The semantics required must be target code that will generate a constant of the appropriate value for use by the program.

vi) *Basic Symbol <symbol>, <internal value>*

This statement can be used when a particular value is desired for new basic symbols in the language. It is also useful where equivalent symbols are desired, as is the case with some versions of ALGOL—the underlined words can be in several languages.

vii) *Statement Form <syntax>, <semantics>*

Refer to the later section on syntax and semantics strings.

viii) *Control Routines <code>*

This statement is to allow the specification of a routine that will be available at run-time of an ALGOL program.

The strings that represent syntax

To represent the syntax of statements and constants we use an extension of Backus Normal Form in which the following nine symbols are reserved as meta-symbols.

<	>	for surrounding syntactic units.
|		for use as a choice operator.
{	}	for denoting a sequence of zero or more repetitions of the enclosed string.
[	]	for bracketing a series of choices together.
"	"	for delimiting the whole string.

These characters will not be available on all computing equipment but for any given system suitable replacements can be made. The extensions to BNF are minor and should be readily understood with the aid of an example. The syntax string for the conditional statement of ALGOL 60 follows

"**if** <Boolean expression> **then** <unconditional statement> [**else** <statement> | <nil>"

It has previously been mentioned that the syntax string is composed of the syntactic units of ALGOL and basic symbols, either new or old. This is, of course, an oversimplification. Firstly, there are many syntactic units in the ALGOL 60 Report [14] that are included merely as steps in the development of the syntax; terms such as 'Boolean secondary' or 'compound tail' are units we need not consider useful in this scheme. Secondly, the grammar should be unambiguous and all developments of the syntax string should be recognisable in a left to right scan; this means that such syntax strings as

abc [<variable> | <expression>];

are made illegal because it is both ambiguous for **abc** alpha; and involves backup for **abc** beta [1+m×n]+2;

Thirdly it should never be the case that two syntactic units can be run together, either directly or indirectly, to remove further difficulties, or even ambiguities, for the compiler. Hence it is not allowed to write a syntax string

pqr <variable> <integer> **xyz** or the string

pqr <variable> [**lmn**| <integer>] **xyz**

The strings that represent semantics

For the LACE statements for defining new forms of statements in ALGOL and new forms of constant, there was a parameter for denoting the syntax and a parameter for denoting the semantics. This pair of parameters forms a translation rule just as in the Syntax Directed Compiler or the Compiler-Compiler. Just as the nine symbols

" " [ ] | { } < >

were reserved for use as meta-symbols in syntax strings, so they are reserved here. Of course, there is no syntax units in the semantics string so the symbols and are used in constructing semantic units. The symbol * is also reserved for this purpose. It should be clear that the semantics string will have an identical structuring of the metasymbols " " [ ] | { } to the corresponding syntax string. Apart from the metasymbols the semantics string is made up of a sequence of semantic units. There are 3 types of semantic units as follows:—

i) A semantic unit may be any symbol of the target language. Special conventions will have to be made for representing symbols which are reserved as metasymbols.

ii) A semantic unit may be a pointer to one of the syntactic units in the syntax string and has the form <integer>, for example, <6> or <12>

The syntactic unit to which it refers is that one with ordinal number equal to the quoted integer. The target code that this initiates is the code normally associated with the syntactic unit. This code is well defined for such units as <statement>, <expression> but for others like <label> would be empty, and so a pointer to <label> would be useless.

iii) A semantic unit may also be a call of a system function designator which will have certain parameter requirements. A call on a function has the following form.

*function name (<parameter list>)

The parameters which form the list will normally be pointers, symbols or integers. There will be

functions for all the attributes that the various syntactic units have and so an example of a function is

*ADDRESS (<2>) which gives as a result the address of syntactic unit number 2 in the syntax string which may be <variable> or <label>.

The effectiveness of the system will, of course, depend to a great extent on a suitable set of functions. However, experience derived from an actual implementation would affect the choice so much that no more than a few examples are included in this paper.

A simple illustration of the correspondence between the syntax strings and the semantics strings is now presented in such a way as to demonstrate an instance of the usefulness of LACE. Many programmers realize that the time spent in duplicate address calculation in the following types of statement is wasted but there is no way out in current ALGOL.

```
xyz[i, j, k, l]  :=xyz[i, j, k, l]  + 1;
```

With a compiler extension scheme available the frustrated programmer could write a more efficient and more suggestively worded

```
step xyz [i, j, k, l]  by  1  ;
```

The following text would be a suitable LACE definition.

```
NSF ("step <variable> by <expression>",
    "*load address (XR1, <2>)
     *fetch indirect (XR1) <4> *apply diad(
     +, *type (<2>), *type(<4>))
     *store indirect (XR1) " )
     where XR1 may be an index register.
```

A second and more complex example is now given which is the definition of a statement to permute the contents of any number of variables.

```
NSF ("rotate  <variable>  <comma>
    { <variable> <comma> } <variable>",
    "*load address (XR1, <2>)
     *fetch indirect(XR1)
     {*load address (XR2, <4>)
     *fetch indirect(XR2)
     *store indirect (XR1)
          CPYREG  XR1,XR2}
     *load address (XR2, <6>)
     *fetch indirec(XR2)
     *store  indirect  (XR1)
     *store  indirect(XR2)")
```

In the above example the string CPYREG is an instance of the object code output required. In both examples some notional stack is assumed.

Implementation

This part is concerned with the design of an ALGOL compiler in such a way that it will be extendible; obviously special considerations are required in formulating its structure. ALGOL, in particular, is chosen because of the difficulty of talking in the abstract about translation techniques. The schemes are modifications of well known ones and consequently much familiar material is presented, but only in the interest of a clear description.

Only two areas of the system will be discussed; these are the ones that present most of the challenge. These areas are expression analysis/compilation and statement analysis/compilation. Other areas affected are the basic symbol input routine, simple variable declarations, array declarations and procedure specification parts.

It should be noted that the following are demonstrations that the problems that arise in implementation are soluble. It may be that the techniques proposed are not immediately applicable to certain languages which are not ALGOL-like or not immediately compatible with certain optimisation requirements for the compiler. The proposals should, in any case, suggest the method of revision for alternate techniques.

Expression analysis routine

The technique that seems most convenient is the operator priority technique for translation to Reverse Polish notation. Early references to the

method are DIJKSTRA,[10] HAMBLIN[11] and HIGMAN[12] (all appeared in 1962).

The technique is normally described in terms of an input string (in source code) and an output string (in target code). At the start of an expression a left parenthesis is put on the operator stack to protect the contents of a previous recursion and if the first symbol is "if" then appropriate action is taken; otherwise there is a normal simple expression. All primaries encountered during the left to right scan generate output code immediately (including sub-expressions) but operators may be stacked according to the following rules.

i) Operators which can be either monadic or diadic are treated as diadic if they follow a primary, else they are treated as monadic.

ii) Operators which are monadic are added to the operator stack, *provided* there is not an operator of higher priority on the stack. This condition would correspond to a failure.

iii) Operators which are diadic must occur immediately after a primary or else an error exists. Also a monadic operator or another primary must follow. Diadic operators first *displace* each operator on top of the stack which is of equal or higher priority. It is then added to the stack with the current value of the variable, BETA, which indicates the type of the left argument of the operator.

iv) An expression terminator *displaces* all operators until it finds a left parenthesis which it removes. It then causes exit from this level of the expression routine indicating that it found an expression of type given by the value of BETA.

v) A *monadic* operator is *displaced* as follows:—The type of quantity to which the operator is to be applied is given by the variable BETA. The appropriate operator significance for that operator and the type of quantity is then added to the output. The type that is given as the type of the result of the operation is placed in BETA and the operator removed from the operator stack.

vi) A *diadic* operator is *displaced* as follows:—The type of the 1st argument of the operation is stored with the operator and the value of BETA gives the type of the second argument. The operator significance appropriate to that operator and that pair of types is selected and added to the output. This may be one of the operator significances specified by the programmer or may be one which was specified, but compounded with one or more generalising transfer functions.

The expression analysis routine described above is one pass, left-to-right scan and does not produce optimum code. The ideas could, however, be extended to other expression analysis techniques.

It has been noted earlier that there are transfer functions which are defined for the extended language both explictly and implictly. If a new data type **abcd** is defined and a generalising transfer function from type **real** to type **abcd** is also defined, then a new generalising function is implicitly defined from type **integer** to type **abcd.**

It is clear that it is more efficient for the *extender* program, after having absorbed all the transfer functions which are explicitly defined, to generate, as well, all the other transfer functions that are implicitly defined. New generalising transfer functions are generated as minimum combinations of explicit generalising transfer functions. Implicit degeneralising transfer functions are generated as minimum combinations of explicit degeneralising functions or as the combination of a generalising transfer function (which need not be explictly defined) and a string of degeneralising transfer functions.

It is also profitable for the extender program to generate, and form as a list in the compiler, all operator significances which are implicit. An implicit operator significance is formed when a generalising transfer function is applied to convert the quantity to an appropriate type for some operator significance, which is given for the operator, to act on.

Statement analysis

The extender program has the task of constructing, from the syntax and semantics strings which are supplied, that part of the compiler which will analyse the statements of the extended-ALGOL programs. We shall talk about this process assuming that the compiler is constructed along the lines commonly known as syntax routine method of compiling.[13] This means that wherever a syntactic unit appears in a syntax string this transforms into compiler code as a call on that syntax routine. The syntactic routine will scan the code till it exhausts that syntactic unit.

A basic symbol which appears in a syntax string will be transformed to a test or a check in the compiler code generated. Because the choice operator, | , separates alternatives, a series of tests

must be made with one branch made to a section of code for each alternative. At the end of each of the alternatives, there must be jumps to a common point.

The transformation of a pair of braces { } which denotes optional repetition, is a loop in the compiler code. The contents of the braces, when transformed, form the body of the loop. The decision to jump out of the loop is made at the start with a test to see whether the next symbol to be fetched from the input is appropriate to continuing or discontinuing the loop.

The above is a description of how the syntax string imparts structure, tests and syntax routine calls to the code generated for the compiler. The semantics string, of course, has the same structure and so where the tests and routine calls are made for some particular element of the syntax string, the corresponding element of the semantics string is designated for output.

CONCLUSIONS

This paper has been an introduction to a solution of the problem of general purpose languages being inflexible; a two segment system which allows the user to introduce new data types and new data manipulations. The scheme is intended for high-level general purpose languages to make them more general but was illustrated with ALGOL. A generalisation to other appropriate languages should not be difficult.

There are several questions that should be asked of this compiler extension scheme. The first of these is whether the direction of extra data types and extra operations on these types is the right way to provide for extensions. The second question is whether it is ambitious enough (i.e., powerful enough), and the third is whether it is feasible (i.e., not too ambitious). The author's opinion on these questions is that they should all be affirmed. However, the areas are subjective since there has been no implementation of such a system *yet*, but the contents of the paper explain and give weight to the point of view.

A fourth and less subjective question asks how it compares with rival schemes. We compare it to four other systems which can be used to provide compilation of extended languages.

i) *Syntax Directed Compiler (SDC).*

The use of a SDC for compiling an ALGOL-like language requires a standard primary 'program' which defines the whole syntax and semantics of the language, rigorously and completely. To make use of SDC for user extensions, the primary program would require suitable alteration. This would demand knowledge of the target code (as does the scheme here presented) but also demands quite intimate knowledge of the primary program—both its language and structure. This is probably significantly easier than amending a hand coded compiler but suffers from the same disadvantages.

ii) *Compiler-Computer (CC)*

This is also a semi-automatic method of extending a compiler and suffers the same disadvantages as SDC.

iii) *Cheatham's approach* [8]

This is an attempt to provide macro facilities for high level languages, and, in the terms of this paper, only in the area of new statement forms. That is, no direct attack is made on the area of provision of extra data types, which is crucial. It is readily admitted that macro facilities are very useful and a powerful macro preprocessor such as LIMP (WAITE [15]) can help to remove some of the problems of tedious programming speech. However, it is also true that if it is impossible to write code for some given task (in some language) that will be compiled both efficiently and compactly, then macro processing is not going to help. It is contended, therefore, that the present paper has a better answer as regards efficiency.

iv) *ALGOL C—Galler, Perlis* [4]

As with the above scheme (Cheatham), considerable importance is given to provision of macro facilities in ALGOL-like languages. For the same reasons it is potentially less efficient than the approach of extended compiler generation. Recall it was mentioned in the introductory section, that the case of complex numbers is a classic example of tedious and inefficient ALGOL; yet Galler and Perlis still have complex numbers implemented as 2 element arrays. On the other hand their approach means that extensions are not machine dependent.

A second disadvantage is that it stops short of providing new statement forms. A last and quite significant disadvantage is that all the definitional facilities are in the compiler and therefore the same work is done over and over again, wasting time and compiler space.

REFERENCES

1 J FELDMAN D GRIES

Translator writing systems
Comm A C M 11 Feb 1968 pp 77–113
2 E T IRONS
The structure and use of the syntax directed compiler
Annual Review in Automatic Programming Vol 3 1963 pp 207–227
3 T CHEATHAM
The introduction of definitional facilities into higher level programming languages
Proc AFIPS 1966 Vol 29 pp 623–637
4 B GALLER A J PERLIS
A proposal for definitions in ALGOL
Comm A C M 10 April 1967 pp 204–219
5 R A BROOKER et al
The compiler-compiler
Annual review in Automatic Programming Vol 3 1963 p 229
6 D J FARBER et al
SNOBOL A string manipulation language
J A C M 11 Jan 1964 p 21
7 C L HAMBLIN
An algorithm for polynomial operations
Computer Journal 10 Aug 1967
8 G E COLLINS
PM A system for polynomial manipulation
Comm A C M 9 Aug 1966 p 578
9 N WIRTH C A R HOARE
A contribution to the development of ALGOL
Comm A C M 9 June 1966 pp 413–432
10 E W DIJKSTRA
Making a translator for ALGOL 60
Annual Review in Automatic Programming Vol 3 1963
11 C L HAMBLIN
Translation to and from Polish notation
Computer Journal 5 Nov 1962 p 210
12 B HIGMAN
Towards an ALGOL translator
Annual Review in Automatic Programming Vol 3 1963
13 B RANDELL D J RUSSELL
ALGOL 60 implementation
Academic Press Inc LONDON 1964
14 P NAUR
Revised report on the algorithmic language ALGOL 60
Comm A C M 6 Jan 1963 pp 1–17
15 W M WAITE
A language independent macro processor
Comm ACM 10 July 1967 p 443

Program composition and editing with an on-line display

by HARVEY BRATMAN, HIRAM G. MARTIN, and ELLEN CLARK PERSTEIN

System Development Corporation
Santa Monica, California

INTRODUCTION

An Interactive Programming Support System (IPSS) has been under development at System Development Corporation since 1965.* The purpose of the system is to permit all of the programming processes—composition (in a procedure-oriented language), editing, execution, testing, and documentation—to be carried out as parts of a single, coordinated activity centered around an interactive compiler. IPSS attempts to unify techniques that are usually embodied in separate functional programs, so that the programmer need not know which particular program is performing a specific task. The system is intended for a time-sharing environment, with user interaction via a small tabular display or typewriter-like terminal.

In the development of IPSS, emphasis has been placed on the functions of program composition and editing, particularly via an interactive display. The current object of this work has been to investigate the use of a small tabular display by a professional programmer to compose and edit programs and to retrieve dynamic information about the state of his program. This paper reports the results of three aspects of that work.

- It describes how the display is used, how the display compares with a typewriter-like terminal, and how the user interacts with the display to accomplish the tasks of program composition and editing.
- It describes the advantages of doing syntactic analysis concurrently with program composition and editing, how program information is obtained, and how IPSS does the analysis and editing.
- It describes the software interface between IPSS and the display hardware, how IPSS can be used with various displays, and what the characteristics of an "ideal" display are.

*This work has been supported in part by the Advanced Research Projects Agency of the Department of Defense.

Background

The Interactive Programming Support System operates under SDC's new time-sharing system (known as ADEPT–50) on an IBM 360 Model 50 computer. Since one of the major purposes of ADEPT–50 is to support large-scale program development, testing, and debugging, IPSS is considered to be an essential element of the ADEPT-50 system.

A small, tabular display (rather than a typewriter-like terminal) was chosen as the primary interactive device in this work for the following reasons. First, we are looking forward to the near future when relatively inexpensive display devices are expected to be available. These devices could be installed directly in the programmer's office or work area, and could be connected to a central time-sharing system. Second, a display provides faster response by presenting many lines of output in the same time a hardcopy device requires to print one line. Thus we hope to eliminate the need for the programmer to search through numerous pages of program listings, hoping to find some single piece of information, but not knowing if the listing really corresponds to the current state of his program. Instead, he will be able to display all the information he wishes to see, and will be assured that it reflects the true state of his program at that instant. Third, the display scope minimizes the amount of typing the programmer must do to compose and edit his program. The programmer is able to use the "replace-character" feature of a display scope to modify parts of his program, rather than having to retype whole lines. Previous systems (such as the Q.E.D. Time–Sharing Editor developed at the University of

California, Berkeley,[1] or the editing routines used under the M.I.T. Compatible Time-Sharing System[2]) show the ingenuity and complexity required to perform similar functions with a typewriter terminal.

The display device used in this project is an IBM 2250. This device is more powerful and expensive than necessary for this work, but we can use it to simulate the characteristics of displays such as the Sanders 720, Computer Communications Inc. 300 TV display, or RCA video data terminal—which are typical of the devices we are considering.

The current system is limited to the use of the JOVIAL programming language (since this is the most readily available language at SDC). The operational principles derived in this work, however, are applicable to other languages such as FORTRAN or PL/1. The user can choose either the 2250 display or an IBM 2741, a typewriter-like device, as his interactive medium. However, some IPSS functions such as automatic program display and minimization of programmer typing can be done only on a display scope; other functions may be too time consuming to be used often on the IBM 2741. The current capabilities of IPSS allow the user to compose and edit his program interactively. All program statements are checked for syntactic correctness as they are input, and correct statements are automatically displayed to show the current status of program composition. The programmer is informed of errors in syntax, which he can correct immediately. Syntactic checking combined with editing insures that the edited program is free from the more common programming errors. Thus, the programmer need not be afraid of unpleasant surprises when his program is compiled. The programmer can display parts of his program, he can obtain current "where set" and "where used" information about variables in his program, and he can obtain all the references to any program label.

Use of the display

The display is used to convey information both from IPSS to the user, and to IPSS from the user. IPSS either requests input and waits for the user to respond, or it outputs information in response to user actions and again waits for the next user response. User responses may be answers to specific questions from IPSS, program statements, or requests for IPSS action.

The face of the display has been organized into three logical "blocks" to facilitate these uses (see Figure 1). The number pair shown in Figure 1 represents block and field numbers; e.g., 3,0 represents block 3, field 0. Each line is composed of one field except for block 1, where the line is divided into two fields. A "block" is roughly defined as a physical segment of the display surface, set aside for some specific function. Block 1 is the area where IPSS requests user actions. IPSS signals the user when it is ready for input by displaying two question marks (??) in position 1,0 (block 1, field 0) and by placing a cursor in position 2,0. A descriptive message is written in position 1,1. Block 2 is the work area where the user responds to IPSS input requests. Since the cursor has been placed in position 2,0, typing on the keyboard causes input to appear there. The user is restricted by IPSS from typing into block 3. Block 3 is the area where IPSS displays information requested by the user or automatically generated in response to some user action. Every line but the first contains one field of 72 characters. The first line is divided into two fields of 2 and 72 characters each. Block 1 consists of 1 line, block 2 consists of 5 lines, and block 3 consists of 47 lines. The size of the blocks can be varied for experimental purposes.

The IBM 2250 has been available to us only since February 1968. (Prior to that time, we used the IBM 2741 to test many IPSS functions.) Hence our ideas on how to use the display are still evolving as the result of experience. We have decided on some general principles to guide us in the use of block 2, the user work area, and block 3, the program's output area. To meet the stated goals of IPSS, the minimum sizes for blocks 2 and 3 are 2 and 20 lines, respectively. Hence any practical display scope should hold about 25 lines of at least 50 characters each.

We wish to minimize the amount of typing the user must do. As shown below, IPSS will, in the case of error correction and editing, regenerate program statements on block 2 so that the user can modify them as necessary, and does not have to retype entire statements. The user can add new displays to block 3 without destroying the display already there. He can change a current dis-

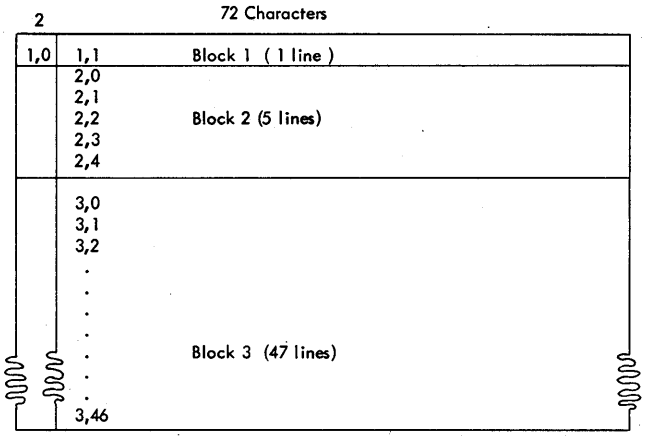


FIGURE 1—Display organization

play to make room for a new display, and thus make effective use of the display as a substitute for leafing through program listings.

Display interaction in program composition

An example of program composition is shown to describe the use of the display and also to point out some advantages of a display over a typewriter-like terminal. The example corresponds to the program shown in Figure 2. Figure 3 shows how the display is used in program composition. Block 3 on the display surface contains an automatic display of the program thus far composed. Statement numbers corresponding to the last statement on each line have been added. The program label is formatted on the same line with a statement so that its statement number does not appear. For editing purposes, INITIAL. is statement 2 and XX = 10000$ is statement 3. Block 1 indicates that IPSS is ready for more input. The user has typed more of his program in block 2. There are two incorrect statements in this example. They will be discussed later. When the user is satisfied with the contents of block 2, he sends it to IPSS by pressing a "SEND" key.

Figure 4 shows the results of IPSS action on the in-

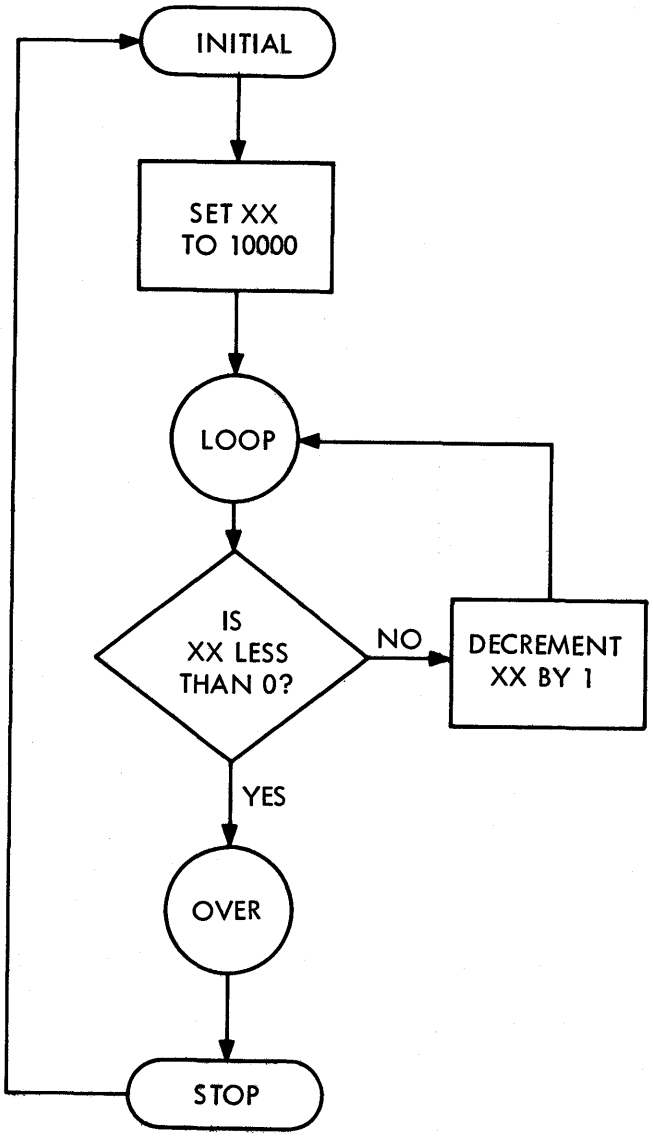


FIGURE 2—Example program

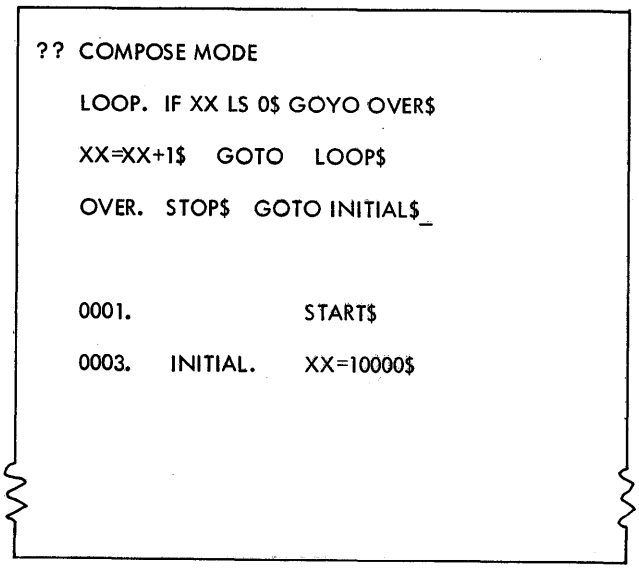


FIGURE 3—The display scope used for program composition

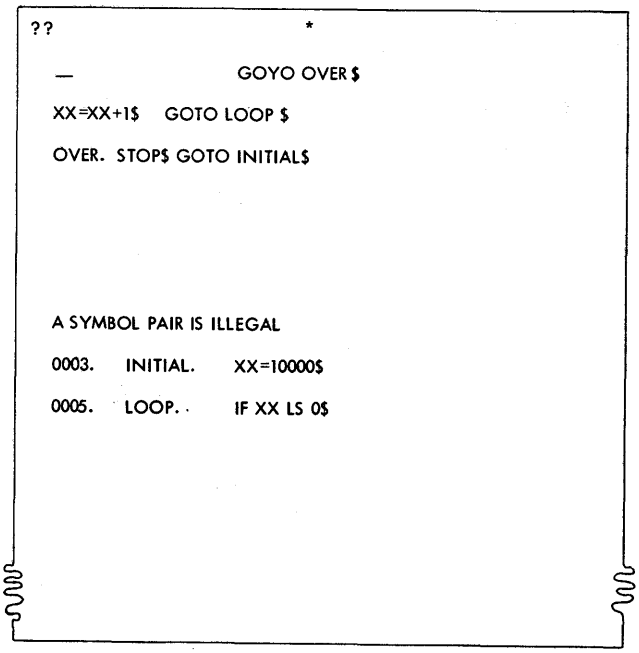


FIGURE 4—The display scope after an error has been detected

put. Correct statements 4 and 5 have been deleted from block 2; they have been reformatted and added to block 3. The incorrect statement

GOYO OVER$

is flagged by an asterisk (*) in block 1 and the type of error is explained at the top of block 3. The cursor has been placed in block 2, ready for user input. The user can correct the statement to read

GOTO OVER$

by positioning the cursor under the Y, typing T and sending the entire block 2 again to IPSS. He does not have to retype the rest of his previous input because IPSS has restored it to block 2. If he had been using a typewriter-like terminal, he would have had to retype the entire incorrect statement and everything after it. (The statement XX = XX + 1$ is logically incorrect; it will be used later to illustrate editing and debugging.)

Display interaction in program editing

The user can change his program in several ways: he can delete lines, replace lines, insert new lines, copy lines, and move lines. He can also change "signs" within statements in a particular area of his program. These tasks are all done by typing the appropriate edit commands in block 2. The user can save himself extra typing if the change consists of modifying statements. The command*

#COPY M, N $

brings lines M through N to block 2. The user can then treat the lines as if he had typed them there. For example, suppose that after compilation, execution, and debugging, the programmer finds that statement 7 is logically wrong and must be replaced. Figure 5 shows the response of IPSS after the following sequence. The programmer types and sends the command*

#COPY 7 $

The response by IPSS is to bring statement 7

XX = XX + 1 $

to block 2. The user then types the command*

*All IPSS commands have both a long and short form. The short form consists of the first two letters of the long form, e.g., #CO and #RE are the short forms for #COPY and #REPLACE.

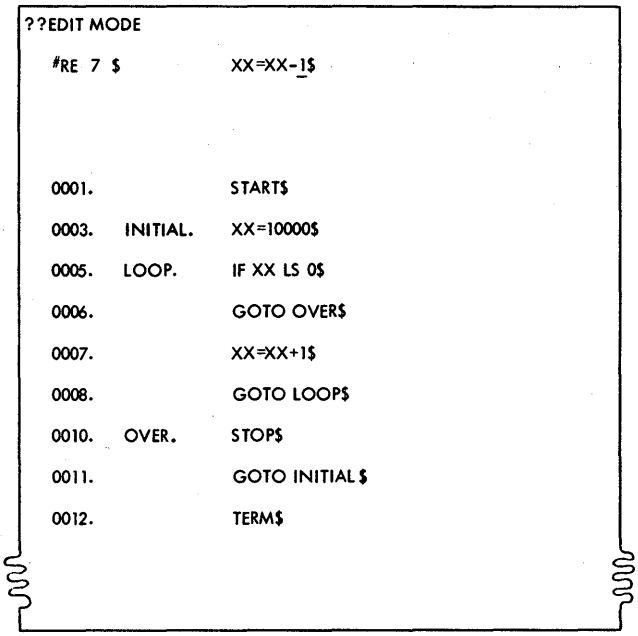


FIGURE 5—The display scope showing statement replacement

#REPLACE 7 $

He positions the cursor under the "+" and changes it to a "–." When he sends block 2 to IPSS, statement 7 will be replaced. Block 3 will be modified to show the replacement.

The length of the program is not limited to the length of block 3. When the program exceeds the length of block 3, lines at the top are removed and the new lines appear at the bottom. The line identifiers M,N in the command

#COPY M,N

can refer to any lines of the program, visible or not. The seemingly more straightforward technique of moving the cursor to the appropriate position in block 3 has not been included in IPSS. The technique of determining which lines have been changed in block 3 is difficult in the type of display equipment we are considering. We have avoided the use of expensive, complex hardware such as light pens and function keys. We are assuming just the ability to move a cursor and send an interrupt to the main computer. (The TVEDIT program, which runs under the THOR Display Based Time-Sharing System at Stanford University,[3] provides an example of what can be done with more complex equipment).

Other display interaction

Consider the previous example of program editing.

Suppose that in the course of debugging, the programmer knows that the variable XX is being set incorrectly, but he does not yet know where. Assume that the program in question is a large one and that the programmer does not remember every place the identifier XX is used.

The following sequence of commands shows how the programmer can make use of saved program information, and can use displays to find and correct errors in program logic (see Figure 6).

The user first types

#DISPLAY SET USED XX

IPSS clears the display and outputs at the top of block 3:

XX S3 U5 B7

This indicates that the identifier XX is set in line 3, used in line 5, and both set and used in line 7. The user then types**

#ADD DISPLAY 3

which adds statement 3 to the current display, as follows:

0003. INITIAL. XX = 10000$

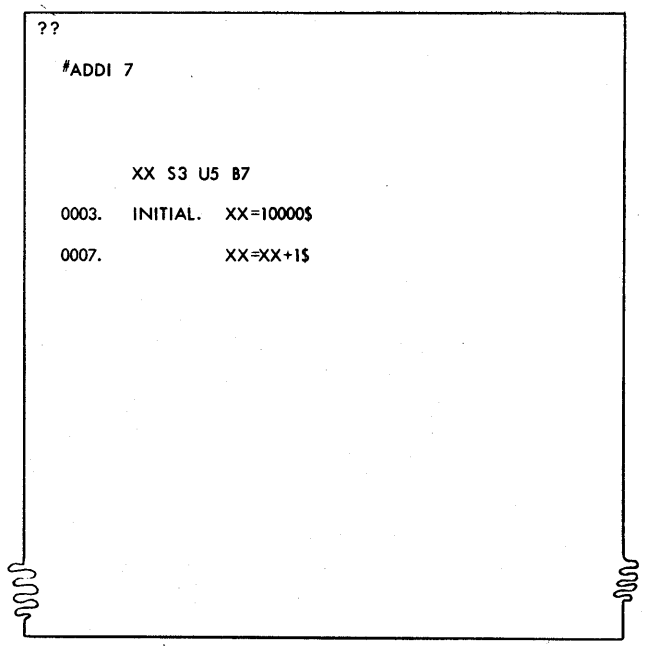


FIGURE 6—Program debugging

He then types**

#ADD D ISPLAY 7

which adds statement 7

0007. XX = XX + 1$

Upon reflection, he determines that statement 7 is wrong and changes it as previously described.

Other IPSS commands

This paper will not describe in detail all the commands available to the user. In brief, the user can display all or part of his program using either statement numbers or symbolic names for the areas. He can request that the current program display be rolled up or down like a scroll. He can display the names and corresponding identifying statement numbers of every procedure and statement label. He can display the statement number of every reference to any identifier. In addition, any display can be directed toward a hard-copy printer. The editing commands are described in the following section.

When the programmer is satisfied with his program, he can compile, execute and debug it using facilities available in the ADEPT–50 Time-Sharing System. Eventually, we plan to include compilation of modified program segments and execution-time debugging in IPSS.

Techniques of program editing used by IPSS

The previous sections of this paper have described the user interaction with IPSS via a display scope. The display scope was chosen for the convenience of the user and because it offers facilities that—on other devices—are cumbersome or impossible. The techniques IPSS uses to carry out the user composition and editing requests are, in general, independent of what interactive device is employed. Their only requirements are that they are efficient in their use of time and do not impose undue restrictions on user actions.

Editing is permitted in any part of a program and in any order that the user desires. The program is blocked, and blocks are output to disc as required. IPSS keeps a record of what statement numbers are included in a disc record, and permits insertions or deletions of any length anywhere in the program. The disc records need not be in order according to statement numbers. This makes it easy to insert statements anywhere in the program,

**The short form of the command (#ADDI) is shown in Figure 6

since new disc records can be added at the end. An attempt is made to combine two records into one after statements are deleted, when this is possible without causing extra input/output. A disc record freed in this way becomes available for reuse whenever needed. Output of records is minimized by not outputting until a user request forces it (unknowingly, of course, since he has no concern with the blocking and input/output of his program.) Input of records is minimized by remembering what disc records are currently in core.

Editing is done on program segments where a program segment is a single statment or a consecutive series of statements identified by its starting statement number or starting statement label, and by its ending statement number or ending statement label. (The ending statement need not be given if only one statement is involved.) Such program segments may be deleted from the program, moved to (inserted after) another program statement, copied after another program statement (this is similar to moving except that the statements are also retained in their original place), or replaced by a new series of statements that the user inserts.

The statement numbers assigned by IPSS during program composition permit up to 1000 insertion statement numbers between statements. During editing, IPSS computes insertion numbers based upon the number of statements to be inserted at a time, and the numbers of the bracketing statements. The user may override the automatic assignment of statement numbers by prefixing the first of a series of statements with the starting number he desires. Automatic assignment of following statement numbers depends upon the number he uses and the next statement number in the program.

The DELETE, MOVE, and COPY commands are complete in themselves; IPSS assumes that the user wishes to continue program composition when they have been executed. The INSERT and REPLACE commands assume that any number of statements may follow in one or more inputs; therefore, IPSS continues to assign statement numbers depending upon the numbers available where the statements are inserted. The user may indicate that he wishes to continue program composition by inputting COMPOSE. Block 1 of the display scope shows the user whether he is in the edit mode or the compose mode.

A COPY command that does not indicate where the statements are to go causes the statements to appear on block 2 of the display scope. The user may then modify them as desired and give a command to insert them anywhere in the program.

A SEQUENCE command causes the entire program to be resequenced. An option permits specifying the starting statement number and statement increment to be used in the resequencing.

All editing commands previously discussed apply to entire program statements. There is also a CHANGE command that permits any specified string of signs (except a null set) within a statement to be replaced with any other specified string of signs (including a null set). The range of statement numbers to be searched for the substitutions may be limited if desired. At the end of a statement in which one or more substitutions of signs have been made, the old statement is deleted and the new version of the statement is checked for syntactic correctness; it then replaces the old statement. This process continues until the end of the specified statement range is reached or until an error in syntax is detected.

IPSS tries to be permissive to the user and often provides optional ways of accomplishing the same result. For example, a REPLACE N command followed by a series of statements accomplishes the same result as a DELETE N command and an INSERT AT N command followed by the series of statements. User convenience is considered more important than optimum efficiency.

Program information storage and retrieval

At all stages of program composition and editing, a list (ordered by statement number) is maintained for each identifier in the program. Each list shows every current reference to the identifier and denotes whether the identifier is declared, set, or used in the statement. Deletions of statements cause deletions of references in the appropriate identifier lists; insertions of statements cause insertions of references in the appropriate identifier lists. Thus, at any time, the user may ask where a variable is set or used and the information is immediately available for display. A link deleted from the list for one identifier is available for reuse as needed for any identifier. In this way the size of the table needed for the set/used information is not prohibitive.

A dictionary is maintained with an entry for each identifier in the program. Information that helps in syntactic analysis and links to other tables (such as the set/used table and the table of procedures and tables) is kept in the dictionary entry. If a declaration of an identifier is deleted, it is assumed that this identifier will be redeclared and it will then be assigned to the same dictionary entry, so that the proper linkages are maintained. For this reason a dictionary entry is not reusable for another identifier even if the declaration and all references to the original identifier are deleted.

Other program information available to the user by command consists of a list of procedure names and their scope denoted by statement numbers, and a list of statement labels and corresponding statement numbers

within the entire program or within any procedure.

Syntax analysis, error detection and correction

Most of the error detection done by IPSS is on an individual statement basis. All such errors within statements are detected regardless of whether the user is composing or editing his program. Some additional checks involving sequences of statements are made during program composition. During composition, a memory of preceding statements is maintained so that errors such as an incorrect table structure or a conditional clause immediately following another conditional clause are detected, whereas these same errors may go undetected when the user is editing his program. In the compose mode, an error message may tell the user what input is required; in this event only that particular input will be accepted next as a part of his program.

It is possible (though, in some cases, difficult) to detect these same errors in the edit mode but it is not considered desirable to do so, since this would force the user to construct his program in a particular order. It is not required that a program be correct at all stages of program editing. The *final* program is the one that counts; consequently, some kinds of errors involving sequences of statements are left for the compiler to catch. Most of the errors are detected by IPSS and the user is given a chance to correct these before compilation is attempted. When consecutive statements are input in the edit mode, some memory can be built up; the error checking of statement sequences approaches that of the compose mode.

In order that program names occurring in edited statements may be correctly identified as to class and scope, IPSS keeps track of the beginning and ending statement numbers associated with each table and each procedure in the program. This identification is necessary for error detection within individual statements. It certainly would be possible for the user to confuse IPSS, for example, by deleting the END of one procedure and the heading of the next in order to combine the two procedures into one. This may accomplish what he wishes, but some error checking is lost. He can accomplish the same result without the confusion by moving the desired statements from one procedure to another and then deleting the unneeded statements. More complete error detection is obtained through this second method.

When the end of the program is signaled by TERM$ or when the user requests that his program be compiled or saved, each procedure and the main program is checked to see that the BEGIN/END count is correct; if not, the user is told, for each such program area, that there are n too many BEGIN's or n too many END's. In order to determine more exactly just where the problem occurs, he can request that a BEGIN/END count be made for any program segment. When the error is narrowed down to a reasonably small program segment, he can request that segment to be displayed for visual checking. Unfortunately, because of the editing freedom permitted, a correct BEGIN/END count guarantees only that the number of BEGIN's and END's is correct and not that their order is logical. Here again, the final judge is the compiler.

The syntax analyzer was implemented using syntax-oriented techniques. These techniques have been used to construct compilers and have been well documented.[4] The syntax analyzer contains subroutines that parse the statement symbol string and then analyze it using an encoded version of JOVIAL syntax equations. The same technique is used to analyze the IPSS commands. It would be relatively straightforward to change the syntax analyzer to handle another language that could be described in syntax equations. This technique also simplifies the task of changing the IPSS command structure.

Display hardware interface consideration

As stated before, one of the primary objectives of the IPSS project was to include the use of a display scope as an interactive device between a time-shared computer operating system and a user. As usual, there were many display devices available, and their characteristics (particularly their computer interfaces) were quite varied. These characteristics, which are discussed later, exerted considerable influence on the proposed use of a display in IPSS. In an attempt to reduce any delays in the project, an intermediate language was developed that isolated device-dependent computer code from the proposed IPSS design. The language made it possible to plan and implement information manipulations on an available display before the operational display device was chosen.

Development of an intermediate display language that would be easy to use but not restricted to any particular display device proceeded naturally. After looking at descriptions of several displays that were being considered, one common characteristic (which was so dominant that it was almost overlooked) was noted. This characteristic was also an essential part of the human-to-computer interface. The characteristic was simply the ability to read and write in a language already known to the human. The only adequate and completely versatile means of communicating between a human and a computing machine is with words and numbers. If a display can accept and show a written language that is intelligible to the user, there is no limit to the type or amount of information that can be ex-

changed. Each of the displays being consider for use had this capability. With this one characteristic in mind, a language was developed assuming only that a display device would be chosen that would have the ability to display alphanumeric information in a gridded form (a number of lines of characters); that the device would also accept external inputs from a typewriter keyboard; and that it would send and receive the displayed data to and from a controlling computer.

An intermediate display user's language (IDUL)

The Intermediate Display User's Language (IDUL) is a tool for defining the format and content of a display device as specified by the program designer. This is done by dividing the view screen of the display into fields of information with one or more characters of displayed information per field. Fields are located by line and starting column; their lengths are established by field sizes; associated fields are grouped together with block numbers, and information to be displayed is defined by its location or address. This language is not a communication device between the outside user and the machine. Such communications are a function of each particular application—like IPSS—and should be left to the computer program designer's discretion. IDUL is concerned with internal communication. It allows the program designer to establish a display format and to design information exchanges without the burden of considering display specifics.

A language is a communication tool that is used to exchange information. In the case of IDUL, the language does not have the complex form of a natural language. Instead, it is composed of tabular data that have accepted meanings. Figure 7 shows a typical initial display view that can be used to explain the language forms. Notice that this display has 20 lines of 40 columns or positions which can be described as 22 fields, shown in Table I.

It is convenient to group the 22 fields into three

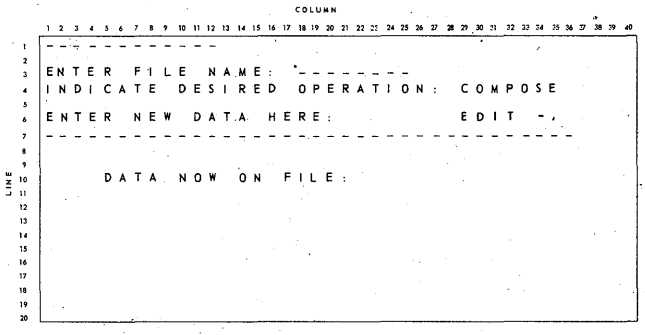


FIGURE 7—Typical initial display

separate blocks: (1) communications or control block. (2) input block, and (3) heading and data display block,

Remember, the display specification language is a programming convenience. The program designer uses it to effect selected information displays in predetermined formats. Therefore, device-specific routines to generate a display or to read indicated display fields as defined by internal tabulations must be available. The programmer sets up the tables and calls the proper routine to perform either the generation or reading of a display. This fact places an importance on the order and structure of the display definition tables.

Table II is a Format Definition Table that shows the fields that were defined in Table I in the IDUL tabular form for format definition. The fields are entered in the table in block, line, column order. The display input fields have zero entries under the "Location of Data" tabular column to indicate that the fields should initially contain blanks. The entries for heading fields in the same column ("Location of Data") show the actual information to be displayed, preceded by the code, LOC, which is an abbreviation for "location." In the internal format definition table, the variable data locations for the heading data, entries 7 through 11, would be inserted by the controlling computer program. The displayed contents of the last ten lines will change as the data file is enlarged; the data descriptions for these fields must be continually updated by the using program. The data fields most frequently read are put in block 1 of the table to make it more accessible. The next most frequently read fields are put in block 2, and all data to be displayed but not altered by the viewer are put in block 3. The block assignments are completely arbitrary with the exception of block 3 fields. Any field with a block number of 3 is protected from external change if this is possible on the display equipment. Block 3 was used as the protection indicator to allow the user more freedom in his block assignments while maintaining an ordered sequence to the field definitions. It is unlikely that any program designer will need more than two blocks to identify input fields, and there is no need to have more than one protection flag. Besides, if the designer wants more field blocking, he can always put the protected field definitions first and start his data blocks with block 4.

The last entry in the format definition table (Table II) is a cursor location. Column 18 of line 3 in Figure 7 is underlined. The underline mark is commonly used by display manufacturers to indicate for the display user's benefit where the next input character will go when he hits a keyboard character key. This mark (universally known as a cursor) is not always an underscore mark. On some displays it is a vertical line to the left of the character position. However, its function is always the

same. It points to the next input position. The cursor is a valuable aid, but it is not essential; information exchange could take place without it. (The operator might have a little trouble getting his data in the right place, but he *could* do it.) IDUL does provide for cursor insertion, as the last entry indicates. It can be recognized as a cursor definition by its position in the table (last) and its zero field size.

The Intermediate Display User's Language (IDUL) is simply a table containing a field-by-field definition of a display view, where each field description contains a block number, line number, starting column, field size, and data location. Flexibility is obtained by establishing some rules for the use of block numbers and by allowing modifications of the master or original format definition with an overlay specification table. This table

FIELD	LINE	COLUMN	SIZE	DESCRIPTION OF FIELD
1	1	1	12	Communications or control data
2	3	1	16	Heading
3	3	18	8	File name input field
4	4	1	38	Heading
5	4	37	1	Code input
6	5	29	7	Heading
7	5	34	1	Code input
8	6	1	20	Heading
9	7	1	36	Data input
10	8	1	36	Data input
11	9	1	36	Data input
12	10	5	17	Heading
13	11	1	40	File data display
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
22	20	1	40	File data display

TABLE I—A field by field description of typical display in Figure 7

(which is identical in content to the format definition table) is used to temporarily change one or more entries in a format definition table and to identify fields to be read from the display. Fields can be deleted by simply zeroing the size, or—if this is confusing—by replacing the block, line, and column numbers with zero, which is an undefined use. Also, block expansion can be provided by leaving an appropriate number of undefined field entries in the tables between blocks.

Device-specific routines to interpret the tabular language can be built and inserted at any time. This allows display device changing with minimal delays, and it makes it possible to support more than one display type through internal selection of conversion routines.

Display characteristics

Required insight into the problems of implementing routines to transcribe such a display definition language was gained through an examination of some display

ENTRY	BLOCK	LINE	COLUMN	SIZE	LOCATION OF DATA
0	1	1	1	12	0
1	2	3	18	8	0
2	2	4	37	1	0
3	2	5	34	1	0
4	2	7	1	36	0
5	2	8	1	36	0
6	2	9	1	36	0
7	3	3	1	16	LOC(ENTER FILE NAME:)
8	3	4	1	38	LOC(INDICATE DESIRED OPERATION: COMPOSE ,)
9	3	5	29	7	LOC(EDIT .)
10	3	6	1	20	LOC(ENTER NEW DATA HERE:)
11	3	10	5	17	LOC(DATA NOW ON FILE:)
12	3	11	1	40	0
.	↓	.	↓	↓	↓
.		.			
.		.			
21	3	20	1	40	0
22	2	3	18	0	0 (Cursor Location)

TABLE II—Format definition table for typical display

characteristics. The three displays discussed here are hypothetical, rather than real. This was done to incorporate as many characteristics of real displays into as few examples as possible.

The hypothetical displays have several common characteristics. Each one has a buffer storage where the display image is retained for automatic periodic regeneration of the picture; each one has a cursor that points to the next input position; each one has an input/output path to a computer; and (as pointed out before) each one displays character information in rows. The actual data transfer technique is not important in this discussion. Data transfer is a function of the sending and receiving equipment, and is—therefore—different in every combination. Yet, the capability must be there. Likewise, the buffer storage medium is not as important as the fact that the buffer is there. Some input/output restrictions are caused by the buffer hardware; these are important. (These restrictions are indicated in the individual display discussions.)

The three typical displays do not cover all of the possible display characteristics. They do exhibit a representative set of visual display features.

Table III lists six characteristics of three hypothetical display devices and an ideal display. The first device has a maximum of 40 lines of up to 50 characters per line. Its data storage buffer has space for 1000 characters of data that includes the format control characters. Thus, the amount of information that can be displayed is limited by size of the storage buffer, not by the row and column limits. Also, note that the buffer storage units are not addressable. The entire buffer must be reloaded to change the picture, or all preceding data must be read to read a field of data. Since the device has no computer interrupt signal, the computer program must poll the display to determine when data are ready for transmission from the display storage to the computer storage. This means that the supporting program must periodically send a message to the display asking for control data. As indicated above, format controls are required to position the information on the screen in the required positions. These characters (such as carriage return, horizontal and vertical spaces, insert cursor, enter format mode, return to start, etc.) occupy space in the display buffer storage, but at the same time they eliminate the need for storing blanks to fill fields and lines, and they protect data from keyboard change. The cursor controls are limited in the sense that the cursor position cannot be determined by the computer. It can be positioned, but it cannot be located if it has been moved.

The second display device is more versatile. It can

TABLE III—Characteristics of hypothetical display devices

	MAX. NO. LINES	MAX. NO. COLMS.	BUFFER SIZE AND ACCESS METHOD	I/O CONTROLS	FORMAT CONTROLS	CURSOR CONTROLS
1	40	50	1000 characters of non-addressable storage	Computer-initiated full buffer with polling required	All displayed fields are relative to a starting position controlled by stored formatting characters; protection provided	Cursor can be inserted but cannot be found by computer
2	50	70	4000 or 8000 characters of addressable storage	Console interrupt signal available	Display positions are controlled by light beam position and display orders; protection provided	Cursor can be inserted into any buffer position and its position can be determined by the computer program
3	20	40	800 characters of addressable storage plus space for format controls	Console interrupt available	Each storage position is displayed in a fixed display position; protection provided	Cursor controlled by Cursor Address Register that can be set or read by the computer program
IDEAL	40	80	3200 characters of addressable storage	Console interrupt; partial read/writes from any location	Fixed storage-to-display positions relationship; data protection	Computer-accessible Cursor Address Register

display a full 50 lines of 70 characters per line; it has 4000 or 8000 characters of addressable buffer storage; it can be signaled to read through a computer interrupt key; display positions are set by positioning the light beam; fields can be protected; the cursor can be inserted at any displayed position; and the computer can determine the cursor location. This more flexible device can also draw lines or vectors between grid points, but this capability is not very common, so it is not handled by the basic IDUL. It can be utilized with simple additional code, but the resultant program becomes restricted to the device.

The last hypothetical display device is more direct and easier to use than the other two. It has 800 fixed storage positions that correspond to the 20 lines of 40 characters per line of displayed data. The first buffer storage character is displayed at line 1, position 1, the forty-first character is displayed at line 2, position 1, etc. This display can signal for computer action with an operator key, and the cursor is controlled by a cursor address register that can be set or sampled by the computer program. The displayed data can be protected from external change.

Of course, these typical display devices do not contain all of the features that are now available. They do point out some of the major differences that IDUL routines or any display support package must consider. The language translator that transcribes the display definitions into computer code must be able to recognize signals from the individual displays; it must convert the tabular definitions into forms acceptable to the display; it must convert data from the display (and in some cases reformat the data) into a form usable by the computer program; it must control the input/output operations; it must be able to effect data protection, and the Intermediate Display User's Language Translator must provide a mechanism for setting cursor locations.

The characteristics for the ideal display were selected to simplify the translation of the Intermediate Display User's Language into computer code. The fixed relationship between the storage position and display positions makes it possible to change or read all or part of the display. This eliminates the transmission of redundant information between the computer and the display. The absolute cursor controls and keyboard interrupt provide for immediate location of communication areas with minimal computer testing. And the character-by-character protection of displayed information permits maximum flexibility in the selection of characters for protection. An IDUL-to-machine-code translator could be quickly and easilt written for these ideal characteristics.

The described Intermediate Display User's Language has been implemented for the Interactive Programming Support System using an IBM 2250 display operating under the ADEPT-50 Time-Sharing System. The 2250 display is not the one that will ultimately be used. Therefore, new routines will be required to interpret the basic IDUL, but the many display views set up by the IPSS routines will (we hope) remain intact, with each view defined in terms of its field length and display view position by line and column as stated in the Intermediate Display User's Language tables.

CONCLUSION

We have found that a display scope for program composition and editing is preferred over a typewriter-like terminal by most users. This preference has also been noted in THOR.[3] This perhaps reflects the fact that typing is a skill not necessarily possesed by all programmers.

None of the displays we have examined are completely suitable for the requirements of IPSS. In addition to the hardware characteristics noted in Table III, an ideal display should be large enough (say 40 lines with 80 characters each) to contain a reasonable sample of data. We have found that the programmer needs a minimum of 20 lines in block 3 alone, in order to correlate displays of several kinds of information. Two final requirements are that the display device must be compatible with computer input/output hardware and the device must be inexpensive.

Syntactic analysis combined with editing gives the programmer two powerful programming aids. He is assured that his edited program is syntactically correct by being immediately advised of errors. He has access to dynamically current program information. IPSS has just begun to develop techniques to process this information and present it in a form useful to the user and thus offer him a significantly improved alternative to personally extracting similar information from program listings.

REFERENCES

1 L P DEUTSCH B W LAMPSON
Reference manual QED time-sharing editor
Document No 30 60 30 University of California Berkeley November 1965

2 *The compatible time-sharing system-A programmer's guide*
The MIT Press Massachusetts Institute of Technology Cambridge Massachusetts

3 J McCARTHY et al
THOR—A display based time-sharing system
Proceedings Spring Joint Computer Conference 1967 pp 623–633

4 R W FLOYD
The syntax of programming languages—A survey
IEEE Transactions on Electronic Computers 1964 Vol EC–133 pp 346–353

New horizons for magnetic bulk storage devices

by FRANK D. RISKO

Bryant Computer Products
Walled Lake, Michigan

INTRODUCTION

There are many types of auxiliary bulk storage devices available on the market now and certainly many more will be available in the future. The second, and in some cases, third generation bulk storage devices are making their appearance on the computer peripheral market now. These devices, much to the surprise of many people, are still rotating magnetic media devices. This paper discusses only state of the art bulk storage devices of the magnetic recording variety. However, since a look at the future is in order, toward the end of the paper, we will attempt to conjecture as to what future memories may look like.

The question of using bulk storage evolves around certain major factors. One factor is cost. This is the primary reason why such items as micrologic or integrated circuit[1] flip-flops have not emerged as bulk storage devices. The cost per bit of these types of devices is extremely high and even with LSI (Large Scale Integration) or woven screen memories, it is still an expensive proposition. Magnetic thin film devices and magnetic cores are both used in most computer systems for the so-called scratch pad, or high speed memory. But again, these devices—in terms of bulk storage—are much too costly.

To put the area of bulk storage into perspective, this paper considers small storage as those of less than five million bits. (The majority of these are head per track devices.) The next memory size range considers a medium range memory to go from 5 million to 50 million bits. (This is the beginning of the positionable-head devices.) Large magnetic memories are then considered from 50 million to 500 million bits. Bulk memories obviously fall above the 500 million bits capacity and have ranges going up to 10 billion bits. Memories above this range which could go up to the trillion* and possibly the quadrillion bit range should be considered large bulk memories. Since a common term makes comparison easier, the bit has been generally used in this paper because character length is manufacture-dependent. When appropriate, character is used and is assumed to have eight bits.

Since the above ranges of memory—particularly in magnetic storage devices—cover a lot of applications, discussion is limited first to state of the art memories in bulk storage category of the contact and non-contact type and then extends these to the near future and beyond. Many papers have been presented in the past[1-7] which have documented many aspects of bulk memories. This paper touches only on some of the more inportant aspects which are affecting present-day thought and equipment design.

Performance factors

There are two major performance factors which must be considered in the discussion of bulk storage devices. These performance factors can be defined as hardware-derived and system-derived.[4] The hardware derived performance factors consider the major areas of:

- Capacity
- Access Time
- Latency
- Data Transfer Rates
- Cost per Bit (or bits per dollar)

The system derived performance factors consider such items as:

- Throughput

*Possibly a new unit of measure such as the Mega-Mega Bit (MMB) for trillion bit capacity should be used since the "mega" is already an accepted term.

— **Indexing Procedures**
— **Memory Allocation**
— **Chaining Provisions**
— **File Activity**
— **Provisions for Queuing**
— **Checking Techniques**
— **Format**

Each of the various types of memory devices such as tape, disc or drum excel in one of the areas listed above more so than in another. Obviously the most important points must be considered in order to optimize the utility of a particular type of system. Because it would be almost impossible to discuss all of the just mentioned factors, this paper is limited to throughput, memory allocations, file activity and format.

Hardware factors

Capacity

Capacity is considered as twofold since this could be taken as total fixed on-line capacity, or total unit capacity, where the unit could be a reel of tape or a replaceable disc, or disc pack. The disc has major advantages when considered as a bulk store. Using any of these three devices it is possible to have virtually rooms full of records up to an almost infinite store. However, the access time of a particular record in that store includes the operators' "fetch" time when comparing it to an on-line fixed store. Most people do not consider "fetch" time when comparing storage devices. Further comparison between these devices is not meaningful because the disc pack outperforms the tape in the area of random access and longevity, and all of these devices have a limited on-line unit capacity.

Consider the fixed capacity on-line store. If enough units (floor space provided) can be put in tandem with the proper controller, it is possible to have in excess of five billion characters available. This means "wall-to-wall" data cells, disc files or drums.

Access time

Access times for very large stores are generally longer than 75 milliseconds. The reason for this is that they are generally moving head devices, or moving strips such as CRAM. The simple fact that an electro-mechanical or electro-hydraulic positioning device is used puts these devices into the long access time ranges. Some head per track devices are only limited by rotational latency for access time, but they generally fall into the large storage area rather than the bulk storage area as we mentioned earlier.

If we consider the larger replaceable disc drive type device, such as the IBM model 2314 with eight disc packs and single data channel operation, we find that we have multiple-seek features for the 3.2-billion bit storage. If, for a moment,

TABLE I

MANUFACTURE		UNIT NO.	CAPACITY MILLIONS OF BITS	ACCESS TIME MILLISECONDS MIN.	AVG.	MAX.	END USER COST DOLLARS	BITS/DOLLARS
Bryant	1	4000 Mod 1	750	60	140	180	260K	2,880
	2	4000 Mod 2	1600	60	160	205	305K	5,240
	3	4000 Mod 2a	4000	60	160	205	440K	9,100
	4	4000 Mod 2b	5000	60	160	205	480K	10,400
	5	Phd 340*	340	30	70	105	125K	2,720
	6	Phd 170*	170	30	70	105	105K	1,820
	7	CD-145*	145	-	17	33	210K	690
CDC	9	818	3750	40	146	226	310K	12,100
	10	814	1200	34	65	110	205K	5,850
	11	6638	1000	25	70	115	325K	3,080
Data Products	12	dp/f5045111	870	50	150	250	227K	3,840
	13	dp/f5085	5000	15	85	150	450K	11,100
Burroughs	8	B475	102	-	20	40	95K	1,080
IBM	14	2302-4	1800	50	115	180	355K	5,070
	15	2311-12	43	30	98	185	26K	1,650
	16	2314	1656	25	88	165	252K	6,580
	17	2321**	3200	50	550	600	140K	22,800
NCR	18	353-5**	360	3	110	125	58K	6,220
RCA	19	70/568-11**	4488	136	500	550	238K	18,800
	20	2488-8**	2400	130	500	640	135K	17,800
Univac	21	Fastrand FRII*	920	5	92	155	168K	5,480

* DRUM
** STRIPS OR CARDS
All other units are discs.

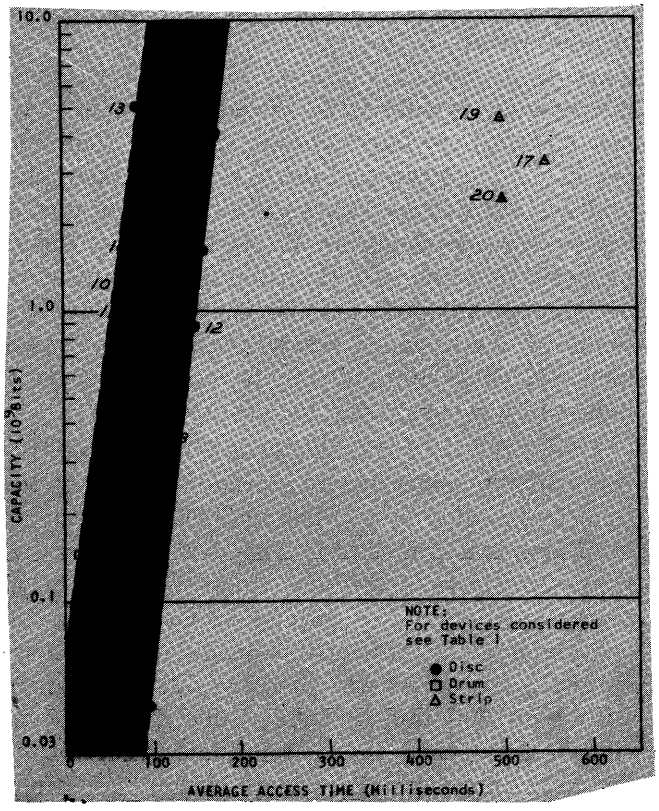


FIGURE 1-Bulk store—capacity vs. access time

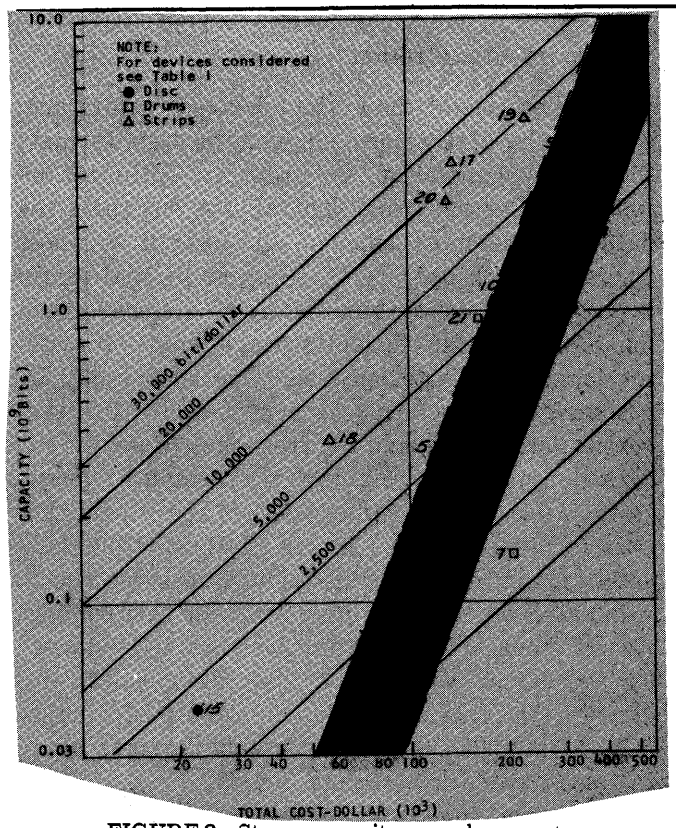


FIGURE 2 Storage capacity vs. end user cost

we want to consider only cost versus access time, we can get the cheapest bulk storage device by buying a unit which has very low cost, but with a long access time. The IBM model 2321 gives 22,800 bits per dollar but at 550 milliseconds access time. A similar price comparison for a Bryant Series 4000 Model 2B Disc File gives us 10,400 bits per dollar and an average access time of only 160 milliseconds. In other words, on one hand you can reduce the cost of a memory system by 50%, but the access time is increased by over 300%. See Table I for a comparison of various manufacturers capacities, access time and costs.

To fully appreciate some of these hardware performance characteristics, we may examine Figure 1 which shows the capacity of the various electro-mechanical types of devices versus access time. This area shows that the higher the capacity of a given store the longer its average access time. Naturally, everyone wants not only a larger store, but wants access to that store in the minimum amount of time, so the area trend line can be expected to move to the left in the future. Figure 2 indicates the capacity versus end-user cost on a bits-per-dollar basis. Again, the trend is end-user cost reduction with increasing capacity. The trend area shows the cost by capacity of the various types of mass memory devices. Note that the greatest economy is gained with strip-type devices at an increase in access time. Also, that all the units outside the trend area to the left in Figure 2 are strip devices. However, the need for increasing speed to obtain data will reduce this trend and cause the industry to move toward the faster access devices into the high bits per dollar range. One of the major problems with the strip device is its poor reliability and limited magnetic coating life.

Latency time

Latency time and data transfer rates are inseparable. Latency time is defined as the time of a single cycle of the magnetic memory device. Data transfer rate is directly related to how many bits-per-inch have been packed onto the given device surface. Since we are always striving for maximum capacity, we are usually recording at maximum packing density. In order to reduce latency, we need as high a surface speed as possible, which increases the transfer rate. In a typical serial device, the data transfer rate could be five Mega-bits per second. This would be typically a rotating device with a mean diameter of 10 inches, turning at 3600 RPM with data written at 2500 bits-per-inch. Lower surface speed units with parallel data transfer can achieve similar rates.

System factors

System-derived performance factors provide a most interesting, as well as practical aproach to the bulk storage areas. With time sharing users becoming more sophisticated, a larger amount of on-line storage is required. This larger on-line storage, however, necessitates a higher throughput. One of the ways throughput (or transactions per second) can be improved upon in a bulk store is to increase the total number of time sharing accesses to that store.

Throughput

Consider for a moment an imaginary time-sharing installation which has a given number of users. To use this computer on a time-share basis it should be obvious that there must be some amount of storage allocated for the users. Storage requirements are a function of how the user is actually achieving his necessary results with his

so-called "own computer." A sophisticated user would probably need on the order of 200,000 characters of storage as opposed to a less sophisticated user who would only require on the order of 50,000 characters of storage. Generally this means that somebody has to wait during the access-time period depending on queuing of the various computer user requests. It could be feasible to reduce waiting time for a given number of users by doubling the computer store access capability. (Additionally, security must be provided so users cannot access each other's data.) Since these people are not accessing the same data, it then seems practical for each user to have his own independent access mechanism. This could be taken care of very easily by going to a group of disc drives (let us say 50 drives for 50 users) in our imaginary time-share system. Obviously, the cost of such a system would be prohibitive because the time-share user cannot afford this luxury.

The next best thing would be to have a number of accesses of positioners on the same device. Some average number of users will always be on-line (let us assume this number to be 25). With 25 on line, how many will be needing access to the bulk storage continuously during any given period of time? The number would probably average 10 to 15. Because of the very nature of the binary application, it would be most feasible to have a binary number of accesses. This may be 8 or 16 positioners. Let us now assume that we do have 16 available; this allows the system performance to increase by a factor of 16 (in regard to throughput) over what a normal single positioner machine would give. One of the unfortunate features of a time-shared system is the fact that at a given point in time we usually have more than one user wanting to access the same positioner. No end to the dilemma. . . .

Memory allocation

Having more than one user accessing a device, in a time-share application, memory allocations for each user is limited in order to serve as many people as possible and still not over-extend the available memory. This is particularly important because the computer itself will want some of the available memory for internal executive and swapping programs. Therefore, it is sometimes necessary for time-share systems to redistribute data in order to make a more economical access and thereby maintain reliable throughput time.

Memory organization

Another side of the system-derived factors are: what is the best record length; the best format; the best organization, in order to optimize the system? The user may not always organize his data for the most efficient throughput or most efficient search operations, so an additional strain is put on the total system. Many studies have been made of the most efficient storage length (or sector length) and storage use. One such study by Brenner[8] discusses this as a probability function versus data length, and the frequency of use versus data length. In weighing performance versus storage use, Brenner showed that for a performance of 90%, a record length of 600 to 700 characters is optimum. His decision to use this length was based on the absolute change in performance versus storage use. Figure 3 shows that storage use decreases very quickly as data length increases. In other words, to achieve a performance approaching one hundred percent, it is necessary to have a very long record length.

Of course, one of the major problems with record length is the fact that storage use de-

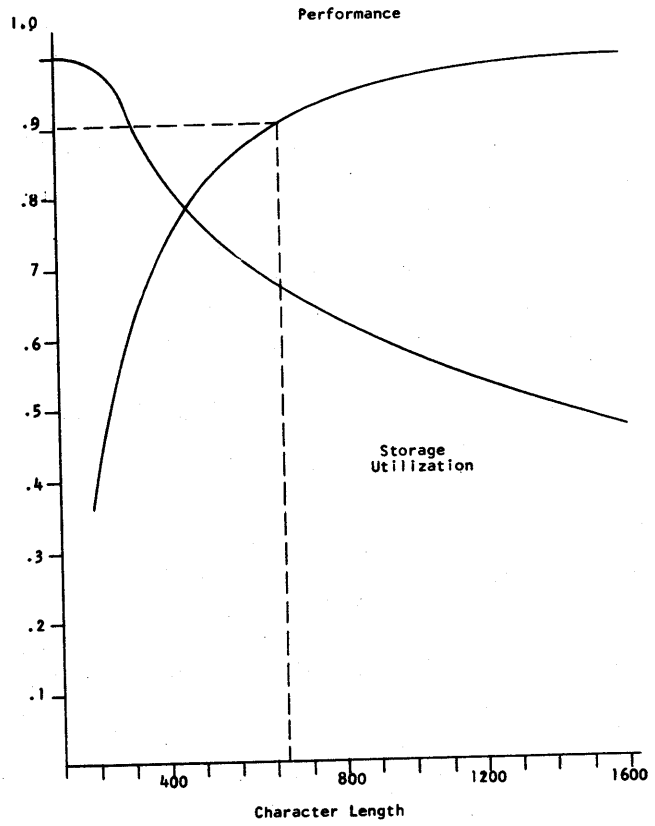


FIGURE 3 Performance and storage utilizati n

creases as length increases because of various over-head functions. This is reflected in an independent (unpublished) study which showed that the most feasible length is approximately nine-thousand bits. In order to make the bit quantity binary, the amount with the nearest factor is actually 256 twenty-four bit words, which gives us approximately 6000 bits as the most useful (or optimum) length. This study is in approximate agreement with the study made by Brenner.

The system and programming aspects must be extended for a given unit in order to allow for the proper indexing, chaining and queuing techniques which are required by the mechanical portions of the storage device. Obviously a head-per-track, parallel system would provide the ultimate in all these areas, but would be very expensive because of the electronics required to handle such a system.

Present and future trends

The present hardware state of the art of the bulk memory field has decreased to 5-mil track widths with 7.5 mil centers between tracks while packing densities have increased upwards to 2500 bits-per-inch. The 5-mil track widths give approximately 128 tracks-per-inch. This gives us a quarter of a million bits per square inch storage capacity as the state of the art magnetic recording of today. However, magnetic recording development is nowhere near being finished in this particular area.

Magnetic recording in the order of 20,000 bits-per-inch have been demonstrated[9,10] and will, no doubt, be the future packing densities by the mid-1970's. Additional hybrid electronic circuits and recording methods will definitely be necessary to achieve these ends. At the present time we're talking of practical playback levels at the head on the order of one to three millivolts. By 1970 to 1975, the playback ranges will be down to the order of 10 to 30 microvolts. This will require amplifiers which must be absolutely immune to noise and spurious signals. These values, I might add, are being achieved experimentally today in the laboratory and this is where we were approximately five years ago. I believe during 1961 and 1962 that Shew from IBM was doing magnetic recording on the order of 1000 to 1500 bits-per-inch under laboratory conditions working with 1-mil wide pole pieces.[11] At that time, state of the art was generally 300 bits-per-inch.

Positioners

A major concern about use of the 1-mil pole piece is that of positioning tolerances necessary so that off-track and peak-shifting problems do not affect playback. The positioning mechanism will have to be at least two orders of magnitude better than present day standards in order to accommodate this type head. Development work at Bryant shows that the next generation bulk stores will make use of a 2.5 to 3 mil track and will be spaced on 3.75-mil centers. This means that we should achieve approximately 256 tracks-per-inch. Considering only a modest increase in packing to 4000 bits-per-inch, we will arrive easily at one-million bits-per-square inch by 1970. The major breakthrough, of course, is the positioning device itself, which must be fantastically more accurate in the area of track selection and positioning repeatability.

To continue our view of future magnetic devices, I would venture to say that a minimum of 16 multiple access moving head-bulk stores, with total capacities of 50 billion bits, should be with us by 1972. The capacity from then on will probably increase logarithmically and the next jump, I would say without any hesitation, would be from 50-billion to 100-bilion bits and then from 100-billion to a 100-trillion bit magnetic memory by 1975. This again will be a magnetic memory in which certain as yet unimagined manufacturing techniques will have been achieved.

Head developments

By 1975 the head will not be a discrete individually-assembled element but will be batch-fabricated heads similar to today's micrologics. The preamplifiers will no doubt be deposited as part of the head assembly. I also believe that many new techniques will have to come from the metallurgical and the chemical industries to give us the needed techniques and materials in order to achieve 16 to 32 heads per pad without interfering crosstalk and frequency limitations. At the present time there are many devices on the market which have from 9 to 20 heads per pads, the Burroughs disc file for one, the Data Disc and Bryan units (which have 9) for another.

The person who wants a tremendous bulk store

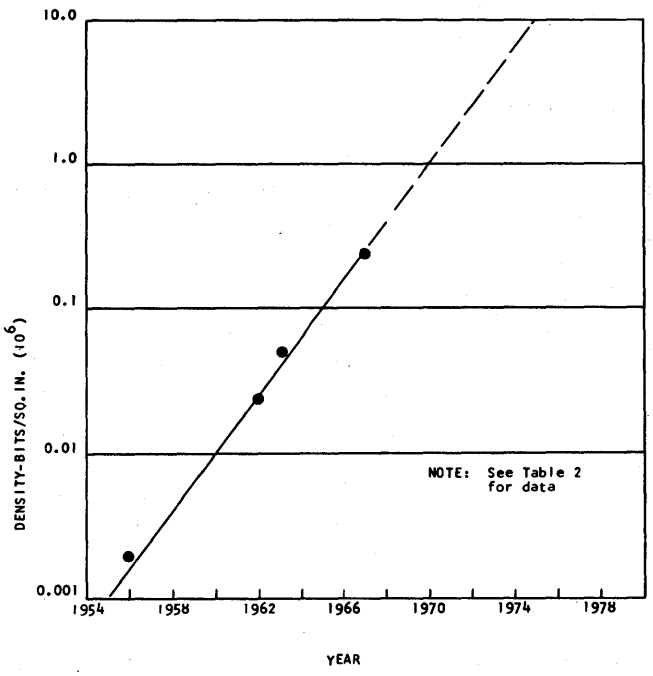


FIGURE 4 Packing density vs. year

is not going to pay any more in 1975 for a trillion (Mega-Mega) bits than he would for a trillion bits right now. The major difference will be that a user will probably only need 1/16 the floor space to hold that device and, no doubt, at a much lower power consumption. The growth of the industry as shown in Table 2, is plotted as a function of packing density (bits/sq. in.) versus the year associated with it.[4-6] This is shown in Figure 4, and is interesting, because the growth has been increasing by an order of magnitude every five years from 1956. At this rate a million bits-per-square inch will be commonplace by 1970.

TABLE 2

YEAR	DENSITY (Bits/Sq. In.)
1956	2,000
1962	25,000
1963	50,000
1967	250,000

The obvious area of improvement along with the batch fabrication techniques envisioned for positioning head devices will be to position the heads over a much shorter distance. The shorter this distance the greater the improvement in access time. Some present day devices move as much as two inches and some as little as a quarter of an inch. I believe that our 1975 devices will use electrical positioners and that the distances traveled will probably be on the order of less than 50-thousandths inch. The reason is that the track width being only 1-mil, or possibly 2-mils, will not allow a positioning accuracy for movements greater than that.

If we consider the present cost of these various devices the future cost ratio will not really change. In other words, the cost per bit is not going to go up appreciably, nor is it going to drop. The manufacturing and material costs will increase, causing the unit cost to increase, but, by the same token, the technology in terms of packing density and bits-per-square inch, as well as bits-per-cubic inch (which, of course, plays an important role in floor space) will also change such that the average cost per bit (or bits per dollar) will not change, but will remain relatively constant.

CONCLUSION

Magnetic type devices are definitely here to stay for another ten years before they give way to the laser/optical type devices. The disc packs provide useful bulk storage and will continue to dominate the small-storage market. The disc packs have a definite place in the bulk memory market although in a different way. The on-line bulk store will be more necessary in the future and will be available with multiple accesses. Some of the problems to be overcome are reliability and improved performance and elimination of the redundant store which is necessary for large systems. Another area of growth will be the medium range (head per track) devices for use in swapping requirements for time-share installations.

Optical type devices are a definite way to go, but they may get competition from LSI devices. The photochromic and laser or optical-type devices which will far exceed total capacity per square inch (25 million-bits-per-sq. in.) have a long way to go to reach the present state of the art of magnetic recording.

Although they are capable of larger capacities in smaller areas, the technology to develop them is still in the future. Magnetic recording of digital information has a good solid 15 years, maybe 19 years, of technology behind it, whereas attempting to employ lasers in reading and writing (*not just read only*) is a tremendous technological achievement that must be made before the laser technique

or beam-memory techniques will become feasible.

At the present time there are many laser read-only memories in operation. I don't want to detract from this achievement, but I feel very strongly that—in order for them to achieve the same state of the art as the magnetic recording media has now (and will have by 1970)—some major technological advances will have to occur. Major research effort is being devoted to this area and the advancement could occur sooner. The holographic displays, although they haven't been publicized much lately, appear also to be feasible and will probably give the laser photochromic devices a good run for their money. However, they still must be nondestruct read/write.

I would like to quote Marvin Camras from the 50th anniversary issue of the IRE Proceedings in 1962[12] extrapolating progress to the year 2012.

> "The first magnetic memory devices resembled tape recorders, just as the first automibiles resembled horse carriages. Eventually tape recorders evolved into the standard memory pack which is presently manufactured in large quantities by specialists. The memory pack of today (2012) is a sealed box about the size of a package of playing cards. It holds upwards of 10^{20} bits of information, and has no mechanically moving parts; the recording, readout, and scanning are all electronic. The storage density is so high that the information of entire libraries is condensed into a few cubic feet. Additional modules are added to extend the capacity to any required level."

I would say we still have a long way to go to achieve that prediction, but the way things can and do happen, a major breakthrough could at any moment occur; I certainly do believe that it can be done.

REFERENCES

1 R A HENLE L O HILL
Integrated computer circuits-past present and future
Proceedings of the IEEE December 1966 pp 1849-60

2 L C HOBBS
Effects of large arrays on machine organization and hardware/software tradeoffs
Proceeding-Fall Joint Computer Conference 1966 pp 89-96

3 L C HOBBS
Present and future state-of-the-art in computer memories
IEEE Transactions on Electronic Computers vol EC-15 August 1966 pp 535-549

4 A S HOAGLAND
Mass storage revistied
AFIPS Proceeding-Fall Joint Computer Conference 1967 pp 235-60

5 L C HOBBS
Review and survey of mass memories
AFIPS Proceeding-Fall Joint Computer Conference 1963 pp 295-310

6 T H BONN
Mass storage A broad review
Proceedings of the IEEE December 1966 pp 1861-70

7 N NISENOFF
Hardware for information processing system: today and in the future
Proceeding of the IEEE vol 54 no 12 December 1966 pp 1820-35

8 F H BRENNER
On designing generalized file records for management information system
AFIPS Proceeding-Fall Joint Computer Conference 1967 pp 291-303

9 K A NORRIS
A technique for high density digital magnetic recording
Presented at the Western Electronic Show and Convention August 24 1967

10 C W STEEL J C MALLINSON
A computer simulation of unbiased digital recording
Abstracts of 1968 Intermag Conference Washington D C p 3 3

11 L F SHEW
High density magnetic head design for non-contact recording
IRE Transactions on Electronic Computers December 1962 pp 764-83

12 M CAMRAS
Magnetic recording and reproduction-2012 AD
Proceeding of the IRE vol 50 no 5 May 1962 p 639

BIBLIOGRAPHY

1 M S FINEBERG O SERLIN
Multiprogramming for hybrid computation
AFIPS Conference Proceedings 1967 vol 31 Washington D C Thompson Book pp 1-13

2 L R GLINKA R M BRUSH A J UNGAR
Design through simulation of a multiple-access information system
AFIPS Conference Proceedings 1967 vol 31 Washington D C Thompson Books pp 437-47

3 A S HOAGLAND
Mass storage
Proceedings IRE vol 50 May 1962 pp 1087-92

4 R L PETRITY
Current status of large scale integration technology
AFIPS Conference Proceedings 1967 vol 31 Washington D C Thompson Books pp 65-85

5 C S POLAND
Advanced concepts of utilization of mass storage
Proceedings of IFIP Congress 65 vol 1 Washington D C Spartan Books 1965 pp 249-54

6 J A RAJCHMAN
Computer memories—possible future developments
RCA Review
vol 23 June 1962 pp 137-51

7 D E SPELIOTIS
Magnetic recording theories; Accomplishments and unresolved problems
IEEE Transaction and Magnetics vol MAG-3 September 1967 pp 195-200

Laser recording unit for high density permanent digital data storage

by K. McFARLAND and M. HASHIGUCHI

Precision Instrument Company
Palto Alto, California

INTRODUCTION

The Laser Recording Unit designed and developed by Precision Instrument Company provides a means for reliably and permanently recording and reproducing digital data.

The Laser Recording Unit uses a new type of permanent recording process which employs a laser to vaporize minute holes in the metallic surface of the recording medium. In this manner digital information is recorded in parallel data tracks along the length of a recording medium strip. The tracks are spaced on the order of five to ten microns (micrometers), center-to-center; each track is composed of bit cells three to five microns in size. For recording or reproducing sequential tracks of digital data, the maximum transfer rate of the Laser Recording Unit is approximately four million bits per second, with an average unrecoverable error rate of one in 10^8 to 10^9 bits, depending on the data density selected.

The Laser Recording Unit includes a programmable Recorder Control Subsystem which can be designed to provide a hardware and software interface compatible with a specified computer system.

The major benefits offered by the Laser Recording Unit as a mass digital-data storage unit are summarized below:

(1) Permanent Storage: Data do not degrade over a period of time of the order of years.

(2) Compact Storage: Data are stored at a density approximately 250 times greater than that of digital magnetic tape.

(3) Unlimited Readout: Data can be repeatedly readout for long periods of time without reduction in quality or damage to the record.

(4) Recording Verification: Essentially error-free data records result from the simultaneous read-while-write verification capability that is unique to the laser hole-vaporization method of permanent recording.

(5) Low Error Rates: The average unrecoverable error rate is approximately one in 10^8 to 10^9 bits.

(6) Economical Data Storage: Recording of large quantities of data on the Laser Recording Unit and permanent storage of the data in PI Record Strips significantly decreases the cost-per-bit of recording and storage imposed by existing methods.

Laser recording unit design

The Laser Recording Unit employs a special recording medium in strip form which is automatically mounted on a precision drum assembly. Using digitally modulated incident coherent laser radiation, diffraction-limited marks or holes, each representing one bit of information, are permanently evaporated in a track pattern in the recording medium as the medium and mounting drum rotate under a focusing microscope objective. Thus, parallel data tracks are recorded transversely along the recording medium.

The major parameters which must be considered in the design of a Laser Recording Unit are:

(1) Optical mode of laser beam.

(2) Diameter and divergence angle of the beam.

(3) Diffraction limits of the optical system which images the laser aperture on the recording medium.[1]

(4) The thermodynamics of the hole-forming vaporization process.[1,2,3]

(5) The kinematics of the optical scanning and mechanical transport of the recording medium.[3]

(6) Characteristics of thin metallic films, deposited onto flexible substrates.[4]

A typical embodiment of these principles is the prototype demonstrator unit in operation at the Research Laboratory of the Precision Instrument Company since January 29, 1968 (see Figure 1).

The recording/reproducing laser is a cw argon II ionic laser operating in zero-order, single-mode (TEM_{oo}) at a maximum power of one watt over all wavelengths. The wavelength of 4880 Å has a maximum intensity of 0.40 watt and is separated from the total output of the laser by means of multiple-layer dielectric filtering.

The light beam emitted from the laser passes through a NBS-calibrated thermopile to determine the total emitted laser power and enters the electro-optical modulator which is a Pockel's cell, which provides changes of the refractive index of the modulator medium (KD*P) as a function of the sine of the applied electric field intensity. The physical configuration of this portion of the equipment is shown in Figure 2.

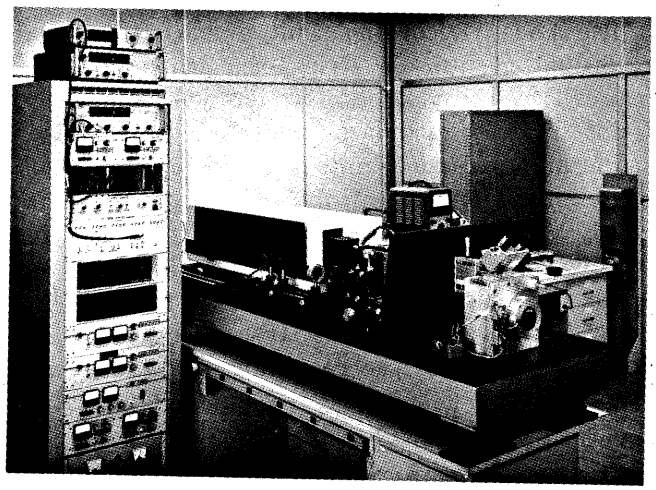


FIGURE 1

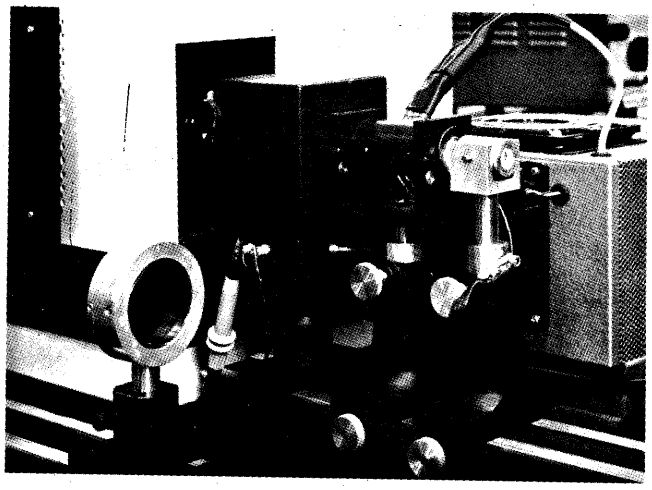


FIGURE 2

After passing the electro-optical modulator and adjacent polarization analyzer (Glan prism), the laser beam enters track widening optics which include a beam folder and an aspheric reflector spreading the wavefront of the laser beam in one dimension. Thereafter, the laser beam passes an optical beam splitter which separates the beam incident to the recording medium from the beam reflected. A periscope adjusts the laser height to that of the recorder so that the laser beam reaches the mirror of the tracking galvanometer. The light reflected back from the recording medium returns again to the optical beam splitter, yielding two separate beams for photoelectric signal detection and error detection, respectively. The appearance of this portion of the equipment is shown in Figure 3.

The recording/reproducing process may be interpreted as follows. Assuming a laser beam is incident to the recording medium at a relative intensity level of 100 percent, reflection and diffraction of the beam occur at the medium. In the absence of a recording, approximately 50 percent of the incident laser power is reflected back towards the incident laser direction. Assuming now the incident laser intensity to be strong enough to vaporize a diffraction-limited section of the recording medium, during the recording process the reflectivity of this area is reduced from 50 percent down to five percent. Hence, separation of incident and reflected light in the path of the laser beam, utilizing a beam splitter, yields an intensity variation of the reflected signal between 50 percent and five percent, thus producing the necessary photoelectric signal output

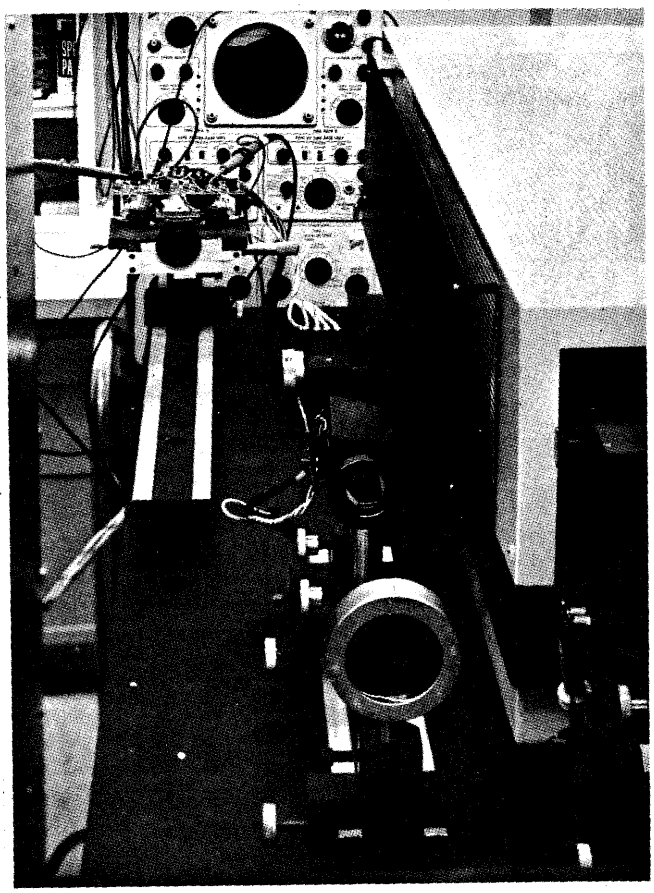


FIGURE 3

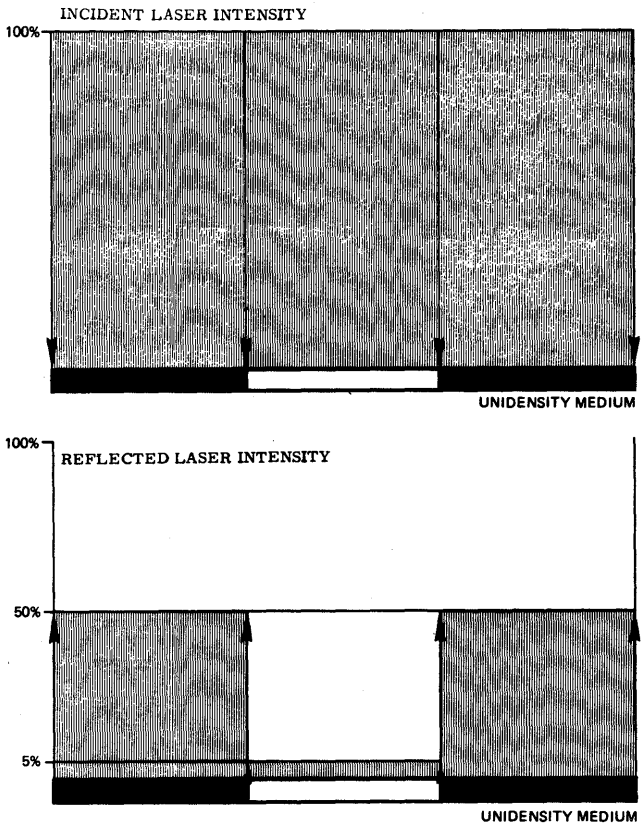


FIGURE 4

for reproducing. Figure 4 is a diagram indicating this relationship.

Figure 5 is a block diagram of the equipment used with the prototype Laser Recording Unit to produce test recordings and evaluate retrieval capabilities. In operation, an unrecorded region suitable for ten or more data tracks is manually set up on the record strip surface. For each track to be recorded, a known arbitrary number is set into the manual count selector switches. The unit is placed in record mode and, when a drum-actuated home signal nevt occurs, the data are recorded on the selected track. To verify retrieval, the tracking servo is commanded to follow this recorded track and repeatedly read out the data thereon. The output of the data sense system is then transmitted to the data output counter, during each drum revolution, and, upon completion of the track readout, this value is transmitted to a parallel identity comparator, to be matched against the state of the count selector switches. Each time an error occurs, as indicated by failure of identity comparison, a count is transmitted to an error counter.

A typical count entered on a data track is about 5,000 repeated data bit patterns. With the drum operating at a nominal rate of 30 RPS, the unit in read mode will repeatedly read and check about 150,000 such bit patterns per second. Repeated runs over a period of several weeks established an average of about 700 seconds of operation between errors, or an error of about one in 10^8 bits.

High density data storage system

Such a system is composed of a Laser Recording Unit and a Recorder Control Subsystem and is typically operated as a peripheral device to a large computing or data processing system.

Laser recording unit

The Laser Recording Unit functionally is a rotating memory with a single movable head for permanent data recording and nondestructive data reproduction. The standard recording-medium strip used with the device is approximately 4.75 by

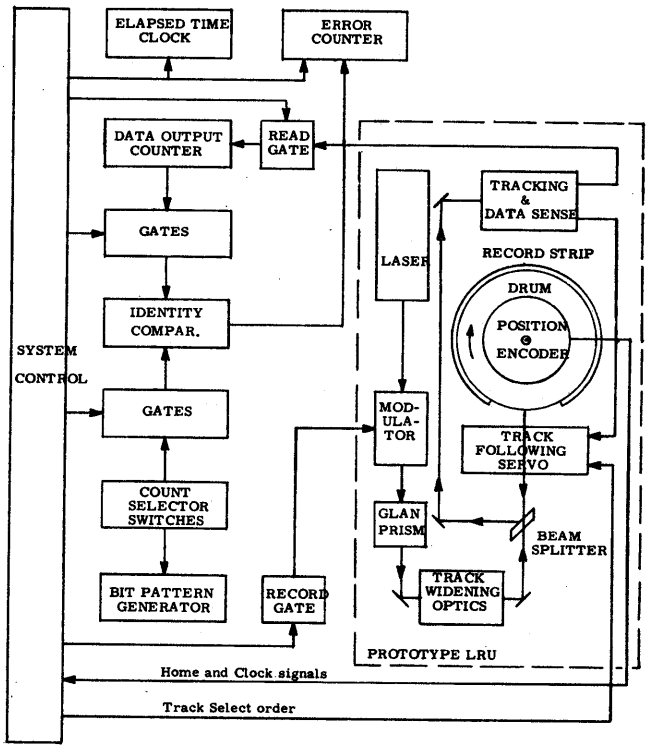


FIGURE 5

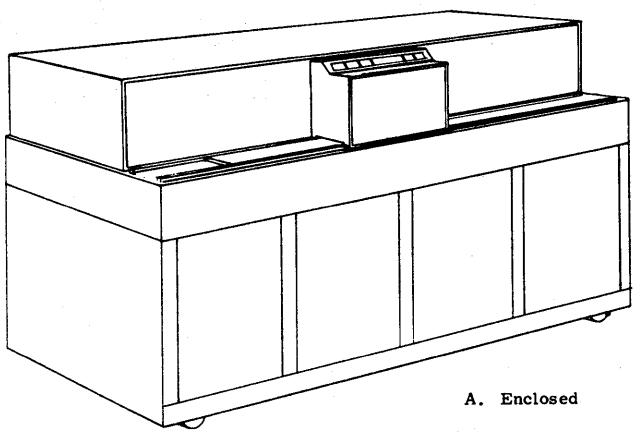
31.25 by 0.01 inches in size; it is wrapped lengthwise around a drum and held down by combined mechanical and vacuum means to provide a precision cylindrical recording surface.

Each standard recording-medium strip can provide a total net data capacity of 1.96×10^9 bits, or 245 million bytes. This capacity will be provided with a bit-cell size of four microns, allowing a bit density of 6,350 bits per inch along each data track and thus approximately 198,500 bits per track; a center-to-center track spacing of eight microns will provide a track density of 3,175 tracks per inch of recording-medium width. Since about 11 percent of the recorded bits will be used for clocking and control purposes, about 175,000 bits per track will represent the net data-recording capacity. With about 11,200 tracks recorded on each strip, the total net data capacity of the strip will be 1.96×10^9 bits or 245 million bytes. For this standard record strip a data transfer rate of 4 million bits, or 500,000 bytes per second can be obtained at a drum rotation speed of 22.9 rps (43.5 milliseconds per revolution). This configuration provides a bit error rate of approximately one bit in 10^8 data bits.

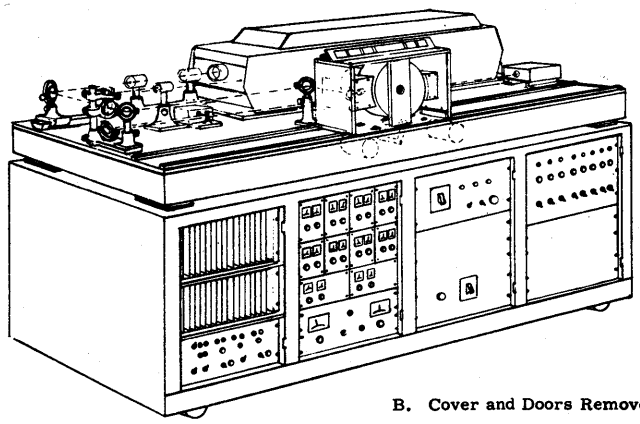
For convenient storage and handling, the recording medium strips are provided with a protective package called a PI Strip Pack. The Strip Pack for a standard record strip is approximately 5.0 by 32.0 by 0.25 inches in size.

Figure 6 is a perspective view of the proposed physical design for a production model of the High Density Storage System incorporating a Laser Recording Unit, configured for use in a computing room environment.

Figure 7 is the overall Laser Recording Unit block diagram. It illustrates the major equipment groups necessary to guarantee subsystem performance. The track-position and carriage-position servos ensure proper positioning of the read and record laser beam. The focus position control group maintains proper objective lens focus position over the full drum width. Laser recording power level is maintained by the laser intensity control group which also provides attenuation of the laser power for the Read Mode. The control logic group provides synchronization, sequencing,



A. Enclosed



B. Cover and Doors Removed

FIGURE 6

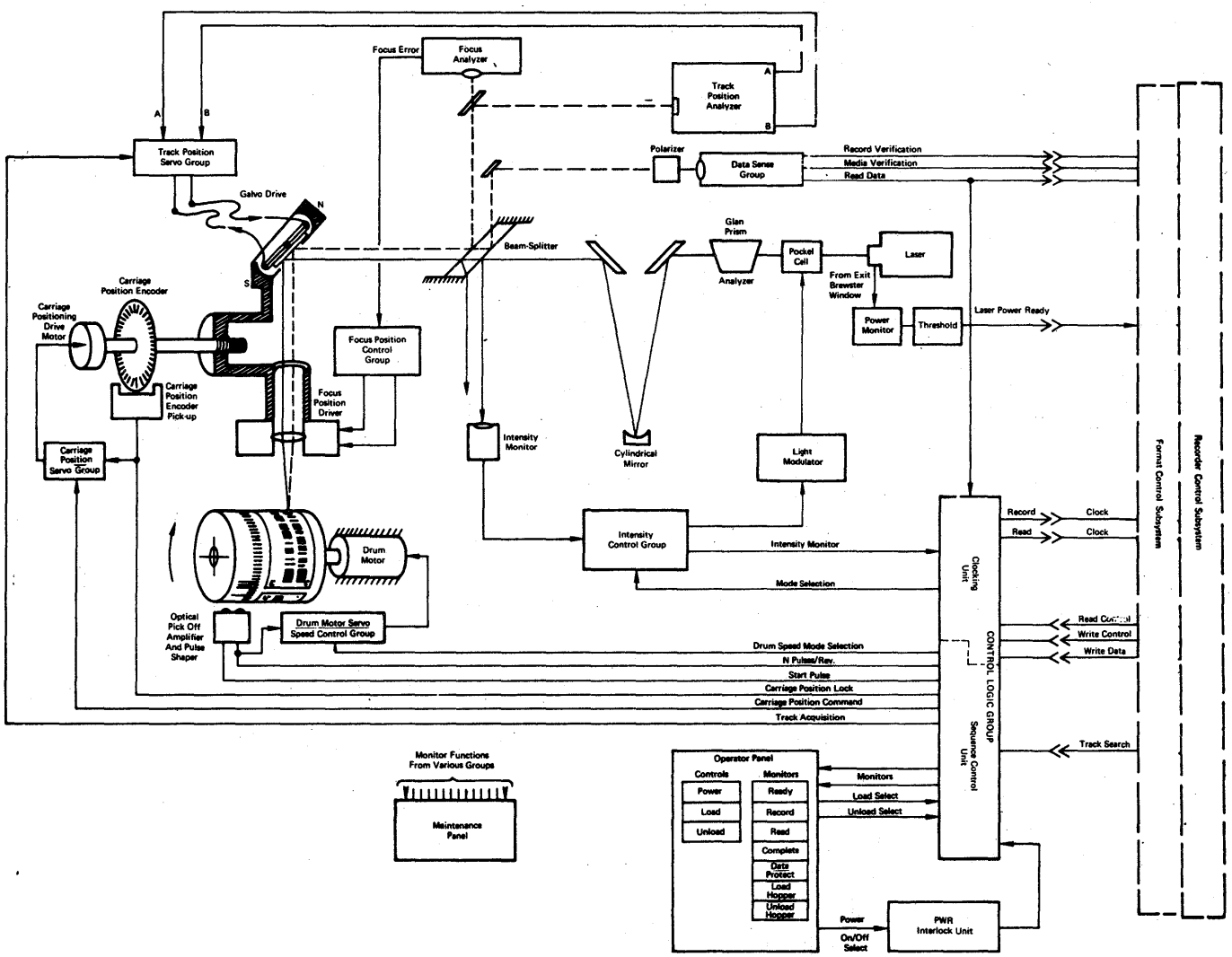
and selection signals for automatic internal control of the subsystem, as well as input/output interface to the Recorder Control Subsystem.

The drum motor servo speed control group provides precise control of record/read drum rotational velocity. The maintenance panel provides controls, indicators, and meters required for routine maintenance and calibration. The data sense group provides media and record verification, as well as data readout. Figures 8A through 8D are block diagrams depicting the functions of the above groups.

Electronics enclosure

The electronics enclosure is located in the laser recorder console cabinet. The cabinet also supports the precision mounting plate, which provides for the mounting of the laser and optics. The top of this cabinet is structurally reinforced to accept the shock mounts holding the precision plate. The base is sufficiently strong to permit the use of a fork lift for moving the laser recorder console. The cabinet consists of four 19-inch standard bays with removable covers or both front and back access. The bay itself is 96 inches long by 32 inches high by 34 inches deep. The strip-pack loading surface is located approximately 38 inches from the floor, and the overall dust cover maximum height is 56 inches from the floor, providing a convenient strip-pack loading height for a standing operator. (Reference Figure 6A.) The laser power supply and heat exchange (used for cooling

FIGURE 7



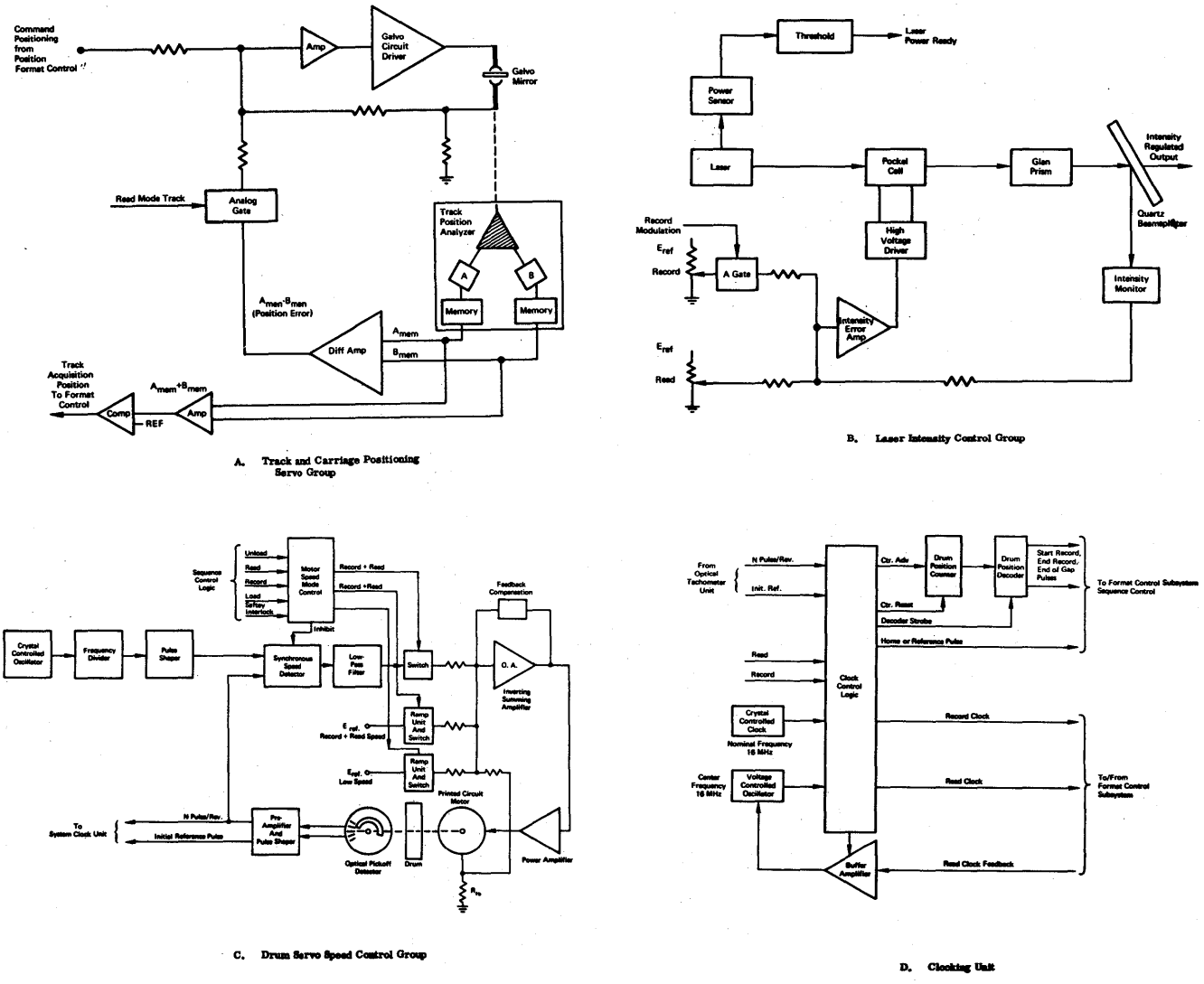
the laser head) will be located in a separate cabinet (not shown.) The space provided in the console cabinet of the Laser Recording Unit is used for the packaging of the signal and control electronics required by the laser recorder, the Recorder Control Subsystem, and the cooling and air-filtration system required for the console (i.e., dust removal in the optical area).

Recorder control subsystem

The Recorder Control Subsystem is physically packaged as part of the Laser Recording Unit. A block diagram of the subsystem is shown in Figure 9. The major element of the subsystem is a special-purpose programmed processor. One output of this processor is to the data channel interface unit of the specified main computing system of which the High Density Storage System is a peripheral device. Other interfaces are to local peripherals of the programmed processor. A final and important interface is to the Laser Recording Unit itself. A block diagram of this interface is shown in Figure 10.

The Laser Recording Unit recording process is conducted on a serial bit-by-bit basis which requires that information transmitted from the Recorder Control Subsystem for recording be not only serialized, but also logically interleaved with two auxiliary signals; the clock and verify signals. The principal objective of the clock is to place, with the stored information, a timing reference for retrieval synchronization. The verify signals will be of two types: the segment (16-bit) verify signal which determines the recording validity of

FIGURE 8



the preceding data segment, and the check sum verify signal, which is a two-character (16-bit) character-oriented accumulated sum verification for each 256-character string.

During retrieval, the Recorder Control Subsystem is required to: (1) determine validity and destination of retrieved information; (2) perform remedial actions and return to normal operation after completion of such actions; and (3) logically remove all clocks, verify and check-sum bits from the serial data while maintaining smoothness of data flow.

Since the function of the Recorder Control Subsystem is to provide communications between the controlling computer complex and the Laser Recording Unit, it is this unit which must be adaptable to the interface requirements imposed by various computers. The Recorder Control Subsystem will accommodate various computer input/output requirements by changing the input interface and the control programs in the programmed processor unit. The remainder of the Recorder Control Subsystem will remain virtually unchanged regardless of the computer complex utilized.

Since a major goal in the design of the Laser Recording Unit—and consequently the Recorder Control Subsystem—is to provide a structure with maximum dependability, additional auxiliary logic is included in the Recorder Control Subsystem to perform a thorough closed-loop diagnostic test of all logical features and characteristics.

During a diagnostic mode, the auxiliary diagnostic logic sets up a logic configuration simulating

FIGURE 9

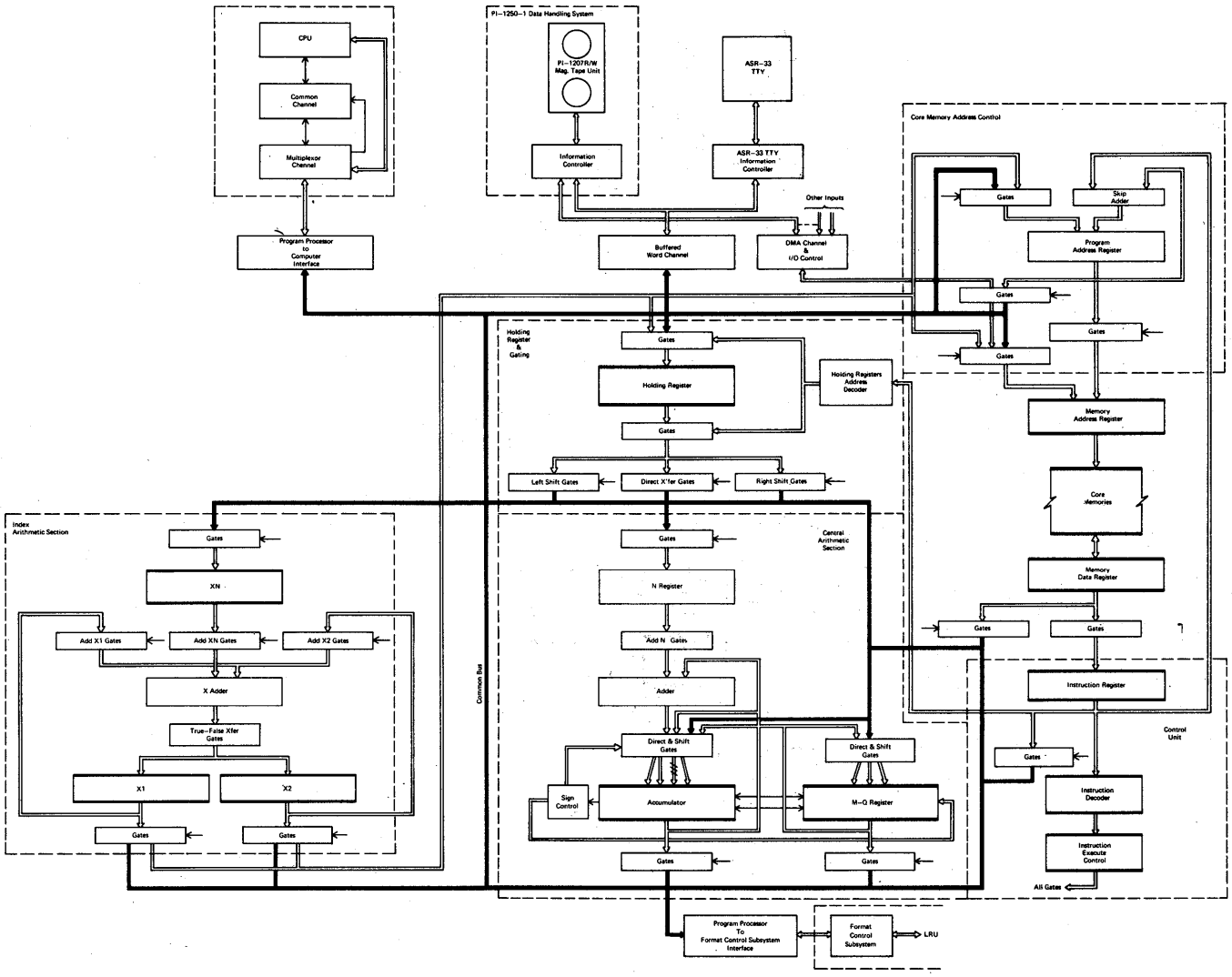
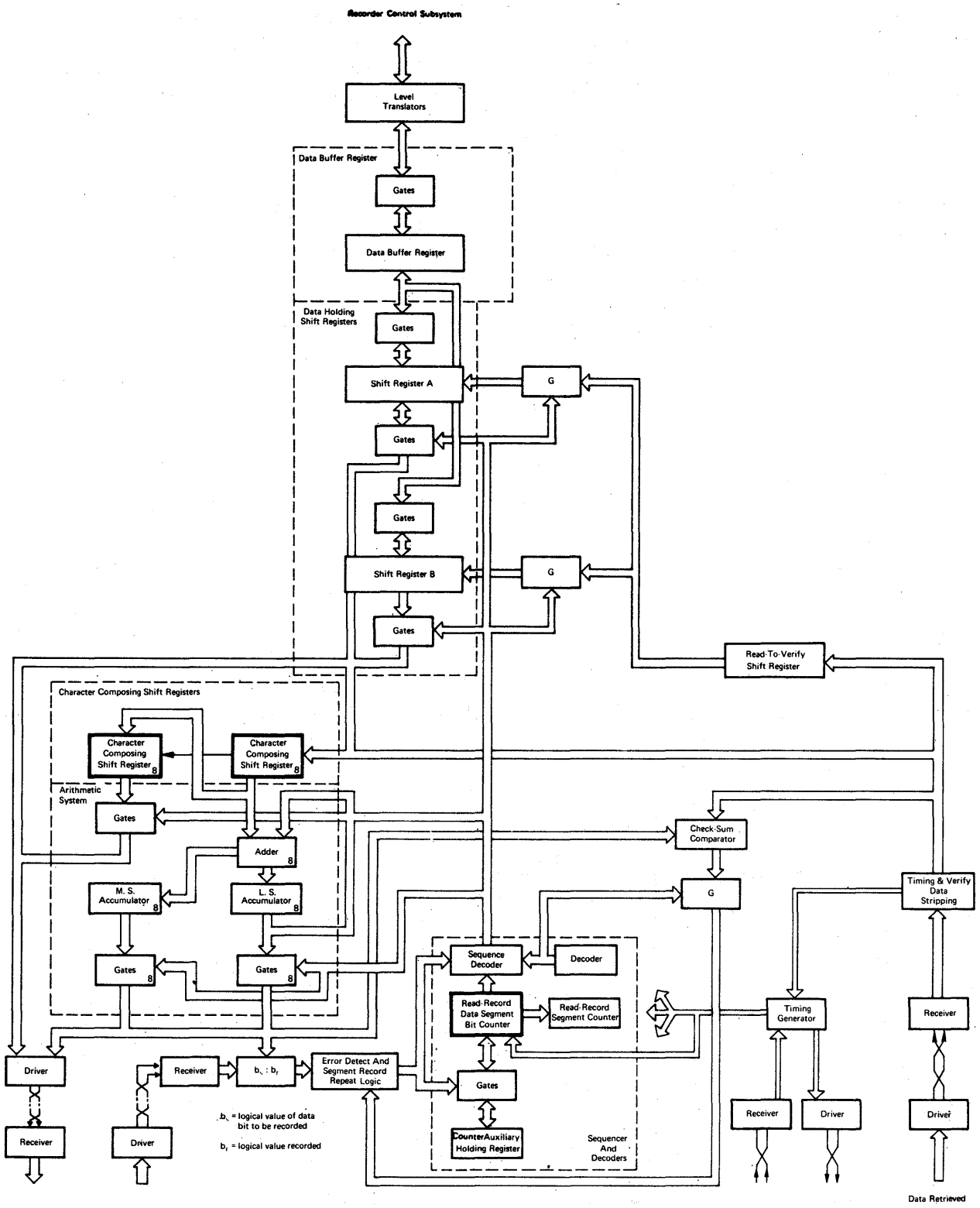


FIGURE 10


Recorder Control Subsystem
Level Translators
Data Buffer Register
Gates
Data Buffer Register
Data Holding Shift Registers
Gates
Shift Register A
G
Gates
Gates
Shift Register B
G
Gates
Read-To-Verify Shift Register
Character Composing Shift Registers
Character Composing Shift Register 8
Character Composing Shift Register 8
Arithmetic System
Gates
Adder 8
M. S. Accumulator 8
L. S. Accumulator 8
Gates 8
Gates 8
Check-Sum Comparator
G
Timing & Verify Data Stripping
Sequence Decoder
Decoder
Read-Record Data Segment Bit Counter
Read-Record Segment Counter
Timing Generator
Receiver
Driver
Receiver
$b_\wedge : b_r$
Error Detect And Segment Record Repeat Logic
Gates
Receiver
Driver
Counter Auxiliary Holding Register
Sequencer And Decoders
Receiver
Driver
Driver
$b_\wedge$ = logical value of data bit to be recorded
b_r = logical value recorded
Data Retrieved

a recording-reading operation; i.e., as data are received from the data channel, they follow a normal-sequence recording path; however, the data transmission is diverted from the Laser Recording Unit and instead follows a closed loop within the Recorder Control Subsystem. A simulated retrieval is performed in essentially every respect, which concludes with the data being sent back to the computer to be examined for evaluation of Recorder Control Subsystem operation. After successful completion of this diagnostic procedure, a further procedure for test of the Laser Recording Unit may be initiated. With the aid of an operator, PI strip packs may be inserted in the Laser Recording Unit under order of a diagnostic program, known synthetic data recorded, and then successfully read back for a diagnostic check of the complete Laser Recording Unit.

Recorded data format

The data recorded by the Laser Recording Unit will be placed on the high-density recording medium through incident coherent laser radiation, causing diffraction-limited holes to be permanently evaporated in the recording medium in a pattern which represents the binary input data.

The recorded data will be placed in tracks running parallel to the length of the medium. In a typical organization of the recorded data, these tracks will be separated by an 8.0-micron center-to-center spacing: this will allow over 3000 tracks per inch of medium width. The data bits to be written along these tracks will be spaced on 4.0-micron centers thus allowing for about 2.0×10^5 bits to be recorded on any given track. A section

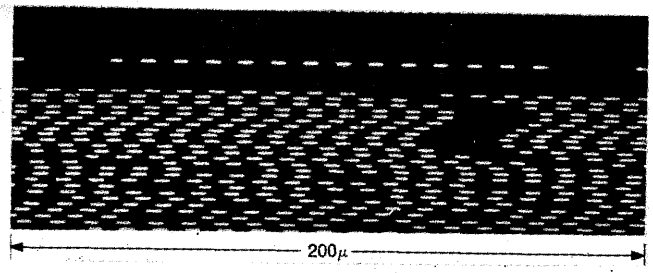


FIGURE 11

of typical laser recording, greatly enlarged, is shown in Figure 11.

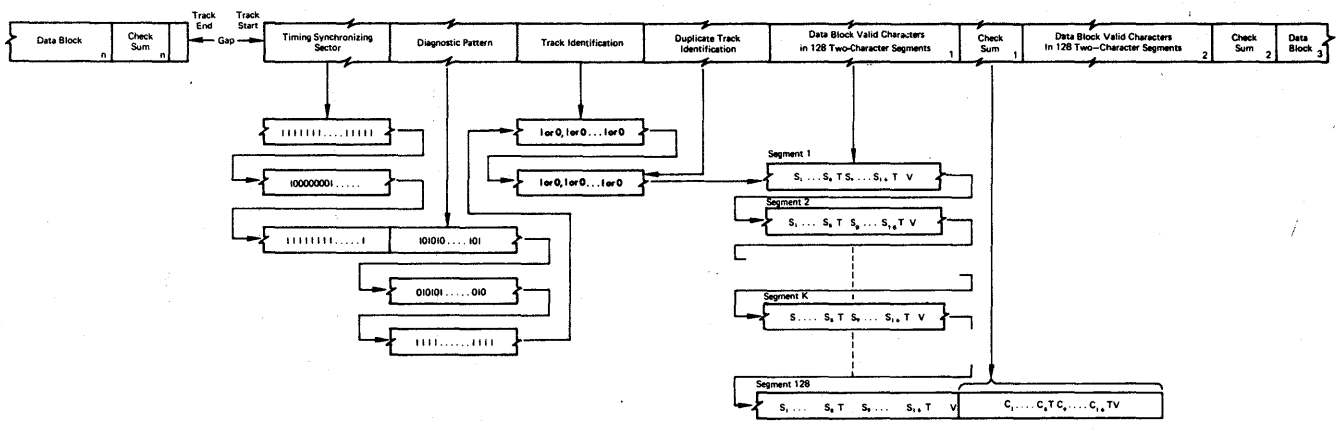
A prime consideration associated with such a recording technique is the format or organization of the data so recorded. A format control portion of the Recorder Control Subsystem controls the format of the data as they are recorded.

Figure 12 illustrates the track-oriented recording organization employed to combine all auxiliary recording signals, such as time-synchronizing pulses, diagnostic patterns, and track identification with the data to be recorded from the computer data channel. The following paragraphs describe the various areas shown in Figure 12 for a typical data track.

Timing synchronizing sector

The timing synchronizing sector will be recorded with two alternating patterns; namely, all ones and character-separated ones. The initial all ones are employed to accelerate the synchronism of the read clock with a voltage-controlled clock (VCC) by performing repeated comparisons to

FIGURE 12



accurately measure their time occurrence difference, thus generating a fine control of the VCC period. The character-oriented clocks will be compared with a counted-down clock derived from the voltage-controlled oscillator clock to establish a synchronism within longer intervals and generate a secondary feedback quantity to obtain further VCC period control adjustments, if necessary.

Diagnostic pattern

After timing synchronizing patterns are recorded, a diagnostic pattern will be recorded. This diagnostic pattern will be employed to determine if the retrieval logic structure is performing correctly (i.e., data retrieval, holding, verification, data stripping, and transmission are exercised by the diagnostic pattern to search for any probable logic or circuit anomalies).

Track identification

A track identification number will be recorded to aid in determining the location of a selected track during data retrieval. During recording, if frequent errors occur, remedial action will be initiated which will record an identical track identification pattern in the adjacent track and repeat the core memory block transfer into the newly identified track. During reading, the search for a track position is greatly aided by the track-identification patterns which will avoid ambiguous track determination.

Data block

The data block will be a block of information which contains the actual data to be recorded from magnetic tape. This block will be of length equal to 128 16-bit verified segments (i.e., if a 16-bit segment verify position indicates an error within the segment occurred during recording, this segment is considered invalid or nonexistent).

Check sum

The check sum locations contain an algebraic sum derived by a process of repeated additions of the values of the recorded valid segments. The algebraic check sum is a 16-bit segment which will be transmitted to the computer only during diagnostic modes. During reading, the retrieved check sum will be compared with a calculated check sum.

Other formats

It can be seen that a data track may be divided into separate sectors with capability of acquisition of track at each sector-start point, rather than for an entire track. Many such modifications may be implemented, depending on record size and format of input data.

PI strip pack

The flexible mylar high-density recording medium strip, which is 4.75 inches wide by 31.25 inches long by 0.010 inch thick, is contained in its strip pack, which is 32 inches long by 5 inches wide by 0.25 inch thick (Ref. Figure 13). The Strip Pack contains edge index slots which prevent the data surface of the recording medium from contacting the Strip Pack's protective surface. Protective caps seal each end of the Strip Pack. The recording medium may be inserted from either end.

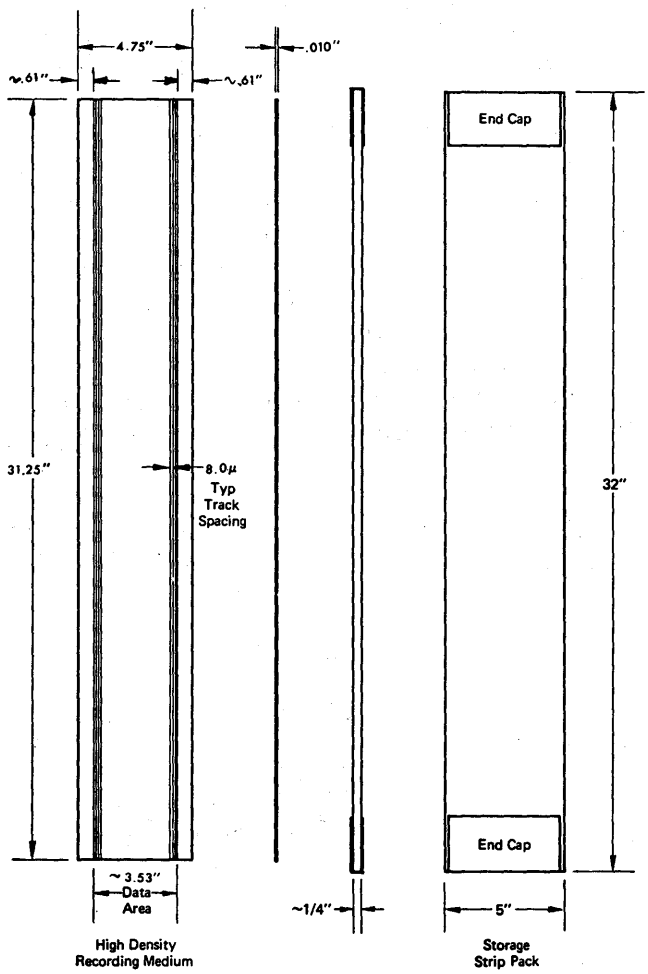


FIGURE 13

System capabilities

Loading

Since the record strips are demountable, any number of record strips can be used to compose a mass data file. To access a block of data in such a file, the PI Strip Pack containing the record strip that holds the required data must be identified. Typically, this action is performed by the data-processing system of which the Laser Recording Unit is a peripheral device: the action is initiated by an input request, which is followed by computer lookup in an index file and then a printout or display for an operator. The operator then selects the requested strip pack from file and places it on the Laser Recording Unit in readiness for loading.

The Laser Recording Unit automatically loads the record strip onto the drum surface and confirms that the requested record strip is correctly loaded. Next, a mechanical carriage-positioning unit, operating under servo control, translates the record/reproduce head to the approximate track address requested by the computer. Finally, an optical tracking servo selects the exact track desired; when the start position for the requested data record is reached, data transfer occurs and continues at the 500 kilobyte rate (except for short gap times) over all sequential tracks containing the requested data.

On the basis of evidence from experiments involving repeated loading and unloading of record strips on a breadboard model of the drum mechanism, it is expected that a particular record strip repeatedly accessed and loaded will return to the same position on the drum periphery within .2 Mils or approximately 50 Microns, maximum. This is well within the tracking range of the galvanometer servo.

The strip-loading process requires about 15 seconds after the operator places the requested strip pack in the load slide of the Laser Recording Unit. However, after a selected record strip is loaded onto the drum, access to each additional block of data on the same strip can take place in less than 400 milliseconds.

Timing

Access motion time

Access motion time is the time required to position the track access mechanism on the LRU to the track containing a specified record on an already mounted and rotating record strip.

In the LRU, the record/read head is positioned to the selected track by a carriage position servo. This unit uses a linear mechanical carriage which is supported by precision ball bushings and driven from a servo motor, through a precision steel belt. A linear incremental encoder plate provides position inputs. For this mechanism, the maximum carriage-positioning time to move from track 1 to track 11,200 on the record strip is less than 400 milliseconds. The minimum access time to move to the adjacent sequential track is less than two milliseconds, and the operation can be accomplished during the gap time at the end of each track on the record strip.

Head selection time

The LRU has a single record/read head mounted on an optical tracking servo. Upon arrival of the mechanical positioner to within two mils of the desired track location, the galvanometer-controlling optical tracking servo will access the track at the center of the field to the objective lens. This field is of the order of 100-200 microns and will cover 10 to 25 tracks, depending on recording density.

At N points around each track the track address number will be recorded, and an optical tachometer on the drum will indicate these locations or sectors. Within 1/N revolution, the track address will be determined.

Due to possible positioning error in the carriage mechanism, the track initially accessed by the optical servo may be one or two tracks distant from the track actually addressed. If the accessed track address is wrong, the optical servo then slews to the correct track; again a wait equivalent to 1/N of a drum revolution may be required to confirm the address. If the specified data were passed in the process, a rotational delay ensues. The galvanometer has a positioning accuracy of one micron and requires about two milliseconds slew time to reach the new track.

Rotational delay

Rotational delay is the time required after the proper track is reached for the correct data to rotate to the read/write head for the start of data transfer.

Assume the LRU device described above has just entered one sector of the specified track and determined that the track address was correct. If the sector contains the desired data, data may be read out immediately.

If the device has entered the wrong sector, the addressed data will be in one of the other sectors of the track, since the machine is on the correct track. Thus, a maximum rotational delay can occur equivalent to (N-1)/N percent of a drum revolution. With N=8 and a nominal revolution time of 43.5 milliseconds, there is a total possible maximum lapse of 51 milliseconds for head selection and rotational delay.

File updating

Since the Laser Recording Unit is a permanent recording device, with nondestructive read, data once written cannot be erased and rewritten as in a magnetic memory. For a large direct access file one convenient organization is that the file master index in key number order be stored on disc pack units under control of the main processor. When it is desired to delete or modify a record in the mass store, the record data are pulled out to the core memory in the recorder control unit, the data modification made, and the entire record rewritten at a new vacant location on the LRU record medium. The new track address is then placed in the disc-mounted index in place of the obsolete address, and this data item will be thereafter accessed at the new location in the mass store.

Where a data record has continuing transactions, which do not obsolete the remainder of the record, at the time of initially placing this record into the mass store, blank spaces may be left, and a local address stored in the disc-mounted index. Then when a transaction is to be added to a mass store record, the address is read from disc, incremented by one and returned, and the mass file track and sector address read, access obtained, local address used to count down the record, and the new transaction written. This may be repeated until the local address regions initially allocated to this record are all filled. Then, either a transfer address to a new mass store track may be utilized for continuation of the record, or the procedure of completely transferring the record to a new location in the mass file may be used.

In a system which contains more than one LRU, the mass file may from time to time have all obsolete material deleted, by passing through the current disc index in key order, and rewriting only those portions of the original mass file which are currently accessible. Also, a special mark or code can be placed on any data record in the mass store to cause the readout system to ignore this data, in a manner similar to that used for read-while-write data validation in which any write error causes the data to be marked as invalid, and the recording repeated.

REFERENCES

1 C H BECKER
UNICON—Coherent light data processing
Instrument Society of America 21st Annual ISA Conference October 1966 Section 16 13-1-66 pp 1-12

2 J F READY
Effects due to absorption of laser radiation
Journal of Applied Physics Vol 36 No 2 February 1965

3 C H BECKER
US Patents 3,314,073; 3,314,074; 3,314,075

4 O S HEAVENS
Measurement of optical constants of thin films
p 193
J T COX G HASS
Antireflection coatings for optical and infrared optical materials
p 239
Physics of Thin Films Vol 2 1964 Academic Press

A random access terabit magnetic memory

by S. DAMRON, J. LUCAS,* J. MILLER, E. SALBU and M. WILDMANN

Ampex Corporation
Redwood City, California

INTRODUCTION

The storage of large amounts of information will, for the foreseeable future, consist of storage on a surface. At present, magnetic surfaces are most widely used, although silver halide surfaces are also finding some limited application where the non-erasability is not a serious limitation. Other media now in the laboratory will undoubtedly find practical use. The problem facing the large memory designer is one of choosing a suitable medium and determining a way of breaking up this surface to make it accessible.

If it is desired to have a memory that can be updated, an erasable surface is imperative, and we are therefore limited, at present to magnetic surfaces. With a magnetic surface, storage densities in the order of one million bits per square inch are feasible at present or will be available in the near future. With this packing density, storage of 10^{12} bits will require one million square inches. (This is approximately one-sixth of an acre, or the size of an average suburban building lot.)

The choice of a suitable medium, with the imposed constraint of erasability, was relatively easy. How to break up this surface for ease of access is a somewhat more difficult problem. However, the storage of information for computers is really not different from the storage of any other information—a problem which has confronted mankind since the invention of writing. Throughout history three principal methods of providing surfaces for writing have been used: solid surfaces in the stone age and for monuments; flexible surfaces in the form of sheets as in most books.

In computers, these same three forms can also be found. Disc and drum files use solid surfaces. Their advantage lies in comparatively rapid access and in comparatively fixed position of the stored bits. However, their volumetric efficiency, i.e., the number of bits per cubic inch, is low. A reel of magnetic tape is the modern equivalent of ancient scrolls; access is slower, the medium stretches, but the volumetric efficiency is the best of any digital storage unit. Magnetic strip files can be compared to books or file folders. Both access and volumetric efficiency are between those of discs and reels of tapes. Since storage on a flexible medium has the advantage of high volumetric efficiency, and since erasability and reusability are possible with iron-oxide coated polyester film, this medium seems to be the obvious choice for a large memory.

Magnetic tape on reels has the disadvantage that a relatively long time is required to move from an address at one end of a tape to an address at the other end. However, if sufficiently high tape speeds and packing densities can be achieved, this disadvantage is less severe and tapes therefore become a very attractive storage medium.

Laboratory developments completed prior to the initiation of the program described here demonstrated that tape speeds well in excess of 1000 inches per second (ips) and packing densities of one million bits per square inch, with high data reliability, were feasible. Using these developments as a basis, a large memory system could therefore be designed. A prototype "small" system with a total storage capacity of 10^{11} bits has

* Department of Defense

been built and tested. The following is a description of such a system and its major components.

TBM system description*

The TBM (terabit memory, i.e., 10^{12} bit memory) random access memory uses magnetic tape as a basic storage medium. Random access is provided by using tape search speeds of 1000 ips (compared to approximately 300 ips used on conventional transports) and by using packing densities of 700,000 bits per square inch (compared to approximately 14,000 bits/in.^{2} for standard computer tape transports). Data recording is done in the transverse mode (to the direction of tape motion) using rotating heads, the technique used for video recording. A redundant recording scheme permits the achievement of error rates of two errors in 10^{10} information bits. The salient features of the tape transport, permitting this high search speed, and of the recording mode and associated data channels, permitting this high packing density, will be considered in detail following the system description.

A block diagram of a typical TBM memory system is given in Figure 1. The memory system is composed of four major building blocks: 1) the tape transport, 2) the transport controller, 3) the TBM controller, and 4) the digital signal channels. In addition to these main components, buffers between the memory and the central processor are usually required. Furthermore, a switching matrix has to be provided between the transport controller and the transports, to permit operation of each transport from any one of the controllers. Similarly, a switching matrix between the transports and the signal channels has to be provided to permit the signal channels to be connected to each of the transports.

The tape transport contains the components necessary to move the tape (reels, capstans, and vacuum chambers), address read and write heads, digital data erase head, auxiliary heads for the control of tape, and rotary digital read and write heads. Vacuum chamber tape position sensor and signal preamplifiers are also provided. Only those functions which cannot be provided in the transport controller or in the digital signal channels are incorporated in the transport. The transport controller contains all the circuitry necessary for control of a tape transport, i.e., the reel servo, the capstan servo, the rotary head servo, as well as

* Trademark, Ampex Corporation

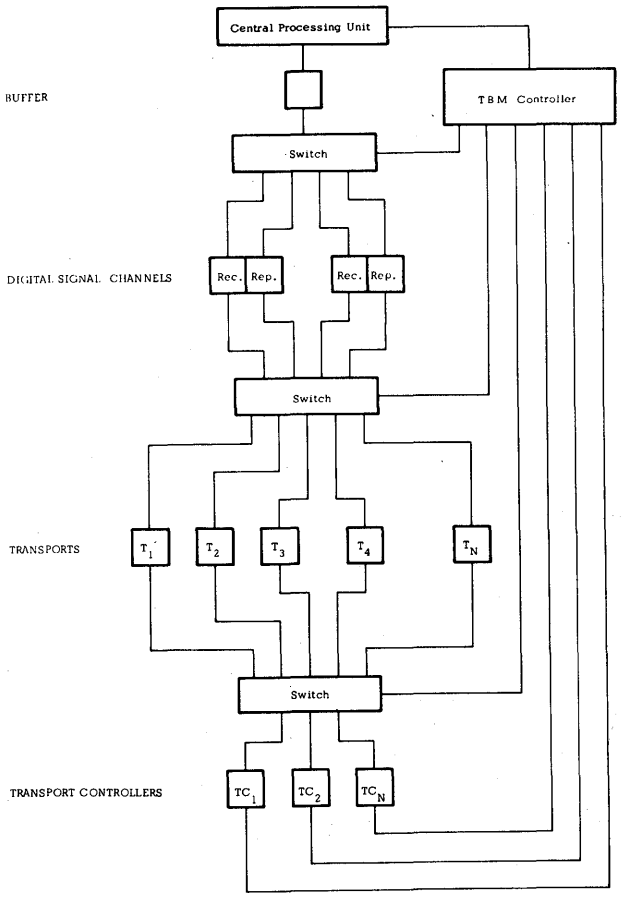


FIGURE 1—Block diagram of TBM memory system

read-write circuitry and the main memory controller interface. The transport controller can be used to operate any one of the transports in the system.

The memory controller is a small general-purpose digital computer. Its function is to provide the command interface between a central processing unit, which is served by the digital memory described, and the various memory building blocks. Because it is a general purpose digital computer, it can be programmed to fill any number of functions specifically designed for any particular application.

The digital data channel accepts the digital input, which is then FM modulated and written in that form on the tape. In reading, the FM data is demodulated and reconstructed into digital format. The read and write channels are independent. This permits simultaneous reading and writing on separate transports. Data erase is an independent function, which does not require the data

channels. The data rate of each channel is 6 megabits per second.

In the transverse recording mode used, the data is recorded across the width of the tape as shown in Figure 2. In addition to the data, information is also recorded longitudinally, i.e., along the direction of motion of the tape. Three longitudinal channels are provided: (1) to record addresses to identify data blocks; (2) a control track containing one pulse for each revolution of the rotating head. (This is primarily needed for operation of the rotating head.); and (3) a tally track to record auxiliary information, such as number of accesses to each data block, illegal addresses or addresses in which tape defects occurred, or to identify data blocks which cannot be erased. The length of each data block is determined only by the length of the longitudinal address on the address track. Data blocks are used which contain one million bits, and use up 0.90 in. of the two inch tape. Different block lengths, down to one-quarter million bits and up to a full reel of tape, may be used without requiring any hardware change. Each data block can be individually erased and recorded. Update consists of first erasing and then re-recording as required. During high speed search, only the control, tally, and address tracks are read and the rotating transverse head is not in contact with the tape. Tape wear on the data channels is eliminated during the search operation.

The system just described is completely modular; i.e., it is made up of a number of components which can be combined in many different ways. This modularity of the system has the following advantages:

Versatility

Different combinations of the basic main building blocks can provide different system characteristics. Total system capacity can be greatly expanded with relatively little increase in system cost by adding more transports. The number of requests that can be answered per unit time by a large memory is usually one of the most important characteristics of such a system. By using more than one transport controller, several transports can be searched at the same time. This simultaneous search operation allows a typical system with five controllers to answer 1200 requests per hour, giving an average time per request of 3 seconds.

The independence of the read and write functions in the signal channel permits independent read and write. For large systems, two data channels are usually required for housekeeping, such as pre-recording of address and control tracks on tapes, rewriting of used tapes, or aligning of new heads without removing the entire system from operation. These channels also provide redundancy.

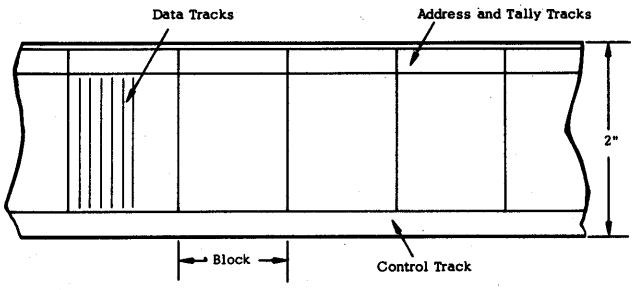


FIGURE 2—Tape format

Reliability

The modular construction of the system results in complete redundancy of all the major building blocks. In an electromechanical system it is the mechanical components which usually exhibit the lowest mean time between failures. This system has only one mechanical component—the tape transport. Because the tape transports are operated in parallel, and because complete switching arrangement is provided both between the transport controllers and the transports, and between the signal channels and the transports, failure of a transport does not result in system failure. If a spare transport is used, the downtime resulting from a transport failure need only be the time required to remove the tape and rotary head from the failed transport and to mount them on the spare. This is an operation requiring, at most, five minutes.

Redundancy is also provided in the digital signal recorded on the tape. Each bit is recorded twice with a separation of approximately three-fourths inch between each recording. The major cause of data errors (tape defects) is thus eliminated because defects are small and rarely have the required spatial relation to cause simultaneous dropouts. It is because of this redundancy that the demonstrated error rate of less than two errors in 10^{10} data bits can be obtained.

Each transport can accommodate up to 45,000 inches of tape, or about 5×10^{10} bits of information. The system described here has the capability of storing anywhere from 5×10^{10} bits for a single transport system to 3×10^{12} bits for a 64 transport system. Larger systems could, of course, be built, but problems such as very large switching matrices and long cables might make such systems unwieldy. To date, a "small" prototype of a system containing two transports, one signal channel, one transport controller, and one TBM controller has been built and tested. A small system such as this (shown in Figure 3) has a total capacity of 10^{11} bits. This is the equivalent to the capacity of approximately 1000 reels of conventional one-half inch computer tape, or of thirty IBM 2321 Data Cells.

Component description

The two components novel to this system are the transport and the data channel. Part of the data channel is contained in the rotary head which is mounted on each transport. The transport and the data channels will now be described in detail.

Transport

The functions of the tape transport are, primarily, to move the tape at high speed for searching, to accelerate and decelerate the tape, and to move the tape accurately during data transfer. A diagram of the transport is shown in Figure 4.

The configuration chosen for the transport has

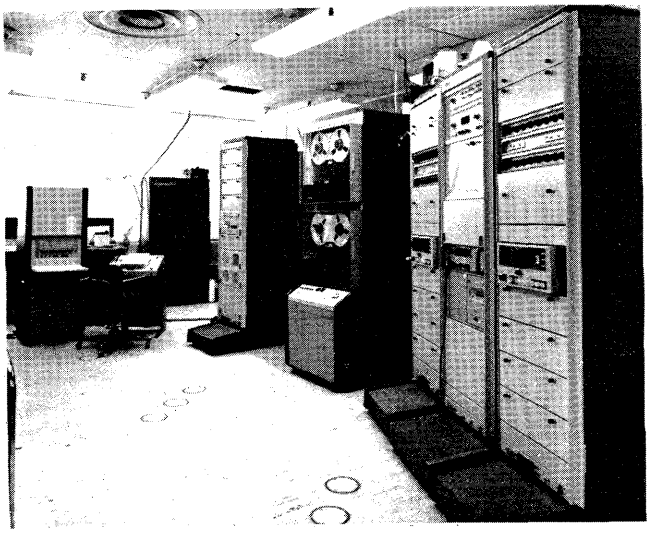


FIGURE 3—Photograph of 10^{11} bit memory system

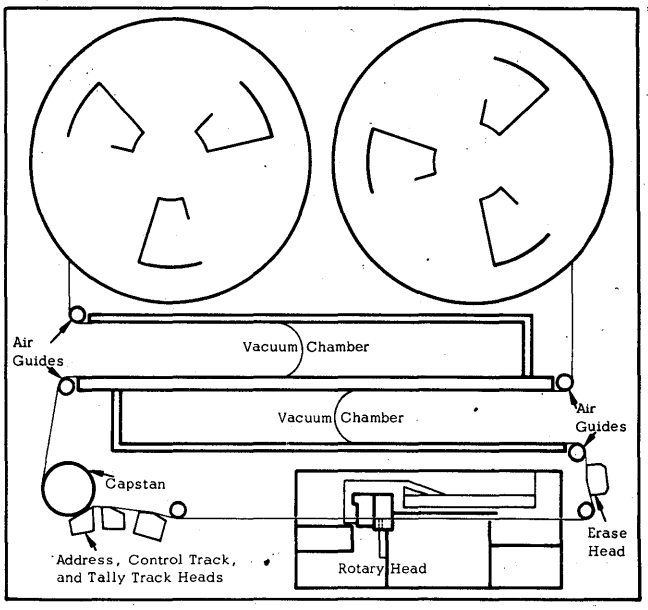


FIGURE 4—Tape transport layout

the following advantages: With the vacuum chambers arranged as shown, a small transport package results, and two transports may be mounted in one rack. The tape path is such that all guide and capstan contact is on the Mylar side of the tape only. Only the longitudinal control track, address head, and rotary head contact the oxide side of the tape. Thus, no damage of the data surfaces can occur from contact with the guides or vacuum chambers. Furthermore, all heads except the rotary head are air lubricated so that, the only contact with the oxide occurs at the rotary head during data transfer. All guides are air lubricated to provide the low friction necessary for successful high speed operation.

At the low speed required for data transfer, the tape speed is controlled by the capstan (friction between tape and capstan is enhanced by supplying vacuum to the capstan). Control at high speed with the capstan is not possible because of entrapment of air between the capstan and the tape. To permit operation at speeds at which such air entrapment would make capstan control impossible, the tape is moved by allowing a difference in pressure between the two vacuum chambers to occur. With this difference in pressure, the tape essentially "flows" from the high pressure to the low pressure chamber. Speed control in this mode is provided by using the control track as a speed reference and by servoing the difference in pressure between the two vacuum chambers to obtain a constant speed.

Two transports are mounted in one rack with room for switches or auxiliary air pressure and vacuum suplies mounted at the bottom. A photograph of the two transport rack is shown in Figure 5.

Data channels

The transverse recording technique was developed for video application in 1956. This method of recording has several advantages over conventional fixed-head recording techniques. It allows high relative head-to-tape velocity while tape speed is low. Thus, accurate speed control to provide the necessary timebase accuracy is easily achieved. Because of the high head-to-tape velocities, good signal levels, and therefore high signal-to-noise ratios can be obtained with narrow tracks which are compared in width to those used for disc recording. This results in high packing densities. Furthermore, high frequencies (up to 20 MHz with the present state of the art) can be recorded and reproduced with good signal-to-noise ratios.

A picture of a rotating head in contact with the tape is shown in Figure 6. In addition to the rotating head with the eight transducers, a tape with the transverse data, longitudinal address, control and tally tracks is shown. The detail of the data tracks, as recorded on tape is shown in Figure 7. The tape itself extends slightly more than 90 degrees of the circumference of the rotating head drum. With eight heads, separated by 45 degrees each, a minimum of two heads are in contact with the tape at all times. By providing the same signal to the two heads which are in contact with the tape simultaneously, all the information is recorded twice. This is also shown in Figure 7.

The data channel takes the digital data and derives a frequency modulated signal from this with a "zero" corresponding to 5.25 MHz and a

FIGURE 5—Photograph of two-transport rack

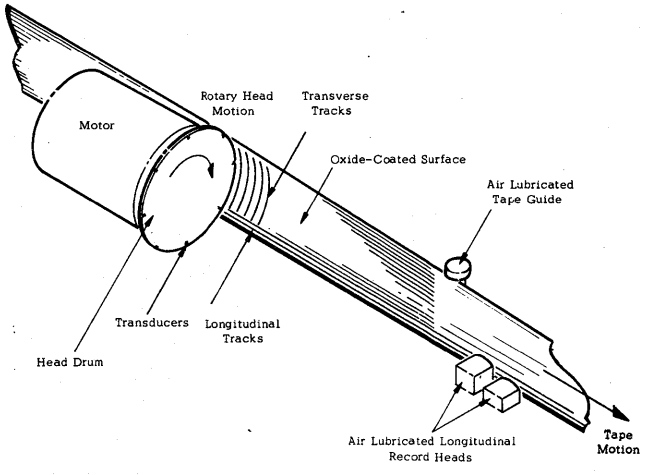


FIGURE 6—Transverse scan rotating head and tape

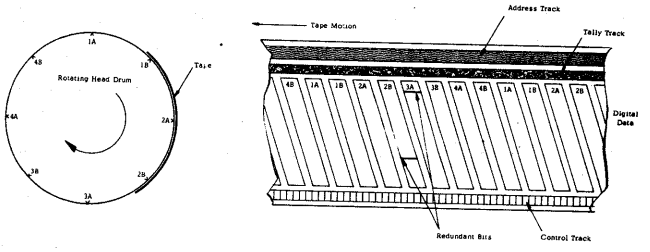


FIGURE 7—Tape format detail

"one" corresponding to 6.75 MHz. In addition to these two frequencies, a 500 KHz pilot signal is multiplexed onto the data and recorded together with the data. Reproduction of the data is done by taking the signal individually from each of the two redundant heads and using the pilot reproduced from tape to generate an error signal, when compared to the system clock. This error signal is used as input to the variable delay line, which accurately corrects the timebase of the signal. After timebase correction, the two signals are combined; dropout in one channel causes a 6 db drop in signal level, rather than a complete loss of signal.

In order to provide a continuous stream of data, rather than an intermittent one, each time one rotating head leaves the edge of the tape and the next one engages the other edge of the tape, there is a small amount of overlap between the signal recorded by each head. In reproducing the signal, the signal level of the head leaving the tape is slowly decreased while the signal level of the head entering the tape is slowly increased. The two signals, after timebase correction, can then again be combined, providing a continuous signal without switching transients. A detailed description of the signal system is beyond the scope of this paper. A description will be found, however, in reference 1*.

The signal system just described is obviously the heart of the memory system. It permits recording of data with a packing density of 1.4×10^6 bits/in.[2] The data rate is 6 megabits/sec. Increasing this data rate to 10 or, even 20 megabits/sec is feasible. Higher packing densities of up to 2 million bits/in.[2] also appear to be possible.

Data accuracy measurements have been made by recording a pseudo-random pattern on the tape, reproducing the data, and comparing this data to the recorded data on a bit-by-bit basis. After eliminating all permanent errors which were repeatable at one location of the tape, the resulting data rate was 1.8 errors in 10^{10} bits of data. A total of slightly less than 10^{12} bits was recorded and reproduced to test the data accuracy of the signal system.

Without the dual channel feature, the error rate is in the order of one error per 10^6 bits of data. The amount of tape that has to be considered as useless because of dropouts amounts to about 0.1% of the total tape tested. This is accomplished by identifying certain addresses illegal. The system performance is not degraded by removal of such a small percentage of storage area.

*E. Kietz, "Transient-Free and Time-Stable Signal Reproduction from Rotating Head Recorders." *Record*, IEEE Tech. Grp. on Space Elect. and Telemetry, 1963.

System specifications

Because the system is modular it is possible to generate a large number of systems specifications to meet a variety of applications. The following specifications were prepared to meet a typical 10^{12} bit application:

Storage capacity	10^{12} bits
Number of transports	36
Tape per transport	30,000 inches
Number of transport controllers	5
Requests per hour	1,200
Average time per request	3 sec
Data transfer rate	6×10^6 bits/sec
Block length	10^6 bits

Two other considerations which enter into the useful measure of a system are tape life and rotary head life. Address and erase heads, being air lubricated, do not wear out. Because of the high data rate, the transfer of each typical data block of one million bits requires 1/6 second contact between the head and the tape. Of the average time per request of 3 seconds, the head is in contact with the tape only about 1/18 of the total time. With a large number of transports, the actual contact time per head is even further reduced, so that in a 36 transport system, each head would, on the average, be in contact only $\frac{1}{3 \times 6 \times 36}$ or 1/648 of the operating time of the system if one data channel is used. Head life of 100 hours in contact can reliably be obtained, so that system head life of over 64,800 hours can be expected.

Data deterioration due to wear of the tape occurs somewhat faster. Experience has shown that at least 1000 data transfer passes can reliably be obtained. Remember that no contact occurs during search. In a large system there should be no data blocks which are frequently used. Such blocks should be stored elsewhere in the system. The purpose of the tally track is to keep count of the

number of accesses on each block so that excessive use can be identified.

Interfacing

Interfacing the TBM system to a typical general-purpose computer requires two interfaces: command and control, and data. The command and control interface with the memory controller is fairly straightforward and consists of several simple commands, such as "write address," "control track," "write digital data," "read digital data," "search for address," "erase digital data," "abort last command," and "return control to local command." The data interface can be handled by providing for the output of the memory system to go directly into the central processor core memory. However, because of the large data blocks involved, sufficient core memory capacity is usually not available and provision of a buffer between the memory and the central processor is therefore required. A small disc or drum is a suitable buffer. In order to avoid the difficulties involved in matching two mechanical devices, such as a tape transport and a drum or disc, a small core-type buffer between the two is necessary. Many other suitable schemes can be visualized.

CONCLUSION

The TBM memory system described, provides a greater capacity than any memory presently available in the marketplace and is based on existing, well developed technology, some of which has been used for ten years. Applications for this system will be found in many computer installations which have large tape libraries because of cost savings, and for large memory stores which should operate on-line.

Dynamic recovery techniqes guarantee system reliability

by D. P. GUSTLIN and D. D. PRENTICE

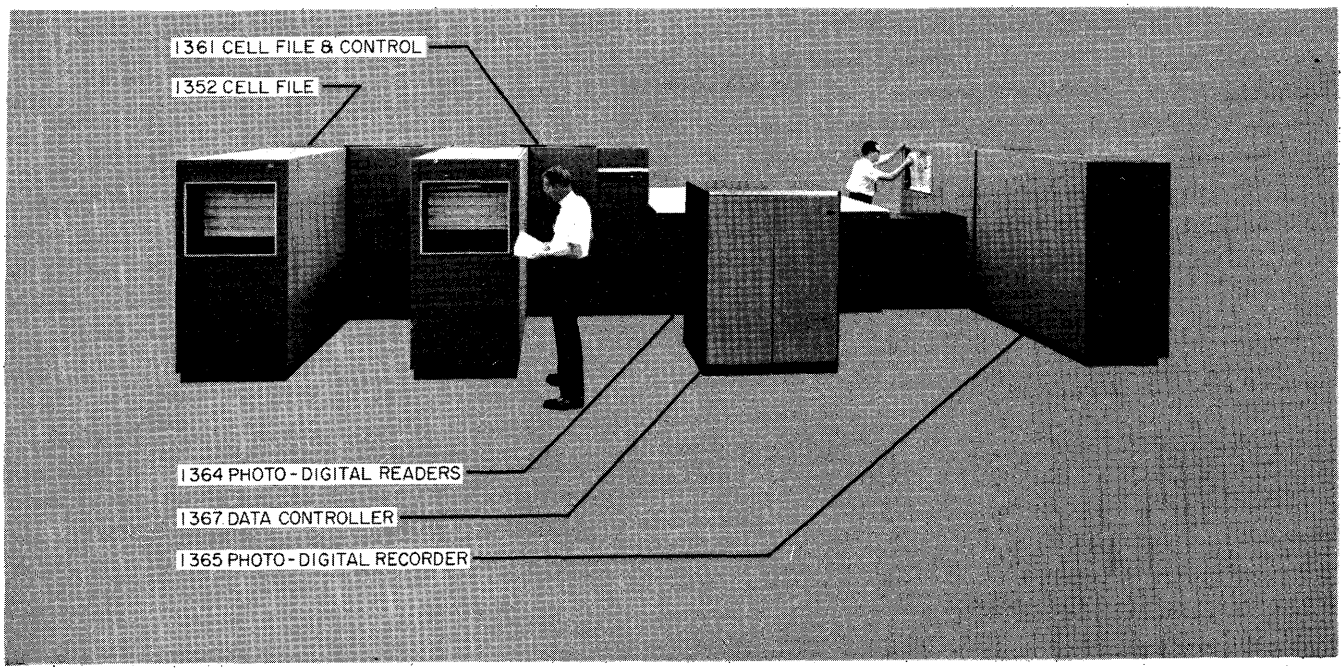
International Business Machines Corporation
San Jose, California

INTRODUCTION

The IBM 1360 Photo-Digital Storage System, shown in Figure 1, is capable in its present form of storing up to 2.38×10^{12} data bits on-line to the main processor. Currently, there are two systems installed, both being less than the maximum configuration. One system is located at the AEC Lawrence Radiation Laboratory, Livermore, California (1.02×10^{12} bit capacity), and the other at the AEC Lawrence Radiation Laboratory, Berkeley, California (3.4×10^{11} bit capacity). In this paper, any specific consideration will refer to the Livermore system, because its larger size tends to better illustrate system complexity and related recovery procedures.

Immense storage capacity in the IBM 1360 is achieved by high-density recording in two dimensions on silver-halide photographic-film chips (4.7×10^6 bits per chip) and by the storage of 32 chips in 1.1 by 1.6 by 3.0 inch containers, or cells (1.51×10^8 bits per cell), which reside remote from the reading device. Figure 2 shows a cell containing chips. The cells are brought to the reader under automatic control in response to main processor commands. Writing is accomplished with an electron beam and reading with a

FIGURE 1—IBM 1360 photo-digital storage system



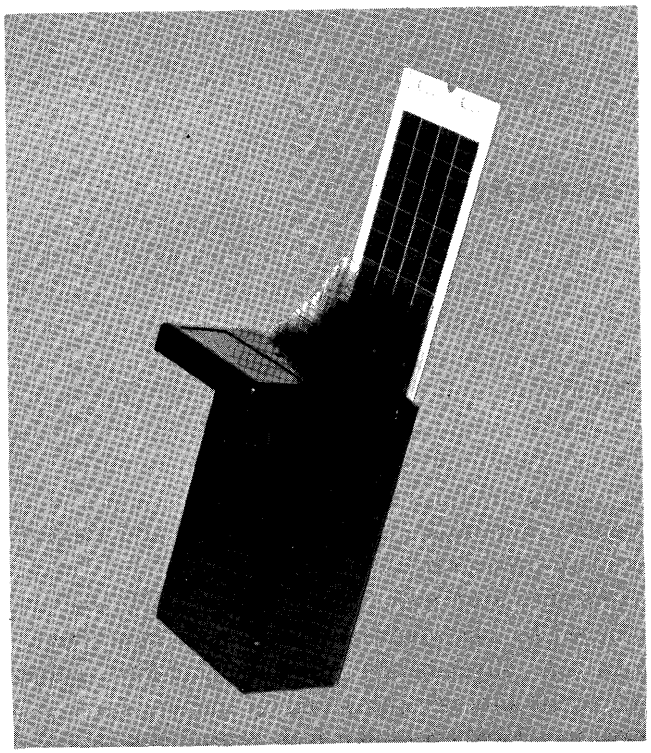


FIGURE 2—Cell and chips. Note the arrangement of data frames on the chip

cathode ray flying-spot scanner. Because of the differing methods for writing and reading, the write and read devices are separate and basically dissimilar physical entities. The residence for cells is called a file. A file is a five-tray array containing 2250 cells. See Figure 3. The write device and the read devices (LRL-Livermore has two), or terminals as they are referred to in the 1360, are connected to the files (LRL-Livermore has three) by a pneumatic transport system.

Overlapped processing operations

The time to move cells between a file and a terminal varies from 2.5 to 4 seconds. Of course, this is long compared to data processing time. To improve effective performance, the 1360 is designed to provide simultaneous operation. For example, one file can be restoring a cell to its resident compartment, a second file can be preparing to send a cell to a terminal, and a cell can be moving in the main pneumatics between a third file and some terminal, all at the same time. Concurrently, one reader can be actually reading and simultaneously readying a second cell and, overlapped with these operations, the other reader can be exchanging

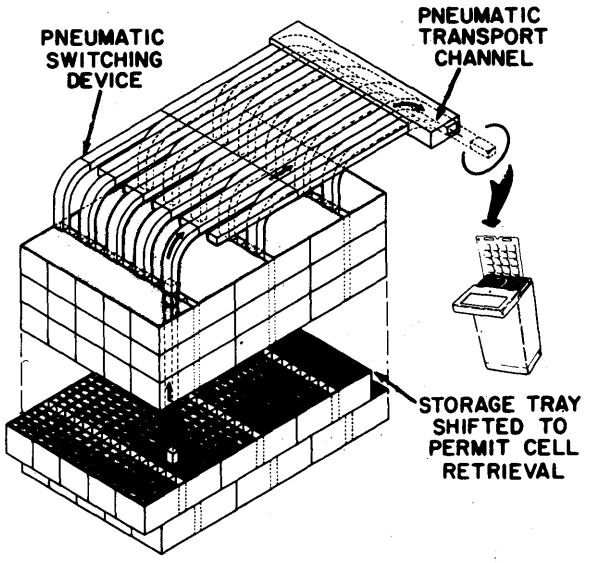


FIGURE 3—The file used for storing cells

chips between the completion of one job and the start of another. Further, the recorder may itself be performing several overlapped functions, such as recording of a chip, on-line automatic chip developing, and extraction (or return) of chips from (or to) cells.

This considerable complexity of overlapped functions and the logical problems deriving from the combinatorial possibilities of operations at any given instant made it imperative to find a sophisticated method of device control. The result was a decision to control the hardware with a stored program processor, a slightly modified IBM 1801 which is the main memory and CPU of the IBM 1800 Data Acquisition and Control System. Having decided to use a control processor, it seemed a reasonable extension to have this processor assume many functions normally performed by circuitry. The programs then control the hardware at a very detailed level, or at what has been termed the solenoid/sensor level. This simply means that the program initiates an action by activating or deactivating a particular single solenoid and determines hardware conditions by individually sensing the state of microswitches or photosensors. The processor used for these actions is referred to as the control processor, and the processor that uses the IBM 1360 as a storage subsystem is called the host computer.

Advantages and objectives

With respect to diagnostics/automatic recovery,

the use of a control processor provides a fundamental advantage of making detailed information available in the event of a failure. The programmed embodiment of the complex logic of the many operations makes it possible to know the system environment in which the specific failure occurred. The information available, together with the intelligence of the processor, makes it possible to perform specific yet flexible analysis and recovery beyond what has previously been feasible.

With these benefits accuring from a control processor which has both intelligence and memory (a 16K word core and a 512K word disc), certain objectives become realizable in the area of diagnostics/automatic recovery, referred to subsequently as D/AR. The primary objectives are the precise identification of a failure (intermittent or solid), the automatic recovery from intermittent failures to significantly increase system availability, and the provision of a complete failure history and sophisticated set of programs to aid the field engineer to analyze and solve problem situations and establish the valid operation of equipment after repair without requiring program execution time or space in the host computer.

In order to elaborate on the accomplishment of these objectives, it seems appropriate first to describe the general structure of the control programs and the related diagnostic procedures.

Organization of operations

Control design is founded on a conceptual organization of the equipment. In describing a complex mechanical/electronic device like the 1360, it becomes apparent that there is a limited set of procedures which are termed operations. Each operation has the characteristic of being invariant as to actions and their sequence, though in some cases the action parameters such as time may be variable. For example, the length of time that air is applied to move a cell out of a file is dependent on the vertical distance the cell must travel in the file. The subsystem functions which are available to the host computer consist of a set of sequences, each sequence being a particular progression of operations. To be initiated, an operation must encounter suitable hardware and system conditions. As examples, a hardware condition would be the proper position of the load mechanism to accomplish a mechanical cell transfer, while a system condition would be the ability to seize a pneumatics path before initiating hardware action that would block that path.

The operations represent the detail control and detail knowledge level of the control programs. The linkage points, in stringing operations to make sequences, represent the environmental control and environmental knowledge level of the control programs.

Sets of operations and logic combining them into sequences have been termed control strings. These strings are totally re-entrant codes composed of symbolic commands. There is an interpretive program called the string processor which examines and actually executes the symbolic control commands. A number of transactions is possible in the system at a given time, each transaction representing a cell-oriented request for action by the host. The string processor defines and maintains a program pointer for each transaction to indicate its current location in a control string.

The background is now established to examine the steps involved in D/AR and the means of accomplishing them.

D/AR structure

There are six clearly defined elements in the D/AR procedure:

1. Manifestation—detecting an abnormal condition.
2. System suspension—arresting the remainder of the system under exact control to examine the particular abnormality.
3. Analysis—investigating the abnormality to determine its nature, location, and degree of seriousness.
4. Automatic recovery—re-establishing proper functioning, if possible.
5. Failure history—recording the detailed data on all failures, whether or not recovery was possible, for use by a field engineer.
6. System restoration—establishing the new status of the system as a result of a failure and the D/AR.

These steps are elaborated on below. A special class of recovery, read recovery, which does not involve the steps above is also described.

Manifestation

Basically, control of an operation consists of activate-time-sense, this being more or less com-

plex, depending on the particular operation. Once the control string has determined that proper hardware and system conditions exist, the operation is initiated. An action is started, such as engaging a clutch, lifting a detent, or opening a valve. Most conditions are determined by program polling, rather than by an interrupt at action completion. There are usually two times associated with an action. The first is normal time or the interval after which the result of the action can be expected. This time is used to avoid taking processor time to look immediately for the end of an action that normally takes a finite time for completion. The second time, called tolerance, specifies the limit for normal completion. The sense is, of course, the determination that the appropriate sensors reflect a successful completion of the action. Both times are measured via system clocks.

Failure manifestation is defined, then, as failure to successfully sense correct action completion within the tolerance time. If such a failure occurs, the string processor notes this fact by setting a diagnostics request bit in the proper transaction pointer, and the next step in the procedure is entered.

System suspension

An abnormality must be handled in such a manner that the valid control of the rest of the system is not compromised. This is particularly important in the 1360 which has many independent, overlapped functions. The method of bringing the system to a controlled suspension to devote resources to D/AR is based on the concept of operation and interpretive control by the string processor. Since an operation is discrete and invariant, it represents a necessary unit; that is, once initiated, it must be carried to unhampered completion in order to be valid. The control string, then, upon initiating an operation, sets a pointer-oriented bit called "stop-suppress," which is an indication to the string processor that it is processing a noninterruptable operation. On completion of the operation, the stop-suppress bit is reset.

When a diagnostic request is made by a pointer with an abnormal condition, the string processor thereafter allows processing only on pointers with a stop-suppress bit set, continuing in this mode until all such bits are cleared. The arrival at this state means that the entire system is in a legitimate, suspended state, and the next step is begun.

Analysis

Analysis is, of course, the true diagnostic element. The diagnostic master program is called and executed. Basic functions of the diagnostic master are to save the failure data and system information, perform gross checks such as testing the relevant component power supply, and determine which specific diagnostic program is required and call it from the disk.

Each specific diagnostic program is related to a particular time/sense failure and is therefore able to further analyze the possible conditions associated with the failure. The usual result is entering the next step, automatic recovery, within the diagnostic program. (The term real-time diagnostics is proper here because the 1360 system analyzes itself without the knowledge or intervention of the host computer; and, in recoverable circumstances, the host experiences only a minor time delay with no effect on the functions in process.)

Automatic recovery

The definition of automatic recovery is the restoration of the subsystem, that is, the IBM 1360, to proper operation by the subsystem itself without intervention by either the host computer or a human. It is too early to provide reliable statistics on the ratio of intermittent to solid failures, but to date a large percentage of failures has been intermittent. The detailed knowledge of the failure conditions and the resultant capability of having very specialized recovery programs greatly enhances the ability to deal with intermittent failures. In its simpler form, automatic recovery involves retry. For example the failure may have been due to an intermittent malfunction of a sensor, which then performs properly on a retry. An extended form of automatic recovery may include several retries at normal or subnormal speeds, which may clear temporary conditions causing nonserious, substandard mechanical performance.

Automatic recovery has the ability, in some circumstances, to deal successfully with solid failures. For example, recovery from a failure to obtain a proper current for the electron beam in the recorder may be achieved by program-controlled rotation of the filament turret to bring a new filament into operation.

The area of automatic recovery will be covered in more detail subsequently. It is certainly a sig-

nificant feature of the IBM 1360. Those familiar with the problems and total system disruptions attendant with malfunctions find it gratifying to observe the subsystem discern a failure, go through an automatic recovery procedure, and then resume complete and normal operation.

The variety of failure causes and recovery procedures do not permit a simple statement on recovery times. However, the considerable majority of recoverable malfunctions represent a delay of less than 20 seconds from suspension of processing to its full restoration. The host computer has been passive in the process and only experiences a time delay in completion of its commands.

Of course, the success or lack thereof by automatic recovery determines the subsequent procedure, discussed under system restoration. However, at this point we briefly examine the failure history element of the D/AR procedural steps.

Failure history

Regardless of the results of the recovery attempt, a failure history is always recorded. The detailed data and the results are formed into a record and stored on the 1360's disk. These records, elaborated on later, are useful to the field engineer in solving a solid problem by isolating the malfunction to a very narrow area. The records are also useful in a preventive sense. The field engineer can, when servicing a solid failure or when performing preventive maintenance, determine from the failure-history output a pattern indicating substandard performance. He can thus give special attention to a marginal piece of equipment and perhaps obviate a solid failure.

System restoration

Once the preceding steps in the D/AR procedure have been accomplished, the remaining concern is to reinstate the 1360 system to a status compatible with the D/AR result. Consider the action first in the two extreme forms, namely, total recovery or total system down.

In the case of total recovery, the failing mechanism is put in the desired state and processing is allowed to continue. Referring to previous discussion on the control programs, it can be seen that this is not a very involved process.

In the event of a nonrecoverable failure which requires that the 1360 system be taken down, there is a somewhat more involved process. First, the host is notified that the system is down. Then, a special program in the control processor "clears out" the system. This consists of taking chips and cells not in their resident file position and moving them to a reserved section of the file, so that the host may identify them and restore them when the system is again operational. A message is also typed on the 1360 typewriter, indicating that it is necessary to call a field engineer.

An intermediate case of a nonrecoverable failure is when a component is down but the entire 1360 system is not down. The availability of a single file, the pneumatics, and a single terminal (either a reader or the recorder) is sufficient to permit operation. For example, if a solid failure occurs in one reader, the rest of the system can operate; or, if the recorder is down, the 1360 is still fully capable of satisfying the read function. In these cases, the host is notified as to which component went down. For a reader or the recorder, a distinction is made between a data failure and a mechanical failure. A mechanical failure means that the component is immediately down and any chips and cells at that terminal are temporarily unavailable. A data error means that the read or write function is lost, but the host has the ability to handle and return chips and cells. After doing so, the host should release the component, and the 1360 will then declare it down. A message identifying the down component is typed on the 1360 typewriter to request field engineer service, but the 1360 is able to function with the remaining components.

This flexibility avoids the "for want of a nail, the shoe was lost" problem. The advantage of this is made even more significant by the fact that the field engineer in many cases is able to service the down component while the rest of the 1360 operates with the host computer.

Read recovery

As mentioned earlier, the reading of data constitutes a special class of recovery not involving the process just described. This is because reading occurs at a fast rate compared to mechanical functions and because reading is a primary, ultimate function of the system and merits priority. The motivation is to provide the host with as much good data as possible, automatically correcting data in error without requiring intervention by the host.

Data error correction is on the basis of 300 bits per line. Line reread is under automatic circuit control for an imperfect line. Further, an error detection/correction code permits a combined hardware-software procedure to correct any combination of up to five 6-bit characters in error for a given line. The time element for software participation is sufficiently small so that the system is in no way suspended.

If simple retry and code correction techniques fail, the programs go into more elaborate retry procedures with the host experiencing only minor delay.

Having now established the structure by which real-time diagnostics/automatic recovery is made possible, the techniques can be examined in more detail.

Typical D/AR application

D/AR can readily be illustrated by three types of recoveries:

1. High-speed recovery such as rereading a line of information (read recovery)
2. Medium-speed recovery such as repositioning mechanical hardware
3. Low-speed recovery such as the failure to obtain proper current from a filament

High-speed recovery

Rereading a bad data line is automatic, and a high priority is assigned to minimize the recovery time. Since the inability to read a line correctly is, in most cases, due to factors related to the chip itself rather than to machine malfunction, no history is recorded. Also, systems suspension is not required because the recovery is executed during normal system operation.

The objective of read recovery is to correct all correctable lines in any block of data in minimum time. This implies that, to correct a line, the sequence of techniques to be selected is the one which statistically yields the most frequent success for the least amount of time invested. Many techniques are used, and each is repeated a number of times before trying another. We will examine two of the several techniques used for recovering from data-read failures. They are rereading and chip repositioning.

Rereads are effective for several reasons:

1. The data detection system is designed so that, if data bit waveforms are neither clearly "1" nor clearly "0," there is about an even chance of being read as a "1" or "0" bit.
2. Electrical noise and mechanical vibrations are different for each reading attempt.
3. A problem line is approached from both of the adjacent lines, with the offset scanning beam sweeping over flaws at different angles to produce a variation in data signal. A line may be reread several hundred times before it is considered unreadable. (In the worst case this takes less than two seconds.)

The probability of success for each reread attempt to recover a line by using the same technique repeatedly decays exponentially. That is, if we do nothing more than reread the line, the chances of recovery after 10 tires are poor. So, we intersperse other techniques to vary the reading conditions.

The chip is held between two air-bearing heads which frame the reading window in the photo-digital reader. The air jets from these heads are shaped to provide an air-brush effect as well as to support the chip. Contaminants picked up by the chip, after the development processes are completed, can usually be "swept off" by the air-brush heads. Sometimes more than one sweep across the chip is required. During repositioning, the chip is moved a short distance and returned. This movement allows for another sweep of the chip and more rereads.

Having illustrated the recovery from data failures, we are ready to investigate malfunctions which require system suspension.

Medium-speed recovery

Repositioning of any hardware component requires system suspension. The time required for this action is a matter of seconds, during which four programs are involved:

1. Diagnostic call
2. Diagnostic master
3. Specific diagnostic/recovery routine
4. Diagnostic return

Diagnostic Call. The limited size of the control processor core storage requires that diagnostic programs be stored on the 1360's disk. Thus, the diagnostic call program replaces the normal operating programs with the diagnostic master. The core image of the present state of the operating

program is stored on disk for use after diagnostics have recovered from the malfunction.

Diagnostic Master. The diagnostic master acts as a clearing house for all machine malfunctions. Its major objectives are to:

1. Grossly check some common system conditions.
2. Record all sense points of the malfunctioning unit.
3. Determine which portion of the hardware is in error and bring in the appropriate diagnostic routine.
4. Record history on disk.
5. Inform the operator if a unit is removed from operation.
6. Inform the host computer if a unit is removed from operation.

Each major unit of the storage system (data controller, file, reader and recorder) has a number of sense points. Each sense point or group of sense points has an individual recovery routine designed to correct a particular malfunction in the corresponding mechanism.

The diagnostic master checks the gross condition of each major unit before control is passed to the individual recovery routine. This gross check assures that the unit is on line and has power, so that the individual routine can control and sense it.

Sense points of the malfunctioning unit are sensed and recorded in typewriter output code on the disk for history information. This information provides service personnel a sense-point check of the unit at the time of malfunction.

Diagnostic Routine. Further analysis of the malfunction is controlled by the individual diagnostic routine which has the ability to control, time and sense any mechanism in the system. Every mechanism's action is recorded by the control program. Using this information, the routines determine the desired position; sensors allow determination of present positions. The routine recovers from the malfunction if possible and, whether successful or not, returns to the diagnostic master.

Each machine malfunction is recorded on disk storage for recall by service personnel. This information is obtained from many sources. All previously recorded malfunction history is checked for the same malfunction. If one is found, then the failure number is increased by one. If none is found, then a new malfunction is added to the present history.

Operational personnel are informed, via typewriter, if any unit is removed from operation by the diagnostic master.

The host computer is informed of any unit that is removed from operational status by a change in the equipment status word. Each major unit has two bits of the status word. One indicates that the unit will accept or send data; the other indicates that mechanical operations are accepted. One additional bit of the equipment status word, the recoverable error bit, is used by the diagnostic master to print out the service required.

Diagnostic Return. The degree of diagnostic return is determined by whether the malfunction was recoverable. If it was, all normal operating programs are set to operate. If the malfunction was nonrecoverable, the program for that unit is not set to operate. This means for example that, if the recorder data bit has been removed due to a solid malfunction, a subsequent request for recording will be rejected. However, the movement of chips in the recorder and the cells to and from the recorder continues. If the mechanical bit is removed, all recorder operations are suspended. Finally, all operating programs and information stored on the disk by diagnostic call are returned to core storage.

Malfunction history is stored for up to 200 different kinds of malfunctions. Each recurrence of a given malfunction is indicated by a count. The field engineer may request that the stored malfunction history be printed out on the 1816 typewriter at any time.

The sample history output in Table I is a record of a failure in the chip-picker mechanism which is used to remove and deposit a chip in the photo-digital recorder. Figure 4 illustrates the chip-picker in the down position. The crankshaft drive motor runs continually. Engaging the selector-latch solenoid for a given time allows the crank to rotate 180°; this places the chip selector in the down position, ready to remove or deposit a chip. Repeating the operation places the picker in the up position (Figure 5).

Low-speed recovery

Recording of data on chips is done with an electron beam that is focused to a diameter of between 1.0 and 2.0 microns at the film chip plane. A pro-

Table I—History output of the chip-picker mechanism

1.	Failures	0001
2.	Pointer Number	0009
3.	Pointer Location	08A5
4.	Bit in Error	00C0
5.	Mask	00C0
6.	Received Word	D069
7.	SIOCC Control Word	CFD3
8.	Malfunction in Recorder	
9.	99C2 FCFF FB59 D069 3E71 FFFB DBAA 5E0D	
10.	Intermittent Failure	
11.	Chip-Picker Position	
12.	At Develop Station	
13.	Picker Revolved or Failed to Move	
14.	Recommand Cleared Trouble	

1. *Failures*—Specifies the number of failures of this type which have occurred since the last time malfunction history was "erased" from the disk.
2. *Pointer Number*—Specifies which one of the 48 program pointers detected a malfunction.
3. *Pointer Location*—Specifies the address of the pointer on the control string at the time the failure occurred. The location implicitly indicates the sequence of events leading to the malfunction.
4. *Bit in Error*—Specifies the particular sense bits that are in error (example: $00C0_{16}$ = 0000 0000 1100 0000_2 indicates that bit 8 and 9 were in error).
5. *Mask*—Specifies the sense word bits which were being tested (example: $00C0_{16}$ = 0000 0000 1100 0000_2 indicates that bits 8 and 9 were being tested).
6. *Received Word*—Shows the condition of a group of sense bits, including the bits in error, at the instant of malfunction.
7. *SIOCC (System Input Output Control Word)*—Identifies the portion of the unit in which the malfunction occurred.
8. *Malfunction in Recorder*—Specifies the major unit in which malfunction occurred.
9. *Hexidecimal Words*—Indicate the status of all sense points of the unit in which the malfunction occurred at the time the recovery routine was called from the disk. This information is useful for observing related mechanism movement during the malfunction. For example, mechanism A may only fail if mechanism B is in position X.
10. *Intermittent Failure*—Specifies that this particular malfunction was recoverable.

11–14. These unique message are recorded by the individual diagnostic routines. They specify the lowest level of mechanical movement detectable by the sensors. The chip selector has two normal positions, up or down.

Possible errors in chip selector position:

1. Selector expected to be in up position found in down position.
2. Selector expected to be in down position found in up position.
3. Selector jammed while moving down.
4. Selector jammed while moving up.

12. *At Develop Station*—There are four positions in the recorder where the selector withdraws or deposits chips.

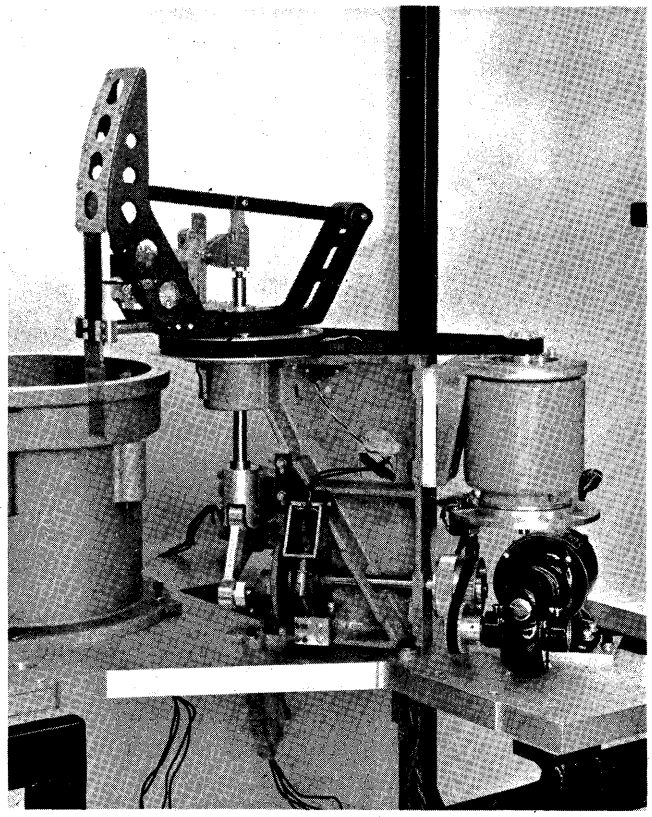


FIGURE 4—The chip-picker mechanism

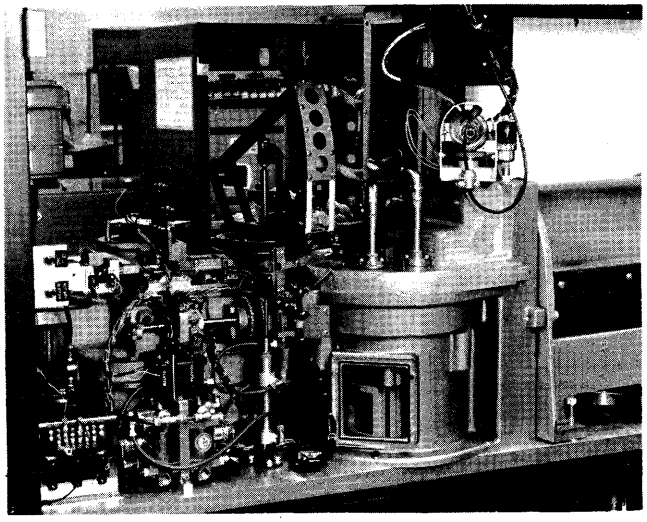


FIGURE 5—Photo-digital recorder. Note the complexity of this unit

gram-controlled electromagnetic electron optical system accomplishes the focusing. Prior to recording each chip, the spot-size of the beam is checked to assure that it is of the proper diameter.

When a failure occurs, the procedure for re-

covery is first to try the techniques of refocusing the beam. If all techniques fail, it is necessary to change filaments in the electron-beam gun. This requires the same procedure as for mechanical repositioning. That is, system operating is suspended until the recovery has been completed. This is the longest automatic recovery routine, requiring several minutes to complete.

CONCLUSIONS

A significant achievement of the IBM 1360 is its ability to discern a failure, go through its automatic recovery procedure, and then resume complete and normal operation. Much of this is made possible by using a stored program control. The program senses and controls the sequence of hardware actions at a detailed and efficient level of operation. It should be pointed out that the diagnosis and recovery is possible only because of the careful mating of programs and hardware during system design. The ability to be diagnosed must be built into the hardware.

ACKNOWLEDGMENT

The authors gratefully acknowledge the efforts of those who were involved in the development of the 1360, particularly those who contributed to the diagnostics and recovery programs.

REFERENCES

1 J D KUEHLER H R KERBY
A photo-digital mass storage system
Proceedings of the Fall Joint Computer Conference pp 753–742 1966

2 K H LOEFFLER
An electron-beam system for digital recording
IEEE 9th Annual Symposium on Electron Ion and Laser Beam Technology May 1967

3 F KURZWEIL JR R R BARBER M H DOST
Automatic control of an electron-beam column
IEEE 9th Annual Symposium on Electron Ion and Laser Beam Technology May 1967

4 K E HAUGHTON
An electron beam digital recorder
Third International Electron and Ion Beams in Science and Technology Conference May 1968

Simulation design of a multiprocessing system

by REINO A. MERIKALLIO

IBM Corporation
Gaithersburg, Maryland

and

FRED C. HOLLAND

MITRE Corporation
Bailey's Crossroads, Virginia

INTRODUCTION

Since 1965 the IBM Corporation and the MITRE Corporation have been involved in the design and implementation of the Enroute Stage A Air Traffic Control (ATC) System of the National Airspace System (NAS). The National Airspace System, as conceived by the Federal Aviation Agency (FAA), will provide advanced air traffic control capabilities in the late 1960's and early 1970's through the use of real time computing systems, thereby automating the data management functions of today's controllers. The system will automate 20 Air Route Traffic Control Centers (ARTCC) throughout the United States.

The system has four major objectives:

1. Provide automated aids for processing and updating of flight information.
2. Aid in establishing and maintaining radar identification of aircraft.
3. Automate CRT display of altitude or flight-level information with aircraft position.
4. Provide an automation base for subsequent improvements in ATC.

Figure 1 shows the overall data flow. Beacon (transponder) and primary (reflected) radar data are accepted into the system and processed from the multiple radar sites in the ARTCC area. The data are correlated with the predicted aircraft positions. Processed radar data plus alphanumeric data such as aircraft identity and altitude are displayed to Air Traffic Controllers ('R' position) on appropriate CRT displays.

Flight plans are accepted from local ARTCC

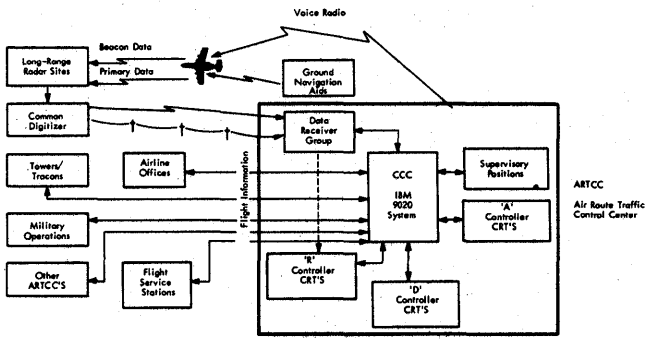


FIGURE 1—NAS enroute stage A-Typical ARTCC

positions and remote sources such as other ARTCC's, airport towers, airline offices, military operations and flight service stations. The data are error-checked, acknowledged, and stored in the system. Updates and amendments to flight plans can be added dynamically. When a flight plan becomes active, facilities are provided to match the flight plan with the corresponding radar track. Prior to departure and at specified intervals during the flight, flight strips containing data on the flight are printed for sectors (control areas) through which the flight will pass.

If a hardware error occurs, the computer program attempts to define the CCC element that failed and, if the failure is solid, configures the failed element out of the system and reconfigures a redundant element in to replace it. Where required, checkpoint data are reloaded from tape and the system recovers from the previous checkpoint. Tabulations of transient errors are main-

tained to permit reconfiguration of elements that may be gradually degrading.

A family of simulation models has been developed to assist in the design of NAS Enroute System. This paper describes the development, structure, applications and results obtained from these models. These models have been of significant value in evaluating alternative designs and locating critical resource bottlenecks. Novel problems have been encountered involving simultaneous processor competition for storage memory cycles and for common data tables. Techniques are described to minimize the degradation of system capacity due to this competition.

NAS IBM 9020 multiprocessing system

References 1-3 provide a detailed description of the NAS Enroute System. The following abstract summarizes those features which are pertinent to the simulation modeling effort.

The basic criteria for the development of the Central Computer Complex (CCC) were high reliability and variable capacity. The IBM 9020 system achieves these goals since it is highly modular with redundant elements provided throughout. Systems may be organized with varying degrees of capability to handle varying demands. The IBM 9020 system consists of five basic elements, which may be configured together to form systems of various capabilities:

- Computing Element (CE)
- Input/Output Control Element (IOCE)
- Storage Element (SE)
- Peripheral Adapter Module (PAM)
- Tape Control Unit (TCU)

Figure 2 shows a hypothetical minimum system involving one of each of these devices.

The Computing Element (CE) is the central logic and computational processor of the CCC. In design it is similar to the mid-range of IBM System/360 Central Processing Units. Each Computing Element contains special logic for configuring the other elements into one or more systems in a highly flexible manner. Up to four Computing Elements may be defined in the CCC.

The Input/Output Control Element (IOCE) provides independent control of input/output operations via two selector channels and one multiplexor channel passing data directly to and from the Storage Elements as requested by the Computing Element configured to it. Up to three Input/Output Control Elements may be included in the CCC.

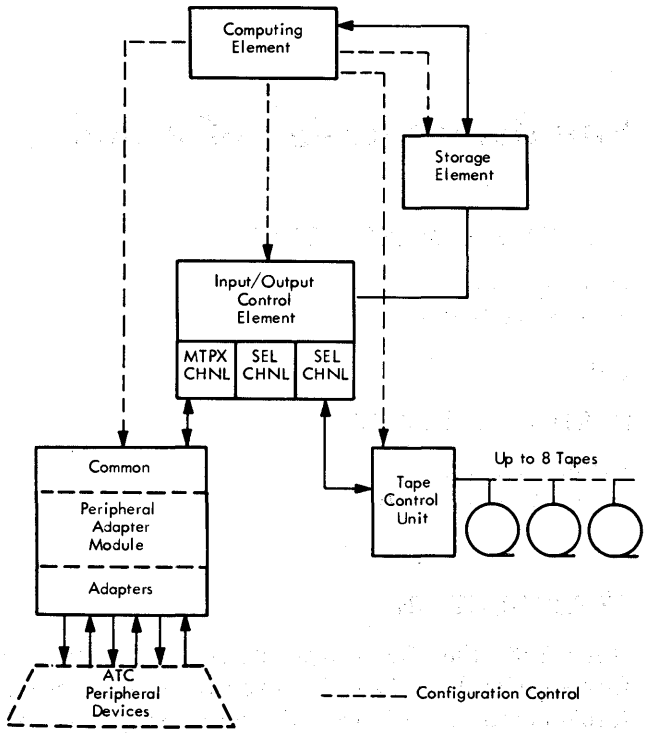


FIGURE 2—Simplex 9020 configuration-basic elements

Each Storage Element has a capacity of 32,768 or 65,536 words. The access times are 1.2 microseconds for single service and 2.5 microseconds for a complete cycle. A 9020 system may have as many as 12 Storage Elements. Each Storage Element can be addressed simultaneously by any Computing Element and any Input/Output Element if so configured. An Address Translation Register (ATR) allows any physical Storage Element to become any logical Storage Element under programmed control of a CE.

The Peripheral Adapter Module (PAM) is attached to the multiplexor channel of an Input/Output Control Element and controls input and output for up to 160 devices. The Peripheral Adapter Module provides for communication between the CCC and the ATC peripheral devices. A CCC may have as many as three Peripheral Adapter Modules. The IBM 7212–01 Tape Control Unit (TCU) provides for connection, to the system, of up to eight Model 2 or 3 IBM 2401 Tape Units. The system also has card readers, punches, printers, and a system console.

The various elements of the IBM 9020 may be mixed to provide different levels of computing and storage capability. The Jacksonville, Fla., ARTCC, considered to have medium-heavy traffic,

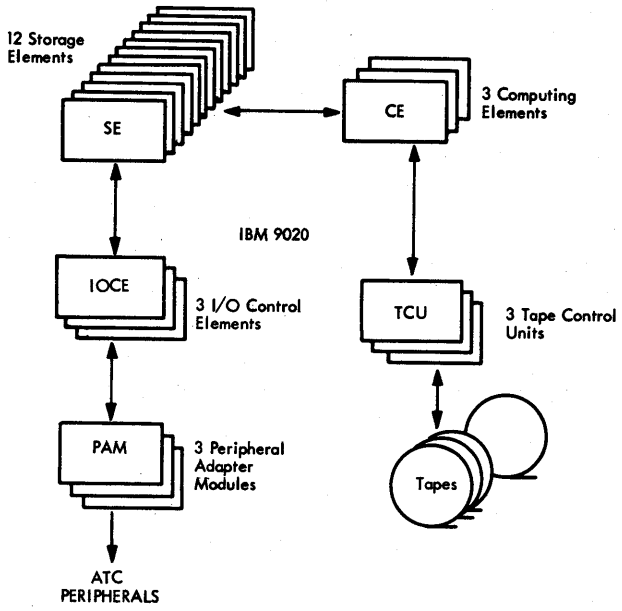


FIGURE 3—Jacksonville CCC configuration

will have the configuration shown in Figure 3, consisting of 3 CE's, 3 IOCE's, 12 32K SE's, 3 PAM's and 3 TCU's.

Advantages of multiprocessing

Several advantages are obtained through the use of multiple processors rather than a single higher speed processor. First, higher system reliability may be achieved in cheaper increments by adding a redundant slower speed processor which is generally cheaper than a faster processor. When an active processor element fails, the failing processor can be configured out of the system and the redundant processor brought on line. Secondly, the number of processors can be adapted to varying data processing loads in different systems. For instance, various NAS air traffic control centers are scheduled to use from one to three on-line 9020 processors plus one redundant processor. These systems can be expanded in cheaper processor increments to meet new traffic demands and additional functional requirements, as opposed to replacing a single processor with the next faster computer in a manufacturer's product line (if indeed such an upward compatible computer exists).

Various key measures of system performance (such as resource utilization, response time and work throughput) may be improved by exploiting and adapting parallel data processing structures to multiple parallel processors. For example, a particular program could either be coded reentrantly or duplicated so that multiple processors could execute the program simultaneously. Individual programs could be decomposed into independent portions for parallel processing.

Finally, whenever the system contains at least three processors, graceful or fail soft degradation can occur when more than one processor fails. For example, if a system consists of three processors (two active and one redundant) then even if two processors fail, the system could limp along with one processor performing either a portion of the overall work load or all of the work load but at a degraded performance level.

Design challenges of multiprocessing

Multiprocessing presents a very challenging software design job in order to achieve the advantages discussed above. The most challenging problem is to minimize the interference between data processing tasks in different processors which require concurrent access to a common data base. Access to a common data base must commonly be restricted to one task at a time, since each task may require that some portion of the data base remain consistent throughout the processing of the task. Excessive data base interference may cause a multiprocessor system to shrink to a single processor system, where the single active processor is executing the task which has acquired access to the most highly used parts of the common data base. The other processors remain idle while tasks queue up to gain access to the data base. Various techniques to enhance parallel processing of a common data base are described in Part 3.

The common data base is merely one of many system resources which the executive program must allocate and schedule optimally for widely varying task requirements. Tables I-III describe the NAS hardware and software resources and the competitors for these resources. Resources can become competitors for other resources. Many resources, e.g., terminals, channels, tasks, leased storage blocks, etc., are, of course, also found in single processor, real time, multiprogramming systems. The multiple processors, both as resources and as competitors for other resources, however create a unique new overall resource optimization problem.

A brand new type of interference develops when several processors compete for the memory cycles of a common storage element. Previously

Resources	Competitors
Computing Elements	Interrupts (I/O, external Monitor programs Operational programs
Storage Element Memory Cycles	Instruction fetches Data fetches and stores I/O data transfers
Input Terminals	External input messages from humans or other devices
Output Terminals	Write operations generated by programs
Duplex Input/Output Terminals	Same as above
Auxiliary Storage Devices (tapes, disks, drums, etc.)	I/O operations generated by programs (reads, writes, seeks, control operations)
Control Units	I/O operations on one or more devices/terminals attached to the control unit
I/O channels	I/O operations on one or more control units/devices/terminals attached to the channels

TABLE I—Physical system resources

(Capitalized words are the names of monitor services requests)

1. *Program Element (PE)*

A program element is an instance of execution of a program. NAS program elements are analogous to OS/360 tasks. When a program is re-entrant there can be more than one program element associated with the program. Resources such as common data locks, leased storage and communication lines are allocated to specific program elements.

2. *Leased Storage Blocks*

A portion of main memory is subdivided into various quantities of blocks of different fixed length sizes. These blocks are LEASEd by program elements, and subsequently RELEASEd and DELETEd. The monitor also leases and releases storage blocks for I/O event control blocks.

3. *Common Data Locks*

Common data locks control access to areas of storage which might be used simultaneously by more than one program element. These storage areas may contain data or instructions, e.g., non-re-entrant subroutines. Common data locks are LOCKed and UNLOCKed by program element requests to the monitor.

4. *Monitor Locks*

Monitor locks are bytes which are set/reset and tested by monitor programs to prevent more than one computing element from simultaneously modifying common monitor data tables. These locks are of short duration and use a test and set instruction.

5. *Communication Lines*

Communication lines control the SENDing and RELAYing of leased blocks to the input queue of a program or to an output channel. The input queue to a program consists of fixed numbers of communication lines for one or more block sizes. Output channels have a fixed number of lines, independent of block size. A communication line must be RESERVEd before a dynamic storage block can be LEASEd and SENDed to a program or channel. Communication lines are RESERVEd and CANCELed by program element SVC requests to the monitor. Monitor programs also reserve and cancel channel lines for various I/O operations.

6. *Main Memory Storage Elements*

The several 32K main storage memory storage elements are considered a software resource which must be allocated to programs, data areas and I/O buffers. This allocation will affect the amount of computing time stretchout caused by memory interference.

7. *Auxiliary Storage Devices*

The manner in which programs and data are allocated and then accessed can have a critical impact on system performance.

TABLE II—Definition of software resources

Resources	Competitors
Program Element of Operational Program	A. Messages in program queues waiting to be ACCEPTed by a program element. B. Messages which have been ACCEPTed by a program element and are waiting to be processed or SENDed to another program or channel. C. Time oriented data processing functions which periodically require the execution of a program by a program element.
Leased Storage	A. Program elements executing LEASE SVC. B. Program elements requesting I/O operations in non-leased storage. These I/O operations result in a monitor lease of an I/O event control block. C. Input channels which lease a block for the next input message.
Common Data Areas	A. Program elements executing LOCK SVCs.
Monitor Locks	A. Processors executing monitor subroutines.
Communication Lines	A. Program elements executing RESERVE SVC. B. Program element I/O operations which require a reserve of a channel communication line. C. Input channels which must reserve lines to input processing programs.
Main Memory Bytes and Words	A. Instructions of programs B. Data areas C. I/O buffers
Auxiliary Storage Device Bytes and Words	A. Programs B. Data tables

TABLE III—Software system resources and competitors

a single processor would only encounter competition from I/O channels which would require memory cycles to transfer data to and from storage. Multiple processors, however, also compete with each other for the memory cycles required to fetch instructions and to store and load data. The IBM 9020 system has the desirable characteristic of being able to employ independent 32K or 65K word storage elements, permitting simultaneous access to different SEs by different CEs without interference. Programs and data can therefore be allocated among several storage elements to minimize the probability that several processors and I/O channels are simultaneously accessing the same storage element. Memory interference may be so severe as to preclude use of re-entrant programs. Instead it may be necessary to allocate multiple copies of programs in different storage elements. Part 3 describes results obtained with a GPSS memory interference simulation model.

Objectives and nature of NAS simulation modeling

The NAS system simulation effort has had several objectives. The basic aim has been to determine system performance in terms of throughput, response time, and resource utilization. The simulation models can determine the sensitivity of the above performance measures under varying loads. By designing parametrized and modular models, various design alternatives can be readily modeled and evaluated, and critical system resources can be determined. The models can assess the impact on system performance of additional functions and data processing requirements.

There is also Murphy's Law of Simulation. The ordinary Murphy's law is all too familiar to anyone in the data processing: "If anything can possibly go wrong, it will go wrong." The simulation modeling variant is: "If anything can possibly go wrong, it will go wrong, YOU HOPE." It is far better to have some perverse combination of events and factors result in a Frankenstein breakdown of the system model than to have it occur in real life where reprogramming is time consuming and expensive. Murphy's Law of Simulation means that every model run could result in a system lockup, and every printout must be scanned for symptoms of some unanticipated design problem.

The process of building a system simulation model provides a detailed understanding of the entire system in an unsurpassed way. Relatively few people work on large scale real time systems from the view of optimizing the performance of the complete system. System simulation requires an understanding of the computer and peripheral hardware, monitor and operational software, and the system functional and performance requirements. The system simulation activity may alone be able to detect critical system resources and bottlenecks.

Need for simulation modeling

A simulation model is merely one of several possible tools which can be used to evaluate the performance measures of alternative system designs. Among possible design analysis tools are:

1. *Experimentation with actual system.* However it may be too costly, if not impossible, to construct all system configurations and load conditions. The actual system usually does not even exist. The extrapolations of results from the few know configurations and loads to unknown ones may be uncertain or unknown. Experiments with performance-degrading loads may not be possible because of safety considerations.
2. *Mathematical analysis* using *average* rates at which resources are required and *average* resource usage times. However, if there is significant randomness in the times between resource requirements and/or in the duration of resource usage, then a mathematical analysis based on average values may grossly overestimate maximum system capacities and underestimate response times.
3. *Mathematical analysis* employing *queuing theory models* to account for randomness in resource requirement rates and/or usage durations. Mathematical queuing models may use significantly less computer time, as well as providing a more powerful and fundamental understanding of the behavior of a system. However, the system may be too complicated to develop a manipulatable mathematical model, or the systems analyst may himself be limited in his knowledge of queuing theory and stochastic processes to develop a sufficiently accurate model.
4. *Discrete simulation model* which describes the interrelated behavior of each system component for each discrete event in the system. The logical behavior of the system must be determined and, where necessary, quantitative parameters, e.g., time durations, resource amounts, etc., must be estimated and incorporated in the model. The model is run for a period of time

during which statistics on desired performance measures are accumulated. The simulated time period merely provides a sample of system performance but should be sufficiently long to make useful statistical inferences on system performance.

There were a number of considerations in selecting the appropriate design analysis tool for the NAS 9020 system. Experimentation with the real system was impossible since there was no complete configuration available at the time that key system design decisions were being made. Significant systems analysis work was done using average computing times and average data processing loads. These average value analyses generally determined whether or not a given system or subsystem design could come close to doing a given job.

However, there are so many significant random factors in the NAS 9020 system to make one wary of any analysis based solely on average values. The NAS 9020 system is, however, too large and complex to consider developing an analytical queuing type model, although individual portions or problems might be amenable to such an attack. Consequently discrete simulation models were chosen as the most practical design analysis tools. The IBM General Purpose Simulation System (GPSS)[4,5,6,7] was the initial simulation language, with some models being converted subsequently to the IBM Computer System Simulator (CSS)[8,9] language.

Structure of the NAS 9020 simulation model

The NAS 9020 simulation model is a *resource competition* model which determines the absolute and relative *performance measures* of a specified NAS *system configuration* under specified load *conditions*.

A system configuration is defined by a specified combination of:

1. Types, quantities and attributes of resources, e.g., computing elements, storage elements, IOCEs, I/O devices and channels, leased storage blocks, common data areas, etc.
2. Competitors for these resources and their resources requirements, e.g., scheduled programs, monitor programs, I/O operations, machine interrupts, etc.
3. Hardware and software rules governing which competitors obtain these resources, e.g., priorities of program elements, priorities of machine interrupts, etc.
4. Lengths of time that the resources are in use, or the logical/physical conditions which terminate their use.

The system configuration can be defined either at a given point in time, or it can be defined variably over a period of time during which load conditions would vary.

The load conditions under which a given system configuration operates are defined by a combination of such major operational factors as:

1. Radar data input rates
2. Flight plans (active and inactive)
3. Tracked aircraft
4. Active sectors in the ARTCC
5. Input rates of message types
6. Cycle times of periodically scheduled programs.

The variation over time of the load conditions must also be specified. In the NAS system there are daily variations, seasonal variations throughout the year, and, finally, long term growth variations.

A given system configuration operating under given load conditions can be evaluated with the following major performance measures:

1. Resource utilization
   - A. Percent of time in use
   - B. Percent of time not in use because other required resources were not available
   - C. Percent of time idle.

   Heavily used resources may be bottlenecks which limit system capacity.
2. Queuing delay times experienced by the various competitors for system resources. Excessive delay times may be associated with heavily used resources.
3. Overall response (transit) times for different inputs, consisting of the sum of resource use times and queuing delay times experienced in processing these inputs through the system. Response times can be represented by the average time, by the entire frequency distribution of response times, or by specified percentiles of the frequency distribution.
4. Percent of the time or the number of times that the system or a specific program successfully performs a function, or conversely, fails to perform a function.

The maximum system capacity is generally expressed as the maximum value(s) of one or more specified load factor(s) which a given configuration can process without exceeding specified re-

sponse time requirements or failure rates. The NAS 9020 system capacity for a given ARTCC can generally be expressed in terms of a single load factor: the number of tracked aircraft. Other key factors such as flight plans, radar data input rates, and message input rates are directly related to the number of tracked aircraft.

Simulation results

Typical uses of models

The simplest use of the simulation models is the optimization of design parameters, which are those features of the hardware and software design that can be modified without changing the overall logical design, e.g.,

1. Resource Quantities
   A. Leased storage-sizes and numbers of blocks
   B. Communication lines for each program input queue and each I/O channel
   C. I/O buffers-sizes and numbers
2. Core Storage Locations
   A. Programs
   B. Leased storage blocks
   C. Data tables
3. Scheduling Algorithm for Program Elements
   A. Priorities
   B. Frequency of operation
   C. Time of operation

The simulation models can be altered to evaluate various major software design alternatives such as:

1. Re-entrant programs versus duplicate copies of programs located in different storage elements to minimize CE interference.
2. Allowing multiple program elements to gain simultaneous reading access to a common data lock area.
3. Aggregating or decomposing the programs executed by various program elements.
4. Different resource scheduling and allocation algorithms.

The system model can also be used to evaluate variations in hardware configurations such as:

1. Number and speed of compute elements
2. Size of storage elements, e.g., 32K versus 65K words
3. Use of peripheral storage devices such as disks and drums

Typical results

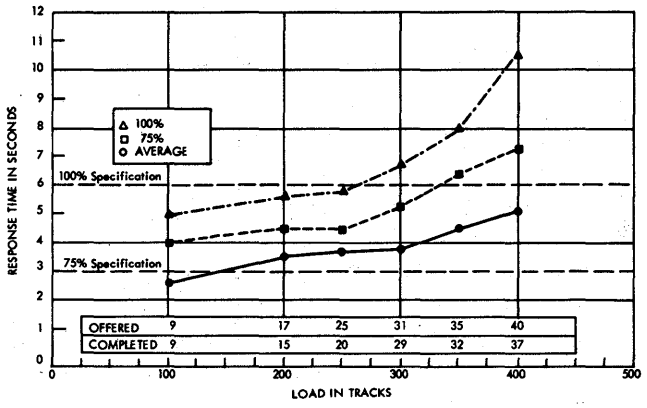


FIGURE 4—Handoff actions response time 2 CEs

Figures 4-8 show typical results from simulation runs in the early stages of NAS system design when numerous alternatives were being considered. Although the latest system design has different performance characteristics, the relative effects and the analysis techniques remain the same. Unless otherwise specified, a two compute element system is being simulated, generally for 60 seconds of real time.

Figure 4 shows how the response time for the important handoff action varies with the total number of tracked aircraft. A handoff of control occurs when a plane moves from one sector, controlled by one R-, D-, and A- controller trio, to another sector in the same or another ARTCC. The "Offered" line in Figure 4 shows the total number of handoff actions entered into the system during the simulated 60 seconds. The "Completed" line shows the number of handoff actions completed by the end of the 60 seconds. If the number completed is less than the number offered, it indicates that the balance is still being proc-

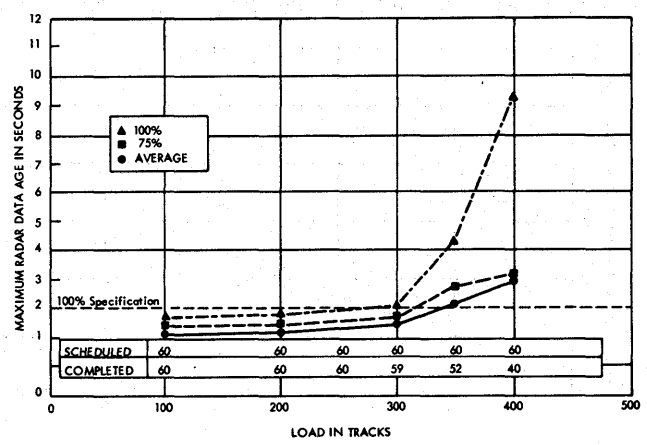


FIGURE 5—Maximum radar data age 2 CEs

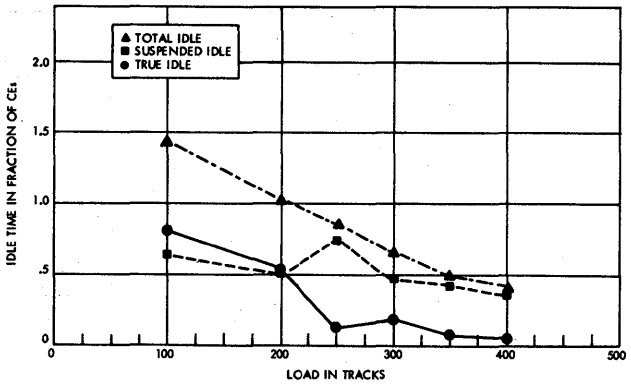


FIGURE 6—Average CE idle time 2 CEs

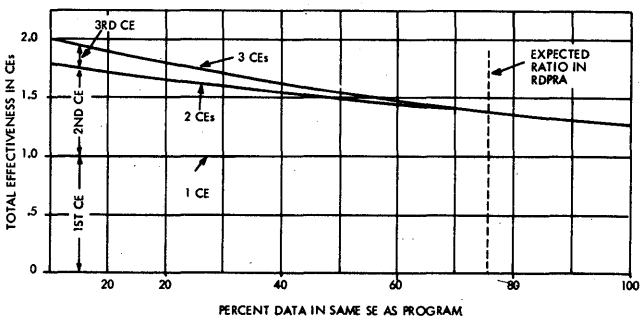


FIGURE 8—Total CE effectiveness-RDPRA

essed. These remaining handoff actions may be encountering normal queuing delays or, more seriously, a backup may be developing because the system capacity has been exceeded.

Figure 5 shows how the maximum radar data age varies with the number of tracks being processed. Radar data is read continuously into wrap-around buffers. One program element (RDPR) is activated every second to process the radar data received since the beginning of the previous RDPR operation. The processed radar data is written out at the end of each RDPR operation to various 'R' controller CRTs. The maximum radar data age is defined as the time from the beginning of the previous RDPR P.E. to the end of the next operation. The specified maximum radar age is 2.0 seconds.

Each RDPR operation results in one measure of maximum radar datage. The "Scheduled" line shows that ideally there would be 60 RDPR operations per minute. However, a particular RDPR operation may be delayed in getting necessary resources and hence, may take longer than two seconds to complete its work. Meanwhile another RDPR operation would normally have been activated for the next second. The system executive, however, lets the delayed RDPR operation continue on into the following second, so that less than 60 RDPR operations per minute can occur, as shown in the "Completed" line. Figure 5 shows that the maximum radar age increases considerably for loads in excess of 300 tracks.

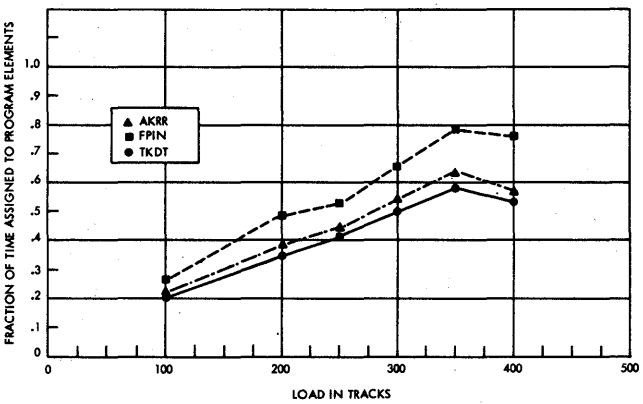


FIGURE 7—Lock area utilization 2 CEs

Figure 6 gives a clue to the maximum system capacity through the relationship between computing element average idle time and track load. Idle time has been subdivided into two components. "True Idle" is the fraction of a C.E.'s time that it is idle and there are *no* program elements (tasks) in any type of resource or wait I/O suspension. "Suspended Idle" is the fraction of a C.E.'s time that a compute element is idle when one or more P.E.s are suspended. C.E.s are ready to perform work but the P.E.s are suspended because of the unavailability of other system resources, e.g., common data locks. The leveling out of the Total Idle curve at 400 tracks in Figure 6 is an indication that the system capacity has been reached. Further work load is being added to the system yet no further computing is being done. Resources other than compute elements are causing infinite queuing to develop. Various response time specifications may, however, have been exceeded at lower track loads.

Figure 7 shows the utilization of three important common data tables (AKRR, FPIN, TKDT) as a function of the track load. The system has been found to approach saturation when the utilization of the most critical common data areas approaches 0.8. The key system design problem has been to reduce the usage of the various common data areas. Whenever such reductions are simulated, system performance improves markedly.

Techniques for enhancing parallel processing of a common data base

A variety of techniques have evolved to enhance the ability of multiple processors to operate off a common data base and to reduce the loads on critical areas. Each technique must generally be tailored to the particular application and data base. The simplest way to avoid interference is to schedule the various functions so that two program elements will never require access simultaneously to a common portion of the data base. This is difficult, however, in a random real-time system.

Access to a specific portion of the data base can be controlled by a common data lock associated with that portion. The restrictive or queuing delay character of these locks can be diminishing by allowing each lock to operate either as a read lock or write lock. Only one program element can gain a write lock, but multiple program elements can simultaneously gain a particular read lock by which they read but do not modify the particular portion of the data base.

The queuing delays caused by simultaneous attempts to lock data areas can be reduced by minimizing the duration of the lock; first, by delaying as long as possible before attempting to lock, and secondly, by unlocking as soon thereafter as possible. The monitor can also facilitate the earlier unlocking by giving higher priorities to program elements which have critical locks. One way to hold off locking is to only add completely processed data to the data base.

If a number of common data locks are required by a program element, they do not have to be acquired at the same time, but can be requested only when needed. This, however, requires that all program elements request common data locks in the same sequence, so as to avoid the following system lockup problem: PE 1 obtains a Lock A while PE 2 is concurrently obtaining Lock B. After additional processing time, PE 1 attempts to get Lock B while PE 2 attempts to get Lock A. However, neither program element will unlock its first lock until after it has obtained the second lock. The lockup could be avoided by requiring all PE's to request locks in the same order, e.g., A first, then B.

Another way to reduce the impact of common data locks is to increase their number by breaking up the data base into smaller portions, each of which has a data lock. For example, consider a data table with n entries, each of which has p fields or items. Several common data locks could be defined, each identified with a subset of the p fields in all entries. Alternatively, multiple common data locks could be defined for subsets of the n entries, where in the ultimate each entry in the table has its own individual lock.

If program 1 transmits data to program element 2 by a communication table, a common data lock on the table can be avoided by treating the table as a wraparound buffer. Each PE maintains a pointer to where it is currently processing. Both PE's can operate simultaneously by not moving ahead of where the other PE is currently pointing. Some monitor services would have to be provided to suspend a PE temporarily when it catches up with the other PE, and to resume the suspended PE when it can do further processing.

Results of memory interference model

With a multiprocessing system, it is possible to have re-entrant programs (which permit several processors to execute them in parallel), thereby minimizing core storage. However, the computing power of the processors would be degraded by their mutual interference in fetching instructions and data from the storage element in which the program is located. Figure 8 shows the results of a typical series of runs with a special memory interference model which determine tradeoffs between using single copy reentrant programs and using more core for duplicate copies in different storage elements. The system simulated in Figure 8 has 1, 2 and 3 computing elements executing the same program in the same storage element (SE). There are no I/O data transfers occurring. Runs were made by varying the percentage of the total data accesses which are located in the same SE as the program (the horizontal axis). For 0 percent all the data accessed by the program is in a different SE.

The stretchout factor is defined as the ratio of the execution time of a program with interference to the execution time without interference. Thus, if the stretchout factor is 1.25, then a program's execution time will be extended by 25 percent. The effectiveness of a CE (the vertical axis of Figure 8) is the reciprocal of the stretchout factor. If two CE's are each executing the same program, with a stretchout factor of 1.33, the effectiveness of each CE is $1/1.33 = 0.75$. Thus, the total effectiveness of the two CE's is $2 \times .750 = 1.500$. If three CE's were executing the same program,

each with a time stretchout of 2.00, then the total effectiveness of the three CE's would be $3 \times 1/2.0 = 1.5$. Thus, the total effectiveness of the three CE's is the same as two CE's and therefore, the third CE, from a system point of view, does not perform any useful data processing.

Figure 8 illustrates the relationship between total CE effectiveness for a particular NAS program called RDPRA and the interference level. When there is only one CE executing a program in an SE, then there is no interference and the effectiveness is 1.0 regardless of the location of the data. When two CE's are executing programs like RDPRA in parallel in the same SE, the total effectiveness varies between 1.78 and 1.28 depending upon the fraction of the data fetches and stores that are requested from the same SE as the programs. Thus, the contribution of the second CE to the total system effectiveness is between .78 and .28 CE's.

When three CE's are executing programs like RDPRA in parallel in the same SE, the total effectiveness varies between 2.0 and 1.28. Thus, the contribution of the third CE to the total system effectiveness would be between .22 and 0.

It should be noted that data can be located in different SE's than the program by modifying the assembler or complier and the loader so that internally defined tables, literals, etc., can be purposely loaded into a storage element(s) other than the program. The other alternative is to duplicate small high use programs so that memory interference is minimized.

Structure of the simulation models

The basic approach has been to develop a family of simulation models to evaluate various design problems. A simulation language was used: the IBM/360 General Purpose Simulation System (GPSS). Consequently the data processing entities of the NAS system (processors, channels, programs, I/O operations, etc.) were modeled in terms of GPSS entities (transactions, blocks, facilities, storage, queues, etc.) and their attributes. Consideration was given initially to the 7090 version of the more specialized IBM Computer System Simulator (CSS). The 7090 CSS could not simulate multiple processors so that GPSS was chosen. The/360 CSS can model multiple processors (up to 256) and so various GPSS models have recently been replaced by CSS models. The IBM Computer System Simulator employs data processing oriented entities and commands. The new CSS models are expected to be faster running, easier to learn and more readily modified in evaluating alternative system designs.

A separate memory interference model was developed which determined the average stretchout or slowing down of a program being executed in a particular storage element as a function of the I/O and processor interference levels in the storage element. The memory interference model simulated the execution of individual instructions and I/O data transfers in order to obtain accurate interference relationships. The overall system model would run far too slow if it simulated individual instructions. Consequently computing operations are represented by large aggregate times which are estimated for interference free conditions. The memory interference which would occur during these computing times is approximated by:

1. Determining the average time stretchout factor as a function of:
   - A. The total I/O data rate in the storage element in which the computing operation will occur.
   - B. The total number of processors executing instructions in this storage element and
2. Multiplying the interference-free computing time by this stretchout factor to determine the actual computing time.

Another model creates a GPSS jobtape which contains a random, time ordered set of external input messages and internal events. The same message-event mix can, therefore, be inputted to a set of alternative system designs. There are several advantages to this approach. First, the overall system model runs faster since it does not have to take computer time in each new run to generate messages and events and their attributes. More importantly, a constantly regenerated message-event mix could vary from run to run, thereby by confounding statistical variations caused by these changes with those caused by true variability between alternative designs. The messages and events which are simulated are only a small portion of the total number of possible message and event types. They, however, constitute more than 90 percent of the total volume of messages and events.

The overall system model was developed in two distinct parts: a monitor/hardware submodel and

an operational program submodel. The monitor/hardware submodel is far more complex, requiring 3500 GPSS blocks while the operational program submodel requires 2000 GPSS blocks, many of which are only used for calling sequence arguments. In the actual system, the portion of the monitor modeled represents about 8000 out of 172,000 words of program. This apparent distortion, however, reflects the fact that the monitor controls the system resources which are the major elements of the system model. The monitor/hardware submodel also uses the great majority of the other GPSS entities (facilities, storages, savevalues, etc.)

The monitor/hardware submodel simulates the following standard monitor programs:

1. Dispatcher program (ZMDP) which allocates tasks to the processors on the basis of task priorities.
2. I/O interrupt processing program (ZMCX) which performs housekeeping functions associated with the completion of I/O operations and allocates delayed I/O requests to available devices and channels.
3. External interrupt processing program (ZMEX) which performs functions associated with periodic timer interrupts and write direct operations between processors.
4. Monitor service request program (ZMSX) which handles various monitor services provided to the operational programs, such as initiating I/O reads and writes, leasing dynamic storage blocks, locking common data areas, scheduling other tasks, and waiting for the completion of outstanding I/O operations.

The monitor/hardware submodel also simulates the I/O hardware (channels, control units, lines and terminals), the execution of various I/O operations on this hardware, and the generation of I/O interrupts of processors. The monitor/hardware submodel has built-in performance statistics which are intended to answer most of the important questions on system performance. Additional special statistics can usually be obtained with ease because of the inherent flexibility of the GPSS language.

The operational program submodel was designed to operate in relative ignorance of the complexities of the monitor/hardware submodel. The major interface between the two submodels is a set of standardized monitor service requests. Each request is a transfer to the monitor submodel with a set of arguments that define the type of service or resource required by the operational program. Consequently, personnel who are less skilled and experienced in the complexities of the GPSS simulation language can model the various NAS operational programs using the simple monitor service requests. They can concentrate their effort on accurately representing the problem state processing times and programming logic. One experienced person has maintained the relatively unchanging model of the monitor programs and hardware.

The overall system model has itself generated some offspring. Because of its complex structure and its relatively slow running time, a scaled down version was produced which eliminated the simulation of various system resources and the gathering of many performance statistics. Dynamic leased blocks and I/O hardware resources were dropped. Memory interference was not accounted for. Instead, the smaller, faster running, system model concentrates on two key resources: the processors which perform computing and the 31 common data locks which govern access to various parts of the system's common data base. This model has proved useful in evaluating different resource scheduling schemes.

Table IV summarizes the size and running time characteristics of the family of models.

Model	Block Size	Unit Time	Run Time	360/50 Time
Memory Interference	50	.1 μsec	10 msec	30 sec
Input Message-Internal Event Generator	200	1 msec	3 min	4 min
Overall System	5500	1 μsec	60 sec	20–40 min
Scaled Down System	1000	1 msec	60 sec	2 min

TABLE IV—Characteristics of GPSS models

CONCLUSION

Simulation modeling should be an integral part of designing complex data processing systems. A family of models will generally be required representing different aspects or portions of the overall system. Simulation models permit an accurate evaluation of numerous design alternatives. They can spotlight critical resource bottlenecks before actual system implementation. Through Murphy's Law of Simulation every simulation run provides

a vital quality assurance function by possibly revealing some unanticipated design problem. Multiprocessing systems present significant new design problems because of simultaneous competition among processors, first, for access to a common data base and, secondly, for storage element memory cycles. Simulation modeling may alone determine the best design to minimize the performance degradation resulting from this competition.

REFERENCES

1 *An application-oriented multiprocessing system*
IBM Systems Journal Vol 6 No 2 1967

2 F K SEWARD
National airspace system enroute
Proceedings 1967 IEEE International Convention

3 *System description national airspace system en route, stage A*
FAA April 1965

4 *GPSS/360 introduction-user's manual*
IBM Form Number H20-0304

5 *GPSS/360 user's manual*
IBM Form Number H20-0326

6 *GPSS/360 OS operator's manual*
IBM Form Number H20-0311

7 *GPSS/360 system manual*
IBM Form Number Y20-0075

8 *Computer system simulator/360 program description and operations manual (IBM Confidential)*
IBM Form Number Y20-0130

9 *Computer system simulator/360 application description*
IBM Form Number Y20-0028

10 T A HUMPHREY
Large core storage utilization in theory and in practice
Proceedings of 1967 Spring Joint Computer Conference

11 W I STANLEY H F HERTEL
Statistics gathering and simulation for the APOLLO real time operating system
IBM Systems Journal Vol 7 No 2 1968

12 E C SMITH JR
Simulation in systems engineering
IBM Systems Journal Vol 1 1963

13 N R NIELSEN
The simulation of time sharing systems
Comm ACM Vol 10 July 1967

14 J H KATZ
Simulation of a multiprocessor computer system
Proceedings of 1966 Spring Joint Computer Conference

A simulation study of resource management in a time-sharing system

by SANDRA L. REHMANN

IBM Corporation
San Jose, California

and

SHERBIE G. GANGWERE, JR.*

Computer Application Incorporated
Palo Alto, California

INTRODUCTION

In general, simulation has two main uses. One is to provide a basis for predicting how the system simulated will perform under varied conditions and for determining which conditions are necessary for optimum performance. The other is more basic. Complex systems can be beyond the scope of human comprehension, and the statistical data generated in a simulation can be used to further the understanding of the interworkings of the system under study. The knowledge gained can, in turn, become the basis for further analysis and evaluation. Results can improve the performance of the system being simulated, as well as broaden performance goals and attainments of future systems.

This paper describes the design and use of a simple time-sharing, multi-programming model. Both goals of the simulation study were met:

1. To estimate the requirements for optimum performance.
2. To obtain a basic understanding, through use of a simple model, of what happens in a multi-programming environment.

Several ideas had been incorporated into the design of the system. The simulation provided us a means of evaluating these ideas.

Response time was selected as the performance guideline in the simulation, since, to the user, response time is one of the most meaningful measurements of a time-sharing system.

Background

The simulation was directed at the problem of "slave-resource" management. Slave-resource management is defined as the management of core memory, disk channel(s) and drives, and the central processor, to accomplish compilations and/or executions on behalf of a user. A compute interaction is the total amount of computing required as a result of an EXECUTE command or user response to a READ statement. A time slice is the maximum amount of CPU time allotted to a program for compilation and/or execution. If the program does not complete in one time slice, it must wait in a queue until additional CPU time can be allotted to it.

The EXECUTE command results in compilation and execution of the compiled code. The READ statement results in a compute interaction. Sequentially the program is compiled, and object code execution begins and continues up to the point of the READ statement. The object code notifies the operating system that input data from the terminal are required. The object program is swapped to disk. All computing up to this point is due to the EXECUTE command. When the user responds with input data, the program can resume execution. Computing from this point

*Now with Advanced Programming, Incorporated, Santa Clara, California.

on is a result of *response* to the READ statement.

System description

The system studied is nonpaging, multi-programming. Users are swapped in and out of high speed core memory via a disk storage unit.

The system does not execute line by line. Instead, all lines for a program are entered; the user then issues an EXECUTE command. The EXECUTE command results in a compilation and execution of the user's program.

All compute requirements up to the EXECUTE command are handled in system's overhead. In the simulation, the only concern was the problem of system management during the compile and execute phase.

A user can cause the system to do a compile and/or execute in two ways:

1. The EXECUTE command causes a compile and execute.
2. When object code is being executed and a READ statement is encountered, execution of that program is suspended, and an appropriate message is sent to the user's terminal. After the user supplies the input data, execution of his program will resume.

The ideas

To the best of our knowledge, H. Cantrell[1] was the first to propose using a time slice to separate short jobs and long jobs. His basic idea was to treat all new jobs as though they would complete in one time slice or less. If a time slice was chosen so that 90% of all jobs would complete in one time slice or less, a slave-resource management system could take advantake of the probability that:

1. 90% of all new jobs would complete in one time slice or less.
2. Jobs that did not complete in a time slice tend to be long-running jobs.

Cantrell observed that, having identified long-running jobs, if the slave-resource management system can keep at least one long job in core, the long-running jobs can use compute time available during I/O time of the new jobs.

Both Dr. Amdahl[2] and H. Cantrell[1] suggested using two queues, one for new jobs, and another for partly-finished jobs that have already used more than a time slice.

In evaluating the proposed system, we found that the probability of being able to hold two jobs in memory simultaneously was greater than .9996.

As a result, our simulation study was based on a further assumption that:

1. Two jobs would always fit in core.
2. A maximum of two jobs would be allowed in core at any time.

With only two job areas in memory, calculations indicated that CPU utilization could possibly exceed 90% when jobs were in the queue waiting for service.

The model

The simulator was written in GPSS.[3] Simulator output was hand-checked to verify the model. Each run required about three minutes of System 360/50 time. A total of ninety-one runs were made and the results tabulated and graphed.

Figure 1 is a block diagram of the model. All new jobs (i.e., compute interactions due to an EXECUTE command or response to a READ statement) are entered in the bottom of Queue 2 (a FIFO queue). New jobs are generated at the rate of one per user per 110 seconds. When a user generates a job, he is removed from the set of users generating jobs. When a user has issued an EXECUTE command or responded to a READ statement, the same user cannot generate another compute interaction request until his first request is completed. When Facility 2 (core memory area number 2) is not in use, the job at the head of Queue 2 is loaded into core. A simulation system study (unpublished) measuring disk chan-

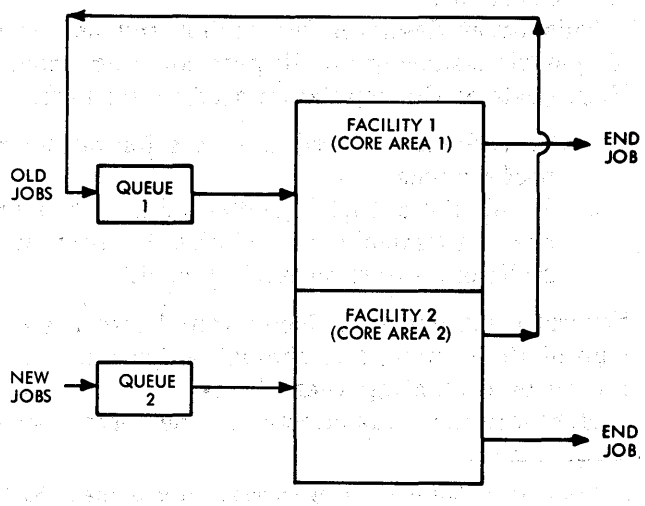


FIGURE 1—Block diagram of model

nel(s) usage had been done by S. Rehmann. Results of this simulation were used to determine requirements for disk I/O time. The model used an I/O time of 150±25 ms (random in the range of 125 to 175 ms). When the load is completed, the CPU is applied to the job until either the compute time-slice is up or the job signals EOJ (end of job).

When the job signals EOJ, the job is removed from the system. If it used a time slice and did not signal EOJ, it is moved to Queue 1 (a FIFO queue). Regardless of the reason for termination, one I/O time is used to remove the job from core.

Assume for a moment that a job is removed because the time slice is up. After completing the I/O (by rolling the job out to disk), the job is placed at the end of Queue 1. When Facility 1 (core memory area number 1) is available, the job at the head of Queue 1 is loaded. When the load is completed, the CPU is applied to the job *if* a new job is not in Facility 2. CPU priority is a simple three-level priority:

1. A new job in Facility 2 gets the CPU.
2. If the CPU is not required for a job in Facility 2, the CPU is applied to a job in Facility 1. When the CPU is applied to a job in Facility 1 and a load completes for a job in Facility 2, the Facility 1 job is interrupted and the CPU is applied to the job in Facility 2.
3. The CPU is idle when a job is not present in either facility.

A job from Queue 1 resides in Facility 1 until it runs to completion.

We can now discuss the model in terms of a job's life span. By definition, there are two kinds of jobs in the system—short jobs and long jobs.

Short jobs

Short jobs are jobs which will complete in one time-slice or less. Refer to Figure 2 for a timing chart of the life span of a short job.

A new job entering the system is placed in the New Job Queue (Queue 2, Figure 1). Since jobs are processed on a first-in-first-out basis, jobs advance until they are at the head of the queue. When core memory (Facility 2, Figure 1) is available, the I/O required to load the job is initiated.

After loading, the CPU is applied to the job for the length of time required to complete the job. A short job will complete when:

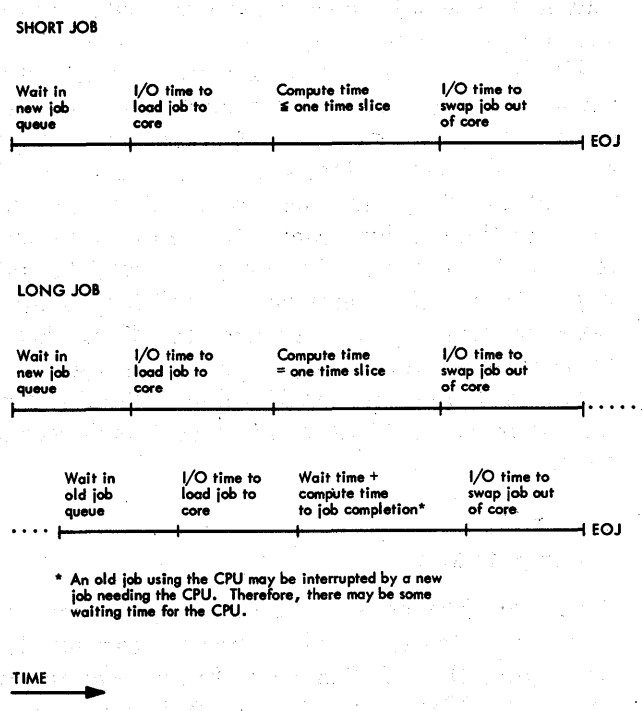


FIGURE 2—Life span of jobs

1. computing is completed, or
2. the job requires input from the terminal.

A job that does not complete for one of the above two reasons is terminated when the time-slice is up. Such a job is then defined as a "long job". Regardless of the reason for termination, I/O is initiated to roll the job to disk.

Long jobs

Long jobs are those which require more than one time-slice. When the time-slice is up and a job has been rolled out to disk, it is placed at the end of the long-job queue (Queue 1, Figure 1). It waits in the long job queue until it has worked its way to the top of the queue. When the core memory (Facility 1, Figure 1) is available, the I/O to load the job is initiated.

A short job in Facility 2 always has CPU priority over the long job in Facility 1. Due to this CPU priority algorithm, the long job will more than likely be interrupted several times when short jobs in memory require the CPU. When the long job terminates, the I/O to roll the job to disk is initiated.

At this point let's examine the results that might be expected from this simple kind of slave-resource management. New jobs are treated as though they can be completed in one time slice.

This is a reasonable conclusion since 85% to 99% will complete, depending on the time slice.

All old jobs (i.e., the ones which are placed in Queue 1) are treated as though they will require a large amount of CPU time. This also is a reasonable conclusion, since the mean compute time of all old jobs is much greater than the mean compute time of jobs completed in one time slice or less. By giving CPU priority to new jobs, jobs requiring one time slice or less will have a short response time. This approach, in effect, favors short-running jobs. Longer jobs (the remaining 10%) will require considerable compute time, and little can be done to improve their response times.

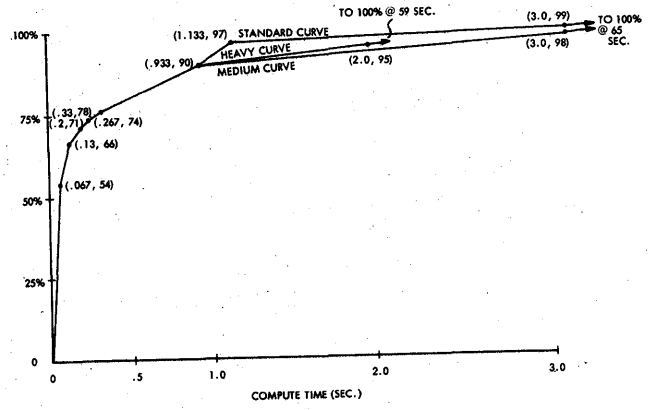


FIGURE 3—System compute load

Compute load

H. Cantrell[1] observed that the computer load curves are almost identical when the compute load data from the GE 265 time-sharing system and the MAC CTSS[4] time-sharing system are scaled (to take into account the different central processor speeds) and plotted on the same graph.

For this simulation (see Figure 3) the first 90% of the MAC CTSS curve was scaled to a two microsecond CPU and used as the first 90% of all three compute loads. The last 10% of the three compute load curves were arbitrarily picked to give various mean compute times. One purpose of this simulation was to determine the effect on the model as the compute load varied. The model is not sensitive to reasonable changes in the first 90% of the compute load, but is sensitive to small changes in the last 10%.

Characteristics of the three compute loads are:

Standard Curve	.61 sec mean
Medium Curve	.99 sec mean
Heavy Curve	1.56 sec mean

When the simulator picks a value from the compute curve, that value is divided by .75 to compensate for the 25% system overhead.

Case problems

Seven case problems were addressed during the course of this study. For each case the result was generated by running 2000 jobs through the model. Data attained from these case problems are illustrated using data tables and graphs. Refer to these tables and graphs in relationship to the conclusions of these problems, which are summarized in the following paragraphs.

Case 1

This run was made with an active user load of from 10 to 200 active users in steps of 10, using the Standard Curve. Refer to Tables Ia, Ib and Figures 4a-4e.

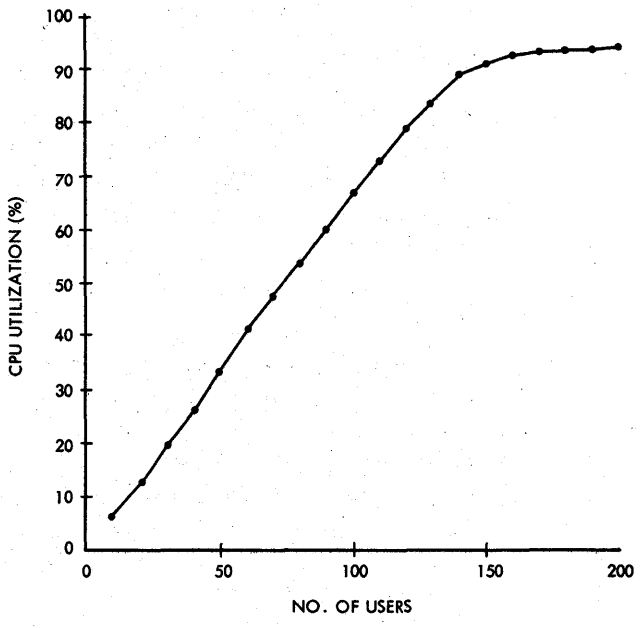


FIGURE 4a—Case 1—CPU utilization

TABLE Ia—Case 1—Data table

		Number of Users										
		10	20	30	40	50	60	70	80	90		
Percent Utilization	Facility 1	3.7	8.4	13.3	18.5	24.1	30.0	37.0	43.9	50.5		
	Facility 2	4.5	10.1	15.8	21.4	26.9	32.5	37.9	43.3	48.5		
	CPU	5.8	12.8	19.8	26.8	33.6	40.6	47.3	54.0	60.5		
Average Time (Sec.)	Facility 1	5.0	5.0	5.2	5.2	5.4	5.6	5.9	6.1	6.3		
	Facility 2	.625	.675	.625	.625	.625	.625	.625	.625	.625		
	CPU	.73	.72	.71	.71	.71	.71	.71	.71	.71		
New Job Queue	Max. Contents	1	1	1	1	1	1	1	1	1		
	Avg. Contents	.00	.00	.00	.00	.00	.00	.00	.01	.03		
	Avg. Time (Sec.)	.00	.00	.00	.00	.00	.00	.00	.015	.04		
Old Job Queue	Max. Contents	2	2	5	7	7	7	10	11	12		
	Avg. Contents	.002	.02	.08	.19	.35	.55	.87	1.26	1.82		
	Avg. Time (Sec.)	.32	1.3	3.0	5.3	7.8	10.2	13.8	17.6	22.7		
Mean Response Time (Sec.)	System	1.15	1.27	1.46	1.71	1.99	2.26	2.66	3.09	3.66		
	Short Jobs	.52	.52	.52	.52	.52	.52	.52	.53	.56		
	Long Jobs	6.67	7.79	9.62	12.1	14.8	17.4	21.3	25.4	30.7		
Mean Compute Time (Sec.)	System	.61	.61	.61	.61	.61	.61	.61	.61	.61		
	Short Jobs	.16	.16	.16	.16	.16	.16	.16	.16	.16		
	Long Jobs	4.5	4.5	4.5	4.5	4.5	4.5	4.5	4.5	4.5		

Case 1. Compute Load = Standard
Time Slice = 933 ms
Number of Users = Variable
Table Ia. Case 1 - Data Table

		Number of Users										
		100	110	120	130	140	150	160	170	180	190	200
Percent Utilization	Facility 1	58.1	65.9	72.9	80.6	86.6	90.0	92.6	92.9	93.1	93.2	93.2
	Facility 2	53.8	58.7	63.3	67.9	71.7	75.2	78.3	81.2	84.1	86.9	89.5
	CPU	67.0	73.2	78.9	84.3	88.6	90.7	92.5	93.3	93.5	93.7	93.8
Average Time (Sec.)	Facility 1	6.5	6.8	7.0	7.4	7.8	7.7	8.1	9.1	8.8	9.0	9.0
	Facility 2	.63	.63	.63	.62	.62	.62	.62	.63	.63	.63	.63
	CPU	.71	.71	.71	.70	.70	.69	.68	.67	.65	.63	.61
New Job Queue	Max. Contents	2	3	3	4	5	5	6	7	9	14	21
	Avg. Contents	.06	.11	.16	.25	.34	.43	.56	.71	.98	1.6	2.8
	Avg. Time (Sec.)	.08	.11	.16	.23	.29	.36	.45	.55	.73	1.1	1.9
Old Job Queue	Max. Contents	14	19	20	25	29	34	41	54	54	64	74
	Avg. Contents	2.6	3.8	5.3	7.3	10.3	14.2	18.4	23.7	28.3	33.2	37.2
	Avg. Time (Sec.)	29.5	38.8	50.7	65.3	86.7	114.3	142.2	173.9	200.4	226.8	247.5
Mean Response Time (Sec.)	System	4.4	5.5	6.8	8.3	10.4	13.3	15.6	15.7	18.9	20.6	22.7
	Short Jobs	.59	.63	.68	.74	.81	.88	.97	1.1	1.3	1.7	2.5
	Long Jobs	37.8	47.5	59.7	75.7	99.4	129.2	160.3	183.2	220.7	248.7	274.1
Mean Compute Time (Sec.)	System	.61	.61	.61	.61	.61	.61	.61	.63	.63	.63	.62
	Short Jobs	.16	.16	.16	.16	.16	.16	.16	.16	.16	.16	.16
	Long Jobs	4.5	4.5	4.5	4.5	4.5	4.5	4.5	4.6	4.6	4.5	4.5

Case 1. Compute Load = Standard
Time Slice = 933 ms
Number of Users = Variable
Table Ib. Case 1 - Data Table (continued)

TABLE Ib—Case 1—Data table (continued)

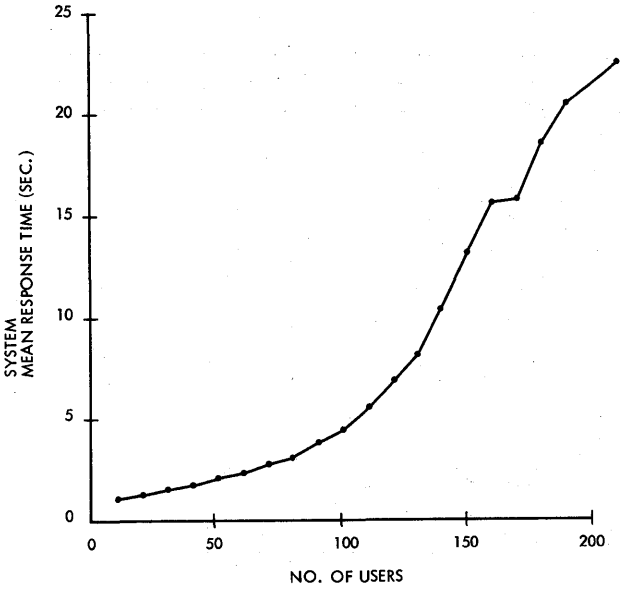


FIGURE 4b—Case 1—System mean response time

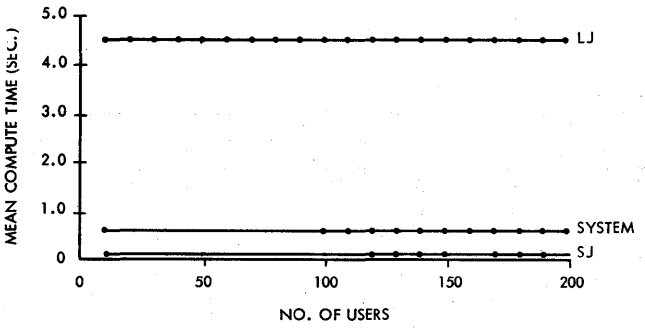


FIGURE 4c—Case 1—Mean compute time

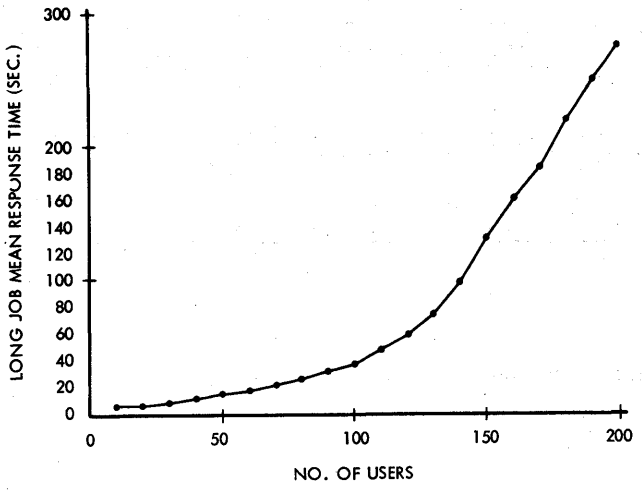


FIGURE 4d—Case 1—Long job mean response time

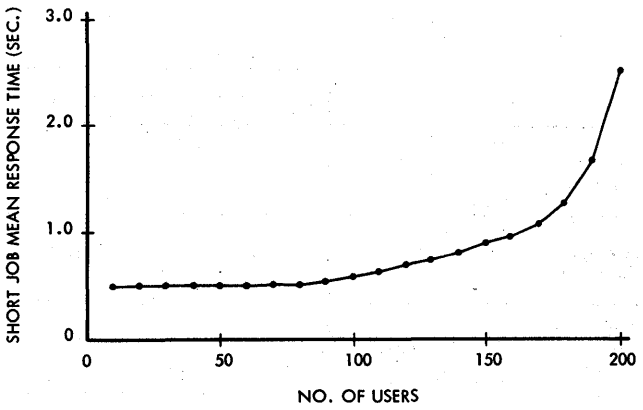


FIGURE 4e—Case 1—Short job mean response time

Analysis

If we pick a point where the CPU utilization is 67% (say, 100 users) and look at the data, we can get a good idea of what is happening in the system. The New Job Queue has an average of .06 users. This means that the system is working off new jobs about as fast as they enter the system. No gains in performance would be realized by trying to put two new jobs in core at the same time.

The Old Job Queue has an average of 2.6 users, but Facility 1 is used only 58.1% of the time. The low usage of Facility 1 means that sometimes the Old Job Queue does not have any entries; hence, Facility 1 is idle or not used. Combining the facts that the Old Job Queue has an average of 2.6 users, but Facility 1 is used only 58.1% of the time, we can deduce that the system, at times, is in a temporary "overload" condition in the Old Job Queue-Facility 1 area, but the system is able to "catch up." During overload conditions, some gain could be realized if two old job core areas were used. The gain may not be as great as one might expect, because if two old jobs core areas were used, there is still only one CPU to apply to the jobs. The only gain would be to use the CPU on the second old job during the I/O times for the first old job and the new job.

During the Old Job Queue-Facility 1 overload periods, the CPU utilization undoubtedly goes much higher. From the data we can estimate what the CPU utilization is during overloads. The old job mean compute time is 4.5 seconds, 4.5 ÷ .75 to compensate for the 25% overhead = 6.0 seconds. The total I/O time is 0.3 second. Compute plus I/O is equal to 6.0 + 0.3 = 6.3. If the

Old Job Queue always had a job ready to load into core and the CPU was used only for long jobs, the CPU utilization would be 6.0 ÷ 6.3 or about 95%. The CPU utilization will be higher than 95% due to the multi-programming gain of having applied the CPU to short jobs as well as to long jobs.

From the foregoing analysis we conclude that the gain to be made by having a second old job in core would be negligible.

Conclusions

(with 100 users)

1. New jobs are worked off about as fast as they enter the system.
2. The CPU is not used up except during temporary overloads.

(with 150 users)

1. New jobs are worked off about as fast as they arrive.
2. CPU utilization is good.
3. Having a second old job in core would not help much, as the CPU is 90% used up at this point.
4. Response time is acceptable.

Case 2

The second case study addressed the problem of the effects of varying the time slice. This case covered variance of the time slice at 120 users using the Standard Curve. Refer to Table II and Figures 5a-5f.

TABLE II—Case 2—Data table

		Time Slice (Sec.)										
		4.0	3.0	2.5	2.0	1.5	1.0	.933	.8	.7	.6	.4
Percent Utilization	Facility 1	69.9	70.6	70.9	71.6	72.5	72.5	72.9	73.9	74.8	76.0	78.0
	Facility 2	74.2	73.0	71.9	70.6	68.8	64.7	63.3	60.5	58.2	55.5	49.4
	CPU	82.0	82.2	81.9	81.8	81.6	79.5	78.9	77.9	77.5	77.0	76.2
Average Time (Sec.)	Facility 1	66.6	67.1	46.6	35.0	27.7	9.0	7.0	5.4	4.7	4.1	3.4
	Facility 2	.71	.69	.68	.67	.66	.63	.63	.60	.58	.56	.50
	CPU	.77	.77	.77	.76	.76	.72	.71	.68	.67	.65	.63
New Job Queue	Max. Contents	9	8	6	6	5	4	3	3	2	1	1
	Avg. Contents	.87	.66	.56	.45	.33	.20	.16	.10	.07	.03	.00
	Avg. Time (Sec.)	.82	.63	.53	.43	.32	.19	.16	.10	.07	.03	.00
Old Job Queue	Max. Contents	3	3	4	5	7	15	20	25	30	30	33
	Avg. Contents	.50	.49	.85	1.08	1.50	4.48	5.3	6.7	7.3	8.0	9.1
	Avg. Time (Sec.)	45.4	44.6	54.1	51.6	56.0	55.2	50.7	48.3	45.7	50.0	39.8
Mean Response Time (Sec.)	System	2.67	2.41	2.66	2.77	3.08	5.92	6.76	8.1	8.8	9.5	10.6
	Short Jobs	1.48	1.28	1.18	1.06	.93	.74	.68	.58	.52	.46	.40
	Long Jobs	120.1	120.1	107.1	91.4	88.6	66.4	59.7	55.5	51.9	48.4	44.2
Mean Compute Time (Sec.)	System	.612	.612	.612	.612	.613	.613	.613	.613	.613	.613	.613
	Short Jobs	.265	.265	.253	.243	.235	.185	.164	.136	.119	.101	.077
	Long Jobs	33.4	33.4	24.2	18.7	15.1	5.57	4.50	3.60	3.18	2.81	2.37

Case 2. Compute Load = Standard
Time Slice = Variable
Number of Users = 120

Table II. Case 2 - Data Table

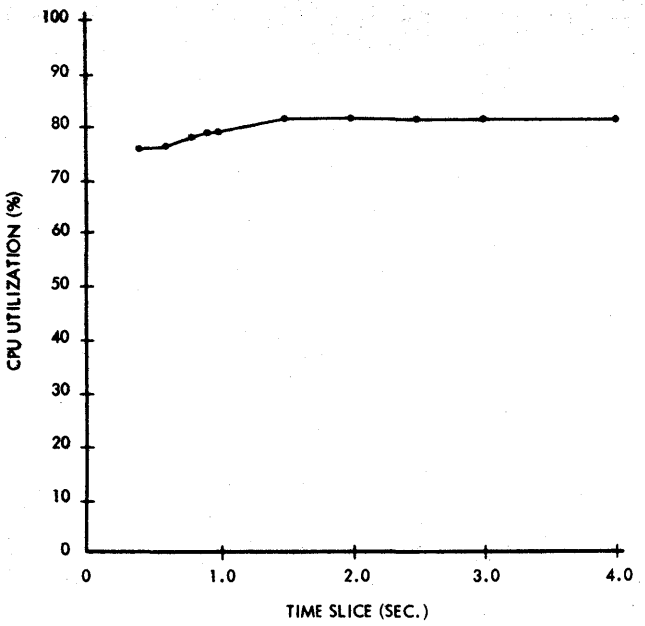


FIGURE 5a—Case 2—CPU utilization

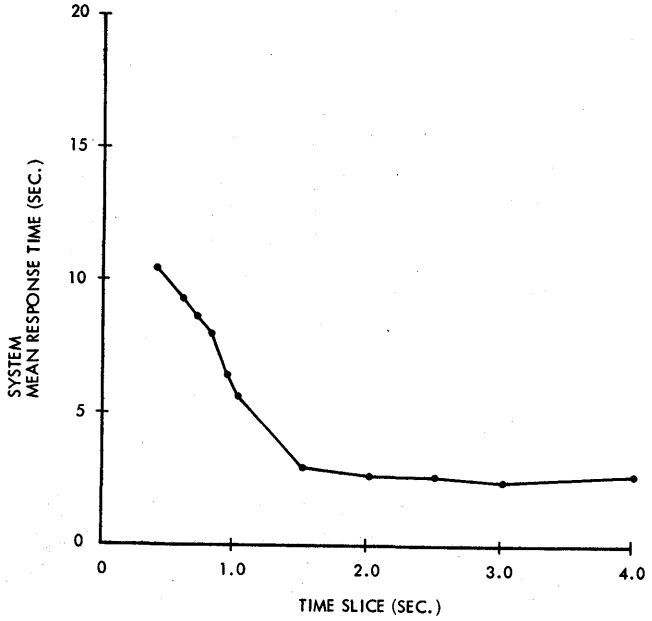


FIGURE 5b—Case 2—System mean response time

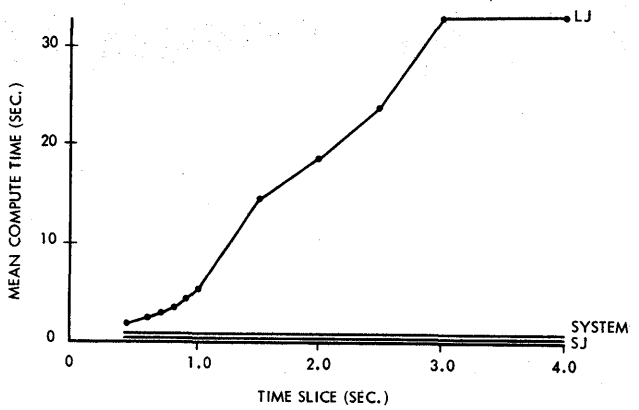


FIGURE 5c—Case 2—Mean compute time

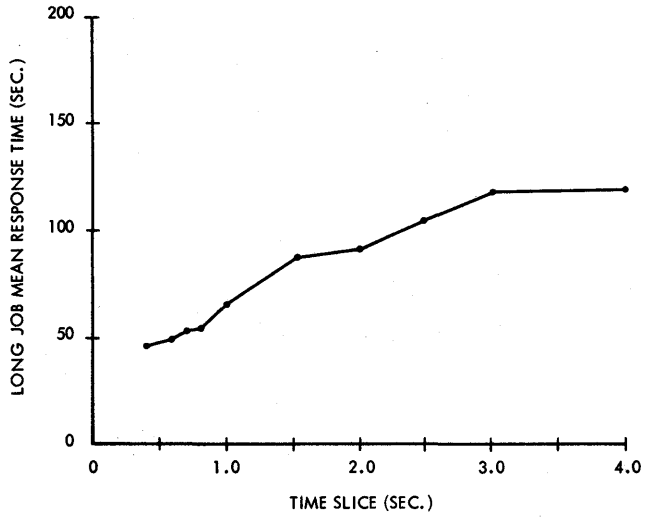


FIGURE 5d—Case 2—Long job mean response time

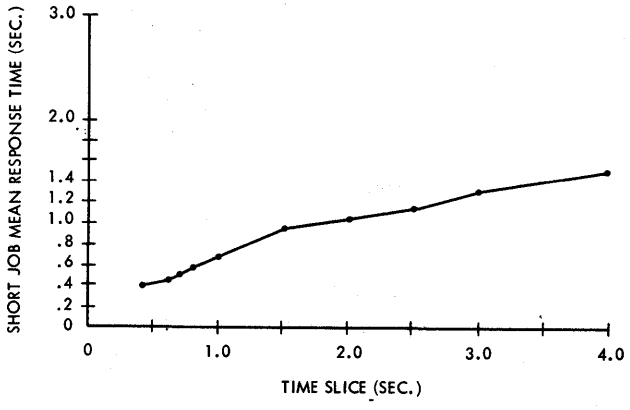


FIGURE 5e—Case 2—Short job mean response time

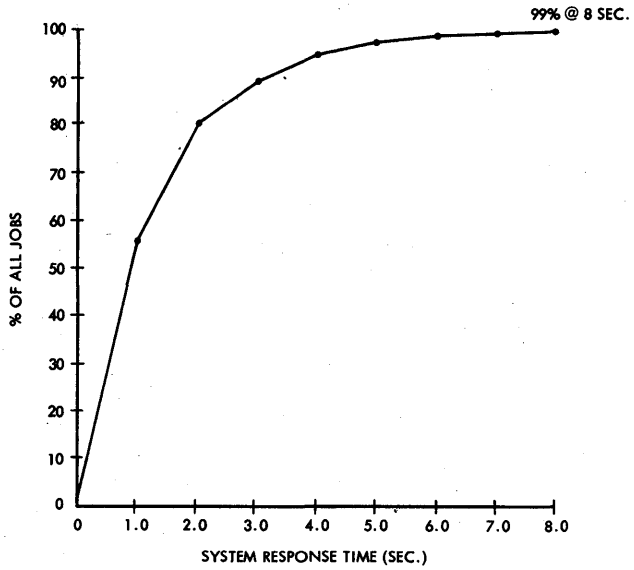


FIGURE 5f-Case 2

Analysis

The SYSTEM MEAN RESPONSE TIME (Figure 5, Part 3) shows that the best response time is at about a three-second time slice. In the compute curve used for this case (Standard Curve), 99% of the jobs will complete in three seconds or less, or within one time slice. The long and short job mean response time curves (Figure 5, Part 4) show that the long job response time is 120 seconds and the short job response time is 1.28 seconds with a three-second time slice.

Since only 1% of the user population will generate a long job, and each user is generating a job every 110 seconds, a single user will generate a long job every 11,000 seconds (110 seconds × 100), or about every three hours.

Conclusions

1. With this particular compute curve, varying the time slice can have a measurable effect on system response.
2. At the point of best system response, the CPU utilization is as good as it can be with this user load.

Case 3

In this case problem the time slice was varied at 140 users using the Standard Curve to determine the effect. Refer to Table III and Figures 6a-6f.

TABLE III—Case 3—Data table

		Time Slice (Sec.)										
		4	3	2.5	2	1.5	1.0	.933	.8	.7	.6	.4
Percent Utilization	Facility 1	86.3	86.6	86.8	87.1	86.7	86.3	86.6	86.9	87.2	87.6	88.9
	Facility 2	85.7	84.3	83.0	81.4	79.3	73.6	71.7	68.2	65.4	62.2	55.3
	CPU	91.9	92.0	91.8	91.7	91.3	89.2	88.6	87.3	86.6	85.7	84.7
Average Time (Sec.)	Facility 1	75.0	75.0	51.2	37.9	29.3	9.9	7.8	5.9	5.0	4.3	3.6
	Facility 2	.71	.69	.68	.67	.66	.63	.62	.60	.58	.56	.50
	CPU	.75	.75	.75	.74	.74	.71	.70	.68	.67	.65	.63
New Job Queue	Max. Contents	13	12	10	9	7	5	5	4	3	2	1
	Avg. Contents	1.9	1.5	1.2	.95	.72	.40	.34	.22	.14	.079	.003
	Avg. Time (Sec.)	1.6	1.2	.98	.78	.59	.35	.29	.19	.13	.071	.003
Old Job Queue	Max. Contents	4	4	5	7	8	25	29	32	37	37	4.4
	Avg. Contents	1.2	1.2	1.9	2.5	3.2	8.8	10.3	12.4	13.5	14.6	16.1
	Avg. Time (Sec.)	94.7	93.7	105.9	104.0	104.1	95.5	86.6	79.6	75.0	69.8	63.2
Mean Response Time (Sec.)	System	3.9	3.5	3.9	4.2	4.6	9.1	10.4	12.4	13.5	14.5	15.9
	Short Jobs	2.2	1.8	1.6	1.4	1.2	.89	.81	.67	.58	.50	.41
	Long Jobs	185.7	183.0	170.0	150.3	141.4	111.1	99.4	89.4	83.4	76.9	68.7
Mean Compute Time (Sec.)	System	.61	.61	.61	.61	.61	.61	.61	.61	.61	.61	.61
	Short Jobs	.26	.26	.25	.24	.23	.18	.16	.14	.12	.10	.077
	Long Jobs	33.4	33.4	24.2	18.7	15.1	5.6	4.5	3.6	3.2	2.8	2.4

Case 3. Compute Load = Standard
Time Slice = Variable
Number of Users = 140

Table III. Case 3 - Data Table

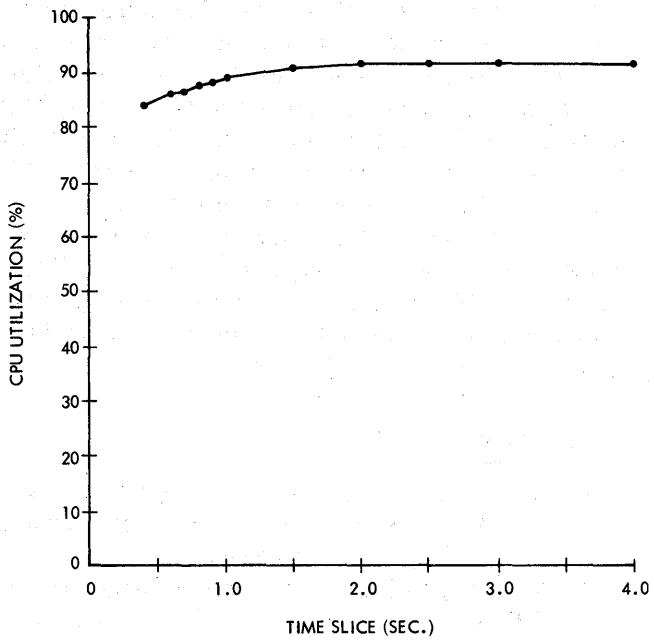


FIGURE 6a—Case 3—CPU utilization

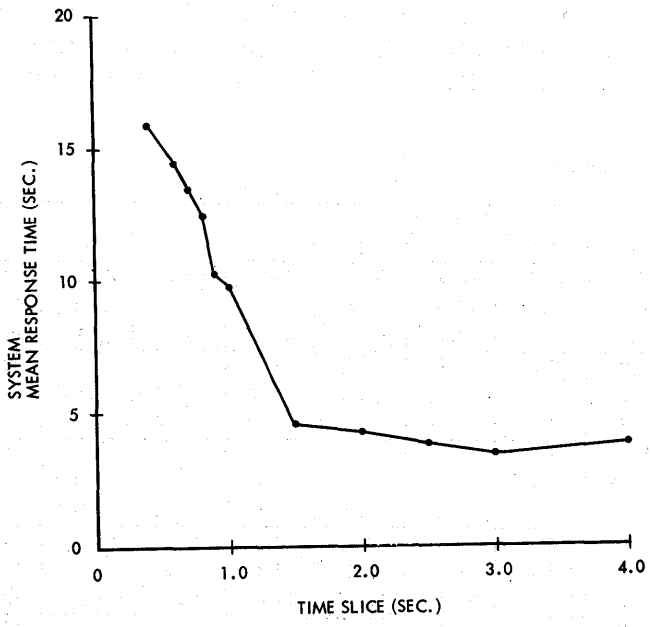


FIGURE 6b—Case 3—System mean response time

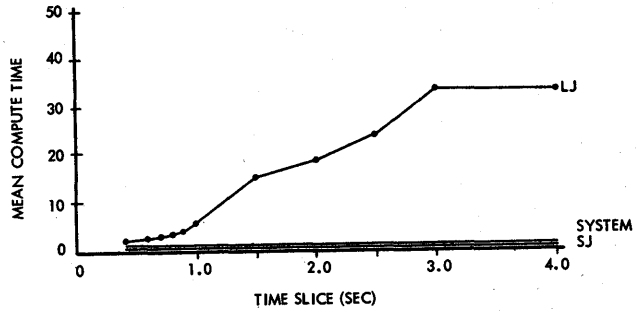


FIGURE 6c—Case 3—Mean compute time

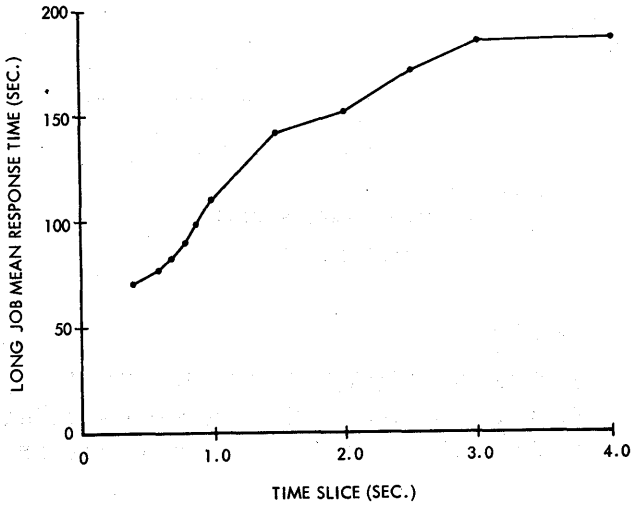


FIGURE 6d—Case 3—Long job mean response time

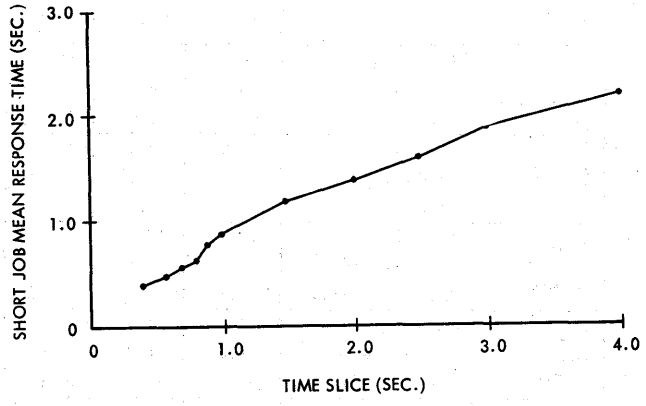


FIGURE 6e—Case 3—Short job mean response time

Conclusions

1. With this particular compute curve, varying the time slice can have a measurable effect on system response.
2. At the point of best response, the CPU utilization is as good as it can be with this user load.

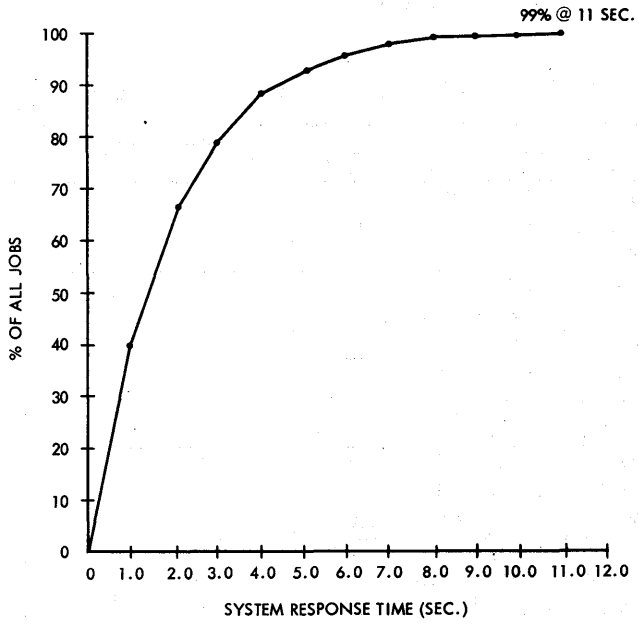


FIGURE 6f—Case 3—Distribution of job response times

		Number of Users										
		10	20	30	40	50	60	70	80	90	100	110
Percent Utilization	Facility 1	7.4	16.8	26.8	37.5	48.6	60.3	71.9	82.5	90.8	94.6	95.7
	Facility 2	4.5	10.1	15.7	21.1	26.6	31.8	36.8	41.1	44.9	48.0	50.3
	CPU	9.4	20.9	32.3	43.5	54.7	65.4	75.5	84.2	90.8	93.9	94.8
Average Time (Sec.)	Facility 1	9.96	10.1	10.3	10.7	11.0	11.4	11.8	12.2	13.0	13.1	13.3
	Facility 2	.625	.625	.625	.625	.625	.625	.625	.625	6.24	6.24	.625
	CPU	1.19	1.17	1.17	1.17	1.16	1.16	1.16	1.16	1.15	1.16	1.08
New Job Queue	Max. Contents	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2
	Avg. Contents	.000	.000	.000	.000	.000	.000	.000	.004	.017	.031	.045
	Avg. Time (Sec.)	.000	.000	.000	.000	.000	.000	.000	.007	.025	.041	.056
Old Job Queue	Max. Contents	2	3	5	7	7	10	11	15	17	28	41
	Avg. Contents	.009	.051	.183	.414	.788	1.49	2.61	4.58	7.81	12.4	18.5
	Avg. Time (Sec.)	1.21	3.09	7.07	11.8	17.9	28.2	42.9	67.2	105.0	156.1	222.0
Mean Response Time (Sec.)	System	1.75	1.97	2.42	2.95	3.62	4.72	6.29	8.81	12.3	17.2	22.8
	Short Jobs	.519	.519	.519	.519	.519	.519	.519	.526	.542	.559	.574
	Long Jobs	12.5	14.6	19.0	24.2	30.6	41.3	56.5	81.7	122.0	176.4	246.6
Mean Compute Time (Sec.)	System	.99	.99	.99	.99	.99	.99	.99	.99	.99	.99	.98
	Short Jobs	.164	.164	.164	.164	.164	.164	.164	.164	.163	.164	.164
	Long Jobs	8.16	8.16	8.16	8.16	8.16	8.16	8.16	8.16	8.13	8.10	8.05

Case 4. Compute Load = Medium
Time Slice = 933 ms
Number of Users = Variable

Table IVa. Case 4 - Data Table

TABLE IVa—Case 4—Data table

3. There is very little difference in varying the time slice at 120 users and 140 users.

Case 4

The Medium Curve (Figure 3) was used in this simulation. The active user load was varied from 10 to 150 in steps of 10. Refer to Tables IV-a-IVb and Figures 7a-7e.

TABLE IVb—Case 4—Data table (continued)

		Number of Users										
		120	130	140	150							
Percent Utilization	Facility 1	96.4	97.0	97.1	97.2							
	Facility 2	52.9	55.6	58.1	60.7							
	CPU	95.5	95.9	96.1	96.3							
Average Time (Sec.)	Facility 1	14.4	14.0	14.1	14.6							
	Facility 2	.626	.63	.63	.63							
	CPU	1.05	10.0	.97	.93							
New Job Queue	Max. Contents	3	3	4	4							
	Avg. Contents	.063	.08	.11	.15							
	Avg. Time (Sec.)	.075	.095	.12	.15							
Old Job Queue	Max. Contents	52	56	64	77							
	Avg. Contents	24.3	29.6	35.2	40.8							
	Avg. Time (Sec.)	275.1	316.1	360.2	398.8							
Mean Response Time (Sec.)	System	23.7	28.1	30.1	30.0							
	Short Jobs	.593	.61	.64	.67							
	Long Jobs	285.2	342.5	385.9	414.3							
Mean Compute Time (Sec.)	System	1.00	1.00	1.00	1.00							
	Short Jobs	.164	.16	.16	.16							
	Long Jobs	8.18	8.12	8.1	8.0							

Case 4. Compute Load = Medium
Time Slice = 933 ms
Number of Users = Variable
Table IVb. Case 4 - Data Table (continued)

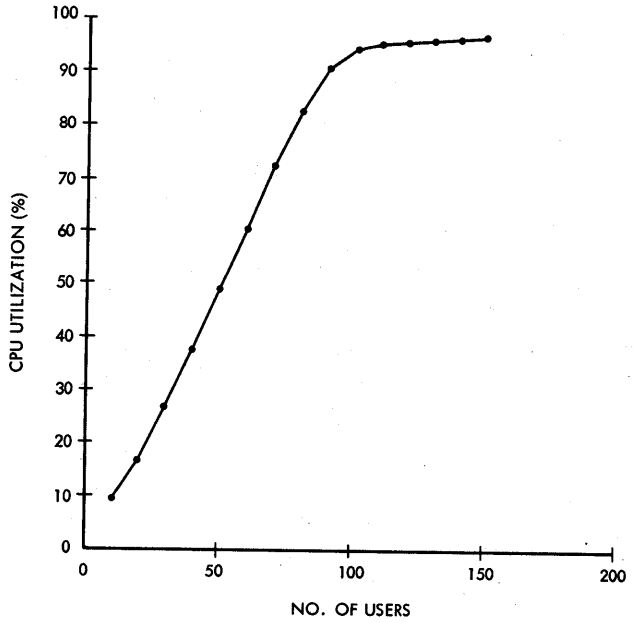


FIGURE 7a—Case 4—CPU utilization

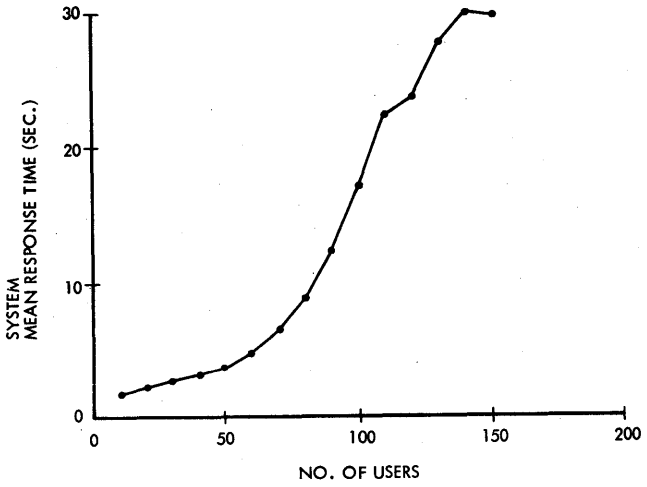


FIGURE 7b—Case 4—System mean response time

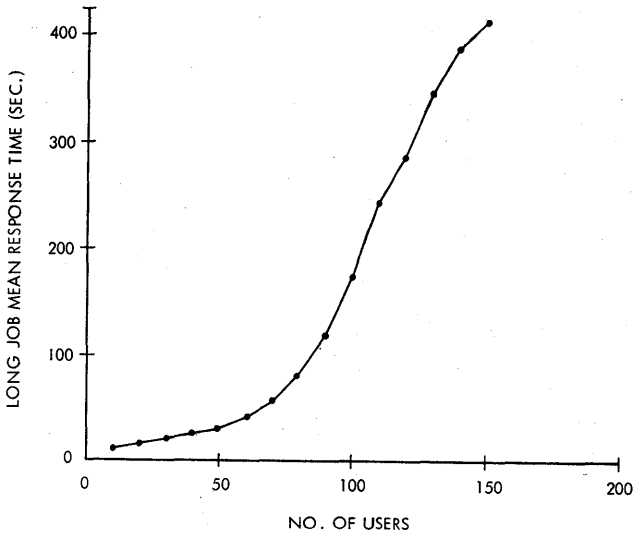


FIGURE 7d—Case 4—Long job mean response time

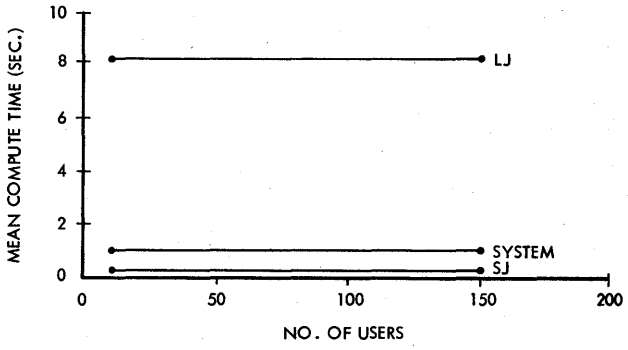


FIGURE 7c—Case 4—Mean compute time

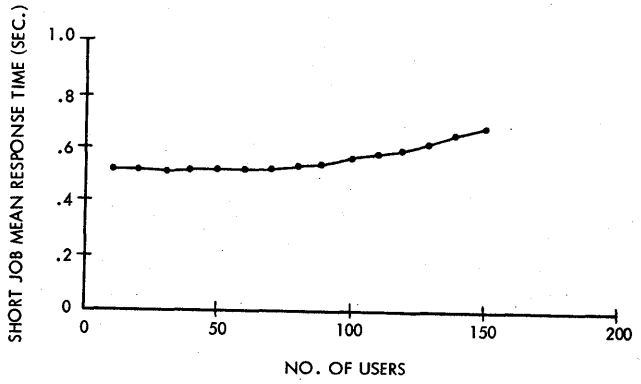


FIGURE 7e—Case 4—Short job mean response time

Conclusions

1. 100 active users is a reasonable upper limit.
2. Short job response times are very good.

Case 5

Using the Medium Curve with a load of 80 active users, the time slice was varied from 4.0 seconds down to 0.4 seconds. Refer to Table V and Figures 8a-8f.

FIGURE 8a—Case 5—CPU utilization

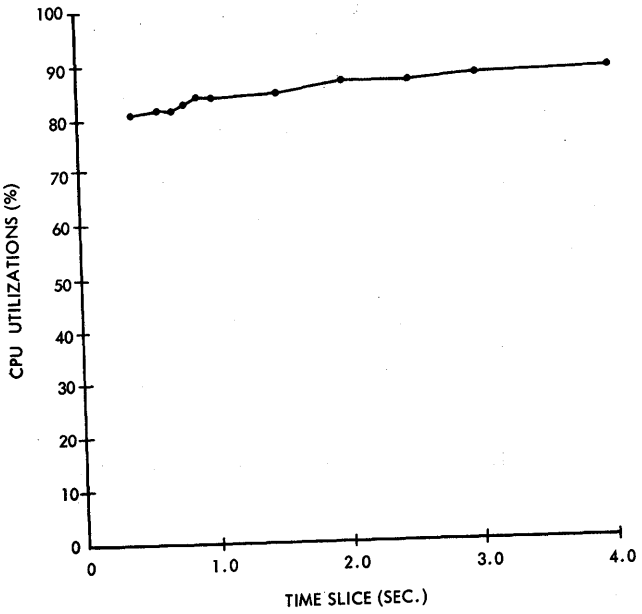
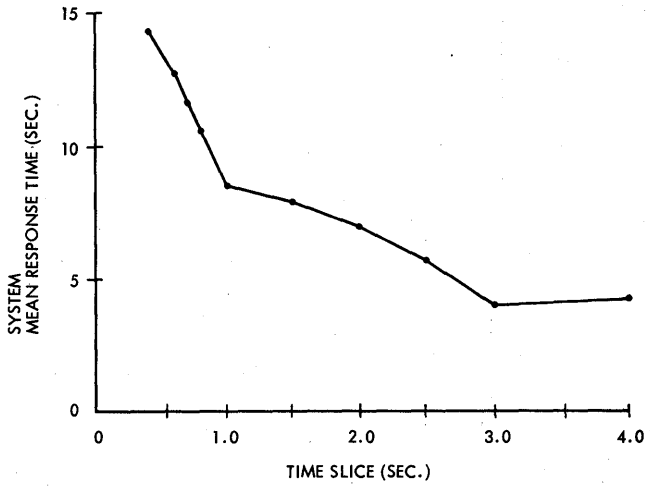


FIGURE 8b—Case 5—System mean response time



		Time Slice (Sec.)										
		4.0	3.0	2.5	2.0	1.5	1.0	.933	.8	.7	.6	.4
Percent Utilization	Facility 1	82.8	83.2	81.9	81.9	81.8	82.4	82.5	82.1	82.2	82.1	82.4
	Facility 2	56.4	54.8	52.5	49.7	46.0	41.7	41.1	39.1	37.6	35.8	31.7
	CPU	87.7	87.8	86.5	85.7	84.7	84.2	84.2	83.1	82.5	81.8	80.7
Average Time (Sec.)	Facility 1	63.5	63.7	30.0	20.7	15.3	12.5	12.2	9.2	7.9	6.8	5.6
	Facility 2	.82	.80	.78	.74	.69	.63	.63	.60	.58	.56	.50
	CPU	1.25	1.25	1.23	1.21	1.18	1.16	1.16	1.13	1.10	1.08	1.04
New Job Queue	Max. Contents	6	5	5	4	2	1	1	1	1	1	1
	Avg. Contents	.41	.31	.23	.15	.067	.01	.004	.000	.000	.000	.000
	Avg. Time (Sec.)	.60	.45	.34	.22	.10	.016	.007	.000	.000	.000	.000
Old Job Queue	Max. Contents	4	4	8	11	13	15	15	17	20	23	26
	Avg. Contents	1.1	1.1	2.5	3.3	4.0	4.5	4.6	5.7	6.3	7.0	7.9
	Avg. Time (Sec.)	81.8	80.8	84.1	80.4	73.2	68.1	67.2	64.0	60.8	57.8	53.6
Mean Response Time (Sec.)	System	4.15	4.0	5.8	7.0	8.0	8.7	8.8	10.7	11.7	12.8	14.3
	Short Jobs	1.33	1.2	.99	.82	.65	.54	.53	.48	.46	.44	.40
	Long Jobs	154.0	151.8	120.3	106.2	92.0	83.0	81.7	75.1	70.4	66.1	60.2
Mean Compute Time (Sec.)	System	.99	.99	.99	.99	.99	.99	.99	.99	.99	.99	.99
	Short Jobs	.32	.32	.26	.22	.19	.17	.16	.14	.12	.10	.08
	Long Jobs	34.8	34.8	17.1	12.8	9.9	8.34	8.16	6.34	5.53	4.82	3.99

Case 5. Compute Load = Medium
Time Slice = Variable
Number of Users = 80
Table V. Case 5 - Data Table

TABLE V—Case 5—Data table

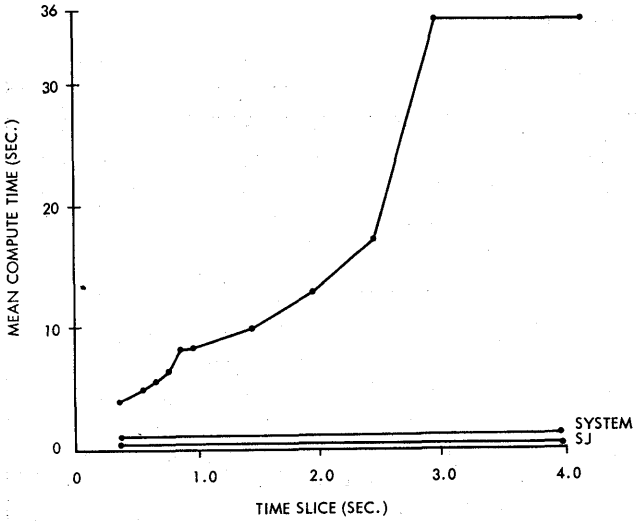


FIGURE 8c—Case 5—Mean compute time

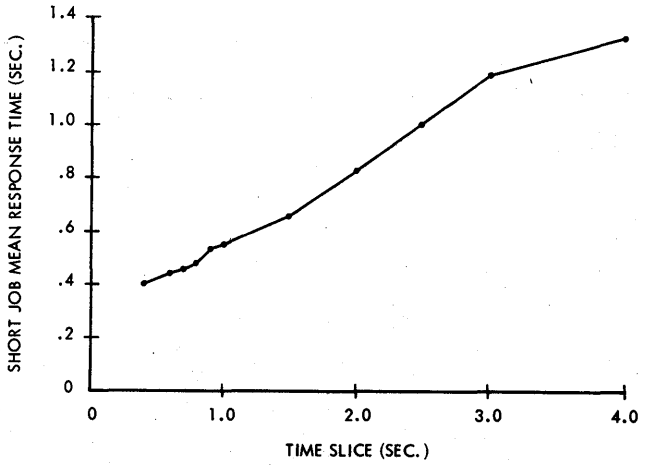


FIGURE 8e—Case 5—Short job mean response time

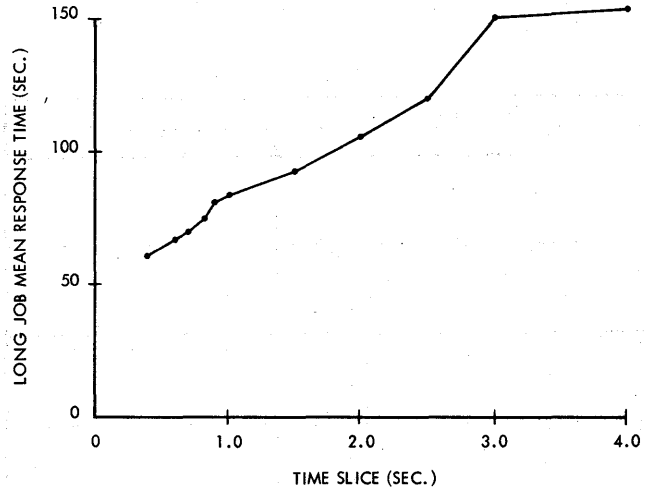


FIGURE 8d—Case 5—Long job mean response time

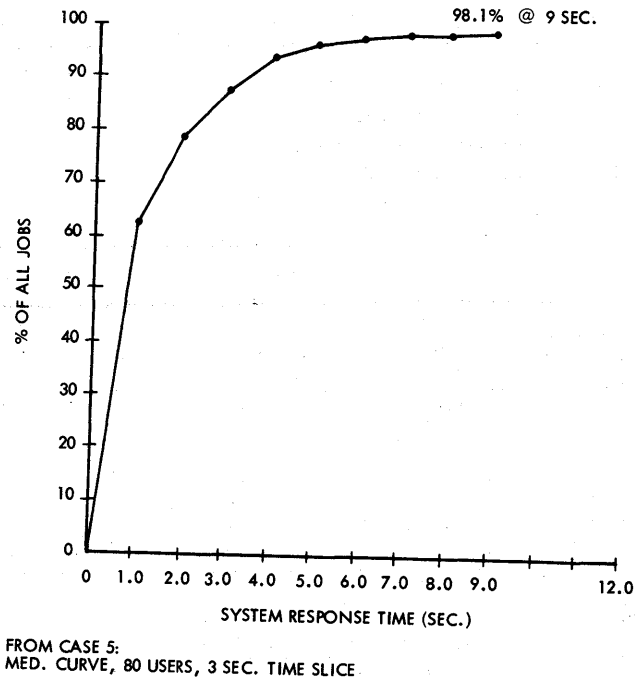


FIGURE 8f—Case 5—Distribution of job response times

Conclusion

A three-second time slice is best.

Case 6

The Heavy Curve (Figure 3) was used in this simulation. The active user load was varied from 10 to 120 in steps of 10. Refer to Tables VIa-VIb and Figures 9a-9e.

TABLE VIa—Case 6—Data table

		Number of Users										
		10	20	30	40	50	60	70	80	90	100	110
Percent Utilization	Facility 1	13.0	29.8	47.3	65.4	82.0	93.3	96.0	96.8	97.3	97.5	97.6
	Facility 2	4.4	10.0	15.4	20.7	25.5	28.8	30.6	32.1	33.7	35.6	37.5
	CPU	14.9	33.1	51.0	68.0	83.3	92.8	95.2	96.0	96.5	96.7	96.8
Average Time (Sec.)	Facility 1	17.6	17.9	18.5	19.3	19.7	20.1	20.5	20.6	22.1	21.7	21.9
	Facility 2	.63	.63	.63	.63	.63	.63	.62	.62	.63	.63	.63
	CPU	1.88	1.87	1.87	1.86	1.85	1.83	1.77	1.71	1.66	1.58	1.51
New Job Queue	Max. Contents	1	1	1	1	1	1	1	1	1	1	2
	Avg. Contents	.00	.00	.00	.00	.00	.00	.00	.00	.003	.006	.012
	Avg. Time (Sec.)	.00	.00	.00	.00	.00	.00	.00	.002	.006	.01	.02
Old Job Queue	Max. Contents	2	3	6	7	8	15	22	34	47	54	64
	Avg. Contents	.01	.09	.33	.99	2.4	6.3	13.3	20.6	28.1	34.7	41.5
	Avg. Time (Sec.)	1.7	5.3	12.8	27.8	56.8	132.3	263.4	387.5	498.0	580.8	656.8
Mean Response Time (Sec.)	System	2.6	3.0	3.9	5.5	8.4	16.2	30.0	40.4	45.7	52.5	56.1
	Short Jobs	.52	.52	.52	.52	.52	.52	.52	.52	.52	.53	.54
	Long Jobs	20.7	24.8	31.4	49.6	78.2	156.2	297.0	433.6	544.3	645.7	726.8
Mean Compute Time (Sec.)	System	1.56	1.56	1.56	1.56	1.56	1.56	1.55	1.56	1.59	1.59	1.58
	Short Jobs	.16	.16	.16	.16	.16	.16	.16	.16	.16	.16	.16
	Long Jobs	13.7	13.7	13.7	13.7	13.7	13.7	13.6	13.7	13.8	13.7	13.6

Case 6. Compute Load = Heavy
Time Slice = 933 ms
Number of Users = Variable
Table VIa. Case 6 - Data Table

		Number of Users										
		120										
Percent Utilization	Facility 1	97.7										
	Facility 2	39.4										
	CPU	97.0										
Average Time (Sec.)	Facility 1	22.4										
	Facility 2	.63										
	CPU	1.4										
New Job Queue	Max. Contents	2										
	Avg. Contents	.02										
	Avg. Time (Sec.)	.03										
Old Job Queue	Max. Contents	75										
	Avg. Contents	48.2										
	Avg. Time (Sec.)	725.2										
Mean Response Time (Sec.)	System	56.5										
	Short Jobs	.55										
	Long Jobs	783.3										
Mean Compute Time (Sec.)	System	1.57										
	Short Jobs	.16										
	Long Jobs	13.5										

Case 6. Compute Load = Heavy
Time Slice = 933 ms
Number of Users = Variable
Table VIb. Case 6 - Data Table (continued)

TABLE VIb—Case 6—Data table (continued)

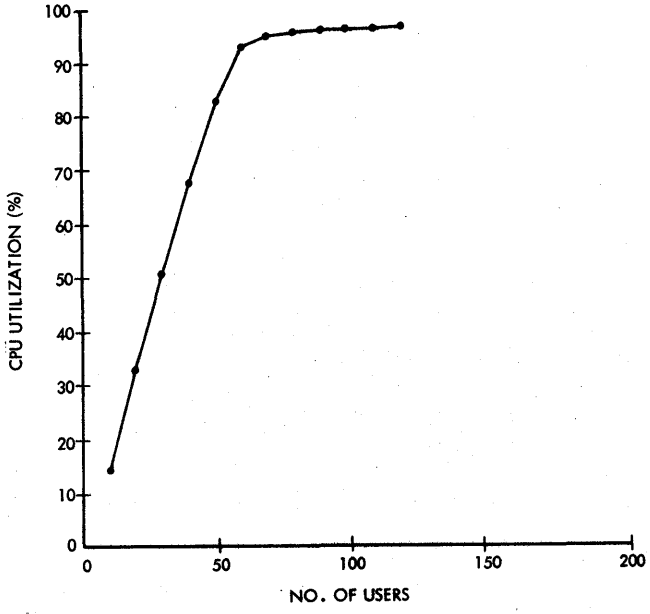


FIGURE 9a—Case 6—CPU utilization

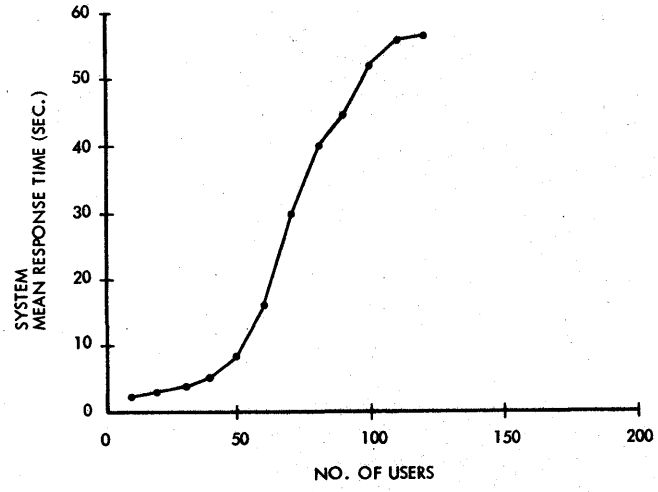


FIGURE 9b—Case 6—System mean response time

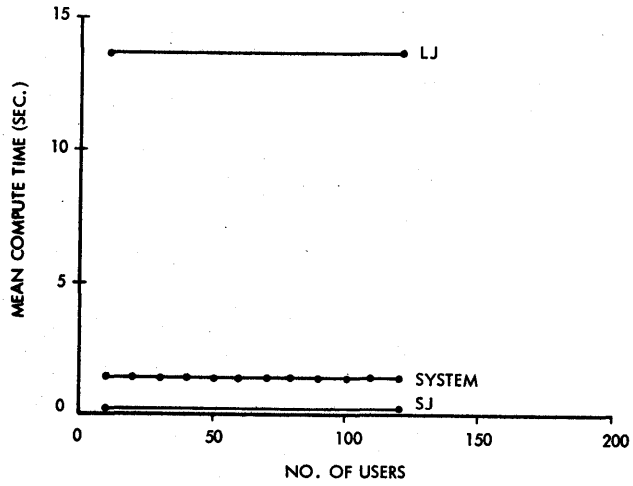


FIGURE 9c—Case 6—Mean compute time

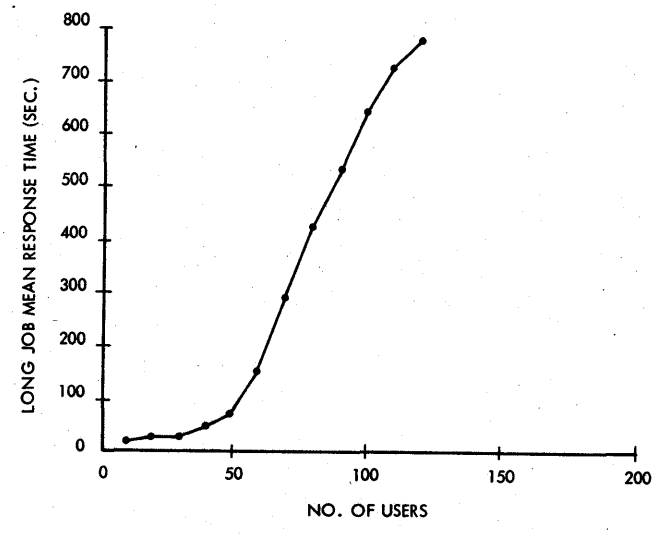


FIGURE 9d—Case 6—Long job mean response time

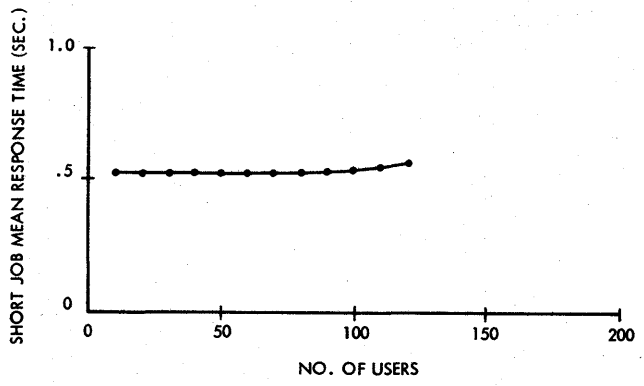


FIGURE 9e—Case 6—Short job mean response time

Conclusion

1. 60 active users is a reasonable upper limit.
2. Short job response time is very good.

Case 7

Using the Heavy Curve with a load of 50 active users, the time slice was varied from 4.0 seconds down to 0.4 seconds. Refer to Table VII and Figures 10a-10f.

TABLE VII—Case 7—Data table

		Time Slice (Sec.)										
		4.0	3.0	2.5	2.0	1.5	1.0	.933	.8	.7	.6	.4
Percent Utilization	Facility 1	81.0	81.8	82.2	82.5	82.2	82.0	82.0	81.2	81.0	80.6	80.3
	Facility 2	36.4	33.5	32.1	30.6	28.5	25.9	25.5	24.3	23.3	22.1	19.5
	CPU	84.7	84.8	84.9	85.0	84.1	83.4	83.3	82.2	81.5	80.6	79.5
Average Time (Sec.)	Facility 1	38.7	37.9	38.0	38.1	26.8	20.6	19.7	14.9	12.8	11.0	9.0
	Facility 2	.88	.81	.77	.74	.69	.63	.63	.60	.58	.56	.50
	CPU	1.94	1.9	1.9	1.9	1.90	1.9	1.9	1.8	1.8	1.7	1.7
New Job Queue	Max. Contents	3	2	2	1	1	1	1	1	1	1	1
	Avg. Contents	.13	.05	.03	.02	.00	0.0	0.0	0.0	0.0	0.0	0.0
	Avg. Time (Sec.)	.30	.13	.08	.04	.00	0.0	0.0	0.0	0.0	0.0	0.0
Old Job Queue	Max. Contents	5	5	5	5	7	8	8	11	11	15	15
	Avg. Contents	1.4	1.5	1.4	1.4	2.0	2.3	2.4	3.1	3.4	3.8	4.4
	Avg. Time (Sec.)	67.3	66.1	65.5	64.8	62.5	57.5	56.7	55.4	52.8	51.3	49.1
Mean Response Time (Sec.)	System	6.5	6.3	6.2	6.1	7.3	8.2	8.4	10.2	11.1	12.2	13.9
	Short Jobs	.92	.74	.69	.65	.56	.52	.52	.48	.46	.44	.40
	Long Jobs	113.1	109.4	108.2	106.9	92.2	80.1	78.2	72.0	67.1	63.6	59.1
Mean Compute Time (Sec.)	System	1.56	1.56	1.56	1.56	1.56	1.56	1.56	1.56	1.56	1.56	1.56
	Short Jobs	.24	.23	.23	.23	.19	.17	.16	.14	.12	.10	.08
	Long Jobs	26.0	25.4	25.4	24.9	18.2	14.3	13.7	10.5	9.1	7.8	6.4

Case 7. Compute Load = Heavy
Time Slice = Variable
Number of Users = 50
Table VII. Case 7 - Data Table

FIGURE 10a—Case 7—CPU utilization

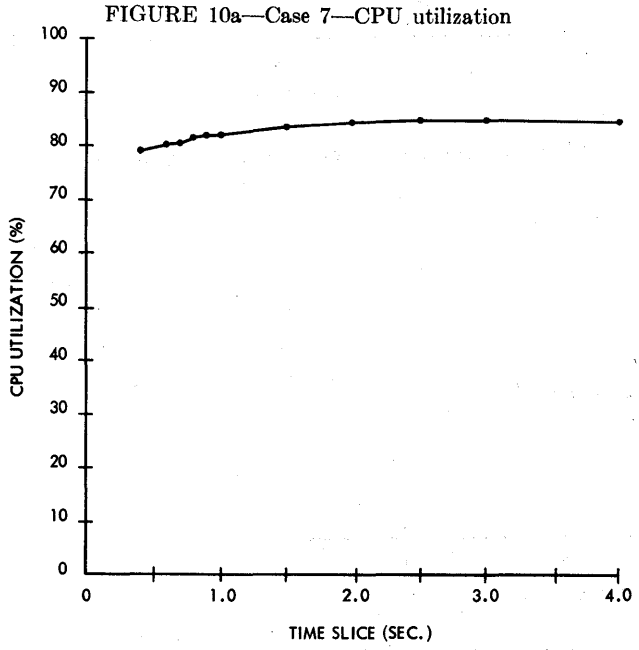


FIGURE 10b—Case 7—System mean response time

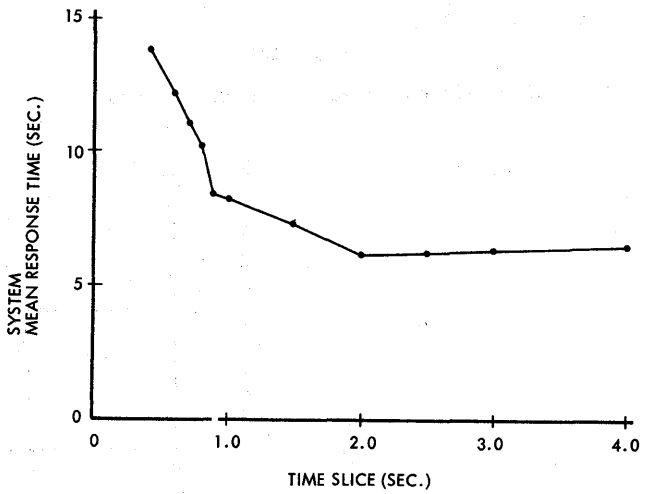


FIGURE 10c—Case 7—Mean compute time

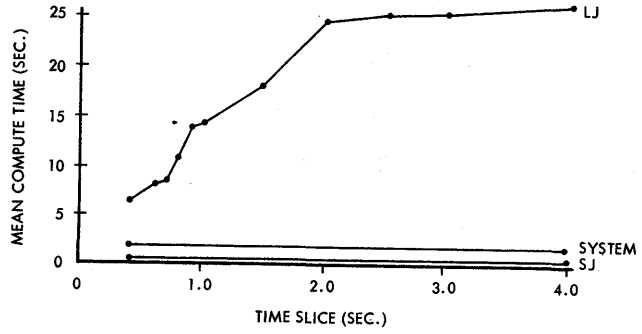


FIGURE 10d—Case 7—Long job mean response time

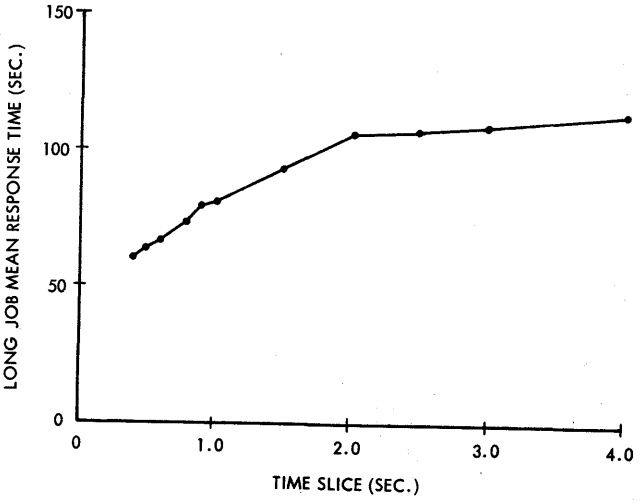


FIGURE 11—User band load

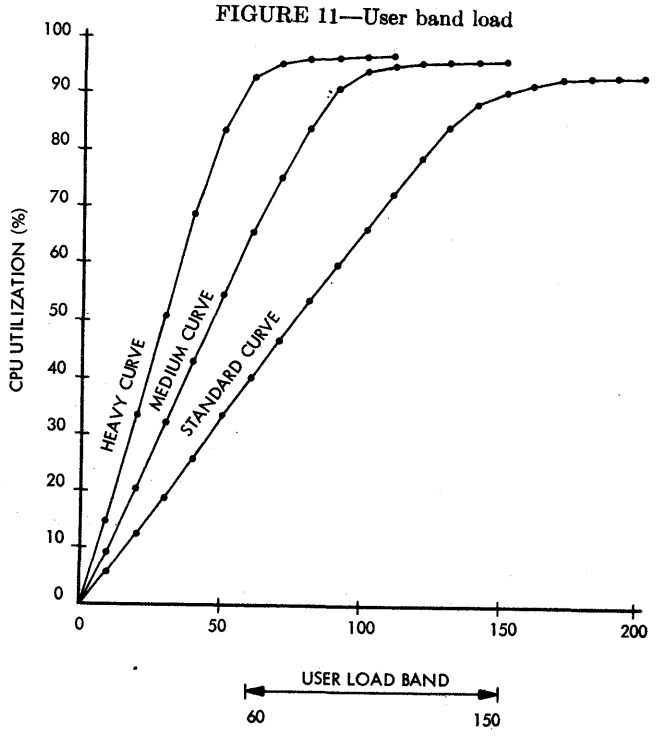


FIGURE 10e—Case 7—Short job mean response time

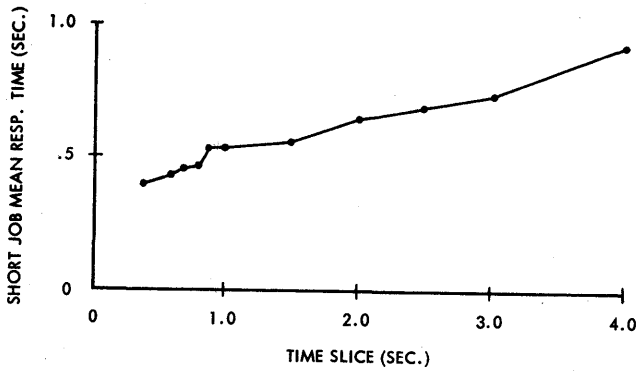
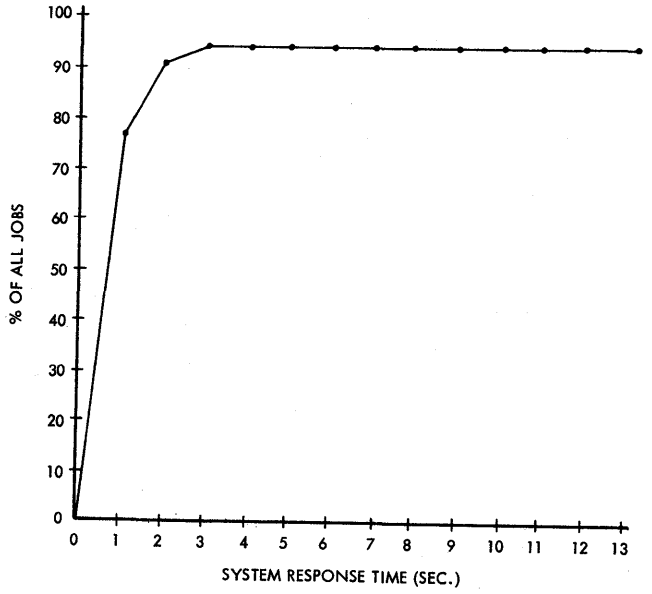


FIGURE 10f—Case 7—Distribution of job response times



Conclusion

A two-second time slice is best.

CONCLUSIONS OF THE SIMULATION STUDY

CPU load curve

The parameter having the most profound effect on the system is the actual compute load generated by the users. This simulation used three different compute load curves. Using any of the curves, the system seems to be able to deliver acceptable response time to the users when CPU utilization does not exceed 90%. If user loads are increased, where 95-98% CPU utilization is seen, response times become unacceptable. As long as CPU utilization is 90% or less, the system has "reserve compute power" to work off temporary overloads.

Figure 11 illustrates CPU utilization versus number of active users for the three compute loads (Standard, Medium, Heavy). Notice that at 90% CPU utilization the system can handle a maximum of about 150 users with the Standard Curve. The Heavy Curve provides a lower limit of about 50-60 users with 90% CPU utilization. We now have a "band" of how many users the system can handle as a function of the CPU load curves.

Time slice

The effect of varying the time slice on system response time is a phenomenon that has not been covered in any known literature. Up to now there have been only opinions as to what the time slice should be. It is interesting to note that the best time slice seems to be that value which maximizes the CPU utilization.

BIBLIOGRAPHY

1 H CANTRELL
Presentation given at Workshop on Models for time-shared processing at the symposium on computers and communication, their system interaction
Sponsored by the IEEE Communications Technology Group and the Computer Group January 1967 in Santa Monica California unpublished

2 G M AMDAHL
IBM Fellow
Menlo Park California

3 *General purpose simulation system 1360 user's manual*
IBM H20-0326-0

4 A L SCHERR
An analysis of time-shared computer systems
June 1965

Performance of a simulated multiprogramming system

by MEIR M. LEHMAN and
JACK L. ROSENFELD

Thomas J. Watson Research Center
Yorktown Heights, New York

INTRODUCTION

In a recent paper,[1] A. Scherr discussed main storage fragmentation and other aspects of the behavior of the MVT option of OS/360 (Multiprogramming with a Variable Number of Tasks). The behavior of MVT running on a System/360, Model 65 CPU was deduced from observations and measurements on a model.

The model simulates a number, fixed for each run, of initiator-terminators (I/T's) each of which accepts a job request from an input stream and retains control of its execution until completion. On completion of any job, the I/T that had controlled it accepts a new job from the input stream and remains with that job until it, in turn, terminates. The storage and CPU power required by the I/T's themselves were ignored in the model.

Each job consists of several job-steps to be executed in sequence. Each job-step is described in terms of three attributes, storage space required, execution time and wait factor. The execution time is the total CPU time required by the step, that is the CPU time that would be observed if the job-step were running alone in the system. The wait factor is the percentage of its running time (total core residence time) that a job-step would be in the wait state (for I/O, for example) if it were running alone in an actual system.

A job-step that has been initiated will either be waiting for main storage space or may have had main storage space allocated. In the latter case, it will share the CPU power with other job-steps then resident in core. The maximum level of multiprogramming will thus be limited by either the number of I/T's or by the size of the store.

All I/T's controlling job-steps awaiting the availability of an appropriately sized block of storage are queued. When storage becomes available on termination of a job-step, an attempt is made to fit each waiting job-step into storage on the basis of a first come, first served discipline. The simulated system satisfies each request for storage from the *first* empty region found that is large enough. The search for an empty region proceeds from the top of main storage. Upon allocation an empty region is generally split into two regions: one, at the top end, being allocated to the job-step, and a new, unused region being created at the bottom end. As soon as a job-step terminates, its storage is released and is coalesced with any neighboring free storage. The next step for that job, or the first job-step of a new job, is obtained, and its initiator is placed at the bottom of the queue waiting for main storage. It is, of course, possible that this new request may be immediately granted.

The job-step that has been in storage longest is given the highest priority for execution, and runs as if it were alone. Thus, its run time equals the specified execution time divided by one minus its wait factor. Run times for other jobs are multiplied by expansion factors (see Table 1) obtained from a simple Markov model, to allow for the fact that each job can run only when all jobs of longer residency in storage are in the wait state. Thus, the expanded execution time in the model represents the sum or the execution time and forced idle time due to CPU unavailability in an actual system.

Scherr's experiments were performed while executing a "standard" job-stream containing FORTRAN, COBOL and Sort jobs. The mix from which the stream was generated contained 21 jobs comprising a total of 97 job-steps (compile, link-

TABLE 1—Expansion factor

Level of Multiprogramming	EXPANSION FACTOR	
	65% Wait Time	90% Wait Time
1	1.00	1.00
2	1.19	1.01
3	1.63	1.02
4	3.13	1.04
5	12.56	1.06
6	230.61	1.09
7	over 1,000	1.14
8	"	1.20
9	"	1.30
10	"	1.48
11	"	1.84
12	"	2.83
13	"	6.96
14	"	47.82
15	"	over 1,000

edit, or go). Average job-step core requirement was 100K with the actual distribution shown in Table 2.

TABLE 2—Core requirements of job-steps in the standard mix

Core Size Required (bytes)	Number of Job-Steps
230K	3
220K	15
88K	36
17K–83K	43

Average job-step execution time was 62.8 seconds, and Scherr used a constant wait factor of 65% in all his experiments. Scherr mentions that this level is typical for many Model 65 installations. The job-stream was obtained by random selection of a job by an I/T from an infinite pool of jobs distributed as in the mix whenever the I/T terminated its previous job assignment.

Scherr's main conclusions

Scherr presents a number of conclusions:

a. System *throughput* is basically unaffected by storage fragmentation, since given enough I/T's and storage space, the algorithms used ensure that fragmentation merely affects the *order* in which jobs are completed.
b. No large jobs were delayed indefinitely during any of the simulation runs, despite the fact that the allocation procedure favors job-steps requiring little storage.
c. Dynamic relocation, modelled in these experiments by single register relocation, does not appreciably increase throughput.
d. Other job-streams with larger, longer-running steps were simulated without yielding appreciably different results. As the number of I/T's was increased, the average number of job-steps held in core invariably approached the ratio of the size of allocatable storage to the time-average job-step size. Maximum main storage utilization rarely left the 80-90% range.

Extensions to the experiments

The present authors raised the following questions not addressed in Scherr's study:

a. Can more discrimination be obtained in the model, so as to enable observation of system behavior when it is storage—rather than I/T- or CPU-limited?
b. What are the maximum delays and the distribution of delays suffered by jobs? How are these related to job storage requirements, job execution times, the number of I/T's, and system storage capacity?
c. What is the time behavior of the model?
d. How does system behavior change when the core requirements of individual job-steps are chosen from continuous rather than from discrete distributions? In the mix used by Scherr, for example, 15 job-steps required exactly 220K bytes (Table 2).
e. While dynamic relocation has no significant effect on system throughput and storage utilization, how does it affect relative performance on the various jobs of the job mix?

New experimental assumptions

Run time

In the course of extending Scherr's investigation, some changes were made in the values of the parameters used. In particular, most of our runs were undertaken with a fixed execution time for all job-steps, equal to the mean run time of the standard mix. This was done to permit analysis

as a function of only one variable, the storage requirement.

The measured effect of using a constant run-time for all job-steps was to increase throughput by 19% at most. There are two basic reasons for this increase. The first is that when all job-steps require the same amount of processing time, the job-steps terminate in precisely the same order in which they are allocated. Thus, a regular pattern of storage allocation is established. This pattern favors the fitting of job-steps into core. Secondly, in the standard job stream, some of the job-steps requiring the smallest amount of storage are those requiring large amounts of processing time. When allocations are made to these job-steps, storage tends to remain fragmented for long periods of time, decreasing throughput when the system is storage-limited. Despite the assumption of constant execution time, however, the results reported are qualitatively correct, as evidenced by runs in which the assumption was discarded.

Wait factor

The 65% wait factor used by Scherr, although determined to be realistic for the job load and the S/360, Model 65 hardware configuration considered by Scherr, does not yield much discrimination. That is, the CPU approaches 100% utilization with only 4 or 5 job-steps in storage, and performance measurements increase very rapidly as the number of job-steps in storage changes. To increase discrimination and to reveal more of the fundamental properties of the system, most of the present experiments were first performed with a wait factor of 90%. Confirming experiments were then undertaken at the 65% wait level, and also for the actual run times specified in the job mix.

Initial transients

Early experiments indicated strong initial transients. Thus, the first 5000 job-steps were generally processed without statistics-gathering. Measurements were then taken over the next 10,000 job-steps. These figures suggest that an actual installation would not necessarily ever reach steady-state. They were used, nevertheless, to enable the determination of some system mean-performance values. However, the cyclic behavior subsequently discovered and described below indicates that the so-called, start-up transient were, in fact, part of the general cyclic pattern. The results of some further runs support this as illustrated for example in Figure 5.

Predicted results

Among the quantities measured were turn-around time, delay to job-steps, number of job-steps allocated to core storage, core storage utilization and processor utilization. These quantities are all closely interrelated, and derivations of their relationships are given in the Appendix. A summary of the relationships follows:

(1) Throughput is given by

$$TP = 3600/\bar{T}, \text{ job-steps per hour}$$
$$= 779/T \text{ jobs per hour,}$$

where $\bar{T}$ represents the mean execution time for a job-step in seconds. This is valid only when the amount of core storage and the number of I/T's are large enough to make CPU power the limiting resource.

(2) Turn-around time (TAT) for a job-step is measured from the time when the I/T for the job-step initially requests core storage allocation to the time when processing for the job-step is complete. The mean job-step turn around time is given by

$$\overline{TAT} = N_{it} \times 3600/TP \ (= N_{it} \times \bar{T} \text{ when CPU-limited})$$

where TAT is in seconds and N_{it} is the number of I/T's

(3) The delay to a job-step is the difference between the turn-around time for that job-step and the time in which the job-step would have been completed had it been the only one in the system This is given by

$$\overline{DELAY} = \overline{TAT} - RT_{js},$$

where RT_{js} is the mean job step run time. The average delay to all job-steps, as the amount of core increases, approaches $\bar{T}(N_{it} - 1/(1 - w))$

(4) Let L be the effective level of multiprocessing ($L \leq N_{it}$), that is the number of job-steps that receive, say, at least 1% of CPU time when the average CPU utilization is, say, 99% (See Table 1 and Appendix). Also let the mean storage requirement per job be $\bar{S}$. Then for a total (job-available) storage capacity in excess of LS, the

system will increasingly tend to delay job-steps requiring small amounts of core storage as a result of being able to more effectively process job-steps requiring large amounts of core storage. In fact, delays to all job-steps will approach the general mean delay as in (3) asymptotically. Thus storage capacity (job-available) of order $\overline{LS}$ represents a point of discontinuity of the system job-delay characteristic as a function of available storage.

Results

Throughput as a function of core size

Typical of the type of results obtained is the plot of throughput vs. core size shown in Figure 1. This is shown for values of the wait factor, 0.65 and 0.90 to illustrate the effect that increasing the wait factor has in expanding the horizontal scale, thus increasing discrimination. Note that as core size increases, throughput increases, rapidly at first but then more slowly as the additional job-steps for which core storage is allocated contribute less and less to throughput. This decreasing contribution to throughput is implied by Table 1, which shows that the first few job-steps allocated use virtually all the CPU resources. The asymptotic behavior of the curves occurs because the system is CPU-limited as core size increases. The asymptotic value of the measured throughput is 57.4 job-steps per hour, which agrees with the predicted value discussed in an earlier section.

Turn-around time as a function of core size

Figure 2 shows measured turn-around time and delay as functions of core size. Figures 1 and 2 verify the predicted inverse relationship between mean turn-around time and throughput. Job-step turn-around time distributions as a function of total available storage are given in Tables 3, 4 and 5. For Table 3, wait time is 90%; in Table 4 and 5 it is 65%. For Tables 3 and 4 all job-steps require 62.8 seconds of CPU time, that is turn-around time when running alone in the system would be 628 seconds and 180 seconds respectively. For the experiments of Table 6 actual job-step times were used. Thus the first two tables

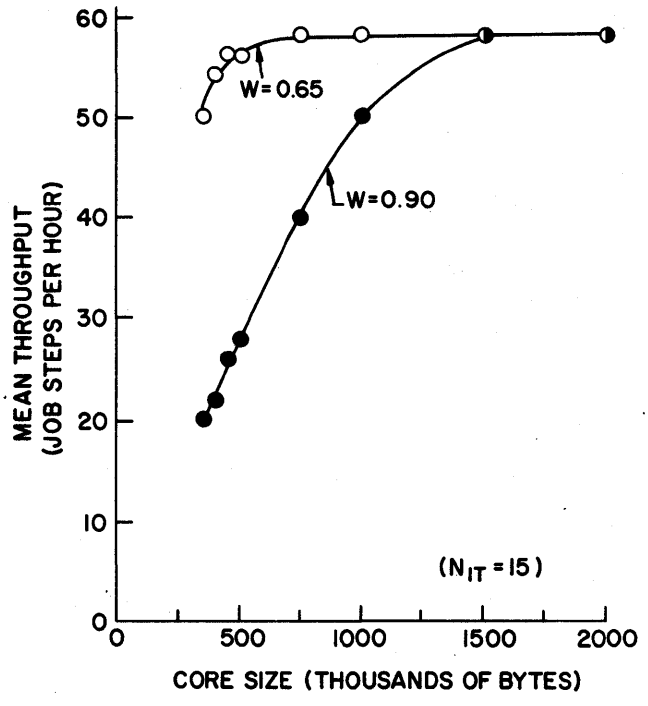


FIGURE 1—Throughput vs. core size

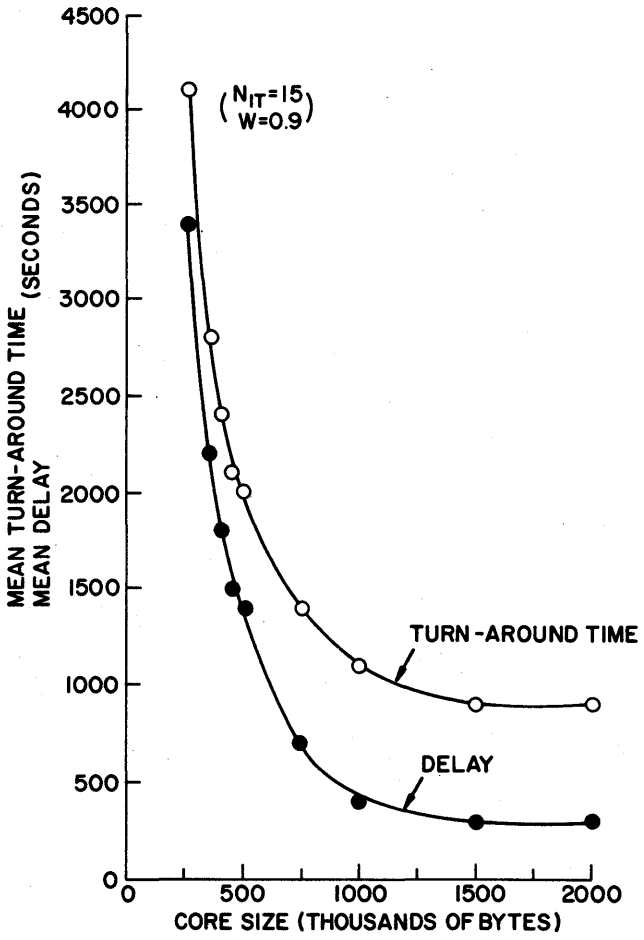


FIGURE 2—Turn-around time and delay vs. core size

TABLE 3—Percentage of job steps with turn-around time in each region (constant job-step time = 62.8 seconds, w = 0.9)

core size (in bytes)	Turn-around Time (in seconds) 600–800	800–1000	1000–1200	1200–1400	1400–1600	1600–1800	1800–2000	>2000	worst case (in seconds)
250K	35.4	0.2	1.7	8.7	0.3	0.2	4.5	49.0	26028
350K	37.3	1.4	2.5	7.7	0.9	0.9	6.2	43.2	23755
400K	31.0	4.4	3.6	10.6	3.7	2.4	8.8	36.5	28466
450K	35.1	4.6	3.8	10.0	3 9	2.7	7.3	32.8	29748
500K	38.5	6.6	5.9	11.4	5.3	3.3	4.4	24.7	31120
750K	45.8	10.9	7.3	7.7	4.3	3.8	2.8	17.4	10283
1000K	42.1	23.6	11.4	5.1	2.7	3.3	3.6	8.3	9321
1500K	1.9	85.1	9.5	2.7	0.7	0.0	0.0	0.0	1481
2000K	0.0	98.2	1.6	0.2	0.0	0.0	0.0	0.0	1383

TABLE 4—Percentage of job steps with turn-around times in each region (constant job-step times = 62.8 seconds, w = 0.65)

core size (in bytes)	Turn-around Time (in seconds) 0–200	200–400	400–600	600–800	800–1000	1000–1200	1200–1400	1400–1600	1600–1800	1800–2000	>2000	worst case (in seconds)
250K	2.3	33.2	8.4	5.4	3.4	2.6	2.5	3.3	3.3	4.5	31.3	8900
350K	0.3	35.1	13.6	7.8	7.5	9.1	7.1	2.0	0.4	0.5	16.7	10457
400K	0.0	26.5	18.4	14.5	11.5	8.1	3.6	1.3	0.9	1.0	14.2	10150
450K	0.0	28.1	19.4	13.3	9.8	7.6	4.6	2.0	1.5	1.6	12.2	13362
500K	0.0	29.7	22.4	14.1	8.8	6.0	2.7	1.1	1.3	1.3	12.6	15499
750K	0.0	3.4	32.0	24.0	12.3	10.2	1.7	1.8	2.9	2.8	8.8	8608
1000K	0.0	0.0	8.4	41.3	23.6	10.0	2.3	3.4	3.5	3.2	4.3	6471
2000K	0.0	0.0	0.0	0.0	98.7	1.2	0.1	0.0	0.0	0.0	0.0	1319

TABLE 5—Percentage of job steps with turn-around times in each region (job-step times given in standard mix, w = 0.65)

core size (in bytes)	Turn-around Time (in seconds) 0–200	200–400	400–600	600–800	800–1000	1000–1200	1200–1400	1400–1600	1600–1800	1800–2000	>2000	worst case delay (in seconds)
250K	21.1	15.0	10.5	8.9	3.5	2.1	2.5	2.2	1.8	1.7	30.6	8094
350K	18.0	19.6	16.2	12.4	7.2	3.6	3.5	2.0	1.1	1.0	15.4	12956
400K	22.6	23.3	15.6	9.9	5.0	3.0	2.0	1.2	0.9	0.9	15.6	14834
500K	12.2	23.0	17.1	11.7	7.4	4.3	3.7	2.6	2.0	1.5	14.6	10383
750K	2.9	19.0	23.3	15.6	10.0	6.7	3.8	3.2	2.5	2.1	11.0	7314
1000K	0.4	11.4	24.9	20.2	12.7	8.1	5.1	3.7	2.9	2.2	8.5	4930
1500K	0.0	1.6	16.1	27.7	21.0	13.0	8.4	5.0	2.9	1.8	2.5	3069
2000K	0.0	0.6	14.4	27.6	22.7	15.3	8.8	5.2	2.3	1.4	1.8	2037

clearly bring out the delay behavior of the system as a function of available storage, a pattern less clear in the third table.

The final columns of each table give the worst case turn-around time for each core size. We discuss this quantity further in a later section.

In Table 3, one observes that under core-limited conditions there are three concentrations of delay, in the ranges 600-800, 1200-1400 and in excess of 2000 seconds. We indicate in the next section that these correspond to three classes of job-steps as grouped by their core requirements.

The data in Tables 3 and 4 illustrate clearly the very large delays suffered under space-limited conditions by the job-steps requiring large amounts of core storage. Moreover, the abrupt change in behavior for a core size around 1400K for Table 3 and 500K for Table 4 corresponds to the behavior as discussed in a previous section. More, particularly, for the present job-mix the average core requirement per job-step ($\bar{S}$) is 100K bytes. Moreover for w=0.9 the effective level of multiprocessing (L) is about 14 and for w=0.65 L is between 5 and 6 (Table 1). Thus $\overline{LS}$ for these two cases is 1400K and between 500K and 600K respectively. The predicted discontinuity in the system delay pattern is clearly confirmed in Table 3 and 4. Also, the turn-around time concentration in the 800-1000 second region corresponds to the limiting TAT given in an earlier section. Thus, job-steps requiring small amounts of storage are actually delayed more when storage is freely available and a high degree of multiprocessing occurs then when the system is storage limited. Table 5 gives data which could be compared to actual data obtained in an MVT system appropriately loaded to test the accuracy of the model.

An important practical conclusion from these results is that merely increasing core size when the system is CPU-limited has its dangers since it increases delays to some job-steps and changes the general pattern of delay behavior.

Time dependencies

An interesting aspect of this study, and one that explains the delay phenomena discussed above, has been the observation of detailed queueing behavior with time. The particular nature of the standard job mix leads to a certain pattern of queueing behavior. Table 2 breaks down the mix by core requirements. The critical aspect of this mix is the number of job-steps requiring large core storage of only two distinct sizes, 220K (these will be called large job-steps), with the remaining job-steps considerably smaller. The consequence of this fact is that the allocation pattern is strongly controlled by the presence or absence of job-steps requiring large storage. When processing for a job-step of a certain size is complete, another job-step waiting for storage allocation and requiring the same amount of storage can fit into the same place.

Consider the situation in which the system has a relatively small amount of core storage—say 350K. Only one large job-step at a time can fit into core storage. Any other large job-step injected into the system must wait in the queue for the availability of an adequately-sized storage block. Suppose now that at some moment a large job-step has been in core and is terminated at a moment when no other large job-step is waiting *at the head of the queue.* Storage will then be allocated to the small job-steps heading the waiting queue. Even one such job-step will be likely to fragment storage so that no other large job-step lower down in the queue can then be allocated storage. As a result, core storage will continue to be allocated only to small job-steps. Gradually the number of large job-steps waiting to be allocated will increase, because every so often a large job-step appears in the job-stream. These queued large job-steps will move to the top of the queue as the smaller jobs are worked off.

Eventually, one of the large job-steps—the top one in the queue in general—will be allocated core space, because either all the jobs waiting for storage are large job-steps or because of some fortuitous release of large amounts of storage. When this large job-step's processing is complete, it will release its large block of storage, and the next large job-step in the queue will be allocated the same block of storage, and so on, since the quantized nature of the large requests almost guarantees fit. Thus most of the large job-steps that have been waiting a long time for storage allocation will have been worked off entirely. However, there may be some exceptions left on the queue, because in the presence of predominantly 220K job-steps one cannot be certain that a 230K step may not be further delayed.

One or more small job-steps will be allocated storage when at least all 220K steps have been processed, and this will reduce the available stor-

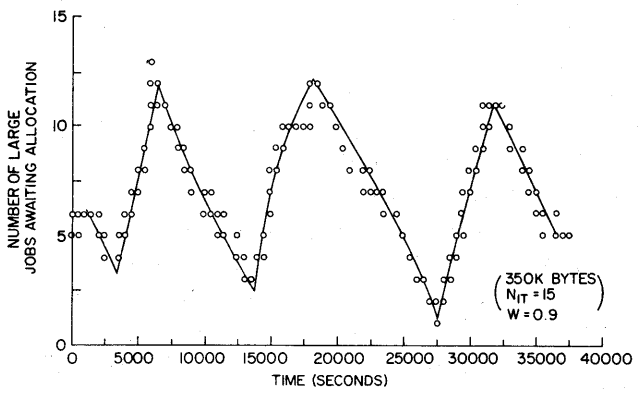


FIGURE 3—Queue of large job steps vs. time for 350K bytes

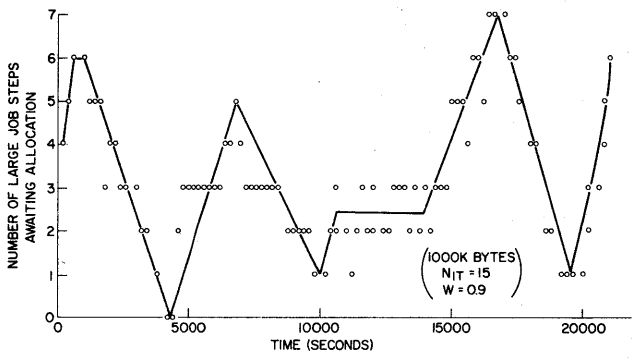


FIGURE 4—Queue of large job steps vs. time for 1000K bytes

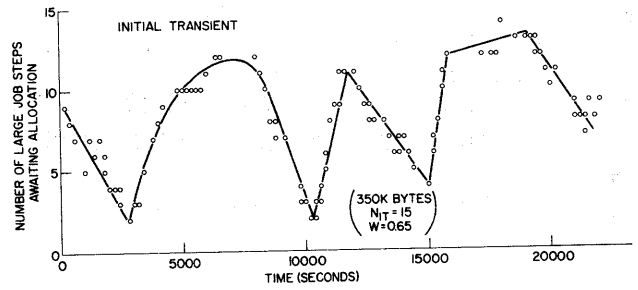


FIGURE 5—Queue of large job steps vs. time-initial transient

age below the critical level. The number of large job-steps in the queue will then begin to build up once more. The cycle of building up the queue of large job-steps and working it off continues indefinitely.

Clear evidence of the cyclic behavior is presented in Figures 3, 4, and 5 for various combinations of load parameters. Each shows several cycles of the variation in numbers of large job-steps waiting for core allocation vs. time. Note that in Figure 3 the period is approximately *four hours* for job-steps requiring some ten minutes of core residence time. For larger core sizes, the period and amplitude of the cyclic pattern gradually decreases, because in systems with larger amounts of core storage, the large job-steps are more easily allocated. This is illustrated in Figure 4, where the system core size is 1000K, compared to the 350K of Figure 3. Note that the scales are different on the two figures. Figure 5 shows the queue of large job-steps from time zero, illustrating that the cyclic behavior occurs from the beginning of a run. For this figure, w= .65, and actual job-step times from the standard mix were used.

Some general calculations based on mean levels of multiprogramming achieved with loads having attributes which correspond to the average levels of the present load have shown that the orders of magnitude of the amplitude and period of the oscillation are predictable. We do not reproduce these calculations here.

It is this cyclic behavior which causes differential treatment of jobs and, as will be shown in the next section, leads to very long delays for job-steps having a large storage requirement. These results suggest changing the present allocation strategy of MVT. For example, the pure FIFO scan of I/T's when allocating storage might be modified by selective scanning. In the simplest plan, when a large area is released, a first pass scan would only look for large job-steps. More sophisticated scans which relate the selection of a request to the availability of storage before allocation, are well known. Reducing the number of I/T's reduces the period of queue oscillation and hence the delay to large job-steps. Thus the number of I/T's can also be used as a control parameter. Hence in fixing the number of I/T's in excess of the effective level of multiprocessing, one must remember that while this helps to maintain CPU utilization, it also has a deleterious effect on the delay pattern.

Job delay

Related measurements of interest to the user are the maximum and the mean delay to any job-step during a run, as a function of its storage requirement. More generally, we must consider the delay to the entire job. The delay to the job-step

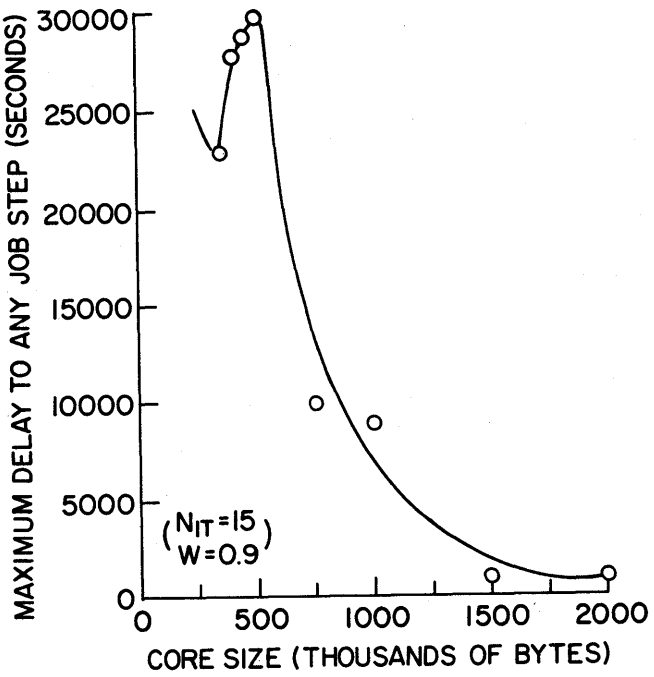


FIGURE 6—Maximum delay vs. core size

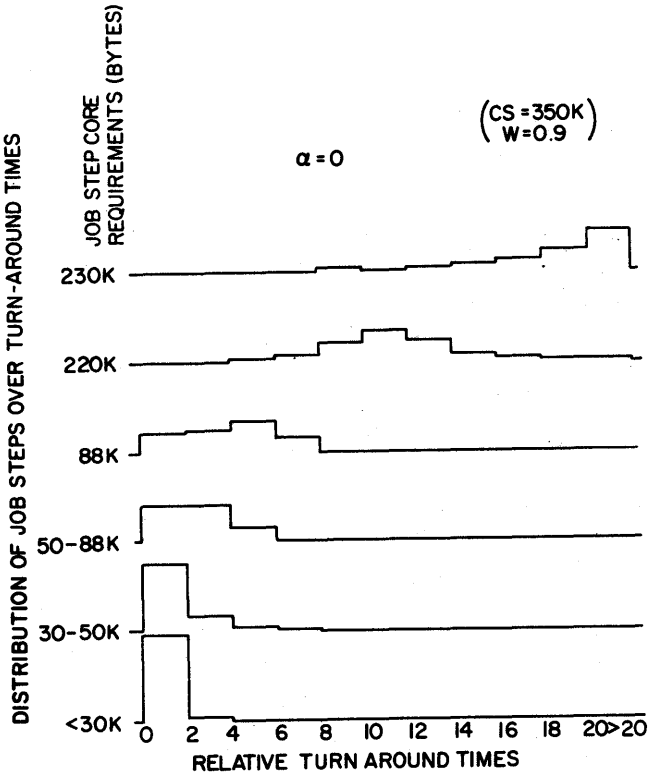


FIGURE 7—Distributions of turn-around times

most delayed during any one experience was determined and is shown in Figure 6, plotted as a function of system core size. Notice that for small core sizes the maximum delay varies sharply with the exact amount of core storage in the system, but for larger core storage systems the maximum delay gradually decreases as core size increases.

The fact that the large job-steps receive the maximum delay has been verified experimentally. Histograms of the delays suffered by job-steps as a function of their storage requirements show that the large job-steps have a high probability of being delayed for times approaching the worst case time. Figure 7 illustrates this point.

Variable sized storage requests

The quantized nature of the core requirements has already been noted. To determine whether this has any effect on performance, we ran experiments with the standard core sizes modified by a factor uniformly distributed between $1+\alpha$, and $1-\alpha$, where α was observed. This is evidenced in Figure 8. This result was at first surprising, since it was expected that for $\alpha=0$ (the standard mix) throughput would be greater. This is because when space for a specified core size is made available, it should be easier for a job-step with identical requirement to fit into the vacated space. However, it is now clear that only large job-steps are affected and the cyclic behavior modified. Since these job-steps constitute only 18% of the load, one would expect a throughput deterioration of less than 9%, which corresponds to the effect observed. More appropriately, it implies that the effect is not so much noticeable in throughput measurements, but could be expected to effect the delay pattern and was so observed.

Maximum and mean delay to large job-steps increased considerably for $\alpha=0.5$ and small system core sizes. In particular, a small number of job-steps with very large core requirements ($>$ 300K bytes) occurs. These job-steps remain in the queue of job-steps awaiting storage allocation much longer than they do with the standard mix. Even for small variations on the standard mix ($\alpha=0.1$), significantly larger delays are experienced, on the average, by the large job-steps. This is shown in Figure 9. This figure, when compared with Figure 7, confirms that non-quantized storage requests increase storage fragmentation, and suggests that the delay pattern of an actual sys-

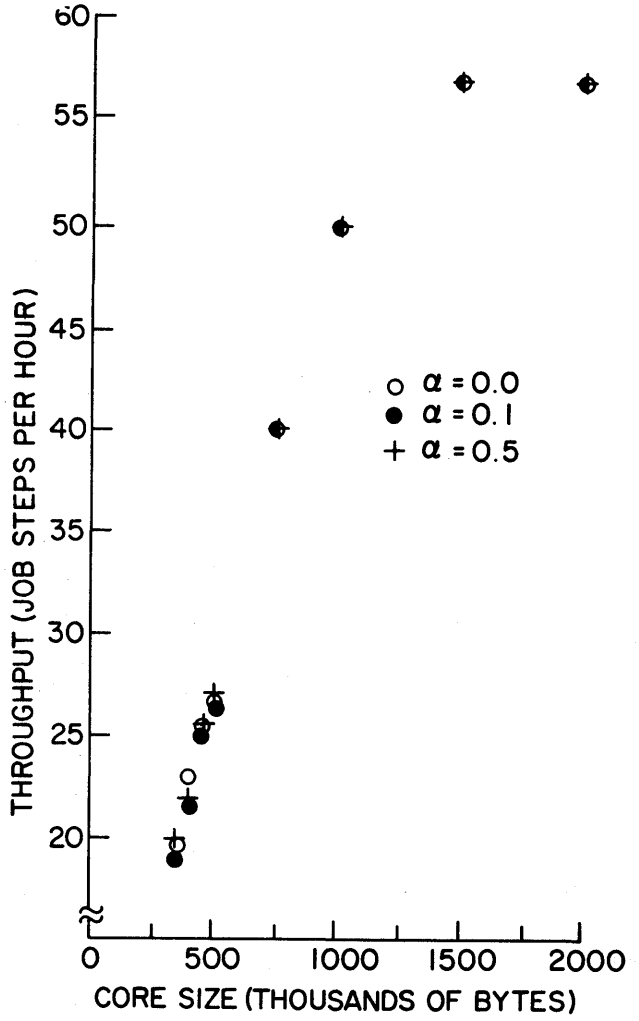


FIGURE 8—Effect of spreading core requirements of job steps upon throughput

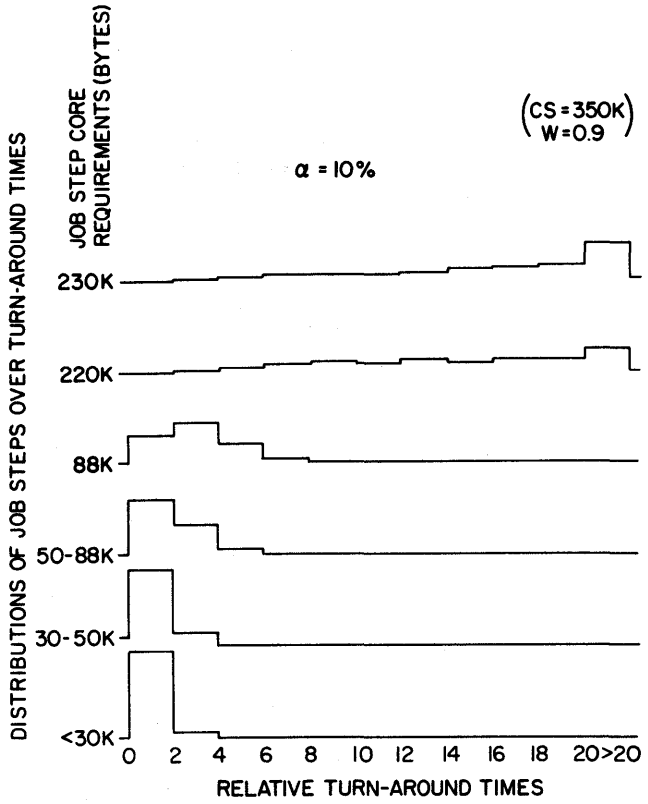


FIGURE 9—Distribution of turn-around times for $\alpha = 10\%$

tem load might well be worse than that shown in Tables 3, 4 and 5.

Dynamic relocation

Another question raised about the allocation system relates to storage relocation. Scherr indicated that for the experiments he performed there was no appreciable improvement in throughput when single register relocation was used. (Single register relocation implies the capability of moving blocks allocated to job-steps within core storage in order to eliminate storage fragmentation.) Experiments that confirm this conclusion were conducted by the present authors. Results are shown in Figure 10. Throughput is slightly improved, since job-steps that otherwise would not fit into storage because of the fragmentation can fit due to relocation.

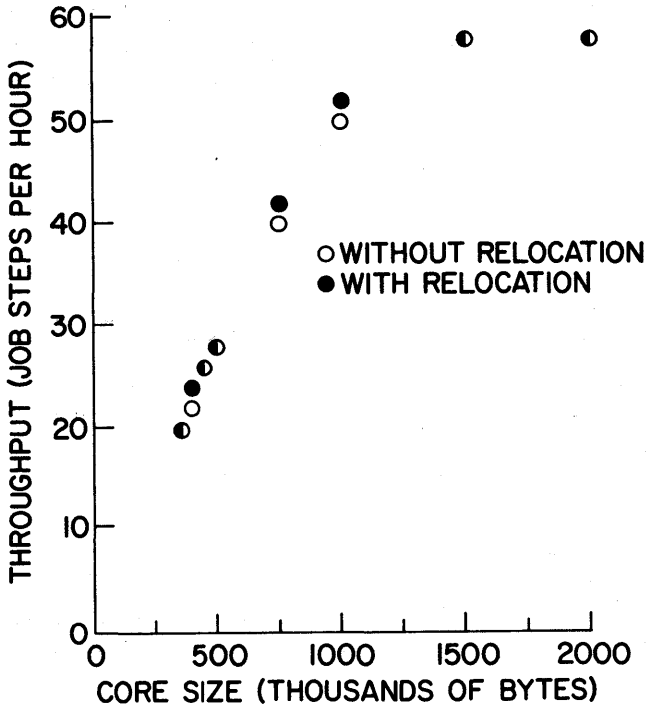


FIGURE 10—Effect of relocation upon throughput

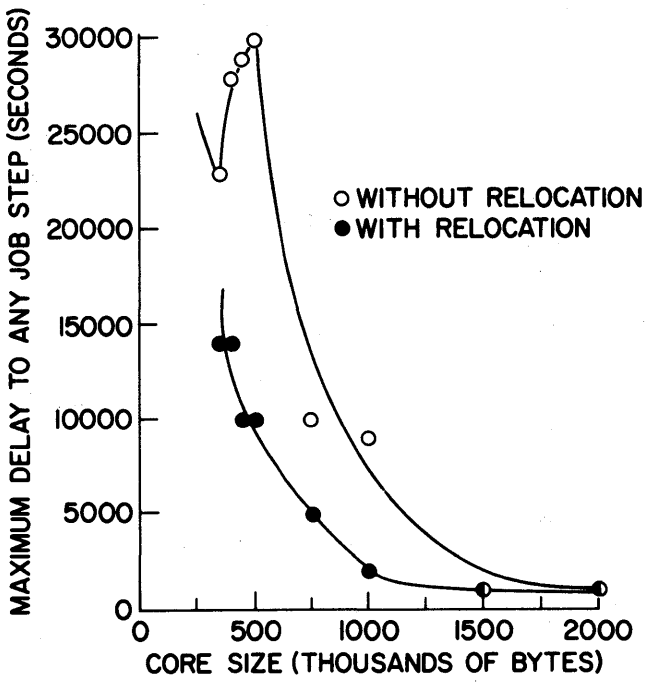


FIGURE 11—Effect of relocation upon maximum delay

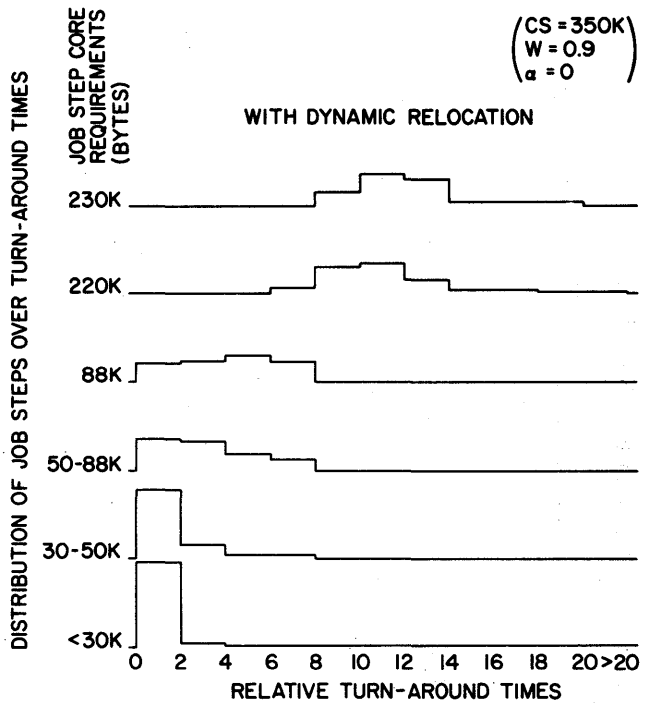


FIGURE 12—Effect of relocation upon distributions of turn-around times

Nevertheless, the significance of dynamic relocation will be seen in the behavior of those job-steps with very large core requirements—the 230-K job-steps. These, in particular, are generally the ones subjected to the maximum delays. Figure 11 illustrates the decrease in maximum job-step delay when relocation is used over the delay (from Figure 6) without relocation. Another aspect of the same phenomenon is demonstrated in Figure 12, in which the distributions of relative turn-around time are shown. Comparing this with Figure 7, one observes that the distributions for all ranges of job-step core requirements are nearly unchanged, except for the 230K job-steps. This distribution changes from one that peaks for relative turn-around time exceeding 20 to a distribution very close to that for 220K job-steps

The reason for this effect is obvious. When a 220K job-step terminates and relinquishes its core storage, another 220K job-step waiting will always fit in, while a 230K step can only fit if an adequate amount of neighboring free space happens to be available. A 230K job-step will therefore tend to be delayed longer than a 220K job-step waiting in queue. With dynamic relocation, this difference is likely to be minimized, since available space anywhere in core will be made available.

Further work

The authors feel that the present results should be validated by actual measurements on an MVT System. If general agreement is seen, then the model can be used for system performance prediction and improvement. We are hoping to perform some such experiments.

CONCLUSIONS

The results of the experiments reported here point up critical aspects of the MVT system and add considerably to our insight into its operation.

While knowledge of throughput (CPU utilization) is helpful in evaluating system performance when sufficient storage and a sufficient number of I/T's are provided, it has been shown that large delays may be experienced by certain classes of jobs. Certain jobs may be kept in the system for as much as 8 hours as a result of fragmentation. In an actual operating environment this could be undesirable, and some modification to the simple alallocation algorithm appears advantageous.

It has also been shown that addition of core to reduce delays is not an unmixed blessing, since it radically changes the general delay pattern, increasing delays to some classes of job-steps. The insight gained by experimentation of the type described suggests algorithm changes and system

parameter combinations that result in desired patterns of behavior.

APPENDIX

Analysis of interrelations among measured quantities

Throughput

Throughput (TP) is expressed in terms of job-steps completed per hour. If we consider first that an indefinitely large amount of core storage is available, the number of job-steps allocated core storage will equal the number of initiator-terminators. The I/T resident in core storage the longest will receive CPU time whenever its job-step is not in the wait state. Other I/T's have the opportunity to use the remaining CPU time in order of core residence priority. The relationship between the priority of the job-step and the expansion of the running time has been given in Table 1. Numbers of job-steps required to utilize the CPU more than 99% for various wait factors are given in Table A.

BATLE A—Number of allocated job-steps required to utilize CPU better than 90%

Wait Factor	Number of Job Steps
.2	2
.3	3
.4	3
.5	4
.6	4
.7	6
.8	7
.9	13

Therefore, when the amount of available core storage is large enough and the number of initiator-terminators is large enough, the CPU constitutes the limiting resource. The throughput then can be determined in terms of average job-step time. Thus,

$$TP = 3600/\bar{T}$$

where $\bar{T}$ represents the mean execution time for a job-step in seconds and TP the throughput in job-steps per hour. The throughput in jobs per hour differs from this by the factor of 21/97 jobs per job-step for the particular work load used. The same factor also converts all subsequent results from per job-step to per job.

Note that if the wait factor is so large that the CPU is not saturated for the number of job-steps allocated core storage, then the above equation does not hold. This is the case when the amount of core storage is not sufficient to accommodate enough job-steps to saturate the CPU. It also occurs when the number of initiator-terminators in the system is insufficient to saturate the CPU. The former case is the one under which many experiments were run. It was not deemed necessary to also measure performance in the latter situation.

Throughput is clearly related to the mean number of job-steps allocated core storage and to the mean percentage of available storage allocated. These quantities can be measured from simulation. The dependency of CPU utilization upon the number of job-steps allocated core storage is, however, non-linear, and no theoretical derivations are made for the relationship.

Turn-around time

Turn-around time (TAT) for a job-step is the difference between the time when the I/T for the job-step initially requests core storage allocation and the time when processing for the job-step is complete. The mean turn-around time is the sum of the turn-around times for all steps during a run divided by the number of job-steps. An expression for turn-around time can be derived by first considering one I/T. At time zero, it requests storage allocation for the first job-step. When the processing for that job-step is complete, it immediately requests allocation for the next job-step, and so on until the run is terminated. Therefore, the contribution of a single I/T to the accumulated turn-around time is approximately equal to the time of the run. If N_{it} initiator-terminators are in the system, then the total contribution to turn-around time is N_{it} x simulation run time. Therefore, the mean turn-around time equals N x run time divided by the total number of job-steps completed. Since run time divided by job-steps completed is 1/TP,

$$\overline{TAT} = N_{it} \times 3600/TP \ (= N_{it} \times \bar{T} \text{ when CPU-limited})$$

where $\overline{TAT}$ is measured in seconds and TP is in job-steps per hour. Therefore, mean turn-around time is directly proportional to the number of I/T's and inversely proportional to the throughput. This holds true whether or not the CPU is fully utilized and all job-steps are allocated space in core storage.

In this relationship, turn-around time is proportional to the number of I/T's in the system because each added I/T provides the capacity for accepting another job-step in the system, allowing it to wait for storage to be allocated and processing to be performed.

Delay

The delay to a job-step is defined as the difference between the turn-around time for that job-step and the time in which the job-step would have been completed had it been the only one in the system. It follows directly from this definition that the mean job-step delay equals the mean job-step trun-around time minus the mean job-step run time. The latter is mean execution time divided by 1 minus the wait factor. This division serves to expand the time into the time a solitary job-step remains in the system. Thus, in the case of a CPU-limited system

$$\overline{DELAY} = \overline{T}\ (N_{it} - 1/(1-w))$$

ACKNOWLEDGMENT

The authors gratefully acknowledge the contribution of J. M. Lee, who coded the present version of the simulation model, ran most of the experiments and contributed to our understanding of the phenomena studied.

REFERENCE

1 A L SCHERR
Analysis of main storage fragmentation
Proceedings of IBM Symposium on Storage Hierarchy Systems
TR 00 1556 December 30 1966 pp 159–174

Hardware design reflecting software requirements

by SAUL ROSEN

Purdue University
Lafayette, Indiana

INTRODUCTION

What was promised

All of the features that have been designed into digital computers may be considered to be, at least to some extent, reflections of software needs.

Starting with some of the earliest machines, the EDSAC[1] provided for address modification on input to make it convenient to use relocatable subroutines. The 4-address code of the EDVAC[2] was designed to make it possible to do minimum access time coding. The Datatron[3] provided off-line tape search, the RCA Bizmac[4] and the Univac File Computer provided special hardware-programmed tape sorting machines. Floating point hardware[5] was introduced on many computers to eliminate the need for software scaling systems and for interpretive floating point systems.

New computing systems, and whole new generations of computers have been introduced at an astonishing rate.[6] Even the small computers of a new generation are powerful machines, comparing favorably with some of the very large machines of earlier generations.

The component design engineers have made great advances over the years in the development of high speed memories, and in the development of circuit components that are cheap and fast and small. There seems to be an order of magnitude improvement in these areas every few years, and the concept of large scale integration[7] promises additional advances on the same scale.

It is the job of the hardware system designers and the software system programmers to translate these advances in components into improvements in system effectiveness.

Perhaps the most important part of the job of computer system design is to determine which functions are best handled by hardware, and which should be provided by software. All of the traditional engineering considerations come into play here. One must consider the added cost of additional hardware measured against possible gains in operating efficiency. One must consider the availability of manpower with the appropriate skills for design and for implementation. One must also consider intangible factors such as the inherent advantages of simplicity and the hidden costs of complexity.

From the point of view of the computer user it makes no difference which features are incorporated into the hardware and which are programmed in software.

It is the combined hardware-software system, the "extended machine"* that the user must rely on to provide the power and flexibility and convenience that have been promised with each new generation of computing equipment. Multiprogramming and interactive systems, problem-oriented languages, on-line debugging, automatic storage and retrieval of information—these and many other developments of modern computer technology promise the user convenience in problem statement, in program preparation, and debugging. They promise to make it possible to obtain timely, useful, meaningful results without the waste of personnel time and equipment time that has been so characteristic of computer problem solving.

So far at least the promises have not been kept. Most problems that are solved on the new generation of computers are solved very much in the same way as they were solved on earlier computers. Most debugging is off-line. Programmers, even in so-called higher level languages still wait through hours and days of turn-around time for

*The term "extended machine" has been used in this context by a number of people. To the best of my knowledge it was first used in this way by Anatol Holt and William Turanski.

their octal and hexadecimal dumps. The hardware machine has changed and developed and improved at a great rate, but the capabilities of the extended machine have grown disappointingly slowly.

What we have

It would take a large volume devoted to the subject of computer organization to discuss all of the hardware features of existing computing systems that reflect software needs. There have been many hardware developments that have contributed to the development and the performance of multiprogramming and multiprocessor operating systems and compiling and translating systems. A few of these are discussed in this section.

Microprogramming

The word microprogramming has been used in a number of different though related ways from the earliest days of computer design.[8]

In the most usual current usage of the word a computer is said to be microprogrammed if it interprets the user or programmer instructions (add, multiply, branch, etc.) as calls on routines (the microprogram routines) stored in a microprogram memory. These routines are themselves sequences of the more elementary operations that are built into the machine hardware. For reasons of speed and economy in the technology now available, the microprogram memory is usually a read-only storage device.[9]

For this reason, the microprogram routines are usually designed and written by hardware engineers rather than by software programmers.

There are microprogrammed computers in which the microprograms are stored in fast read-write storage which is logically or even physically the same as the ordinary program and data storage of the computer. The elementary operations of the computer thus become available to the problem programmers. This kind of system may in a sense permit each programmer to design his own instruction code. If that were true on a sufficiently basic level, it would be very difficult to supply software in the same sense as it is now supplied. Typically, in such systems, only software programmers (i.e., systems programmers) are permitted to design instructions.

Microprogramming in a read-only memory is a relatively simple, easy to understand technique whereby it is possible to "build-in" quite complex operations.

It makes it clearer perhaps than it was before that much of the sequencing that is done by hardware is the same kind of thing that is done in programming.

With the popularization of microprogramming it is easier to talk about performing some traditional software functions in the hardware, i.e., in programs on the elementary operation level rather than on the instruction level.

Such programs could be done in the hardware without the formalism of microprogram control. We shall use the term hardware programming to indicate programming built into hardware, either through a microprogramming system or by other hardware sequencing methods.

Emulation and simulation

One of the most successful applications of the recent microprogramming technology is in the simulation of computers on computers.

The microprogram control and the set of microprogram routines are in effect a simulation program that simulates the programmer's instruction set on a computer whose instruction set is the set of elementary operations. It may be equally possible to simulate computers with other programmer instruction sets in terms of the same set of elementary operations. This, slightly oversimplified perhaps, is the idea of hardware assisted simulation that is now usually called emulation.[10]

Simulation of one computer on another had been done for many years using the programmer instructions. The resulting systems were inevitably so slow as to be almost useless for general computation, and were limited in use to special applications such as checkout, etc. The speed that makes emulation practical for much wider areas of application can only be achieved by operating at the hardware level.

Interrupt systems

One of the critical areas in most operating systems is the handling of interrupts, especially routine input-output interrupts. The time it takes to recognize an interrupt, handle the functions required by the interrupt, initiate new operations where needed, and finally return to the interrupted program, is usually one of the most important performance parameters of the system. If a great

many instructions must be executed in response to each interrupt, an otherwise fast computer may become intolerably slow.

This is an area of programming in which the programmer deals directly with the idiosyncracies of the computer hardware. It is, almost by definition, an area that cannot be handled by software alone. Much of the programming here is best done at the elementary operation level. There are many hardware features of many computing systems that have been designed in response to the need for efficient handling of interrupts. Some of these perform the following functions.

1. Identify the device that interrupted, either by transferring control to a location that is specific to that device or class of devices, or by storing a device identification code in a special register or storage location.
2. Provide special locations into which information about the state of the device is automatically recorded.
3. Provide for automatic storage of the most important registers that will then be automatically restored when the interrupted program is resumed.
4. Switch automatically into a speecial supervisory mode. In some systems the special mode has its own instruction control register and its own special registers for temporary storage and for control functions.

Also important in this area are hardware devices that cut down on the number of interrupts that have to be handled by the central computer. An example is the so-called multiplexer channel that automatically transfers streams of information between core memory and a large number of slow devices, interrupting the central computer only at the beginning and end of each stream.

Dynamic storage allocation

There have been a number of interesting and useful hardware developments to assist in the allocation and addressing of storage. In the Burroughs 5000-6000-7000[11] series an array (or a subprogram) may be addressed as if it is present in main memory even though it may actually be in peripheral storage. Addressing is indirect, by way of a descriptor. A "presence" bit in the descriptor causes an interrupt if the addressed array (or subprogram) is not in core storage. The supervisor may then fetch it from peripheral storage and place it, without change, in any area of core memory that is large enough to hold it. The origin of the assigned storage area is placed in the descriptor, and all subsequent references to any elements are interpreted as relative to this origin. The descriptor also contains the length of the array and the hardware automatically checks for and interrupts on any reference beyond the limits of the array.

Another, perhaps more influential design was the Atlas one-level storage system.[12] In that system core storage is organized into 512-word pages, and programs are organized into 512-word blocks. A block in peripheral storage may be loaded into any page in core storage. The block number is placed in a page address register. The page address registers form an associative memory that is searched by hardware. A program address that consists of a block number and a displacement within the block is automatically translated into a hardware address that is a page number and the same displacement within the page. A reference to an address in a block that is not physically present in memory causes an interrupt to the supervisor which will fetch the block from peripheral storage and place it in any available page and place its block number in the page address register for that page.

A three level addressing scheme[13] was introduced on the General Electric 645[14] and on the IBM 360 model 67.[15] There a program address is interpreted as a segment number, a page number, and a displacement. An active segment has a page table which correlates physical page numbers with logical block numbers. A segment table gives the location of the beginning of the page table for each segment. A program address is converted into a hardware address by an automatic two level table look-up. In both the IBM and GE systems a small associative memory stores a few of the most recent translations, and the address translation is very much speeded up if the segment-page combination has been used recently enough so that its translation is still available in the associative memory.

The GE645 and IBM 360/67 address translation system, and other hardware address mapping schemes that were introduced since the Atlas development were designed as hardware aids to time-sharing systems. Computer manufacturers

and many users tend to refer to any computer that has any kind of hardware address translation as a time-sharing computer.

It takes much more than adding an automatic scheme for address translation to an existing general-purpose computer design to produce an effective time-sharing system.

Supervisory functions

In most systems supervisory functions are initiated in response to interrupts, but they are too varied and complex to be considered as part of the interrupt system.

With the recent and continuing developments in integrated circuits and large scale integration it is becoming both technically and economically feasible to handle supervisory functions by hardware rather than software programming.

In the input-output area such functions may include the maintenance of input-output queues, the handling of priorities and issuance of input-output commands, the response to error conditions and other special conditions.

In the job management area they include once again the management of priority queues. They also include the handling of address translation tables in systems that provide automatic translation, and more generally the management of a multi-level hierarchical storage system.

There are dangers and pitfalls in attempts to handle these functions in the hardware. Programs on the microprogram level may become excessively long. They may be difficult or even impossible to debug. Hardware programmers are not necessarily better than software programmers. They make errors that must be corrected. Their first versions of programs are often incomplete and inefficient and must be replaced by later better versions Hardware programming for really complicated tasks will only be feasible when it is relatively easy to make changes to the programs.

This is still a research area. It seems reasonably certain that some of the functions mentioned above will be handled by hardware programming. Eventually they all may be handled that way.

Compilation

Compilation of programs written in Fortran, Algol, Cobol and other higher level languages is another critical area in the performance of software systems. It is an area in which hardware programming can and in some cases already does handle at least part of the job.

With the microprogramming systems that now exist it is reasonable to handle all of the process that is usually called lexical analysis on the microprogramming level. Microprograms can also be used in the construction and manipulation of tables, and to provide arithemetic capabilities not otherwise provided in the hardware. Much of this has already been done in the Allen-Babcock PL-1 System by adding microprograms to those already present in the IBM 360/50.

As in the case of supervisory systems, here too by making use of the integrated circuit and LS1 technologies it may become economically feasible to do the whole compiling process by hardware rather than by software programming. This too is a research area, and so far none of the proposed designs for building higher-level language capabilities into hardware seem both practical and attractive.[16]

The Burroughs 5500-6500-7500 computers represent a very interesting approach in this area. The machines, especially the later ones, contain very elaborate hardware programs to help in the processing and running of Algol programs. One might say that the block structure of Algol is built into the hardware, along with the automatic interpretation of Polish strings and the automatic handling of push-down lists or stacks. The logic of the 6500-7500 computers is described in some detail in.[17]

What we need

The most important need is for a design philosophy that aims at the design of total information processing systems, and that will eliminate the mostly artificial distinction between hardware systems and software systems. We need a continuing development of the trend toward combined hardware-software programming discussed in part 2. Practically no one would deny that these are desirable goals, and it would serve little purpose to elaborate on them here. The remainder of this section will therefore discuss a few more specific and perhaps less obvious needs.

Peripheral storage

Writers in the computer field have overemphasized the importance of central processor speeds and capabilities in discussing the performance of

computing systems. This is understandable, since there have been such dramatic advances in speed and compactness of circuits and central memories.

The advances in peripheral storage have been fewer and less spectacular. Software programs, along with almost all other information-processing programs, can profit far more from hardware advances in the area of peripheral storage, than from improvements in central computer logic and speed

At a SHARE meeting in 1966, representatives of IBM presented figures concerning the performance of an initial version of the OS 360, the operating system for the 360 computer. For the areas covered in that presentation, the operating system ran almost as fast with the slow model 40 central processor as it did with the very much faster model 65. Both used the same disc storage system, and it was the very frequent reference to disc storage, not the speed of internal processing, that was the limiting factor in the performance of the system.

I have been told that the design philosophy of OS 360 that permits frequent reference to system routines resident in peripheral storage, was influenced by a hardware development program that was to provide very much faster-access peripheral storage for system residence than is provided in current disc or drum storage systems.

A peripheral storage system large enough, cheap enough, and fast enough to permit its use as a rapid access device for system residence would be the most important contribution that hardware designers could make to software systems.

Peripheral core storage devices such as IBM's LCS and Control Data's ECS are too expensive. The cost per bit of bulk core is roughly 100 times the cost per bit of large disc storage ($2¢ - 4¢$ per bits vs. $.002¢ - .004¢$ per bit). The random access time differs by a factor of about 10000. The most obvious design point between these two would aim at a device with random access 100 times as fast as disc at about 10 times the cost per bit—i.e., access time less than 1 millisecond at a cost of $.02¢$ to $.04¢$ per bit.

Software systems deal with file management. with the cataloging, the storage and the retrieval of information. Catalogues and glossaries must be kept in peripheral storage. If the files are at all extensive, the catalogues must be several levels deep. One glossary leads to another which in turn leads to another. all requiring additional accesses to peripheral storage. The faster access peripheral storage mentioned above would be of great help, but even there, and more so in connection with disc and drum systems, hardware that provides for automatic search, for branching from one catalogue to another, for parallel readout from a number of index tracks—these would all provide major assistance to a large segment of software programming.

Time sharing

In recent years there has been a great deal of interest in, and a great deal of effort expended on the design and implementation of large interactive time-sharing systems. The results achieved in 1968 are very disappointing when compared with the promises of 1965.

Hardware performance is only one aspect of time-sharing system performance. Improvements in computer hardware alone will have little effect without corresponding developments in areas of software and communications. However, new computer hardware developments are essential before the promise of interactive time-sharing systems can be realized.

There are many general hardware design improvements that are relevant to time-sharing systems. These include improvements in peripheral storage, faster channels with greater bandwidth. and hardware programming of many operating system functions.

Specific to the time-sharing field there is a need for the development of adequate consoles, especially graphical input-output consoles. There are conflicting requirements for low cost and for many built-in features that minimize the load on the central computer. An adequate graphical console may require built-in hardware equivalent to that required in a fairly sophisticated computer. This is an area in which analogue as well as digital techniques may be important. It is an area in which the new component technologies may make significant contributions.

Analysis and debugging

Software systems, especially on large computers are systems of great complexity. Hardware design can and should contribute to the analysis, to the debugging and to the documentation of such systems.

In order to improve the performance of a software system, we need to be able to evaluate its

performance. In the case of a number of computing systems, separate and very complicated hardware was built to gather information needed in such analyses. The problem is to get the necessary information without loading the system and thereby distorting the information gathered. This can be very difficult to do in software alone. Hardware aids built-in to the initial design may represent a trivial cost compared to the cost of external monitoring equipment that may otherwise be needed.

There has been far too little concern on the part of the hardware system designers with the problems of debugging of complex programs. Hardware aids to program debugging would be among the most important hardware aids to software production. On-line debugging is essential. It should be possible to monitor the performance of software on a cathode ray tube console, without interfering with the performance of the software. It should be possible to examine areas of peripheral storage as well as areas of core storage.

One of the most interesting features of the recently announced IBM 360 model 85 is the microfiche reader built into the system console. This reader will permit reference to the software documentation and at the same time permit reference to the state of various hardware registers.

We should perhaps aim for a software system design and implementation technique that will make it unnecessary to have a microfiche reader at the console for debugging and maintenance, but in systems where documentation has to be voluminous, a simple hardware retrieval system can be of great value.

Compatibility and standards

Advances in the area of standardization and compatibility in computer hardware can be of importance to software developments in a great many ways. There is always a danger that standardization will inhibit progress, but at some point in time this danger must be faced. Over the years the computer industry has become accustomed to change at a frenetic pace, and such change is often but not always accompanied by progress.

The time may be near, or already at hand when there should be standardization of computer instructions, of input-output interfaces, of character sets. Duplication and multiplication of identical and nearly identical programming efforts is typical of the computer field today. It is hard to overestimate the saving in manpower and computer power if such duplication could be eliminated. Standardization of computer hardware would be the major step in the elimination of such duplication. We might even then find that there exists a surplus and not a shortage of system programmers.

REFERENCES

1 M V WILKES D J WHEELER S GILL
The preparation of programs for an electronic digital computer
Addison-Wesley 1951 1957

2 *Functional description of the EDVAC*
University of Penna Moore School of EE Research Division Report 50–9 under contract W–36–03400RD–7593 with the Ordnance Dept Dept of the Army Nov 1 1949

3 J C ALRICH
Engineering description of the electro data digital computer
IRE Trans on electronic computers EC–4 March 1955 p 1–10

4 W K HALSTEAD et al
Purpose and application of the RCA BIZMAC system
Proc of the WJCC Feb 1956 p 119–124

5 S G CAMPBELL
Floating-point operation
In W Buchholz Editor Planning a computer system McGraw-Hill 1962

6 S ROSEN
Electronic Computers—a historical survey
Document CSD TR 25 Computer Sciences Dept Purdue University July 1968

7 D F Farina
Large-scale integration: A status report
Datamation 14 Feb 1968 p 22–29

8 M V WILKES J B STRINGER
Microprogramming and the design of the control circuits in an electronic digital computer
Proc Cambridge Philosophical Society 49 Part 2 April 1949 p 230–238

9 S G TUCKER
Microprogram control for system/360
IBM Systems Journal 6 4 1967 p 222–241

10 J GREEN
Microprogramming, emulators and programming languages
Comm ACM 9 March 1966 p 230–232

11 R S BARTON
A new approach to the functional design of a digital computer
Proc of the WJCC 1961 p 393–396

12 T KILBURN et al
One-level storage system
IRE Trans on Electronic Computers EC–11 April 1962 p 223–235

13 J B DENNIS
Segmentation and the design of multiprogrammed computer systems
Journal of the ACM 12 Oct 1965 p 589–602 Reprinted in S Rosen Editor Programming systems and languages McGraw-Hill 1967

14 R C DALEY J B DENNIS
Virtual memory, process, and sharing in MULTICS
Comm of the ACM 11 May 1968 p 306–312

15 B W ARDEN B A GALLER T C O'BRIEN F H WESTERVELT

Program and addressing structure in a time-sharing environment
Journal of the ACM 13 January 1966 p 1–16

16 T R BASHKOW A SASSON A KRONFELD
System design of a FORTRAN machine
IEEE Trans on Electronic Computers EC–16 Aug 1967 p 485–499

17 A A HAUCK B A DENT
Burroughs' B6500/B7500 stack mechanism
Proc of the SJCC AFIPS Vol 32 1968 p 245–251

What was promised—what we have—and what is being promised in character recognition

by A. W. HOLT

Control Data Corporation
Rockville, Maryland

INTRODUCTION

Like the dishwasher, where you have to wash the dishes first before you use it, character recognition machines have certainly not solved the whole problem of man-machine communications. You still have to clean up your inputs. On the whole, however, it seems fair to say that the engineers have succeeded quite well in delivering what they promised, particularly if you make allowance for the purple prose occasionally delivered by the Fourth Estate. What we have today is systems that work well if the source of character imprinting is controlled. In the future, systems will accept data generated by non-captive sources in the course of normal business transactions.

How can we measure what was promised? One analog method is to look back in a file of old newspaper and magazine articles. The following excerpts are taken from an article which appeared on the front page of a big city daily newspaper. The date was February 26, 1963.

"Slip that new novel into the scanner. Lie back, close your eyes and read.

"That restful arrangement—reading with your eyes shut—came a step nearer to reality today with the demonstration of a new 'seeing' machine.

"It can already read numbers in ordinary print. And ——— said work is going on to refine the device so it can master the alphabet.

"The development," he said, "points toward the building of a universal machine that can read anything intelligible, including handwritten material.

"The prospect raises all kinds of novel possibilities. For instance: driving along the highway, and keeping your eyes on the road, while at the same time, with your scanner-speaker plugged in your ears, reading a book of ancient history.

"Yon won't even need an extra pillow or overhead light while reading in bed".

The author was not given a by-line.

To measure what we have is relatively easier than to measure promises—a list of all constructed machines will suffice. Since nearly 100 different types have been built, however, we will confine the list to the top few, publicity wise.

	Font
Farrington Oil Credit Card Reader	Self Check 7B
Control Data 915 Page Reader	USASI (ISOA ASA)
REI Document Reader	407E
IBM 1287	Carefully Hand Printed Numbers
Philco Reader for Post Office	Omni-Font
IBM 1975 Reader for Social Security	Omni-Font
Control Data Reader for Fort Monmouth	Multi-Font, consisting of: USASI 12 Pitch Elite, Upper and Lower Case 10 Pitch Pica, Upper and Lower Case
NCR 420	NOF
Magnetic Ink Character Reader (MICR)	E13b

These machines, and many others, were promised and delivered, but the mere fact of their delivery and acceptance does not really measure whether they lived up to their promises, because

the designers and users were certainly hoping for applications far beyond the use of only the first prototype.

A way of measuring the ratio of success of these machines to the promises made for them is to count the number of repeat orders. On this basis the MICR machine certainly ranks five stars with at least 500 machines in the field, the NCR 420 ranks four stars. The Rabinow Division has delivered about 100 of the Control Data 915 Page Readers. The Oil Credit Card Readers certainly number over 100 and the IBM 1287 has at least 25 machines in the field. REI has at least 10 Document Readers out, and Philco has probably installed at least 4 machines for the Post Office. Only single models of the rest of the machines have been built.

The significant point about this list is that by far the most successful machines have been those which read a stylized font—that is to say, a set of character shapes which are very limited in number and carefully controlled as to quality. The only exception to this is the good showing of the Philco Post Office Reader, which is, of course, required to do its job on an enormous variety of shapes and sizes of characters. Even so, the Philco reader has not been successful in the business data processing market. At another end of the spectrum, we find the IBM 1287, which is a machine specializing in the reading of carefully hand printed numerals. Before attempting to explain why successful machines have primarily fallen in the category of stylized font readers, we will present a brief discussion of the construction and uses for character readers.

Why reading machines are useful as inputs to computers

First, let us recall that we are *not* discussing the entire field of *pattern* recognition, but only a very narrow corner.

The major attractiveness of character recognition as a method of input to computers is because it is entirely compatible with the way humans communicate among themselves. Why is compatibility good? Because humans produce better accuracy, better verification, better correction and better flexibility using a single language than using a double language. Also, humans seem to get a greater feeling of personal security out of using a familiar set of marks.

There are a number of ways that character recognition can save money over keypunching.

1. The source documents are prepared in such a way that they are used by humans for their non-machine purposes and are then returned to the information collecting station to be machine read. The best example of this is the preprinted account number on the bank checks.
2. Even when the machine readable document does not go out into the flow of general business commerce there are very important savings that can be realized over the traditional method of key-punching. These are:
   a. Higher accuracy. Part of this is due to the fact that a typewriter keyboard is more familiar to people than a keypunch keyboard. Another part of this is due to the ability of instantaneous verification of what key has been stuck (sic)*
   b. Wages and training time for typists are less than for keypunch operators.
   c. The format flexibility of an OCR document is almost infinite.
   d. The document or page is completely correctible in any part without disjointing the whole.

What about mark sensing? It has the potential ability to acquire information from non-captive humans. Certainly there are some good uses for it, like exam answering, but it is still a double language and suffers from all the attendant problems. (It is useful to visualize tab cards as a special case of mark sensing.) The advantages of OCR recognition machines over various forms of mark sensing are:

There is more redundancy in a character than can be built into a mark system. There is considerable space saving. There is also human compatibility, instantaneous verification, and correctability. In fact, almost all of the advantages which can be enumerated for OCR over punched cards can be enumerated over mark sensing.

On the negative side of the ledger, one would expect to find that the cost of a character recognition machine is greater than for a mark sensing machine. True, in comparison with readers for punched cards, present day character readers are more expensive. In comparison with readers for

*In the course of rough-typing this paper, a secretary actually made this mistake and corrected it on the spot as shown.

less rigidly controlled mark sensing documents, however, the character reader is very nearly the same price. The reason for this is that the majority of the machinery to do the two jobs is identical—namely, there must be an input hopper, a paper pickoff, a paper transport and at least two output sorting bins. There must be a complex array of photocells and there must be a data recording device, or at least an interface to a computer. It turns out that only in the area of the recognition circuitry itself does the character reader have a more expensive component.

Optical versus magnetic scanning

The major reason why the banking system chose a magnetic method was that this allowed overprinting to be done on top of the magnetic numbers and still have them read. This relative advantage of magnetic over optical is rapidly disappearing as systems people have finally been discovering that there is no need to overprint on data areas. The optical has a real advantage over the magnetic system in the area of quality control, because OCR makes its basic measurements the same way that humans do. The thickness of magnetic material in an MICR character, in particular, is very critical, and the human being cannot measure this value without a special instrument. Because of this the magnetic ink characters are expensive to print and an ordinary typewriter cannot be used.

Techniques of reading

Let us now discuss a few of the methods which are used to decipher which character is which. There are basically four outstanding classes. The first of these techniques is called "Feature Recognition" and this involves simply taking the character apart into its pieces (big pieces, like bars and columns), recognizing the pieces, and then reassembling them into a truth table. This truth table may very well be in the form of a completely digital decision matrix, although it may have considerable redundancy.

A second method, called "Area Correlation," is to take the character apart into tiny unit areas, each black or white, and then compare it point by point with all templates. No decisions are made on parts of the character—decisions are made by choosing the *best match* correlation between the complete image and the complete templates. This image may be in a purely optical form or it may be in a digital form; the correlation may be done optically or by using weighted resistor networks.

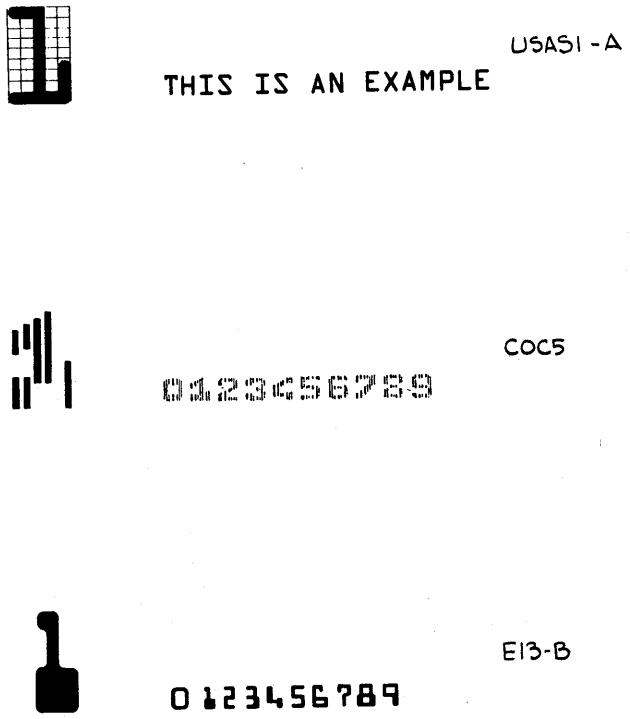


FIGURE 1—Several varieties of stylized font

The third class may be called "Topological Recognition." This is found most commonly in machines designed to read hand printed numbers. The basic method is to track the lines and to discover the relationship of each line to other lines in the form of sharp curvatures, beginnings of lines, ends of lines, joints, and splits.

The fourth class we will brand with the name "Inhuman Recognition." This is a group of special recognition techniques which are useful only with severely stylized fonts. Two of these are illustrated in Figure 1. In this class of reading machines the complexity of the character reading portion of the machine is nominally reduced by designing the character set to have some trick shape which can produce an electrical signal more easily deciphered. One example of this is the MICR Reader which has the font designed to give various coded amplitude outputs as a function of time, the character being scanned horizontally with a narrow vertical slit. Another example of this is the COC5 font. Its trick is that it is made up of combinations of wide and narrow spaces between vertical black bars. The particular set of wide-narrow spaces form a digital code.

All of the first three techniques have been suc-

cessfully used with non-stylized fonts, but improvement invariably results from the use of a somewhat stylized character. The USASI (ISOA) font was developed for this purpose. It is now a national standard and is being required as a font which must be read on all OCR reading machines which are purchased by the U. S. Government. (Note: machines purchased may also read additional other fonts.) The use of a slightly stylized font together with a powerful technique such as area correlation allows an inexpensive central reader and insures a very high accuracy rate.

There are a great many other techniques which have been promised and seem promising to some. The idea of the Perceptron triggered off a burst of studies aimed at self-organizing systems, and the theoretical work is undoubtedly laying the foundations for exotic new fields in Automata. Practically, some interesting extrapolations of the Perceptron have been made at SCOPE and at Andromeda, but it does not appear that the self-organizing method is likely to be useful for character reading in business. One system, called the "N Tuple" method, was very nearly a commercial success for the conversion of plate impressions. Groups of statistical measurements are made by a computer on a training set, and these measurements are later correlated with the unknown characters. Economic justification for a self-learning reader is difficult, the argument being that the self-learning feature need really only be used once (or at most occasionally) and that therefore the requisite machinery for accomplishing this would sit idle most of the time while the reader proper was paying the rent.

A very severe problem common to all machines reading non-stylized alphabets is the separation of characters when they touch. This is particularly bad in a 12 Pitch font with serifs.

A more detailed discussion of the philosophy and construction of reading machines may be found in the references.[1,2]

Scanning

Scanning is the term used for the job of getting the character shape off the paper and into the electronics. For magnetic character readers the job is done by using one or a few magnetic heads arrayed in a vertical slot. These are, perforce, in contact with the paper and so the variety of the possible structures is severely limited.

OCR readers, on the other hand, have a bewildering array of possibilities for scanning, since lenses and mirrors can be used ad infinitum.

Some of these are:

1. A single photocell which peeks through a moving slit or hole. Typically, this hole will be traveling vertically at high speed and the paper will be moving horizontally at low speed, thus producing an area scan.
2. A column or a row of photocells. If a column of photocells is used, the paper will be traveling horizontally, thus giving an area scan.
3. "Retina" of photocells. This is simply an X-Y array of photocells which is typically somewhat larger than the size of a character.
4. Cathode ray tube scanning. This is particularly attractive where there is a very large area to be covered while searching for the small portion of information to be read.

In all of these schemes, moving mirrors may be employed to help scan, but the mirrors are notorious thieves of light. Light level and uniformity is always a major problem. Where only a single line is to be read and the paper can move in one of the directions, the light problem is much easier.

Another major requirement for the scanning system is to be able to resolve down to very fine line width. Omni-Font machines characteristically require the ability to see line widths down to 6 mils. In order to make sure of catching this line, the scanner must resolve about 3 mils. It is important to remember, however, that one should not resolve too finely because then dirt will quantize very nicely instead of integrating. There isn't much danger of anybody getting finer than 3 mils, though, because it is a struggle to get even this much resolution.

Paper handling

The complexities of handling paper are typically very much underestimated, and broken promises exist in almost every design. A tabulating card is a nearly ideal object for being moved around by a machine since it is stiff, has close tolerances. Not so for paper.

The two most critical areas in paper transport are the pickoff and the stacker. In general, these problems get more difficult the wider the variety of paper which is required to be handled. For example, while a vacuum pickoff works very well for 16 lb. paper, it works quite poorly for tab cards; while a "ledge" type pickoff works very well for

tabulating cards it is utterly useless on paper under 20 lbs. Undoubtedly, the most difficult pick-off and stacking problems occur with the use of a very light paper, particularly the type used in airline tickets. For such paper the pickoffs are better than the stackers.

In between the pickoff and the stacker, we have some other problems, the majority of which are concerned with starting and stopping the paper smoothly while preventing skew. The aspect ratio of documents is extremely important. Those documents which can be moved with a long edge in the direction of travel tend, because of some very basic physical laws, to be much more stable than those documents which must be moved the other direction.

Choice of font

As we have hinted before, the detailed shape of characters for OCR is all important. If the shape is laid out on a grid such as a 5×9 matrix, the reading machine gets cheaper and more accurate than one which has to read non-stylized. How about reading Chinese characters, or worse yet Arabic alphabet? Symbols of the Arabic alphabet often extend under each other for several characters. Let us try analyzing the font problem from the point of view of information content. Firstly, it is clear that the binary code is the most efficient —good for the machine but hard for the human. A compromise must be established which has good redundancy, has good recognition ability by humans, and also can result in an inexpensive reading machine. There are several examples of such "stylized" fonts that have been constructed with the reading machine in mind. Figure 1 illustrates the USASI-A, the COC5 and the E13b.

It is educational to calculate the redundancy of a character. Let us say that we have 2^6 characters in the set and a 45 point black-white grid. The redundancy of any character ought therefore to be $2^{45} - 2^6 = 2^{39}$.

Don't be fooled, however, by this superficial redundancy figure. It is a poorly recognized fact even today that in every OCR business application the critical parts of the font must be carefully controlled. No matter how good the reading machine is, there is some limitation to the degradation which can be allowed. Because this requirement is only now beginning to be firmly understood, many past users have had a hard life living with old typewriters. Suppose the potential user already owns several thousand typewriters which carry the 12 pitch Elite font. He figures that if he can get a reading machine which can read these existing typewriters, he can save the price of new typewriters and can, therefore, afford to pay a fancy rent. If the system is going to equal the accuracy of a USASI font machine, however, he must make sure that the elemental areas (or features, if the machine is a feature reading machine) stay very well defined. Since there are more critical dimensions in the Elite font than there are in the USASI font, he has to spend extra maintenance and care in printing characters. This means better paper, better ribbons, more frequent changing of ribbons (or use of one-shot ribbons), more frequent cleaning of the type, and most desirable, the use of an electric typewriter instead of a manual typewriter. This costs!

Applications

The following paragraphs present a partial list of applications classified according to their most likely requirements in terms of font.

A. *Stylized font:* (Characters can be generated by printing, typing, plates, high speed printers, cash registers). Point-of-sale readers (credit sales, charge plates), bank checks, tally roll records, inventory control, production control, insurance payments, way bills (transportation), stock certificates, message entry, prize coupons, insurance payments.

B. *Multi-Font:* More than one font, but probably less than 5 fonts.
   1. Typewriters: mail correspondence, Social Security returns, tax returns, banking, rental companies.
   2. Plate Imprinters: Mass mail, conversion to computer language, credit cards.

C. *Omni-Font:* Post Office, Social Security, tax returns, books, magazines, newspapers, information retrieval, language translations, readers for blind people.

D. *Hand-Written:*
   1. Numeric: Post Office, tax returns, mail orders, credit sales, bank checks, sales slips, meter reading.
   2. Alphabetic: Computer programming, exam papers.

Systems including character recognition

The amount of effort needed for systems analysis is only now beginning to be recognized.

The greatest application of character readers is undoubtedly in the business community where

man must communicate with the machine. One historical difficulty is that forms design evolved from information typed on a piece of paper in locations which seemed suitable to forms designers of twenty years ago. These forms were deciphered and interpreted by humans. The ability of a human reader to perform a whole host of operations—not part of the character reading itself—have been very poorly appreciated. For example, it may be that a girl is handling 30 different forms from which she is only copying the serial number and the total amount of the invoice. In old systems these items might appear in as many different places as there are forms. The most obvious thing to do to correct this, of course, is to design all forms to have similar information in similar locations, but this is often impractical, if not completely impossible. There are not only the problems of space, but inventory, training, and a host of others. Worse yet, your requirements may change again before you get the new forms on line!

One of the important breakthroughs in character recognition systems came with the concept of attaching a very small computer to the optical reading machine. Initially, the first experiment in this direction was not too satisfactory because the cost of the computer used outweighed the cost of the reading machine by a factor of 2 to 1. More recent systems, however, such as the Control Data 915 Page Reader, use an on-line computer representing only a small fraction of the total system cost. Such a computer can accommodate not only the formatting of read information, the buffering of this, and the writing of it all to a mag tape—it can also (and this is the breakthrough) *control* the reading machine according to a stored program. Thus, the program may read the number of a form at the top of the page, select the appropriate stored program, instruct the reading machine to move 2 inches down, 3 inches across, read 8 numeric digits, advance paper by one inch, read an alphanumeric number of unknown length, inspect the last 2 characters on that line to make a decision as to whether to continue to read more information on that form, or to immediately use a fast paper advance to shoot the document into any one of two or more hoppers. Each form can have a separate program. We thus say that the concept of *stored program control of the reading machine* can result in enormous flexibility in system design and forms design.

In order to reduce the amount of programming necessary to accomplish this flexibility, compiler programs are available. To use the compiler, the parameters of a new program are typed on a sheet of paper and read by the reading machine. The compiler then generates a program for reading the desired specialized form.

Perhaps the difference between a "Page Reader" and a "Document Reader" should be emphasized at this point. A "Page Reader" is designed as a reading machine which can read many lines on a page; the size of the page is normally very flexible, ranging from a $3\frac{1}{2} \times 4$ inch document up to a 10×14 inch document and including Fanfold. A "Document Reader" is normally designed as a machine which can read three lines or less on a document; the paper motion is parallel to the lines of typing.

Outside of Business Data processing there are only a few solid opportunities for OCR. The Post Office problem is a major one, but the systems problem is so difficult that it may never be satisfactorily solved without much cooperation from the public.

Summary of performance promises

As mentioned earlier in this paper, the engineers have, by-and-large, delivered machinery which worked quite well when compared with the specifications which were agreed upon at the time of contract. In fact, the accuracy rate has gone sky high for machines using fixed fonts such as the USASI. Most users never check the accuracy of the machine at all. A typical machine guarantees an accuracy rate of less than one substitution per 50,000 characters read; this is a point at which the errors produced by humans in the systems are far more significant than any errors produced by the reading machine. In the area of making a system work more flexibly and easily, the presently available generation of program controlled Page Readers has given the systems designer a new dimension of freedom.

With respect to manually formed characters, promises were made only for *carefully* hand printed numbers. The reports from users of the IBM 1287 machine are so encouraging that many new applications for this form of data reporting are being considered.

There is a very interesting hole in the spectrum of successful promises as measured by the number of repeat orders. This hole appears in the machines which were designed for Multi-Font and Omni-Font work. The specifications for accuracy

of a machine reading 5 different fonts is something in the neighborhood of one reject per 1,000 characters and one substitution per 5,000 characters. In practice, these machines probably get one reject per 500 characters and one substitution per 2,500 characters. (It is quite correct in most cases, to allow an engineer to point out that the difference is almost invariably due to the fact that the quality of typing or printing didn't meet the specifications.) In Omni-Font reading, the only really powerful machines have been built for the Post Office and Social Security; the reject and substitution rates are probably of the order of one in a hundred and one in 500. Although these readers probably meet the negotiated specifications, it is clear that for Omni-Font machines a much better technique of recognition would result in greatly increased usage. Here, indeed, exists an enormous challenge for the engineer. The production of new ideas about how to read noisy junk continues at a splendid rate, and this all to the good. It is not unreasonable, if you want to call names, to say that the engineers have not yet kept their promises with respect to the Omni-Font reading machine.

With respect to the Multi-Font readers, however (those that read just a few fonts, well controlled) the promises that have not been kept are primarily those promises of the users, the maintenance people, and the systems designers. The point is that the reject and substitution rate of such machines is satisfactory for most business applications. The sum total of print quality problems is not unsurmountable for these Multi-Font applications. The most likely reason for having so few Multi-Font reading machines in service is that the systems design of such applications is so enormous that the whole job of re-design couldn't get finished. The basis for this complaint may vanish shortly, since there are some very impressive people working right now on Multi-Font systems.

What is being promised

Stylized font readers

We have, of course, both Page Readers and Document Readers using stylized fonts. We will certainly see more flexibility in both these applications, and this means a wider range of paper sizes, paper thicknesses, computer orders to the reading machine, etc. It is even possible that a pickoff may be offered which can handle documents of varying sizes and thicknesses within one stack, but this may require a fractional bending of the laws of physics. We will certainly see lower prices and higher throughput rates, perhaps both in the same machine. The systems support will be much wider and better. Along this line many general purpose programs will be available, and the user can also expect to find several varieties of compilers and assemblers available for use. Reliability and maintenance will both be improved. For certain types of remote terminal devices and point-of-sale devices, an exceedingly low cost is possible.

Recently there has been considerable publicity about optical techniques using the Fourier transform and the hologram. These ideas are fascinating, but some severe technical problems need to be overcome before such machines go commercial. Their most likely first application is in the reading of numeric sets.

Laser scan would undoubtedly be a big help to OCR if it can be developed to have non-mechanical wide angle deflection at a low cost. It would then replace the cathode ray tube for scanning.

Multi-font readers

A likely collection of fonts to be found in one reader might be as follows: 1) An upper and lower case stylized font such as USASI (although a lower case companion for the present USASI font has not been yet formally approved, it is expected that this will occur certainly within the next year). 2) A high speed printer font. This may either be nominally the same font as the USASI or it may be one of the sans-serif fonts in use on familiar printers. It is worthy of note that there is now in the field at least one high speed printer (such as the Control Data 1742) which makes a strong effort to produce high quality printing. 3) A 12 pitch font containing serifs, such as Elite. 4) Other fonts recognized by this type of machine will be primarily numeric sets, such as those generated by charge plates, cash registers, billing machines, etc. The chances are that the user will be able to order a collection of these which is specialized to his needs.

It is also promised that these machines will be able to read through a whole line of characters each one of which touches its neighboring characters. Most of the comments made above for the new generation of stylized readers will apply here also, with the exception that it is not likely that the price will go down. In the application of these Multi-Font machines, a very large effort will go into prior systems study. The Document Reader type of organization, rather than the Page Reader, seems to be the most profitable way to organize a Multi-Font machine at present, and it is predicted

that most of the sales of Multi-Font readers will be in Document Readers.

The Multi-Font Page Readers will be asked to perform Herculean tasks of separating readable information from information which is not supposed to be read by the machine, in addition to following large skew angles, large instantaneous jumps of characters with respect to the characters next door—all within the framework of a 6 lines to the inch situation! Instantaneous character reading rates of better than 10,000 characters per second will be offered. All of these requirements, while they can be fulfilled in a single machine, lead to nightmares for the engineer and, one suspects, for system managers.

Topological readers

The attractiveness of having human beings act as free keypunchers is so great that we will work very hard to give you the right machinery. The most likely possibility right now is the recognition of hand printed numbers. It seems not unreasonable that bank checks, for example, will have boxes for the numeric amount of the check. The customer will then fill in the amount in arabic numerals, and the reader will read these optically. Many banks have already adopted deposit slips with boxes for the account number. The advantage of boxes is twofold: First of all it separates the numerals well and secondly, it tends to normalize the size of the characters. It is predicted that machines can shortly be built which will read these numbers with the same accuracy that a human being can read them. The recognition of hand printed alphabet has now been announced by Recognition Equipment, Inc., and sales experience will indicate how valuable this capability is.

Readers for "Inhuman Fonts" (COC5, CMC7, E13b) will undoubtedly proliferate in special purpose situations in spite of their miserable appearance.

Magnetic ink machines will command an ever decreasing share of the market, however, because the scanning modes possible in OCR are so much more flexible than in MICR. This is primarily due to a world-wide shortage of magnetic lenses.

Omni-font readers

These will improve and be used with some success in situations where the input is historically uncontrolled. In order to bring real savings to these applications, however, strenuous efforts will have to be made to improve the quality of the inputs.

Cursive script readers

The target application is the U. S. Post Office, and the existence of this problem gladdens the heart of every OCR researcher. By 1975 a machine should be available to read 80% of the script addresses. More important than the reader, however, will be the vast increase in knowledge about automata and the human brain which will accompany the success.

CONCLUSION

There are two categories of "What Was Promised."

1. Promises made by authors.
2. Promises made by engineers in their specifications for machines.

The performance on category one has been pretty spotty, and may get worse, but performance on specifications has been good for most projects.

"What We Have" is an excellent capability for most intra-company business data, a modest capability for inter-company transactions, a poor capability for data collection, and practically no capability for uncontrolled communications.

"What Is Being Promised" . . . source data collection will expand one hundred times in five years . . . forms for inter-company transactions will be standardized by public and private agencies . . . discovery of methods for reading cursive script will illuminate the problems of human language.

Perhaps, before we lay down our soldering irons, these machines will send letters to each other in clear text:

"Dear Complex 479:

Please be advised that your 47th print hammer is badly out of synchronization . . ."

REFERENCES

1 L N KANAL (editor)
Pattern recognition
Thompson Book Company Inc Washington DC 1968

2 A W HOLT
Comparative religion in character recognition machines
Computer Group News IEEE November 1968

High-speed logic and memory—Past, present and future

by ARTHUR W. LO

Princeton University
Princeton, New Jersey

INTRODUCTION

By 1950 the logic and memory functions required in data processing were successfully implemented by vacuum-tube technology to demonstrate the feasibility of electronic computers. In the decade of 1950-1960 a large variety of solid-state switching and storage elements and associated circuit schemes were proposed, and pursued to various extent through the research, development and production phases. The present decade, 1960-1970, has witnessed the consolidation of hardware technology to semiconductor logic circuits and magnetic random-access memories to achieve cost and performance advantages in practical systems. The overwhelming success of the semiconductors and the magnetics appears to deter the development of other technologies. Whether or not this trend will continue in the next decade, 1970-1980, depends on the magnitude of the demand and the advancement of science and technology.

This article provides a brief account of what was promised, what we have, and what is being promised in high-speed logic and memory hardware. It aims to show the reasoning behind the past promises, why we have what we have, and what is reasonable to expect for the near future. A chronological sketch of the major developments in hardware technologies for high-speed logic and memory is shown in Tables I and II to aid the discussion.

No hardware technology can be successful unless it lends itself to the fulfillment of certain universal device and circuit requirements pertinent to digital operation. The "function-composition" nature of digital information processing systems makes possible the synthesis of complex system

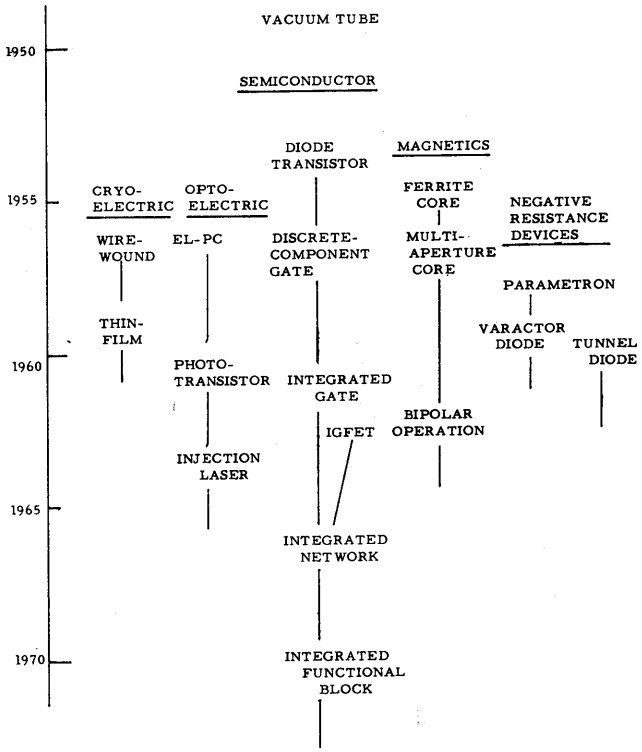


TABLE I—Chronology of high-speed logic development

functions from primitive operations. The physical realization of such systems relies on the interconnection of a large number of elementary circuits to form complex electronic networks. The requirement of extensive and flexible *interconnection* dictates the fundamental properties of the elementary circuits. In order that arbitrary combinational and sequential logic functions may be implemented with least constraint, it is essential that the elementary circuits (the logic gates which performs the primitive logic functions) can be "free-

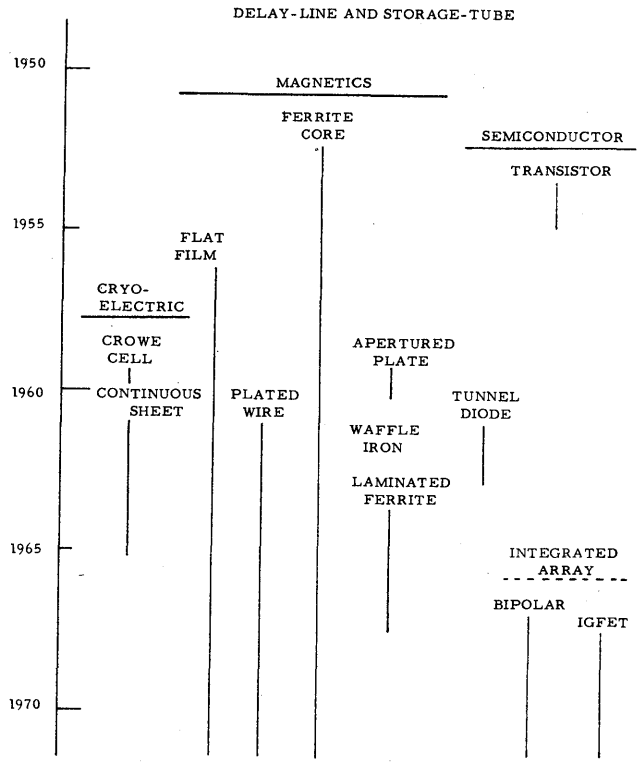


TABLE II—Chronology of high-speed memory development

ly" interconnected with one another without obstructing their basic operations. The interconnected gates must respond to one another according to the design and with little delay. These basic requirements demand that the elementary logic circuits must have the following characteristics:

(a) Signal Quantization. The provision of signal quantization ensures that the output signal of the elementary circuit is more true to the specified "0" or "1" representation than the input signal, so that there is no signal deterioration in a logic network of any size.

(b) Directivity and Isolation. The provision of directivity and isolation of signal flow ensures clear designation of "cause and effect" to prevent spurious interactions between connected circuits.

(c) Fast Switching. The minimization of the switching delay time between switching circuits ensures the operation speed of the digital system.

Most of these desired characteristics of the elementary circuit are provided by the *switching element(s)* employed in the circuit. The quantization requirement demands switching elements having a sharp and uniform switching threshold, as well as an operating region of high gain. These properties ensure reliable switching of the element by small input voltage and current excursions, under the realistic constraints of noise, component tolerance, and fan-in and fan-out loading. The requirement of directivity and isolation of signal flow calls for switching elements of unilateral isolation property or the use of unidirectional coupling elements. The switching delay time of a digital circuit is essentially determined by the time required to charge or discharge reactances in the circuit to produce the specified signal voltage or current excursion. Reduction of component size and high density circuit construction are essential to reduce the intrinsic and parasitic reactances associated with the devices and their interconnections. In summary, the physical implementation of high-speed logic demands high-grain, highly nonlinear unilateral switching elements which can be readily fabricated to very small size and high uniformity, and interconnected to one another in high density.

In the case of memory, very-high-speed, small-capacity information storage is usually intimately connected to the logic circuits, and the same hardware technology serves for both logic and memory. For memories of storage capacity of thousands of bits to millions of bits, some efficient, random-access address-selection arrangement is indispensable. Such memory is usually arranged in the form of an array, or planes of array, of storage elements with provision of selective addressing. The major considerations for high-speed, medium and large capacity memories are as follows:

(a) The storage elements must provide discrete, static or dynamic remanent states to retain the stored information, with little or no standby power.

(b) The storage elements should provide simple means of selective addressing, and they must be immune to repeated disturbs (produced by digit-drive, partial word-drive, and spurious coupling). A foremost requirement is that the storage cell has a sharp, stable and uniform switching threshold.

(c) High-speed operation requires that the storage elements can be made to minimal physical size (and with high uniformity) and "wired" into high density arrays.

(d) The large number of storage elements needed in each memory unit puts the

emphasis on high-yield, mass fabrication techniques for the production of storage elements and memory arrays at reasonable cost.

This outline of the fundamental device and circuit requirements for high-speed logic and memory implementation provides the basis of a systematic appraisal of the merits and limitations (leading to the success and failure) of the various hardware technologies, present, past and future.

High-speed logic

What was promised?

The large variety of promised hardware approaches for the implementation of high-speed logic are summarized under the following categories.

Semiconductor diode, transistor and integrated circuits

From the earliest stage of development, the semiconductor transistors and diodes, with their obvious advantages in size and power consumption, offered good promise of their replacement of the vacuum-tubes in logic circuits. The advancement of semiconductor technology soon revealed the significant advantages in reliability and life, switching speed, and manufacturing cost of the semiconductor devices, several orders of magnitude over what could be achieved with vacuum-tubes. A unique property of the semiconductor junction devices is that the pn junction voltage-current characteristic provides a switching threshold which is not only very sharp but also "naturally" uniform in the sense that the threshold value is insensitive to the physical dimensions and the fabrication processes of the device. Thus diodes and transistors can be readily fabricated to extremely small size (where dimensional variance is unavoidably large) and in large quantity, while having uniform switching thresholds. The sharp and uniform threshold permits reliable switching by voltage excursion of less than a volt. The minimal device capacitance (from the small physical size) and the high transconductance of the device provide fast switching. Early promise of logic subsystems composed of thousands of transistor logic gates with logic delay of a fraction of a microsecond and cost below $1.00 per gate were readily surpassed in the late 1950's. The phenomenal success of planar silicon transistor technology, consisting of the processing steps of epitaxial growth, oxide passivation, photolithographically controlled selective impurity-diffusion and metal-deposition, made practical the batch-fabrication of large number of components (diodes, transistors and resistors) of extremely small size and great uniformity at low cost. The same technology readily makes the interconnections between components to form logic gates, and the interconnections between gates to form logic networks, all at the surface of a silicon chip. Following the undisputed demonstration of the producibility of integrated-circuits and their advantage over discrete-element assembly, came speculations of larger- and-larger scales of integration, all the way to "computer on a chip," as well as delay-time "limited only by the speed of light" and cost of "practically nothing."

In the decade 1950-1960, when the need of practical high-speed logic hardware was urgent and much of the potentials of transistor logic circuits were not fully appreciated or realized, a large number of solid-state devices and circuits were seriously considered for logic implementation. The major developments and expectations of these approaches are outlined in the following sections:

Cryotron logic circuits

The early wire-wound cryotron offered an electronic analog of the electromechanical relay, with the potential advantages of higher speed, longer life and lower power consumption, despite the obvious cryogenic requirement. The advent of thin-film cryotron structure prompted great expectation of batch-fabrication of large and complex logic networks by the simple process of vacuum-deposition of homogeneous materials. An inherent drawback of cryotron logic circuit is the low operation speed dictated by the large L/R time constant (even after the introduction of superconductive ground-plane which drastically reduced the circuit inductance). Efforts to improve operation speed by size miniaturization and the use of higher-resistance metal-alloy encountered difficulties in the uniformity of the switching threshold, which is directly related to line-width and critically dependent on temperature (thus ohmic heating in switching), as well as material and fabrication imperfections. The success of cryoelectric logic system hinges on the ability to batch-

fabricate large, complex logic networks economically, but the present status of cryogenic science and technology has not warranted such undertaking. The promise of entire logic and memory system batch-fabricated at low cost by the cryoelectric technology is not likely to be fulfilled in the near future.

Opto-electric logic circuits

The early EL–PC pair, employing polycrystalline electro-luminescent phosphor and cadium-sulfide photoconducting material, promised low-cost production of logic circuits by the "printing-press" process. However, the cost of assembling, interconnecting, and protective packaging of these circuits for sizable logic systems was not low, and the operation speed was intolerable. More recent opto-electric logic, using semiconductor light-emitting injector-laser and light-sensitive photodiode (or transistor) still could not achieve operation speed comparable to transistors. The major difficulty is that, for any realistic network interconnection, only a small fraction of the emitted light from a driving element can be effectively coupled to the active region of a receiving element to realize the high gain required to achieve effective quantization and fast switching. There has been few promises of practical opto-electric logic except for special purpose applications.

Magnetic logic circuits

Square-loop magnetics provide static storage and stable switching-threshold, but they are handicapped by the lack of gain and unilateral property. The development of "all-magnetic" logic was more a challenge than a practical solution. The use of sequenced clock-pulse energization and multi-aperture cores has produced logic circuits operating at low speed and high power level. The more recently developed "bipolar" operation overcame much of the earlier drawbacks, but the cost and performance advantages of the magnetic circuit were, by then, lagging far behind transistor circuits. The promises of magnetic logic were limited to its high reliability against noise, its immunity to radiation damage and its apparently unlimited life.

Parametric phase-lock oscillator logic circuits

The phase-script, carrier-powered, logic circuits (beginning with the Parametron) promised immunity to noise (but not spurious interaction between circuits) and the potential of extremely high frequency operation (using varactor diode and microwave structure). The majority-logic operation, however, demands uniform oscillation amplitude and coupling attenuators, as well as precision phase-control of the transmission of signal and carrier power at microwave frequency. The practical aspects of interconnecting complex microwave structures make the implementation difficult and costly even for small logic systems.

Tunnel-diode logic circuit

The two-terminal, negative-resistance, semiconductor device (with its majority-carrier conduction mechanism) attracted early attention on account of its subnanosecond switching time. The problem of uniformity of switching thresholds among units (their peak-current being directly proportional to the tiny junction area) resulted in low gain, and consequently longer switching time than promised. The balanced-pair circuit (requiring only matched-pair instead of uniformity of all units) overcame some of the operation difficulties, but the need of sequenced clock-pulse supply and precision coupling attenuators makes the realizable operation speed no better than modern transistor logic circuits. The voltage-limiting property of the high-speed device might still be useful if it could be conveniently integrated to the more conventional semiconductor elements.

What we have

Presently the field of high-speed logic is monopolized by the semiconductor integrated-circuit technology. Technical developments are concentrated on smaller component-size and higher component density. A typical transistor at present may have emitter strip-width of 0.1 mil and base width of 0.4μ. Such a transistor can be switched reliably in one nanosecond by a voltage excursion of less than one volt. The basic logic gates, composed of transistors, diodes and low-precision resistors have logic delay time of 2-20 ns and power consumption of 10-15 mW. (The integrated-circuit technology has all but obsoleted the TRL and DCTL gates, in favor of the DTL and the TTL gates for cost-oriented applications, and the ECL gates for speed-oriented applications.) The present scale of integration is 10 to 50 interconnected gates on a chip (corresponding to density of $10^4 - 10^5$ components per square inch), for the

cost of about 25 cents per gate.

The Insulated-gate Field-effect Transistor has been successfuly developed to rival the bipolar transistor as practical basic switching element. In comparison with the bipolar transistor, the IGFET is handicapped by the lack of a "naturally" uniform switching threshold and its low transconductance at lower operating voltage. Thus the basic IGFET inverter requires a larger voltage-excursion and a larger "load" resistance for reliable switching, and consequently has a longer switching delay time (ten times that of the bipolar transistor inverter). The shortcomings of the basic IGFET circuit are partially overcome by the pulse-mode circuit (at the expense of sequenced-pulse power-supply) and the complementary-symmetry circuit (at the expense of more complex device fabrication). The IGFET circuits are not likely to match the performance of the bipolar-transistor circuits for general-purpose, high-speed logic, without some significant advancement in IGFET technology. The inherent advantage of the IGFET is its simpler device structure which lends itself to higher degree of miniaturization and higher fabrication yield (than the bipolar transistor), thus more suitable for practical large-scale integration.

The limiting factor of larger-and-larger scale of integration on a chip is essentially determined by the yield of the completed network. Presently there are two distinct approaches to large-scale integration. The fixed-pattern wiring (metalization) approach requires all the components covered by the interconnection pattern to be free of defects. Integrated logic networks of 10 - 30 bipolar transistor gates and over 100 IGFET gates have been produced in this manner. Larger integration by this approach suffers severe yield shrinkage. The discretionary-wiring approach allows reasonable imperfections among the fabricated components to make larger integration feasible, only at the expense of tedious testing at the component level, considerable data processing to formulate interconnection patterns, and the cost of preparing individual metalization masks. At the present time the fixed-pattern technique is used for the majority of integrated-circuit production. and the discretionary-wiring technique is used for some products.

What is being promised

The present trend of logic hardware is larger-and-larger scale of integration on a chip, since the interconnection between chips is largely responsible for the degradation of speed, reliability and cost of present logic systems. This fact prompts the promise that by early 1970's we will have 1,000 - 10,000 interconnected gates on a chip, with component density of $10^5 - 10^6$ components per square inch and switching delay time of 2 - 5 ns, at the cost of 5 cents per gate. With this many interconnected gates a single chip would constitute the entire electronic network of a complete logic subsystem, thus greatly reducing the number of terminal leads to the chip. The manufacturing of "functional-units" of such complexity can be achieved only by automation to handle the tasks of logical design, device and circuit layout, and diffusion (and metalization) mask design, as well as automatic fabrication and testing. The fulfillment of such promise is largely governed by the law of demand and supply. The functional-units which have large-volume demand will be developed and produced, and the others will be neglected. The progress in this direction will depend heavily on the standardization of computer structure and the consolidation of computer types. Never before is there such urgent need for interaction between software and hardware planning. "What to make" and "How to test them, when made" deserves as much attention as the technology of producing the hardware.

It is of interest to note that the sobering experience of past developments and a better understanding of the basic requirements of digital circuits have refrained speculations of revolutionary logic-implementation in recent years. Nevertheless, there are expectations that new and significant progress will be made *if* and *when* certain difficulties associated with a number of hardware technologies are overcome or bypassed. Germanium transistor operating at low temperature can provide very high speed logic if the fabrication techniques of germanium can reach the same level of silicon technology. The feasibility of all-optic logic for extremely high speed operation depends on the development of practical means of efficient light-coupling between laser-like elements. Large-scale cryoelectric logic and memory systems (at extremely low fabrication cost) could become practical through the discovery of superconductor material of higher normal resistance and higher operating temperature (say, liquid nitrogen temperature). Practical use of two-terminal, bulk-

effect semiconductor devices for high-speed logic awaits the development of uncomplicated means of providing directivity and isolation. All-magnetic logic based on flux steering of magnetic domains within homogeneous material (thus eliminating signal-path interconnections) could be feasible with the development of practical means of providing flux quantization and signal directivity.

High-speed memory

What was promised

A major consideration in the development of high-speed, large-capacity memory systems is the cost of producing and assembling the large quantities of storage cells. The magnetics has been, and still is, the dominating memory technology.

Ferrite core-memory

The ferrite core memory technology has the distinction of having promised little (in storage capacity, cycle time and system cost), and delivered much. By 1955 the main memories of all major computers were being implemented with this technology. The ferrite core, with its inherent static-storage property (without standby-power) and material stability (polycrystalline ceramic material), easily obsoleted the electro-acoustic delay-lines and the storage tubes. The switching threshold of the magnetic core is directly related to its physical dimensions and material imperfections, which are difficult to control in the high-temperature, high-pressure processing of the polycrystalline material. The cores, therefore, are tested individually to meet the actual operating conditions (including the large number of disturbs) before wired into the array. This enforced device uniformity, and the tremendous demand of random-access memory of adequate speed and capacity to match the existing logic circuitry, resulted in the production of million-bit memory with read-write cycle time around 5 μs by 1960. The cost of the core-memory systems was found to depend more and more on the driving and sensing circuitry, which followed closely the advancement of the semiconductor technology. Three types of memory organizations, 3D, 2D and the middle-ground 2½D, were adopted to fit the compromise between storage capacity, cycle time, and system cost for any particular, random-access memory application. The feasibility of multi-aperture core for reliable nondestructive read-out, and two-core-per-bit partial-switching operation for higher speed were also demonstrated, but not widely adopted on account of the added complexity and cost. The promised batch-fabricated ferrite memory, such as apertured-plate, laminated ferrite and waffle-iron, did not achieve the expected cost advantage, mainly because of yield and the burden (of switching excess magnetic material) imposed on the drive-and sense-circuitry.

Magnetic thin-film memory

The anisotropic magnetic thin-film shows the promise of exceedingly fast coherent-rotation switching, convenient orthogonal drive operation, and planar structure for easy batch-fabrication. The development of practical flat-film memory, however, has been slower than expected. The open-flux path structure imposes a compromise between dot size and film thickness, and the output signal voltage, thus prohibiting very high density array structure. The desired uniformity of switching and disturb thresholds are difficult to achieve with present material and batch fabrication techniques. The more recently developed cylindrical-film (plated-wire) memory elements provide a closed flux-path in the easy axis (thus permitting the use of small-radius wire and thicker film) to overcome some of the difficulties (particularly disturb sensitivity) of the planar thin-film. The promise for these thin-film memories is that, once the material and fabrication problems are solved, improvement of memory storage capacity, cycle time, and system cost by one order of magnitude over the core memory should be readily achievable. A comfortable thought is that there is no lack of large demand for multi-million-bit, sub-microsecond memory at present and in the foreseeable future, provided the price is not outrageous.

Cryoelectric persistent current memory

The development of cryoelectric memory, (from cryotron-cell, to Crowe cell, to continuous-sheet and cavity-sensing) promised very-large capacity random-access memory operating at modest speed. The task of producing high-uniform, high-density array with good yield remains a challenge to our material and fabrication technology.

What we have

Core memories of 3D or 2½D organization,

with capacity of multi-million-bits and cycle-time of 1 - 2 μs, dominate the field of internal main memory in the present generation of computers. Some flat-film and cylindrical-film memories of 10^6 bit and 0.5 μs cycle-time have been installed in working systems. Memory systems cost is difficult to assess, but it appears to be in the range of 1 - 5¢ per bit. In the area of high-speed, small-capacity random-access memory, the magnetic thin-film technology (both flat-film and cylindrical film) has produced memories of 5×10^4 bits with access-time (for read or write) of 100 - 200 ns.

Semiconductor, integrated-circuit technology is making an inroad to the field of very high-speed memory. Development efforts on the fabrication of large arrays of semiconductor memory cells and interconnections has shown that IGFET memory of 10^4 bits with access-time of 100 ns, and bipolar-transistor memory of 10^3 bits with access-time less than 50 ns are technically feasible.

What is being promised

The normal progress of core-memory technology will see some further reduction of cost so that it will be economically feasible to have tens of millions bits of random-access storage (with cycle-time around 1 μ s) in the present generation of computers. There is strong expectation that the evolution of thin-film technology (flat-film and plated-wire) will finally reach the goal of low-cost batch-fabrication, thus providing memories of $10^6 - 10^7$ bits capacity, 200 - 500 ns cycle-time, at the cost of a fraction of a cent per bit. Integrated IGFET memory is expected to dominate the area of small-capacity storage (less than 10^6 bits) with access time of 100 - 200 ns, at the cost of 1c per bit. The semiconductor technology also shows good promise for merging the logic and the memory functions to make medium-capacity associative-memory economically feasible and to allow much more sophisticated computer organizations.

Much attention is currently focused on the development of block-oriented random-access memories. One prospect is the magnetosonic delay-line memory which employs magnetic storage and block-access by semiconductor electronics (to cause the propagation of a sonic wave in the selected line). Nondestructive read-out is derived on the digit lines in sequence by the propagation sonic wave, and write-in is carried out by the coincidence of digit currents and the propagating wave. Another prospect is the opto-electric read-only memory, where the stored information on high resolution photographic plate is block-selected by optical means, employing light-beam deflection or an array of light-emitting elements. The optical readout (of all the bits in the block in parallel) is converted to electric signal by an array of photo-sensitive elements. Holographic techniques are proposed for the implementation of the high-density photographic processing. The practicality of these block-oriented systems are too early to be realistically appraised.

CONCLUDING REMARKS

The present hardware technologies for high-speed logic and memory have reached a high level of proficiency, making the central processing unit far ahead in cost and performance in computer systems. Evolutionary progress of these technologies, particularly in batch-fabrication techniques, will provide more logic and memory at less cost, but only slight improvement in operation speed. There is, however, no sign of any revolutionary hardware implementation of high-speed logic and memory in the near future.

Real time systems and public information

by C. WEST CHURCHMAN

University of California
Berkeley, California

It is perhaps obvious that computer technology will open up a new era in public information in the coming decade. What is not so obvious at all is the proper design of the future information systems. Nor do I sense any simple answers.

The questions that come to mind with regard to public information systems are these:

1. Is it feasible to build systems which will more accurately and quickly inform the public on major issues of social policy?
2. If so, is it desirable to build such systems?
3. Is it feasible to build systems which will more accurately report to the legislative and executive branches of government the opinions of the public on social policy?
4. If so, is it desirable to build such systems?

As I say, none of these questions seem to me to have an obvious answer. In the case of the first, it is easy to point out that there exist "manual" systems, like the *New York Times* or the radio-TV, which do a reasonably good job of informing the public, but which are not widely used. Are we to expect that merely increasing the facility of access will lead to wider use? But is a more informed public socially desirable? And if legislators could react more quickly to changes in public opinion, would this be desirable?

In order to make the point that these questions have no simple answer, I'd like to contrast two "world views" of the public and its value system, both world views being plausible, but each leading to a different answer to the questions I've posed.

In the first world of the public, the citizen is viewed as a living entity who seeks to attain certain "life goals." In a democratic society, these goals may vary considerably and often will come in conflict. The aim of government is to resolve such conflicts fairly, and to use the national resources to come as near as possible to aiding as many citizens as possible to attain their respective goals. In such a world, information plays a very important role. The citizen needs to know the policies of the government with respect to matters that are relevant to him. If it is the case that the government plans to implement a policy, the citizen should be able to inform his representatives about the consequences of such a policy *vis-a-vis* his goals. The measure of performance of the information system is the extent to which it removes ignorance, ignorance on the part of both citizen and representative. Consequently, the answer to all four of my questions, according to this world view, is "yes," but the critical point is that the information system needs to be supplemented by a vast educational program. The citizen needs to learn the value of information. He must come to realize that acquiring and transmitting information is an essential part of his value system, so that he will freely seek to be well informed. Hence the government must see to it that the information systems are of the highest quality, and that the public is made aware of their value.

Admittedly, this first world view is a bit simplistic but when painted in more complicated detail it becomes quite attractive to the social planner. It justifies a social commitment of large resources to the building of better information systems.

The second world view is also plausible, but by no means as attractive to the system designer. It says that there is no such thing as accurate or objective information, especially in the context of social policy. Instead, so-called "information" is simply one kind of incentive, which can be used by one person or group to influence the behavior of another person or group. It is, in fact, a commodity with its own price, a commodity that serves the purpose of shaping social action. When a person "informs" another, he uses up some of his commodity for the sake of getting the other person to act in ways desirable to him. Thus one man "informs" the citizens of the USA that the North Vietnamese massacred 60,000 Catholics, while another "informs" us that the United States Air Force in its bombings massacred an equivalent number of North Vietnamese civilians. Both are us-

ing their information commodity to attain a reaction, either for or against the war. Note that both must be concerned with timing; the commodity is in fact "used up" when information is transmitted. The repeated use of the same information has a quite different effect from the first application.

According to this second world view of public information, the answers to my four questions are generally in the negative. By creating more powerful information systems, we shall be giving certain power groups a tool to manipulate public action. Which group benefits depends on who designs the system, but the net effect will be one in which political power is enhanced. Inevitably, therefore, a reaction will set in, either civil war or fascism, if only one public information system is designed. What needs to be examined at the outset is the legal and moral aspects of such systems, and this examination will inevitably lead us to conclude that there should at least be competing information systems. In other words, there should be a free market of the commodity.

This world view is also rather simplistic. In order to enrich it, we might listen some more to the protagonist of the first world view. In his rebuttal, he will point out that of course the designers of the system need to inform the public about the reliability of the information. It is true, he says, that so-called information is often mere subjective opinion. But well documented and objective facts are not the inventions of scheming minds; they are "there" and cannot be changed. A fact is a fact, after all.

To this rebuttal, the other side points out that there are two ways to deceive a person even though what you tell him is the "truth". One way is to tell him only a part of the story, i.e., to select the data. It is not the case that the *New York Times* prints all the news that's fit to print, because such a task is impossible. Instead it filters out "uninteresting" news, using its own value system to do so. The second way is to interpret the data. No fact is "pure". Whenever I tell you something, I must include shades of meaning and inference, either explicitly or implicitly. This is especially true when the data are supposed to be relevant to policy making. If I tell you that the USSR will produce more engineers in 1970 than the USA, this is not a pure fact, especially if the telling occurs in the environment of alarm about the arms race. The "datum" is attached to a world view, and cannot be detached from it. Hence it is simply impossible to warn the citizen about reliability in any real sense. Furthermore, it is doubtful whether there is any true scale of reliability of information.

The debate is not apt to have any end, because both sides argue in the context of ignorance. Neither one has knowledge of the real social system, though each can tell a plausible story about his view of it. My sympathies are with the second side, because I suspect that the first is somewhat dishonest: by trying to insist on objective information, it also tries to escape from moral commitment. Its proponents want us to believe that a fact stands there independent of one's purposes. Hence they believe that the designer of information systems does not have to declare what his opponent takes to be the implicitly assumed moral assumptions. Furthermore, I believe that all too many legislators and executives aid and abet this moral deception.

I wouldn't vote for the second viewpoint if it ended in relativism, i.e., in the position that information can mean whatever one wants it to mean. Rather, I take it to be saying that (1) there is such a thing as morality in social policy making, (2) the morality dictates in part which goals are legitimate and which are not, and (3) that information is valid which directs us towards the moral goals. That is, the design of an information system ought not to be accomplished without an explicit moral commitment. Finally, no one should claim expertise on morality, which is the voice of all people, past, present and future. Morality is above all a matter of experience, or even of experiment, in which all of us are both subjects and observers.

The unpopular antithesis I've been developing in this paper says in effect that as a culture we are morally immature, and that this immaturity comes to be a serious threat every time anyone proposes a technological innovation. We don't understand the moral basis for building freeways, supersonic transports or public information systems. We are not mature enough to deserve the fruits of technological change. Because we are childish in our morality, we believe we can keep shifting moral responsibility. We say that it's not up to the system scientist to find the moral basis for an information system; this is a matter for managers or the public to decide. No study of morality ever seems to get into the proposals or RFP's. We even childishly assert that it's meaningless to try to study moral issues because people have been trying for centuries "without success." Only a child believes that "success" means satisfaction and removal of the problem. Every adult knows that serious problems are never resolved, and must be studied continuously and deeply.

The antithesis is bound to be unpopular and probably will go unheeded. No child likes to be told that he's behaving like a child, especially when *physically* he is quite strong and mature.

National and international information networks in science and technology

by H. BORKO

University of California
Los Angeles, California

The 20th Century has been variously called the atomic age; the age of the second industrial revolution and the century of the computer. However the historians choose to label this period, it is clear that science and technology has had a central role and a significant impact on our society. Research and engineering are no longer considered esoteric disciplines, for their products are clearly visible. Although the importance of science has been recognized and accepted for many years, only now are we becoming concerned with the results of these scientific activities. It is not enough to produce new knowledge; the information must be made available so that it can be tested, evaluated, and applied for the benefit of the community.

New knowledge is being produced at an ever increasing rate by an ever increasing number of scientists so that the sheer volume of output is clogging and constricting the information transfer processes. Scientific literature has increased to such an extent that the unaided scientist is unable to keep abreast of, and make use of, the information relevant to his own research. The growth and proliferation of the literature could rapidly produce chaos and unnecessary duplication of effort. This could happen and would happen were it not for the development of information science centers that control the collection and dissemination of science information.

A few such centers are already in existence and many more are needed, as are more efficient procedures for the storage, retrieval and dissemination of scientific information. Conceptually, the idea of an information center is an old one since such centers evolved from the research library. However, the modern computerized information center is a relatively new development requiring, in addition to the high speed computer, a large capacity storage device and specialized programs for the processing of requests and the manipulation of files. These capabilities now exist and are being used in a number of specialized information centers. Although these centers are reasonably efficient, they are expensive to operate and there are frequent delays in responding to the user's request.

The information transfer problem is world wide and must be approached at both national and international levels. A number of significant steps—both technical and administrative—have already been taken toward the creation of information centers. It is important for us to review these activities in order to understand their present status and the implications that an efficient information exchange can have on the social, economic and political milieu in which we live. In the United States, our leaders have taken cognizance of the new technology and have recognized the importance of expediting the exchange of scientific information. As an illustration of this fact let me quote from a congressional report of the U.S. Senate Committee on Government Operations, June 1965. This paper is usually referred to as the Humphrey Report. It states:

> "Information is an agency resource, a federal, national, and international resource.
>
> Modern information technology is making it possible to place much of the accumulated knowledge of the human race within the reach of a man's fingertips, so to speak. The potentialities of this access to power are awesome, in terms of improving the well-being of our own and other people, as well as in terms of improved education for young and old alike.
>
> If man's collected knowledge is to become truly

accessible, plans and programs must be made, priorities assigned and resources allocated." [1]

Note Senator Humphrey's statement that, "Modern information technology is making it possible to place much of the accumulated knowledge of the human race within the reach of a man's fingertips. . . ." Clearly, he is talking about computer technology, and the reference to "a man's fingertips" alludes to remote access via time-shared systems. The phrase that priorities need to be assigned and resources allocated is indicative of a growing sense of urgency. Of course, after this report had been written, other national problems have received even higher priorities, but the need for information transfer has not diminished. The problem remains and has been re-emphasized by Carter's report on National Document-Handling Systems for Science and Technology. [2] Carter writes:

> "The essence of scientific and technical information transfer is contained in the process of communication. The development of scientific knowledge depends on the communication of new theories and new experimental observations to others. No new idea is afforded acceptance in the scientific community until it has been discussed and evaluated by people knowledgeable in that area, whose knowledge of the idea and background for evaluating it is derived from formal and informal communications."

As a result of the work of these and other study groups, federal agencies have given increased attention and support for activities in the field of scientific and technical communication. Important improvements have been made and more are under way as both governmental and non-governmental agencies move toward the development of an effective national program. The goals are threefold: To increase the effectiveness of R & D management in the Federal government; to make readily available to all U.S. scientists and engineers the world's current and past output of significant scientific information; and to encourage the transfer of defense generated technology to the civilian sector in order to induce comparable advances in the total economy.

Within the federal establishment, the significant actions in the scientific and technical information program may be summarized as follows: [3]

- The Department of Defense has redesignated the Armed Services Technical Information Agency as the Defense Documentation Center in order to reflect its broadened mission and the enlarged scope of its document collection. Under the supervision of the Director of Defense Research and Engineering, the Center has two specific new assignments: the maintenance of an index of current "Research Development Tests and Evaluation" programs within the department and the establishment of a centralized directory and referral service on available DOD Scientific and Technical information activities.
- Within the Department of Health, Education, and Welfare, committees on scientific information have been established to survey information needs and to examine scientific and technical information activities in the Public Health Service, including the National Institutes of Health, and the Food and Drug Administration. The National Library of Medicine, which is a component of the Public Health Service, is currently operating a computer-based Medical Literature Analysis and Retrieval System called MEDLARS, and a National Drug Information Clearinghouse.
- The National Bureau of Standards, under the Department of Commerce, has established a National Standard Reference Data System which provides scientists and engineers with critically evaluated numerical tables in the physical and engineering sciences.
- The Office of Information Service in the National Science Foundation has sponsored numerous research projects in information science so as to improve the collection and dissemination of information. In addition this Office has helped support the establishment of many specialized information centers including the Science Information Exchange which monitors research in progress, and the National Referral Center. This latter agency is housed at the Library of Congress and directs any scientist or research organization to the most appropriate source of information on any scientific or technical question. In so doing, the Referral Center interconnects the immense resources of information services currently available in the United States.
- One of the largest sources of report literature is the Office of Technical Services in the Department of Commerce. This agency administers the Clearinghouse for Federal Scientific and Technical Information, and provides a central and ready source of information from which the user can get copies of government reports, scientific papers and translations of Russian and other foreign works in specific fields.

In the private sector, there are in existence a large number of specialized information centers in the fields of agriculture, engineering, physics, chemistry, etc., and more are being planned.

All of this activity is convincing evidence that a number of very significant steps have been taken to disseminate information in a manner that will make government-sponsored research available for public use, minimize duplication, and aid in the development of new products and processes. But while these efforts are coordinated, they are not truly integrated. There is a need for a national document-handling system for science and technology, and by this I mean an integrated system in which all of the individual centers are interconnected and cooperate with each other in sharing their resources, and this includes the responsibility for gathering and translating foreign language periodicals. No one organization has sufficient resources—material and personnel—to collect everything. Only through an organized cooperative effort can complete coverage be obtained.

An integrated system also requires the standardization of procedures, format rules, and terminology. This will be very difficult to achieve as anyone working with the United States Standards Institute will testify. And no organization that has an existing operational system would be interested in changing their procedures. A national scientific and technical information center will also be expensive, and it is only reasonable to ask whether the effort and the cost will be worthwhile. Unfortunately, there does not seem to be any way of placing an acceptable value on information, and it is even more difficult to determine the value of that information that has been lost or not retrieved.

While the problems are numerous, the social and economic implications of such a network are also great, and this is particularly true when we broaden our perspective to include the international scene. Whereas within the United States the main impetus for the establishment of an information network was to transfer information from one part of the economy to another—from military support programs to civilian utilization—the international problem is a different one. Information is seen as a means by which a developing nation can increase its gross national product, raise the standard of living for its citizens and narrow the gap between the have and the have-not nations of the world. In short, information is seen as a very valuable commodity which should flow freely across national boundaries to the mutual benefit of all. There are a number of international science information activities either in existence or in the planning stage, but they are for the most part narrow in scope and reflect the dominant interest of the sponsoring organization.

One such information exchange agency is sponsored by the Committee for International Cooperation in Information Retrieval among Examining Patent Offices. The committee's objective is to facilitate all types of patent searches, and to cooperatively develop and share procedures for improving patent retrieval. By working together, the 21 national offices have developed special indexing systems and have agreed on the standardization of microforms, of punched-card formats, of specification headings, and of other relevant matters.

EURATOM, the European Atomic Energy Agency, has successfully operated an international center on nuclear information. The center collects documents and provides fast and reliable access to its files. Additionally, a proposal has been made by the International Atomic Energy Agency to create a computerized International Nuclear Information System which would provide for more efficient storage and transfer of nuclear information.

Other agencies, such as the Food and Agriculture Organization, the World Health Organization, the International Federal of Library Associations are also engaged in science information activities, but as I indicated, these are all specific, uncoordinated, and separate.

Recently, in December 1967, the International Council of Scientific Unions and the United Nations Educational, Scientific and Cultural Organization convened a joint project to study the feasibility of a World Science Information System. Such a world system is now possible; although there are a host of problems that need to be solved before the system could be implemented. I do not wish to minimize these problems for they do exist and they are concerned with such technical things as communication lines, memory size, file organization, data elements, compatible terminology, etc.

Because the members at this audience are experts in computer systems and data base management, I will not dwell on these technical matters. If we thought about it for a few minutes, each of us could list a dozen or more problems. And then if we thought about it a little longer, we might even come up with some solutions. There are no

insurmountable problems in technology. However, the management problems are less obvious and because of the political implications, more difficult to solve. Namely:

- Who should manage such a world science information system? Should it be a committee of interested and involved countries? Should the Center be organized under the auspices of the United Nations? Or should the leadership come from the professional societies such as IFIPS or the International Federation for Documentation?
- Should the service be free to all qualified personnel East and West? Or should there be a metered charge for each delivered item of information?
- Granted that scientific information is a valuable commodity, should the United States, share this information without any strings, or should the government insist upon a *quid pro quo* arrangement?
- Who shall be responsible for gathering and reviewing the documents that are to be part of information system? Shall this responsibility be delegated to the country, the professional societies or some international body?

These and similar questions are being debated by a number of international organizations. The debates will be long and arduous, but eventually solutions will be found, for a World Science Information Center is a necessity.

The very concept staggers the imagination. Such an Information Center will provide a means for helping to equalize the differences between the poor and the wealthy nations of the world. A data bank of scientific and technical information will enable a farmer in one country to learn how to combat a plague of destructive insects by using a technique developed in another country. A food processor could learn how to preserve crops and thus alleviate a famine in a distant land. Technicians, engineers, and scientists will be able to pool their knowledge internationally and all will benefit. Educational opportunities will be more readily available and resources will be spread more equitably. Students, wherever they are, will be able to learn from the best of instructors and have the most up-to-date material. Doctors all over the world would be better trained and more knowledgeable, and health care will improve. Perhaps people will even learn to trust and respect one another and there will be no need for war.

As you see, my imagination has been stimulated and I am blinded by the grandeur of the vision. In truth, I recognize that information itself will not change the world; although the utilization of information may. A World Science Information Center is a means of making knowledge available so that it can be utilized. As such it has a small but necessary role to play in improving our social and economic status. Furthermore it is a practical first step, and one that we can and should take. Here is an application of computing power that is worthy of data processing technology. Here is a project that gets us out of the area of techniques and into the arena of real social problems. Let us allow ourselves to be stimulated by this concept and let us be challenged to solve the technical problems so that a World Science Information Center can become a reality.

REFERENCES

1 *Summary of activities toward interagency coordination*
US Congress Senate Committee on Government Operations Report No 369 89th Congress 1st Session Washington DC GPO June 1965 The Humphrey Report

2 L F CARTER et al
National document-handling systems for science and technology
John Wiley & Sons New York 1967

3 *Status report on scientific and technical information*
Federal Council for Science and Technology June 18 1963

Real time computer communications and the public interest*

by MICHAEL M. GOLD

Carnegie-Mellon University
Pittsburgh, Pennsylvania

and

LEE L. SELWYN

Boston University
Boston, Massachusetts

INTRODUCTION

The development of remote computer access has resulted in among other things, the employment, by the data processing industry, of an important new raw material—telecommunication services. In the United States the principal source of this raw material is the communications common carrier industry. Each member is a "regulated common carrier"; a firm granted certain monopoly privileges and which, in return, must subject itself to review and regulation by the Federal Communications Commission (or appropriate state regulatory bodies) acting to safeguard the public interest. All of this might be very interesting but, until now, has been of relatively little concern to members of the data processing industry. However, as the computer becomes more dependent upon communications services, so too does the data processing industry become more subject to the structure and practices of an industry that has now become a principal supplier of one of its key inputs of raw material.

In our present discussion, we are concerned with the nature and, hence, the adequacy of the services that are made available and furnished by these common carriers to the computer industry. Our view is that their services are inadequate for the requirements of computer systems. They are designed to the same requirements as that employed for voice telephony; because they do not reflect the unique characteristics and requirements of data communications, they are highly inefficient services for such purposes. As a result, they cost too much. This high cost may well result in the restriction in the most widespread use of remote access computing systems, since the communications costs of the services now available and their associated tariffs are often prohibitive.

The technology required for significant improvement in the production of data transmission services has been developed. Its implementation would significantly reduce the cost of such services. To the extent that implementation of this technology is delayed, the most widespread use of remote-access computer systems will also be retarded.

Government regulatory agencies have the responsibility to encourage the use of the most appropriate technology for the production of communications services by the regulated firms. At the present time, the Federal Communications Commission is investigating the extent and direction of their responsibility in this area. It is our hope that, as a result of this inquiry, regulatory policies cognizant of the unique requirements of the data processing industry will be promulgated. As the custodian of the public interest, it is the responsibility of government regulation and public policy to foster the development of such services that will not impede the growth and development

*This work was supported in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-146) and is maintained by the Air Force Office of Scientific Research. Partial support was also provided by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number Nonr-4102(01). Reproduction in whole or in part is permitted for any purpose of the United States Government.

of data processing but will permit the latter to develop along its own economic course.

The need for a more data-oriented communications service exists. The technology for providing such a service, with its resulting reduction in unit cost, is available. Our purpose is to suggest that this technology must be implemented so as to satisfy the present and future demands already evident. Several techniques for providing the indicated services have been suggested. (See, for example, Davies, 1967 or Mills, 1965.) We propose, as an example, one such technique and present the potential benefits of its implementation.

The development of remote real-time computer access

The concept of time-sharing a real-time computer is not new—as such it was proposed as early as 1959 and had been implemented at various places on a limited basis as early as 1963. Although there are many advantages to the user of such systems, the major ones seem to stem from the ability of a computer user to work at his most advantageous speed, communicating with the remote computer at a time and place convenient to the user. From the standpoint of efficient usage of the computer system, such usage also has its advantages: the larger computer is not required to spend valuable time waiting for a human to make decisions, communicate a line of input, or even wait for a mechanical device to transmit signals. While the user, for example, is typing a line of input to the system, thinking about what he is going to do next, or seeing a line of output being typed, the computer can be servicing many other users.

Of potentially greater advantage, the availability of remote access computer systems reduces the user's dependence on having close physical access to the actual computer system and instead, allows the user to gain access to the computer system through communications channels—independent of his location. This ability to use a computer from afar has led to the concept and development of the "remote service bureau," or "computer utility," to provide the user with computing power. Although many different systems have been developed under various names, they all have one thing in common—many remote users are connected simultaneously via communication channels to a central computer.

The problems with remote real-time computer access

At the present time, relatively little "remote use" of these systems from distances of more than a few miles has occurred. This is a result of the high cost of using a communications network—designed for voice—for the transmission of data. The characteristics of data transmission are sufficiently unique and the potential demand sufficiently great, that a system designed for voice communication cannot be expected to provide the required services, efficiently and economically, over the long run.

In order to make effective use of most real-time computer systems, the user's remote console must be connected to the computer facility for the entire time he is working. This is expensive. For other than "local" service, where the cost is nominal, the cost of remote usage creates an effective barrier against geographically extended use of computers.

To provide a more concrete example of these costs, we have presented the approximate charges per hour for non-permanent telephcne and teletype grade lines between Boston and several cities in Table I.

TABLE I—Typical hourly costs of dial-up communication channels

Between Boston and:	Telephone Message Toll	Teletype TWX
New York	$12	$21
Washington, D.C.	$18	$24
Chicago	$24	$30
Dallas	$27	$33
Los Angeles	$30	$36

These fixed hourly charges are incurred as a result of being connected to the interactive computer system by "long-distance" telephone or teletype lines, irrespective of the actual amount of work done or the number of messages transmitted between the user and the computer. During an average hour of console hookup, a user might be billed for $10 to $20 of actual computer (main frame) usage. A local user (within the local telephone service area) of a Boston-based system would be billed $0.60 per hour for communication channel usage; a user in Washington, D. C., $18,

and one in Los Angeles, $30. The tariff structure under which this communication service falls is based upon the assumption that most of the time the communication channel is actively connected, it will be used to transmit information and, thus, one full-time circuit would be required for each connection. Although this assumption of relatively full utilization of the communication channel is satisfactory when voice communication is being transmitted, it is not when these channels are employed for remote, interactive use of a multi-access computer system. With such systems, only a small percentage of the facilities of the communication channels are utilized—the primary characteristic of this type of usage is the transmission of a number of small data-content messages with relatively large delays between successive messages. The communication channels used for data transmission have the capacity to transmit over 12,000 characters per minute. Currently used remote console devices have a maximum data rate of 900 characters per minute or less than 7.5% of the channel capacity. Measurement of users interacting with existing time-sharing systems indicate maximum use of less than 10% of the capacity of the console—and, thus, less than 1% of the capacity of the communications channel (Gold, 1968).

The high "overhead" charge to communicate with the system will have a detrimental effect upon the remote computer user. In addition to the rather obvious one of having to pay the "phone bill," the user may be pressured into working fast, thinking fast, editing fast, and making mistakes fast. This, of course, tends to negate a basic feature of interactive real-time systems—the ability of the user to work at his own pace, unhurried by external pressures. Research comparing on-line computing and the more traditional batch-processing computing indicates this real-time or time-sharing access can result in higher performance (a higher quality product) if the computer is available to the user upon demand and the output is generated within a reasonably short amount of time (Gold, 1967). It would appear, then, that the existing communications service offerings and the resulting tariff rates—which discourage remote users from having a computer available upon demand—will be a real and serious barrier to the effective development of remote use of such systems. In view of the unique communication requirements of such systems, a reassessment of the means of effecting this service appears warranted.

Requirements for remote real-time computer access

What appears to be needed is a communication system which will allow the user to communicate with the remote computer as the need arises—with a minimum of effort and delay to the user and at a price not significantly greater than that incurred by a user located at the remote computer site. Under the existing service and rate structure, the user of a remote computer utility is faced with several alternative methods of using the communication channels. First, he may maintain the connection between his console and the computer for the duration of his computing session, even if, for the majority of the time, the communications facility is lying idle. This, of course, is most expensive but provides minimum bother to the user. Or second, if he wants to economize, he (or the computer) may break and subsequently re-establish the connection, by dialing, for only as long as it will actually be needed. This second alternative has immediate economic advantages to the user, but would certainly prove extremely cumbersome in practice or, for real-time operations where immediate access is normally a requirement, would prove impossible. In either case, such a scheme would negate many important aspects of real-time systems—especially the ability of a user to communicate with a remote real-time computer at a time and place convenient to him.

It is clear that any attempt to minimize the relatively high cost of the communication channels for remote computer access through somewhat devious means would result in a substantial downgrading of the services provided by a time-sharing system —as compared with those received by a user at the facility. Simply a cheaper system for communicating with the computer utility is not sufficient—an adequate system must provide service similar to that available to a user located at the computer facility, and must do so at a cost which: (1) does not act as a barrier to remote use of the computer system; and (2) is somehow proportional to the actual usage of the communication channel and not simply to the time the user is actively connected to these channels. This objective of eliminating the effective barrier to remote computer usage may be achieved by adapting existing communication plant to the specific requirements of such systems. The result will necessarily be:

(1) a change in the basis of communication charges—from elapsed time to the amount of information transmitted; and (2) a substantial reduction in the cost of the communication service. This would attempt to change the cost structure by eliminating the full time devotion of one circuit to each user—a major factor in present remote computer usage costs.

Proposed data-transmission service

Computer time-sharing is founded, in part, on the idea that of the many users actively connected to the computer at any time, the computer pays attention only to those requesting service and does not waste its time by trying to provide service to the others. Thus, the system as a whole can operate efficiently—charging the users only for the time the computer works for them. It should be possible to extend this concept to the communications facilities that are associated with such remote access computer system—that is, provide a pool of communication resources for a number of active users of the system such that, at any instant of time, when one user is not using the facility it is available to another user for transmission of his data. As an example, a computer utility in, say, Boston might have fifty active users in the New York area. Rather than establish fifty independent connections between New York and Boston as must be done presently, it should be possible to connect each New York user to some collection point in the New York area; either through a permanent connection or by dialing, and then have some smaller number of pooled circuits between this collection point and the computer in Boston. Equipment at the collection point would take care of assembling complete messages from the user (and the computer) and assigning an open line to transmit the entire message at full line transmission speed. Once the message was sent, the line could be made available for the next message to be sent—in much the same way as the main control unit at the central computer assigns the various components of the computer system among the various users. Although this would reduce the "rent" on the communication circuit to a small percentage of its previous cost and, thus, provide more efficient usage, there would be a cost for the switching hardware which could be substantial if shared only by a small number of users.

In the more general case, however, it would perhaps be unwise for each computer utility to establish such collection points. The economies of a larger scale operation would be significant, while the cost of any one collection point operated independently might well exceed the savings derived from it—such a system would only function efficiently when a sufficiently large group of users made it economically practical.

However, this problem can easily be eliminated by pooling the entire system of remote computer users and providing one communication system which they could all share—possibly within the structure of present or proposed communication systems. Such a system could operate more efficiently because the requirements of one large group of users could more accurately be predicted than those of several smaller individual groups. In addition, the entire system could afford to maintain (in the short-run) a portion of the network which does not have a sufficiently large number of users—something not possible with a number of independent collection points.

There appears to be ample technical and economic justification for such a system. First, it is technologically possible for a common carrier to operate public "collection points" of the type mentioned above—in fact, this has been done for overseas telephone calls. In the case of overseas telephone conversations, the pool of two-way circuits is shared among a larger number of individual calls. The callers have access to the pool only when they are actually talking, and in only the direction that the talk is traveling. Second, as the demand for this type of service increases, the number of users of different computer utilities. Thus, Bossible to share communication facilities among users of different computer utilities. Thus, Boston users of a New York computer utility might share a pool of circuits with New York users of, say, three different Boston computer systems.

To provide a concrete example of the savings which could result from implementation of the proposed system, we have presented hourly communication channel costs under both the present system and the proposed system based on permanent service between Boston and several cities in Table II. The calculations include the costs involved in sharing the lines and transmitting data. Local termination charges (as much as $0.60 per hour for a local dialed connection) are not included.

TABLE II—Comparison of hourly charges under present and proposed communication systems

Between Boston and:	Dial-up Cost	Proposed System's Cost 50 User*	100 User*
New York	$12	$0.44	$0.42
Washington, D.C.	18	0.47	0.43
Chicago	24	0.52	0.46
Dallas	27	0.59	0.49
Los Angeles	30	0.70	0.54

*These figures are based on the use of the system 40 hours a week. Substantial reductions in these costs could be effected through the use of this equipment more than 40 hours per week. See the Appendix for the derivation of these costs.

CONCLUSION

It is essential to the public interest that both the common carriers, who provide communications services, and the public agencies who regulate them recognize the potential importance to the public of modern computer technology and offer communications services that will serve and encourage its development. Rates for such service should not be out of line with the value it may add to the total system; they should be based not upon the elapsed time or duration of the connection but instead upon the amount of data transmitted by the carrier. Changes from the present technique of full-time use of a circuit to a method enabling a large number of active computer users to share a small number of these circuits can meet this goal. Although such techniques might be implemented on a limited scale by small groups of computer users and computer service suppliers, the full development of a widespread network of such data-oriented communications facilities is rightfully the responsibility of the communications common carrier. It is essential that the communications companies and the regulatory agencies take appropriate action now to bring about the economically viable shared-facility data communications services.

BIBLIOGRAPHY

1 American Telephone and Telegraph Company Tariffs
Federal Communication Commission Nos 132 130

2 D W DAVIES et al
A digital communications network for computing giving rapid response at remote terminals
Symposium on Operating System Principles ACM October 1–4 1967 unpublished

3 Digital Equipment Corporation
Price list for PDP–8
Maynard Mass 1967

4 M M GOLD
A methodology for evaluating time-shared computer usage
Carnegie Institute of Technology 1967

5 M M GOLD
Communication circuit utilization during experimental time-sharing operation
Unpublished report 1968

6 R S GRISETTI
The synthesis of communications and computers
In F Gruenberger (ed) Computers and Communications Toward a Computer Utility
Prentice-Hall Englewood Cliffs 1968 pp 209–219

7 R G MILLS
Communications implications of the Project MAC multiple-access computer system
IEEE Convention Record 1965

8 P E MUENCH
Common carrier approach to digital data transmission: terminals transmission equipment and future plans for the computer utility
in F. Gruenberger (ed) Computers and Communications—Toward a Computer Utility
Prentice-Hall Englewood Cliffs 1968 pp 209–219

9 L L SELWYN
The Information Utility
Industrial Management Review
Vol 7 No 2 1966 pp 17–26

10 D DIAMOND L L SELWYN
Considerations for computer utility pricing policies
Proceeding of 23rd National ACM Conference 1968 pp. 189-200 Brandon Systems Press Princeton, New Jersey

APPENDIX

Typical Hardware Costs

PDP–8 (one end)	
Extra 4K memory	
CAB–8B	
681 Interface	
685 Multiplexor	
W750 Modules (64)	
DF32 Disc file and controller	
CAB–B–S	
637B Dataphone interface	$ 49,350
PDP–8 (second end)	49,350
PDP–8 (2 - one each for redundancy)	98,700
Total hardware cost for 64 lines	$197,400
Equivalent monthly rental	$ 4,400
Equivalent hourly rental (40) hrs./wk)	25
Equivalent hourly rental per full-time line	0.39

Full duplex leased line monthly cost with channel terminators

	Monthly Cost	Hourly Cost (40 hrs./wk.)	Hourly Line Cost per User (50 Users)
Boston to New York	$ 445	$ 2.53	$0.05
Boston to Washington, D.C.	721	4.10	0.08
Boston to Chicago	1144	6.55	0.13
" " Dallas	1750	10.00	0.20
" " Los Angeles	2656	15.20	0.31

If we further assume only 50% system utilization, we should double the above costs, e.g., for Boston to Los Angeles, the hourly hardware cost would be $0.39 and the line charges (with 50 users), $0.31—for a total of $0.70. With only 50% utilization the cost would be $1.40.

With more than one shift operation, the costs would be reduced, e.g., Boston to Los Angeles for 80 hours a week (with 50% utilization) would cost $0.70 per hour.

It should be emphasized that these figures are used for demonstration purposes only.

The two extra computers are for redundancy only; more economical methods of providing this redundancy would probably be available.

These costs are based on the research investigating the amount of data transmited by users of time-sharing systems (Gold, 1968). In all cases studied, the circuit utilization was significantly less than 1% of the advertised half-duplex capacity. It would thus appear that up to 100 simultaneous users could be handled by such a circuit; the cost data, however, is for full-duplex circuits. Use of inexpensive scope displays indicates data transmission requirements of these devices may be double those of existing consoles—or a maximum of 50 users per circuit.

Toward education in real time

by PERRY E. ROSOVE

System Development Corporation
Santa Monica, California

INTRODUCTION

The Bureau of Research of the U.S. Office of Education, in January 1967, expressed its concern with the potential impact of social and technological changes on the formulation of educational policy. This concern was manifested in the creation of five, nine-months pilot centers for educational policy research with funds made available by the Bureau of Research under Title IV of the Elementary and Secondary Education Act (PL 89-10). The primary task of the centers, according to the Bureau, was "to analyze future educational needs and resources and, in light of these analyses, provide educational policy makers at all levels with relevant information and techniques for decision making."[1]

The Request for Proposal (RFP) distributed by the Bureau of Research reflected a concern with what may be referred to here as a *temporal lag* between the future needs of education and current educational policies, objectives, and programs. The RFP called attention to alleged "revolutions" in population growth, civil rights, urbanization, automation, etc., and asked the centers to address themselves to the implications of these revolutions for current and future educational policies. In the wording of the RFP, the U.S. Office of Education explicitly recognized that the response of the educational community to large-scale social and technological changes was excessively slow and that it was necessary, therefore, for educators and other responsible authorities to attempt to "invent alternative futures."[2]

The problem of anticipating the future is especially difficult for educators. Policy makers in noneducational areas may be required to plan for a future that extends five years, or perhaps ten years, ahead. But in education, decisions must be made today for a future that is a generation away. A child starting school this year (1968) will be in the prime of his adult life in twenty years. Will the education he will receive during this period of time prepare him adequately for a lifetime of work and self-fulfillment? How relevant is today's education for life in the 1980's and beyond? Do today's educators carry out their tasks with that future in mind or is it the past and the present which provide them with guidelines?

These, then, were the general issues which the five pilot centers were requested to probe. To carry out the Bureau of Research's objectives, approximately $110,000 was awarded to each of the centers. They included the Stanford Research Institute, a consortium of Syracuse University and the General Learning Corporation, Western Behavioral Sciences Institute, the National Planning Association, and the System Development Corporation.

Each of the pilot centers was required by the Bureau of Research to design "a model operational center." It was the intention of the Bureau, based upon the performances of the pilot centers, to select two or more of them to become operational centers with long-term contracts and with expanded staffs. Selection of the pilot centers would depend heavily upon their designs for operational centers. In addition to this requirement which was levied on each pilot center, the System Development Corporation's pilot center, named the Educational Policy Research and Support Center (EPRSC), elected to focus its attention on two areas:* (1) the applicability of available forecasting methods to the study of the future

*The EPRSC subsequently developed studies in other areas but they are not discussed in this paper.

of education and its social and technological environment; and (2) the possible roles of educators in the late 1980's. This paper describes some of the results of the work done in these areas, specifically with respect to the concept "real time."

The first objective was selected for fairly obvious reasons. The field of forecasting, if it may be referred to as a "field," is relatively new. It seemed necessary, as a preliminary step, to investigate what forecasting methods were available and to determine which, if any, might be appropriate to employ in a study of the future of education.

The second objective, the possible roles of educators in the late 1980's, was selected since such roles are central both to the educational process and to education as an institution. Educational policy making at all levels is concerned with them. If the roles of educators were to change drastically during the next twenty years, as many experts predict, this change would require important policy decisions affecting broad areas of education.

Contextual mapping

Early in the life of the SDC pilot center, the writer conducted a survey of forecasting methods to determine which method or methods might be relevant to the substantive issues referred to above. As a result of this survey, it was decided to concentrate upon and experiment with an extrapolative method known as "contextual mapping," which may be briefly defined as a graphic display of the logical and causal depedencies of functionally related phenomena.[3] The selection of this method by the EPRSC staff was based on the philosophy that the formulation of educational policy would be facilitated by the development of a forecasting method which would lend itself to public scrutiny and debate.

Over a period of approximately three months, a contextual map was constructed. The map was composed of a two-dimensional matrix containing 36 cells (see Figure 1). The vertical axis of the

Basic, Long-term Trends	Major Sub-Trends	Social and Technical Implications	Implications for Education	Educational Functions	Possible Future Roles	Major Issues
(Cultural Sector) Increasingly sensate, empirical, humanistic, pragmatic, utilitarian culture.	1A	2A	3A	4A	5A	6A
(Socio-cultural Sector) Increasingly sensate, empirical, humanistic, pragmatic, utilitarian culture.	1B	2B	3B	4B	5B	6B
(Economic Sector-National) Transitional, mass-consumption society characterized by higher GNP and personal incomes, affluence (among better educated).	1C	2C	3C	4C	5C	6C
(Economic Sector-International) World-wide industrialization and modernization.	1D	2D	3D	4D	5D	6D
(Science & Technology Sector) (I Organization) Institutionalization of change, especially through research, development, innovation & organized diffusion.	1E	2E	3E	4E	5E	6E
(Science & Technology Sector) (II Information) Accumulation of scientific and technological knowledge.	1F	2F	3F	4F	5F	6F

FIGURE 1—An EPRSC contextual map (roles)

matrix was divided into rows, each of which represented "basic, long-term trends" of Western civilization. These trends were adapted from the work of the Hudson Institute.[4] To facilitate the mapping process, the basic, long-term trends were grouped, as indicated by the double lines in Figure 1, into three major "sectors" and a total of six subsectors were identified as rows of the matrix: cultural, socio-cultural, economic-national, economic-international, science and technology-organization, and science and technology-information. Since the basic, long-term trends were too broadly conceived to lend themselves readily to the derivation of logical and causal dependencies, each of the six sectors was further divided into three functionally distinctive areas providing a total of 18 rows in the matrix, each row identified as shown in Table 1.

TABLE 1—Eighteen rows of the contextual map matrix

Basic, Long-Term Trends	Rows of Contextual Map
Cultural Sector	1. Pragmatism 2. Values in Art, Religion, Philosophy 3. Permissiveness re: Mores
Socio-Cultural Sector	4. Rising Aspirations of Disadvantaged 5. Urbanization 6. Changing Aspirations of Middle Class
Economic Sector-National	7. Occupational Changes 8. Controlled Economy 9. Social Growth
Economic Sector-International	10. Industrialization-World wide 11. Internationalism 12. Transportation and Communication
Science and Technology Sector-Organization	13. Technological Replacement 14. Tempo of Change 15. Technological Complexity
Science and Technology Sector-Information	16. Information as Wealth 17. Growth of Knowledge 18. Information Technology

The horizontal axis of the map matrix was divided into six columns which were designed to show the logical and causal sequences of events, trends, conditions, and processes associated with each of the 18 functionally distinctive areas or rows. As shown in Figure 1, the six columns included:

- Major Subtrends
- Social and Technical Implications
- Implications for Education
- Educational Functions
- Possible Future Roles (for educators)
- Major Issues (in education)

The selection of these column headings reflected both a logical sequence and the substantive concerns of the SDC pilot center, e.g., the implications of social and technological trends for education and the roles of educators.

Trends, events, conditions, and processes were represented in each cell of the matrix by "entries" (see Figure 2). Each entry was enclosed in a rectangular box and numbered to facilitate identification. Entries which were regarded as especially important were set off from other entries by heavier lines. Lines joining the entries indicated hypothesized logical and causal sequences. Dependency relationships flow from left to right. The content of initial entries for the first column of the map in Figure 1 was obtained from an extensive review of the published literature dealing with social and technological trends in addition to the work of the Hudson Institute referred to above.

It should be emphasized that each entry was regarded as an hypothesis which was and is subject to debate and revision as relevant data become available. While the construction of the map in the EPRSC was the work of one individual, it was anticipated that it would be the responsibility of an interdisciplinary team to review and develop it further in the operational center.

It is not necessary for our purposes here to reproduce the contents of the entire map. A document describing the contextual mapping method and containing all the map entries may be obtained from SDC or the writer upon request.[5] However, to illustrate the content of entries and how entry relationships are graphically portrayed, Figure 2 reproduces a segment of the map, cell 1B. The extrapolation of current social and technological trends into the future resulted in the conclusion that existing educational organization and

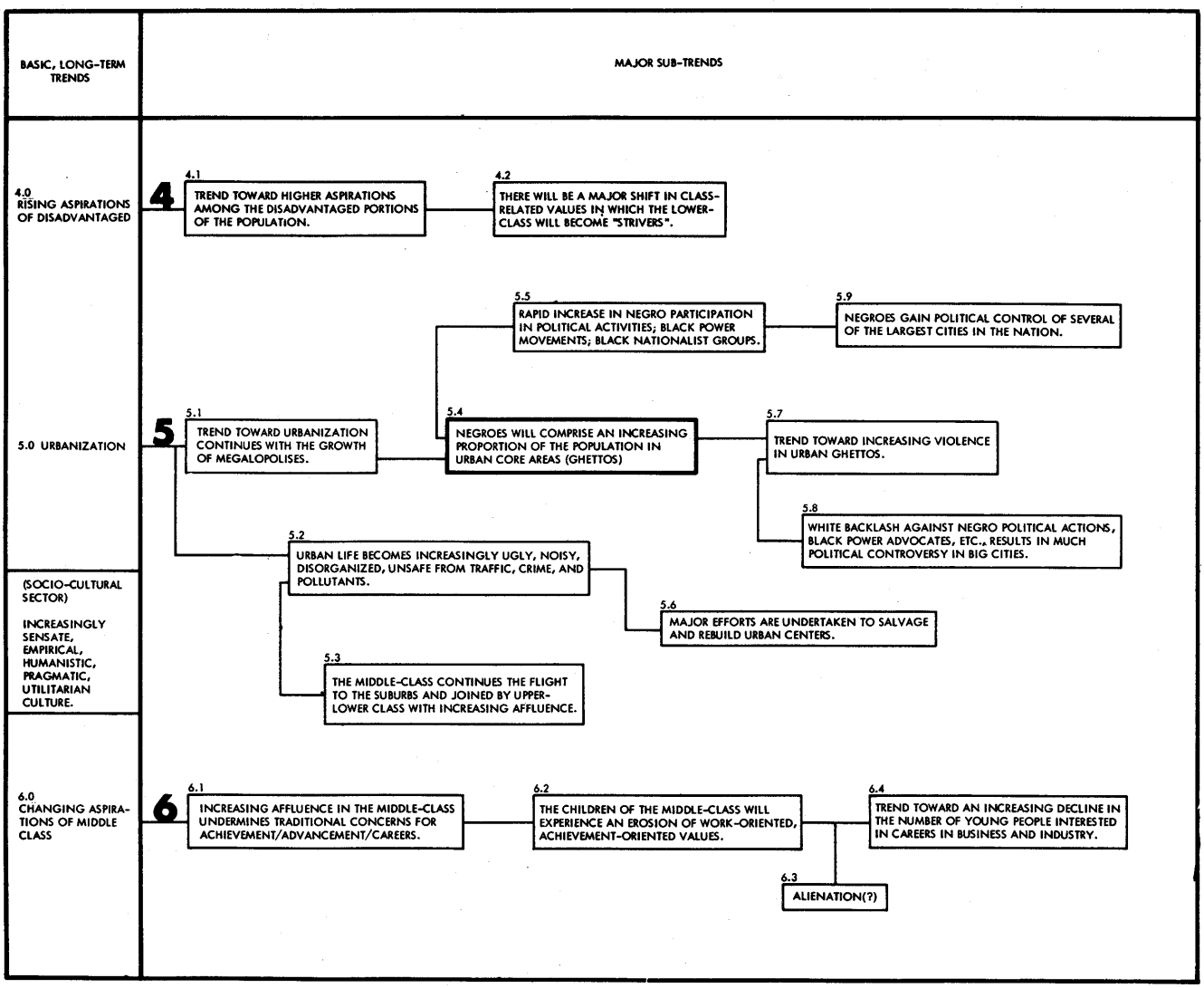


FIGURE 2—Segment of the contextual map: Cell 1B

the generic concept of the role of the "teacher" would be incompatible with future conditions and requirements.

Our review of the published literature dealing with educational reforms, which was conducted over a period of three to four months following the construction of the contextual map, revealed that while current reform efforts are highly significant and major undertakings, they can only be characterized as piecemeal and inadequate efforts. They represent a form of tinkering with existing educational roles, curricula, and organization which leaves the contemporary educational system essentially unchanged.

The remedy proposed most frequently by educators to accelerate adaptive changes in education is the improvement of the educators' education. But this does not appear to be an adequate response to the problem. The extrapolation of current trends, particularly in the six rows of the map comprising the science and technology sector (see Table 1), highlighted the exponential growth of knowledge, the obsolescence of existing knowledge, and the proliferation of new specialties. How can the pre-service training of educators be altered to conform with such changes on a *continuing basis?*

The fact is that schools at all levels of the educational system below the university level, including the schools that prepare educators, are autonomous institutions and relatively isolated from developments, new knowledge, and changes

occurring in other areas of society.[6] Although research is a major function of universities, it is not conducted at lower levels of the system. The new Research and Development Centers established by the U.S. Office of Education do not alter this basic pattern in which the generation of new knowledge, even new knowledge concerned with learning processes, is the responsibility, not of teachers, but of other professionals such as experimental psychologists. Thus there remains the fundamental question of how new knowledge, either in subject fields or in the learning process, is injected into the elementary and secondary schools. As academic critics of contemporary education have too often noted, what is taught in the schools may be as much as thirty years behind new developments;[7] and the isolation of the teachers from sources of new knowledge too frequently results in the dissemination of either "trivia" or "untruth."[8]

Three basic organizational concepts for the education of the future evolved out of our construction of the contextual map and the review of the literature on educational reform. The sequence of logical steps and tasks which were followed to derive the organizational concepts and a new generic role concept of the teacher ("learning facilitator") is shown in Figure 3. In the balance of this paper, we shall limit our remarks to the conception of the learning environment as a real time facility.

Education in real time

In the language of the information sciences,* education currently occurs in a non-real time mode but it may increasingly have both a need and a capability for operating in real time. The

*The term "information sciences" is used here to apply, in the broadest sense, to those fields concerned with information—cybernetics, computer science, data processing, system analysis, information storage and retrieval, etc.

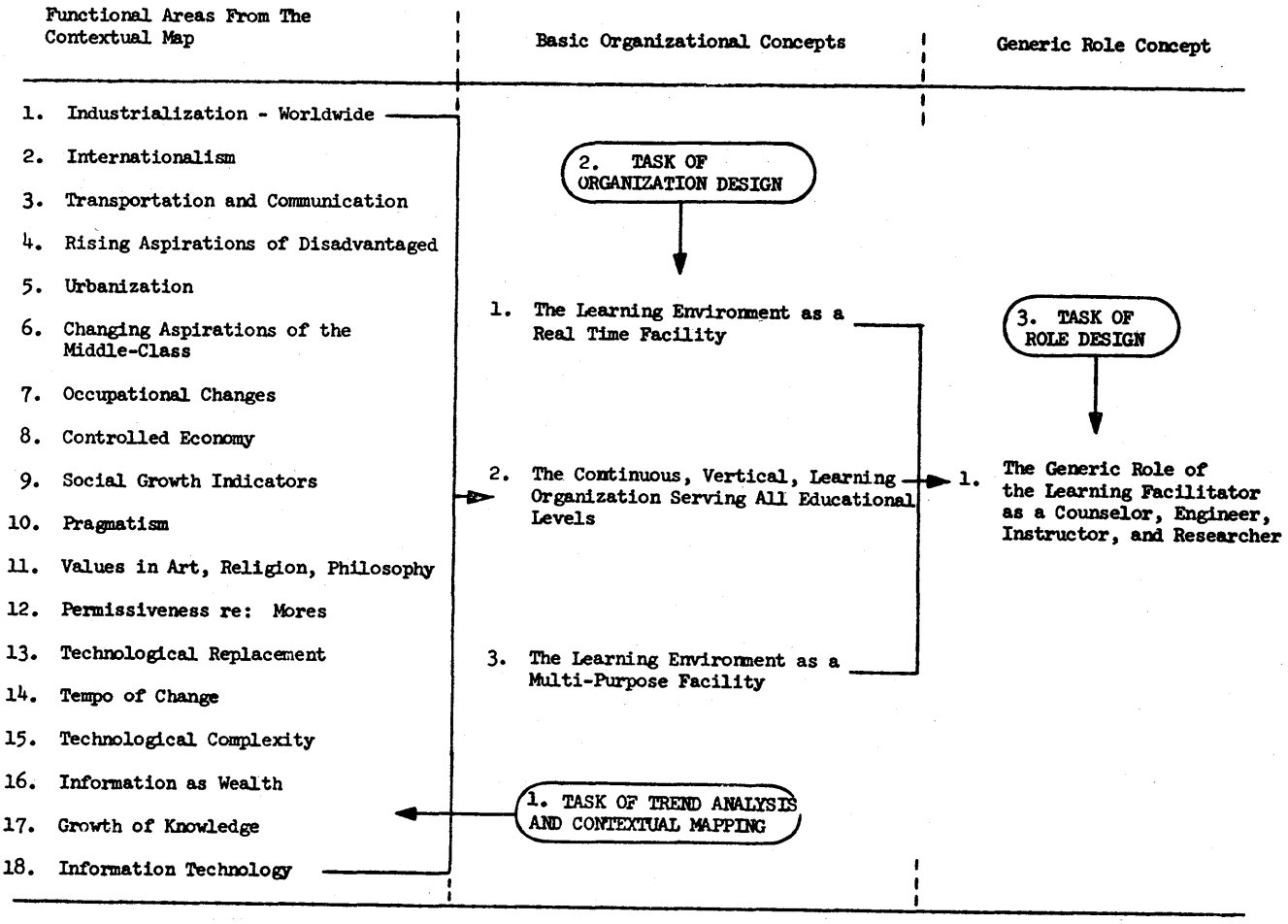


FIGURE 3—Sequence of logical steps and tasks to derive the generic role concept

changes displayed on the contextual map—changes in society, in the economy, and in science and technology—imply a reduction in the amount of time available for institutional adaptation. Education, from the point of view of its timeliness, can operate effectively in isolation from other institutions when change is relatively slow; it cannot be equally effective in isolation when change is relatively rapid, as is now the case. Hence we are led inexorably to the idea that within the next twenty years learning environments ought to be designed more like real-time information processing systems.

In the United States since the turn of the century, with the advent of the free public school system, formal schooling has become increasingly isolated from the "real" world both in time and space. In primitive and peasant societies, as well as in apprenticeship programs within our own society, learning is closely geared to the world of reality. The child in a nonliterate society may be taught his cultural traditions by his kinsmen as he participates with them in hunting or food-gathering expeditions. In peasant societies skilled craftsmen, artists, and priests transmit their stores of knowledge to their designated successors most typically as an on-the-job experience. In agricultural communities, children acquire the cultural heritage and necessary technical skills as they labor beside their elders in the fields or rice paddies. In all of these situations, one learns by doing in the real life environment. Formal schooling, if it exists at all, represents only a fraction of the learner's time. But, increasingly, as our institutions have evolved, as specialization has increased, and as the amount of knowledge required to perform occupational tasks has accumulated, the amount of time required of the individual to devote to formal schooling in institutions physically separated from the nonschool world has increased. Since the end of World War I, welfare legislation has required the child to spend more time in formal school environments while apprenticeship opportunities have declined in number. Indeed, if contemporary trends continue, it may soon be mandatory for all citizens to remain in school through the associate of arts or the baccalaureate degree level. Thus, unless there is a reversal of this trend, the "school" as an institution, very much like the penitentiary, will prolong and exacerbate the isolation of the young from the real world.

In the traditional notion of the "school" as the basic learning environment, its isolation from the world of reality or "real life" has taken three forms: (a) education is separated from the world of work; (b) learning is separated from research on the learning process; and (c) education as a discipline is separated from the other academic disciplines. We propose here a concept of the learning environment in which these separations are drastically reduced.

A real time approach to education and work

Formal schooling as a prelude to entering specialized occupations and professions is based upon an assumption that the fund of knowledge needed by the student in preparation for life is relatively static. It is also based on the assumption that the number and variety of occupations and professions are unchanging and that each individual will be prepared during his formal school years to enter and remain in one occupation or profession throughout his lifetime. Indeed, in the past if an individual changed his specialty too often it was not uncommon to interpret this as a sign of personal instability or, worse still, as opportunism. The "jack-of-all-trades" has never been regarded as a social success. "Finding one's self" for a male was synonymous with identification with one's specialized field.

These assumptions contributed to the conception of formal education as something separate in time and space from the world of work. One went to school for a fixed period of time. One then graduated and went to work, presumably, until retirement or death, in the field of one's choice. The readers of this paper, perhaps more than other audiences, can appreciate how archaic such notions are. Who among us is in the occupation or profession for which he prepared himself so assiduously in college or graduate school? Today it has become almost a cliché to comment, since Margaret Mead first made the remark, that young people will have four or five different and successive jobs or careers in their lifetimes. Thus the references to education becoming a "continuous, lifelong" process have become commonplace in the educational literature.

If one accepts the idea of continuous, lifelong education seriously, then we should no longer retain the notion that the lives of future students

should be divided into two distinct phases—an educational phase and a working phase—dichotomized by an ancient ritual called "graduation." If we take seriously the idea that each individual will have several different jobs, occupations, or careers in a lifetime, then education has no termination; there is, rather, interspersed periods of education, work, reeducation, work, ad infinitum. From this perspective the concept of the "school" as *the* place where education occurs is obsolete.

Professional educators are attempting to cope with the impact of an accelerating rate of change in science and technology on educational institutions or, rather, the lack of impact. However, it does not appear to this writer that their recommendations are adequate for the future conditions which they themselves anticipate. Grant Venn,[9] for example, maintains, in his study of post-secondary vocational and technical education, that the problems of an excessive student dropout rate and the unemployment and underemployment of dropouts can be attributed to the liberal, academic orientation of the educational community which is of no interest to 80 percent of the student population. The time has come, he asserts, for schools and colleges to recognize that the new occupations spawned by science and technology "must be taught where they are best able to be learned." Venn believes existing educational institutions are the best places to achieve this, although, as he points out, they have failed to perform this task successfully in the past. He hopes that they can be reformed. But are they failing, as Venn maintains, merely because of a liberal, academic orientation, or is it not also because today's educational system operates in a temporal mode which is out of gear with the rate of change in science and technology? It is our contention here that the institutional isolation of the typical school, at elementary, secondary, and college levels, makes it impossible for it, with contemporary methods of pedagogy, to cope with the rapidity of change characteristic of what Venn calls the "new technology."

Whether the objective of education is "liberal," "academic," "humanistic," etc., designed to prepare the student for citizenship and self-fulfillment, or whether its objective is to provide the trained technical and professional manpower for a highly industrialized society, the isolation of the formal school system from the real world remains a fundamental problem.

In addition, the individual while on the job should, in the future, remain closely tied to educational processes. He may have as much need for information and new knowledge, if he is a technician, scientist, or professional, as the formal student in today's school environment. The acceleration of production of new knowledge, the obsolescence of existing knowledge, and the proliferation of new specialties which we can expect to follow in the wake of scientific developments, suggest that the learning process must become a real time process in which it will become increasingly difficult to differentiate between work and education. This is the essence of a "learning society." To the extent that knowledge generation becomes a basic process of our society and knowledge the major product, developments which have already been predicted by Machlup,[10] then the knowledge bearers in society must become lifelong learners and education must become a real time, continuous process.

A real time approach to research on learning processes

Traditionally the function of teaching and the task of conducting research on learning processes and educational technology have been separated. The isolation of the elementary and secondary teacher from the sources of new knowledge in these areas, while on the job, has been accepted as inevitable. But, less excusably, the education of the educators in teachers' colleges has also been isolated from the research on learning conducted by university scholars and other types of professional researchers.

A real time approach to research on learning processes becomes technically feasible as soon as one introduces the computer into that process and begins to conceive the learning environment as a man-machine system. In man-machine system theory, the system need not be regarded as a static entity but as evolutionary in character. The system may be perceived in the spirit of experimental inquiry. Every employment of computer-assisted instruction may be regarded as an experiment, the object of which is to evolve a more effective man-machine learning process. The components of the experimental situation include such items as: the computer programs, the displays, the data formats, the mechanisms provided for student-computer interaction (input and output devices), the sequencing and branching of subject matter, the measurement, recording, and evalua-

tion of the learner's performance, and his associated behavior. The results of the learner's performance can be recorded and stored in the computer's memory for later review and analysis, or the results may be presented immediately for review and study by the learning facilitator ("teacher"). Computer languages written in natural English may make it possible within twenty years to modify subject matter or its sequencing or branching on the spot or to make revisions immediately following a man-computer run. Sackman, for example, foresees a time in the near future when the computer facilities of man-machine educational systems would make possible a continuous series of experiments "directed to their own operations and aimed at the improvement of their own design and performance."[11] Such educational systems would undergo an evolutionary development as each experiment is carried out.

Computer-assisted instruction makes it possible for the learning facilitator to obtain almost immediately upon the completion of a lesson an analysis of that lesson to determine which parts of it caused difficulties and for which students. The learning facilitator's function is to modify the instructional programs where the difficulties are found and then evaluate the changes after another computer-run with the same students or another, comparable, group. With this sort of an experimental attitude toward the learning process, the approach to the design of educational systems is altered so that if the student fails to learn, it is considered to be the fault of the system rather than of the student, and the system is modified as many times as necessary until a pre-established criterion of learning success is achieved.[12]

This point of view has profound implications for educational policy. It places the "teacher" in the nerve center of the man-machine learning system. It makes possible a transformation in the traditional role of the teacher. While the machine takes over the function of dispensing facts or data, or the explanation of a concept or theory, the teacher (1) serves to facilitate the learning process by providing assistance, guidance, and counseling as needed by the student; (2) designs and conducts experiments on the learning process using the actual learning situation as his laboratory; and (3) contributes to the development of improved man-machine learning systems and procedures.

The transformation of the role of the teacher into that of a learning facilitator as a component of a man-machine learning system and as a system designer is fraught with obstacles. The education of today's teachers at the elementary and secondary levels does not contribute to respect for either machines or experimental inquiry,[13] particularly if the subject of the experiment is the learning process involving the teacher as one element. Insecure teachers do not appreciate or welcome having administrators or supervisors monitor their performances in the classroom. A revolutionary change in attitude by both teachers and administrators toward the learning process as a phenomenon to be investigated while it takes place—in real time—will be required. It is questionable if such an attitude can be taught in the colleges which produce our teachers. Who will provide the necessary instruction? More likely, an attitude which regards research into the learning process as a proper function of the teacher while on the job will occur as computers are gradually installed in school systems and as individual teachers learn, by trial and error, how to maximize the machine's capabilities.

A real time approach to subject matter transmission

The institutional isolation of professional educators from their colleagues in the academic disciplines and research organizations should be considerably reduced within the next twenty years. A number of trends suggest this. To reduce the widely recognized lag between the creation of new knowledge in the sciences and the introduction of this material into the curricula of the elementary and secondary schools, a large number of multidisciplinary committees, composed of professional educators and academic specialists, have been conducting extensive curriculum revisions in the major disciplines during the past decade.[14] Such committees have been active in biology, physics, chemistry, mathematics, English, social studies, anthropology, and other fields. The inherent weakness of these efforts is that, for the most part, they represent relatively infrequent modifications to the existing curricula. The participating academicians come and go, while the summer institute is the most common institutional device for preparing the professional educators to teach the new curricula. New knowledge, however, is continuously being generated in all fields. There are,

as yet, no procedures for up-dating the curricula in a given field as new concepts and data are generated.

In recognition of the need to reduce the time lag between the generation of knowledge and its dissemination to school children, some communities have instituted arrangements with corporations engaged in research to have their top engineers and scientists present lectures. In New York City, for example, nine corporations have agreed to pool their scientific manpower resources to provide lecturers each week on such subjects as nuclear physics, jet propulsion, and space technology as part of the City's high school program on contemporary science.[15] Unfortunately, few communities throughout the United States have this sort of skilled manpower available locally. From a national point of view, this is hardly a very satisfactory solution.

An associated dilemma is that elementary and secondary teachers are ill-prepared to present the basic language and concepts of a particular discipline. As an anthropologist who served on a curriculum revision committee for his field at the elementary school level states, "relatively few teachers have taken formal courses in anthropology."[16] They are prepared as generalists not as subject specialists. This accounts, unfortunately, for the frequency with which elementary teachers, and secondary school teachers as well, tend to teach facts without the conceptual framework which makes the facts meaningful. Yet it is increasingly recognized by professional educators, not only the academic scholars, that it is more important to convey concepts to the student rather than mere factual data out of context. Does the long-range solution to this problem lie in better education for educators or in bringing what the academic scholars and professional researchers have to say directly to the student in a form he can understand?

Educational technology already exists which can bring real events into the classroom as they occur. The significance of such a capability cannot be exaggerated. It is not too far fetched to hypothesize that today's student rebellions at the secondary and postsecondary levels might not have occurred if educators had attempted to bring the real world into the classroom as a proper subject for discussion, as a source of raw data, and for scientific analyses. As Carpenter has pointed out in Congressional hearings on educational technology before the Subcommittee on Economic Progress, "what is taught (in the classroom) is so often perceived by students to be remote from reality, abstract and sterile."[17]

Computer-assisted instruction and other types of innovative educational technology, such as closed-circuit television, video tape recordings, film cartridges, microfiche, etc., should make unnecessary during the next two decades the existing intellectual isolation of teachers in the lower educational levels from the universities and the research community. With appropriately designed information storage, dissemination, and display systems, the prototype or experimental forms of which already exist,[18] it should be possible for new knowledge in specialized fields, generated at the university level or in research laboratories, to be made available to learning facilitators at all educational institutions or in private homes as soon as it is in transmittable form. In terms of technical feasibility, there would be no need for elementary and secondary teachers to transmit trivia and untruth to their students if they and their students were linked via appropriate communication networks to the sources of new knowledge. The teaching of a subject such as biology need not suffer from a lag of thirty years in the elementary schools if *both* the learning facilitator and his students could view *together* the lectures of the most prominent researchers in biology via television, or could operate a computer-driven program dealing with the most recently developed biological findings and theory. Such programs, of course, would have to be written and maintained by the specialists in biology and other sciences. The qualifying expression here is "with appropriately designed information . . . systems." Our assumption is that within twenty years the "book" will be replaced, at least in part, by more efficient types of data transmitters.[19] For example, we assume that the research papers of investigators in biology will be available for worldwide transmission via electronic reproducing and transmitting equipment as soon as they are written. We also assume that important experiments in biology and other fields will be observed via TV networks *as they occur*.

One of the most important functions of the learning facilitator in the types of learning situations suggested above would be acting as an intermediary between the scholar or researcher and the students. His task would be to translate the

concepts and findings of the knowledge generators into a language appropriate to the educational level of his students, and to serve, where the learning situation permits, as a discussion leader, guiding and directing the real time interactions between the scholars or researchers and the students so that a genuine intellectual exchange can occur.

It should be noted that the installation of carrels in the rooms and dormitories of students at the college and university levels, whereby they can switch themselves at any time into instructional systems, by-passes the teacher (as a data dispenser) and makes his basic, contemporary function obsolete. If, in addition, two-way communication between academic scholars and students is provided while the learning process is taking place, the intelligent or gifted student may become a participant in the generation of new knowledge. With such capabilities, which are already within the technical state of the art, the temporal lag between knowledge generation and its dissemination would be greatly reduced. We can reasonably anticipate that the transmission of new subject matter will thus gradually evolve in the direction of real time.

CONCLUSION

Even a relatively unsophisticated form of conjecturing about the future of our society, particularly in the area of its knowledge generating power, leads to the conclusion that education in real time should replace, eventually, the contemporary mode of education. The prototype and experimental systems for real time education already exist, but their impact on our educational institutions has, as yet, been insignificant.

Before this impact can become significant, the designers and builders of real-time information processing systems will have to join with professional educators in a concerted effort to define educational objectives more precisely than heretofore, and then proceed to specify the system requirements to meet those objectives. This may prove to be a difficult task, but ultimately a more rewarding one, in terms of its social consequences, than the introduction of real time systems in the industrial, commercial, and military spheres.

ACKNOWLEDGMENT

The author would like to express his appreciation to his colleagues at the System Development Corporation who comprised the staff of the Educational Policy Research and Support Center: Dr. Marvin Adelson, Director; Dr. John F. O'Toole, Jr., Associate Director, Jack Jaffe, and Thorington B. Robertson. Their direct and indirect contributions to the research on which this paper is based is hereby acknowledged. I am also indebted to Dr. Harold Sackman whose independent work on real time systems contributed to the development of the ideas expressed in this paper.

REFERENCES

1 *Policy research center program*
Request for Proposal Bureau of Research US Office of Education Department of Health Education and Welfare Washington DC January 1967 p 1

2 Ibid

3 E JANTSCH
Technological forecasting in perspective
Organization for Economic Cooperation and Development Paris France 1966
pp 178–181

4 H KAHN A J WIENER
The Year 2000 A framework for speculation on the next thirty-three years
Vol II of the working papers of the Commission on the Year 2000 Hudson Institute Inc Croton-on-Hudson 1967 Chapter I

5 P E ROSOVE
An analysis of possible future roles of educators as derived froma contextual map
System Development Corporation Santa Monica California SP-3088 8 March 1968

6 *Technology in education*
Hearings before the Subcommittee on Economic Progress of the Joint Economic Committee Congress of the United States 89th Congress Washington DC US Government Printing Office June 6 10 and 13 1966

7 B GLASS
Information crisis in biology
Bulletin of the Atomic Scientists October 1962 pp 6–12

8 R J SCHAEFER
The school as a center of inquiry
Harper & Row New York 1967 p 46

9 G VENN
Man education and work: Postsecondary vocational and technical education
American Council on Education Washington DC 1964

10 F MACHLUP
The production and distribution of knowledge in the United States Princeton University Press Princeton New Jersey 1962

11 H SACKMAN
Computers system science and evolving society
John Wiley & Sons Inc New York 1967 p 531

12 D L BITZER
"PLATO: Research in automatic teaching at the coordinated science laboratory University of Illinois The automation of school information systems
National Education Association of the United States Monograph No 1 (ed.) D D Bushnell 1964 pp 103–105

13 R J SCHAEFER
op cit pp 33–40

14 J I GOODLAD et al
The changing school curriculum
The Fund for the Advancement of Education 1966
15 *Technology in education*
op cit p 8
16 W C BAILEY F J CLUNE Jr
Preparation of elementrary school units on the concept of culture
Human Organization Vol 27 No 1 Spring 1968 p 8
17 *Technology in education*
op cit p 78
18 J W LOUGHARY
Man-machine systems in education
Harper & Row New York 1966
19 J C R LICKLIDER
Libraries of the future
The M I T Press Cambridge Massachusetts 1965

A public philosophy for real time information systems

by H. SACKMAN

System Development Corporation
Santa Monica, California

INTRODUCTION

The electronic digital computer has been with us for only a single human generation. When spawned in World War II, the human motivation behind its birth and development was in the tradition stemming from the ancient abacus up to the digital machines used by Pascal, Leibnitz, and Babbage—to relieve the tedium of laborious computation through machine assistance. Babbage was attracted to computers because of the endless, repetitive work necessary to produce logarithmic tables; von Neumann's early interest in computers grew from its potential as a shortcut computational aid in such problems as the analysis of atomic reactions.

The social implications of computers caught the public eye in the 1950's when the first real time information systems were applied to military command and control, to manned spaceflight, and at the end of that decade, to initial applications of online systems in business and industry. The time-sharing developments of the 1960's have extended the twin concepts of online man-computer communication and real time control in the form of the impending information utility. The speed of technological change in the computer world, coupled with the lack of an experimental and humanistic tradition in the application of computers, has contributed to a philosophical void in the social ramifications of real time information systems.

The computer scientist was initially able to take refuge in the once-respectable but now largely discredited notion that philosophy and human values lie in a transcendental subjective domain far removed from the objective operations of science. On the social side, the experience of the atomic scientists in World War II has underscored the social accountability of all scientists in radioactive human debris. On the psychological side, the analysis of scientific problem-solving—in the real world, not the paper world—has shown the all-too-human behavior of scientists. Human values and scientific method are passengers in the same boat, traversing the same stormy sea, sharing a common destiny.

The public has been insulated from philosophical confrontation with computers because they rarely had direct interactions with computers. However, the inquiry of the Federal Communications Commission into the computer/communications industry and the imminence of public computer utilities irrevocably changes the role of the public from spectator to participant.

The refusal to think problems through and to take a stand is itself a philosophical position—a position of passivity, drift, solipsism, nihilism, or agnosticism—depending upon one's temperament. The admission of the need to start thinking problems through and basing action on rational, tested belief is the admission of a need for a philosophical quest, for a public philosophy of real time information systems.

The need for a public philosophy

The diverse needs for a public philosophy on the use of computers for the regulation and control of social affairs stem from many cultural roots. Perhaps the most fundamental source is the accelerating tempo of contemporary change spurred by the advance of science and technology. Situations and events seem to be moving faster than we can recognize and cope with them. Social solutions which once had a useful half-life spanning decades now have useful total lives over much shorter periods and have to be constantly revised and updated along the way to keep pace with changing conditions.

The concept of the real time information system—a system that monitors events in a specified environment and controls the outcome of such events in a desired

direction—is a leading technical concept that is being increasingly applied to cope with fast-moving changes in many walks of life. The power of computerized real time information systems to meet rapidly changing problems and situations has been garnered from over a decade of experience in computer-assisted command and control. The technique is well-known: continual surveillance over the object environment to permit early warning of critical situations; identification of problems; corrective regulation and control in accordance with established standards of system performance; and evolutionary adaptation of system design and operations to meet changing conditions.

The real time information system is a new class of social institution, a more radically powerful and rapidly responsive social form to recognize, meet and deal with specified problems at the time they occur and in time to modify their outcome. If we neglect to formulate desirable social consequences for these new systems, we neglect them at our own peril, and at public peril.

Information power is a new dilemma in modern society. Social control of information power is a focal problem for a public philosophy of real time information systems. Prior to the advent of real time computing systems, information was collected and stored in a manner that tended to separate knowledge from action, as in books and films. Radio and television allowed more timely collection and dissemination of information, but these mass media of communication were still not linked to direct social action. In real time computing systems, however, the collection, organization and storage of information leads directly to action, to integrated surveillance and control over the object environment. This dynamic marriage of information and control in real time systems is a fusion of knowledge and action, and, through directed action in real time, information is expressed as power.

As more and more social knowledge becomes computer-accessible, so will more extensive, interlocking, and more powerful real time systems come into being. As surely as the night follows the day—or the day follows the night, depending on your outlook—so will computer-accessible information be followed by real time control. In a democracy, the public is the ultimate source of social power, and information power, accordingly, is ultimately a public trust. A public philosophy on information power needs to account for new democratic forms and procedures bearing on the organization and equitable distribution of social information.

Philosophical challenges are encountered not only in general areas such as meeting the tempo of contemporary change and coping with the institutionalization of information power, but also and perhaps most critically in the problems of social implementation. A workable philosophical position should provide guide-lines for social method, for putting principles into practice. While many agree on broad principles, consensus often vanishes when details of implementation are hammered out.

There are many knotty questions facing the implementation of real time information systems in the public interest. Where does the domain of public information end and where does the domain of private information begin? Is the information utility a genuine public utility and, if it is, what kind of commodity is public information? Should information be distributed to the public on a metered basis, as we do with gas, water and electricity, or should it be freely available as in radio and television? Should the cultural store of computer-accessible public information be available to all as a basic human right, supported by the government and the taxpayer, or should such public information services be supported by private enterprise, or do we need a judicious mix of public and private support? Who tests and evaluates real time information systems for social effectiveness and who evaluates the evaluators? What legal changes and what new social agencies are required to safeguard the public interest and to protect the private interest in the field of information services?

It should be apparent from the foregoing that the challenge of a public philosophy for real time information systems is, in many respects, unprecedented and extremely complex, at general levels, in details, and in implementation. At the same time, the need for such a philosophy is vital and long overdue. The next section is devoted to an inquiry into key elements of a public philosophy of real time information systems; this inquiry then leads to a synthesis of these elements into an initial philosophical framework.

Elements of a public philosophy

The desiderata of a public philosophy are developed in three stages, starting from definitions of the area of inquiry, proceeding to scientific and technical aspects, and culminating with social considerations. Each stage builds upon and incorporates the preceding stage. While the social stage represents the broadest set of elements and requirements, it does not attempt to describe a substantive public philosophy of real time information systems, per se, which is the main business of the last part of this paper.

A philosophy on real time information systems presupposes some definition of the concept of "real time". Historical interpretations of time, and by extension, "real time," have assumed the varied forms of the conceptual containers into which notions of time, like a liquid, have been poured. These interpretations range

from the ceaseless flux of Heraclitus, to the flickering unreality of Platonic change, to Newton's geometrization of time, to Einstein's space-time, to probabilistic and indeterminate temporal constructions in quantum physics, to ecological statistical trends in evolutionary time. While probabilistic and contingent interpretations of real time seem to be gaining increasing ground in the physical, biological and social sciences, controversy has been and still is the rule.

For present purposes, three aspects of real time are distinguished: real time events, real time information systems, and real time science. Each is defined and discussed in turn.

Real time essentially refers to events—their appearance and duration, their passage and succession, and the hypothesized interrelations of events as empirically tested and demonstrated in any referent system and its environment. Real time is thus the way events happen, our description of how they happen, and our best interpretations of why they happen as they do. The definition also implies that warranted interpretations are those empirically certified by experimental method in a system context.

Real time information systems refer to systems that 1) continually sense and respond to selected changes in an object environment 2) in a manner and in time to enable regulation and control over some ongoing events in the system and its environment while they occur, 3) within the bounds of minimal or acceptable levels of system performance as determined by continual test and evaluation of feedback from system events.

As mentioned earlier, the central feature of real time information systems is direction and control over selected system events while they take place; and in order to exert such cognizance it is necessary to maintain constant surveillance, identification (decision making), and control to modify the environment as required. Note that the definition does not mention computers. It essentially states that any system that is organized to sense and respond to an object environment according to some criterion of effectiveness is, in principle, a real time information system. The crux of this definition is that real time systems are not merely passive spectators of their own events, but are creators of desirable outcomes, that they are active agencies that mold a partially plastic environment in accordance with a preconceived image.

The next definition—real time science—moves into the second stage toward a public philosophy: the technical and scientific stage.

Real time science deals with temporally and situationally contingent events amenable to experimental method, and it results in an extension of human mastery over such events; it is broadly eclectic, borrowing freely from the methods and findings of the pure and applied sciences, and from any mix of interscience and new science as required and needed to understand and control real world events.

As we enter into the era of computer-catalyzed real time information systems, we need a scientific discipline to develop the theory and practice of real time systems, and the suggested discipline is real time science as defined above. This definition is different from conventional concepts of science in several leading respects. First, it explicitly fuses knowledge and action together as a single entity; no pretense is made for the pursuit of antecedent, abstract knowledge for its own sake. Second, the pursuit of knowledge is for human purposes, for improving human effectiveness. Third, the proper object of real time science is real world events—real time science belongs where the action is. Laboratory events and abstract constructions are not excluded, but they are preparatory rather than consummatory in the sense that they contribute toward the ultimate objective of understanding, shaping and controlling real world events for human ends.

The eclecticism of real time science is a restless, fast-moving, aggressive eclecticism, itself changing in real time with new methods and findings. Real time science borrows freely from any established or new experimental discipline that contributes to improved real time system performance. Anyone who has worked in the design and development of real time information systems is acutely aware of the eclectic and pluralistic nature of such systems, of the requirement to optimize interdisciplinary teamwork, of the necessity to adopt new science and technology into system design and operations, of the open-ended, jazz-like need to improvise against residual uncertainty, of the need to continually test and modify system configuration throughout the entire life cycle of the system.

The relation of real time science to traditional forms of science is that real time science borrows experimental method and findings wherever and whenever they are useful for understanding and directing real world events. The common denominator is experimental method. With the advent of computer-serviced societies we may expect a flowering of new species of computer-catalyzed experimental method, particularly in real time information systems embedded in real world happenings.

A criticism that may be levelled against the foregoing definitions of real time events, real time information systems and real time science is that they seem to be so broad and all-encompassing as to become meaningless; little is left out. The antidote to indiscriminate extension of real time concepts lies in the distinction from non-real time concepts. There are two basic senses

in which non-real time may be construed—as an entity in its own right, and as failure in real time systems.

In the first sense, a non-real time information system is one that does not continually sense and respond to selected changes in the object environment in a manner permitting control over events at the time they occur. Analyses of past events and planning for future events fall into the non-real time category. In computer systems, batch processing is generally conducted in non-real time, and abstract simulations are typically non-real time operations.

The results of analyses of the past, of planning for possible futures, of batch information processing and of non-real time simulations may eventually be applied to a real time systems context, and as such, non-real time behavior may be interpreted as preparatory to real time behavior. Non-real time behavior may even have its own characteristic real time pace (as in accelerated real time simulation), but, insofar as immediate control is not exerted over ongoing events, such behavior is interpreted as non-real time for concurrent events. From a practical point of view, the heart of the distinction between non-real time and real time is the distinction between knowledge disembodied from immediate and concurrent action versus knowledge expressed in action.

The second sense of non-real time is failure of a real time system to meet some specified standard of performance in controlling the system environment. Thus, in computerized real time systems, if SAGE does not destroy a hostile bomber before it reaches its target, if the Apollo spacecraft is not being picked up by the ground tracking system, if SABRE airline reservations are swamped with erroneous manual inputs, if the executive program of a time-sharing system has to handle too many users at one time, to that extent the real time information system deteriorates in performance and regresses to a non-responsive or non-real time mode of operations.

Turning now to the last stage, the social elements of a public philosophy of real time information systems, we are primarily concerned with social effectiveness. If the object of real time systems is to regulate selected ongoing events in the system environment, and if the object of real time science is to extend human mastery over real world events, then the aggregated effectiveness of such efforts is real time social effectiveness.

But just what does social effectiveness mean when applied to real time information systems? It was mentioned earlier that real time systems probably represent the most advanced technical means available for regulation and control of social change, and that information power, in a democracy, ultimately resides in the public. Social change via real time information systems is thus self-change. The public, ideally, authorizes and warrants social change. Each individual is thus both experimenter and subject in the development of real time public systems; a new level of participant democracy is needed well beyond anything that has been attempted so far. A socially effective public philosophy correspondingly requires educational changes in the general population that can lead to enhanced participant democracy. The alternative is the eclipse of the public by a new technological meritocracy.

The way out of the dilemma of overconcentration of information power in some new elite is modification of existing democratic procedures with the aid of new technological capability. Pluralistic checks and balances between competing groups and interests, conducted in the open forum, is a time-honored method for preserving a dynamic democratic equilibrium. The design of pluralistic checks and balances for diverse real time information systems and public information services would be pouring new real time wine into old democratic bottles. The new real time information services can be applied to enable the public to exert closer scrutiny over elected officials by more frequent voting, and more frequent expression of public opinion by electronic polling on key issues as they arise. The electronic potential for public control is so great that we should also be concerned with overcontrol of public officials by a fickle and changeable public, overcontrol that could lead to a more virulent form of the tyranny of the majority (as Alexis de Tocqueville[1] described it more than a century ago). The knife cuts both ways—more power means more work and greater responsibility for the public and its representatives to maintain equitable equilibrium between shifting majorities and diverse minorities.

The foregoing should make it obvious that the determination of suitable checks and balances between competing groups, competing real time information systems, between the public and various elites, between majorities and minorities, will require a long and continuing course of social experimentation. Doctrinaire solutions are no match for systematic social experiment and verified empirical demonstration. An essential requirement of a public philosophy, then, is the institutionalization of social experiment in public affairs with a corresponding internalization of experimental values in thought and outlook.

Summing up, what are the key elements of a public philosophy for real time information systems? The philosophy requires an outlook that links knowledge with action; it needs the support of eclectic real time sciences concerned with the extension of human mastery over real world events; it is characterized by diverse democratic means to achieve pluralistic social ends; and it requires an extension of experimental

method and experimental ethos to social affairs. Do we have a philosophy that brings these elements together, or do we have to invent a new philosophy for the era of real time systems in computer-serviced societies?

The promise of American pragmatism

The thesis put forth in this final section is that we do have the fundamental elements for a public philosophy of real time information systems in the legacy of American pragmatism. The following discussion develops the grounds for this position in four steps: a brief description of the historical development of American pragmatism; pragmatism as a philosophical system founded upon and profoundly influenced by real time concepts and a real time outlook; pragmatism as a coherent experimental approach to the democratization of real time social control; and the extension of pragmatism into systems science and the era of computer-serviced societies.

The recurrent theme throughout the rest of this paper is that we already have the basic elements of a philosophy of real time in American pragmatism, constructed over almost a century of hotly contested philosophical labor, as represented by its principal originators, Charles Peirce,[2] William James[3,4] and John Dewey,[5] and their successors. These three founders portray the three faces of pragmatism—Peirce the mathematical and scientific side, James the psychological side, and Dewey the social side.

"Pragmatism" is derived from a Greek root signifying action. According to James (1907),[3] pragmatism was first introduced into philosophy by Peirce in 1878. For Peirce, the meaning and value of a statement consisted of its conceivable consequences in practice, of its bearing on deliberate human control over future events. Peirce clearly envisaged the union of knowledge with action.

At the heart of Peirce's belief was his conviction of the superiority of experimental method over other methods for gaining and implementing useful human knowledge. The "truth" of statements is operationally determined by empirical verification of testable consequences achieved by iterative experimental inquiry, as it occurs in scientific progress. Peirce was the first to use the concept of inquiry in this context, a term later adopted and elaborated by Dewey. With Peirce, meaning, truth, and experimental inquiry were cast in a temporal frame, contingent upon and responsive to the cumulative consequences of ongoing human action. The long-range, evolutionary advance of science served as the idealized model for Peirce's vision of pragmatism which contained the seeds for a philosophy of real time science.

James was the popularizer of pragmatism and its most eloquent, almost poetic, spokesman. While agreeing with Peirce that the validity of statements is to be continually tested by their consequences, James broadened the domain of pragmatic meaning over the whole of human experience. His earlier preoccupation with the shifting stream of consciousness was expanded into an all-encompassing concept of experience(which he described as radical empiricism) that incorporated subject and object as an undifferentiated unity, a unity consonant with that described by Bertrand Russell (1945)[6] as "neutral monism." For James, the temporally conditioned stream of experience, flowing in a "pluriverse," displayed the same strung-along, partially connected, mosaic character as his stream of consciousness. This strung-along pluriverse was contrasted by James against the "block universe" espoused by idealists of all callings who believed in the Platonic tradition of a fixed, antecedent structure of the universe laid out in some grand, sweeping design. James' philosophical pluralism, more than that of other pragmatists, lays the groundwork for a virtually unlimited multiplicity of real time sciences and intersciences modeled after the kaleidoscopic configurations of real time systems.

Dewey was deeply concerned with the accelerating tempo of scientific and technological advance and the need for continual social reconstruction to keep pace with such changes. He seized upon the element of human control in experimental method, developed previously by Peirce, as the method of choice to implement and guide social reconstruction.

In a remarkable anticipation of systems science, Dewey attacked the efficacy of conventianal notions of true and false and urged, in their place, the adoption of operational measures of effectiveness for human, organizational, and social performance. Social behavior is not true or false—it exists, for better or worse—and our concern, according to Dewey, is to find out how effective it is, and to do it in a manner that will permit us to improve upon it to meet new conditions.

For Dewey, every existence is an event, and all events are potential experiments. Contrary to the prevailing laboratory view of science, Dewey saw the universal prospects of real world experimentation with real time events in his doctrine of experimentalism—the extension of experimental method to human affairs. Dewey effectively anticipated a philosophy of real time science by urging increased human control over social events through scientific method.

Dewey's concept of increasing experimental control over social affairs was consistently qualified as democratic control by an enlightened public. In *The Public and Its Problems* (1927)[7] Dewey put forth his prophetic vision of free social communication and democratized public control in a new machine age:

"We have but touched lightly and in passing upon the conditions that must be fulfilled if the Great Society is to become a Great Community; a society in which the ever-expanding and intricately ramifying consequences of associated activities shall be known in the full sense of that word, so that an organized, articulate Public comes into being. The highest and most difficult kind of inquiry and a subtle, delicate, vivid and responsive art of communication must take possession of the physical machinery of transmission and circulation and breathe life into it. When the machine age has thus perfected its machinery it will be a means of life and not its despotic master. Democracy will come into its own, for democracy is a name of free and enriching communion. It had its seer in Walt Whitman. It will have its consummation when free social inquiry is indissolubly wedded to the art of full and moving communication."

The above sketch is only crudely indicative of the philosophies of Peirce, James, and Dewey. It is beyond the scope of this paper to set out the distinguishing characteristics and the current diversity of American pragmatism in any detail. But these brief remarks should suffice to point up the pronounced temporal thrust of American pragmatism, the continual reconstruction of present belief toward future behavior, with vigilant appraisal of fresh consequences leading to new guidelines for further action. Peirce stressed the self-corrective aspect of the inquiring process; James focused on the human implications of the pluralistic stream of experience; Dewey emphasized instrumental means and experimental control over growing social problems in a precarious world. The flux and pressure of real time events is written large in these philosophies, and social mastery over this flux is most apparent and most comprehensively expounded in Dewey's work.

Social control has become a terrifying notion when coupled with computers. It conjures up visions of Orwell's[8] Big Brother and Wiener's[9] Golem. Dewey was always a great believer in democracy even though he was acutely aware of its many limitations as he saw them in his time. He also had an abiding faith in the public. He felt that if, in some manner, available social knowledge could be made freely accessible to the public, that the effective intelligence of the public would be released, and the excellence of democracy would be enhanced. Is the imminent emergence of the public information utility the instrumentality through which Dewey's dream can be realized? If we design the computer utility to gather and distribute public information on an equitable basis to all, and if we integrate this utility with new, experimentally evolved democratic procedures that will enable the public to use this information wisely, then, to that extent, democracy stands to be the beneficiary of the new concentration of information power, not its victim.

The development of such new democratic procedures would involve experimentation with prototype computer utilities to test alternative methods of man-computer communication in the management of social information. Results of such tests could be made widely available, openly discussed and debated, and incorporated into improved versions by public approval or through authorized agencies appointed and monitored by the public.

The concept of deliberate, institutionalized, continuing public experiment for public affairs is a new evolutionary force in democratic advance, a challenge that requires new attitudes and revised values. Each individual will have to learn to think of himself as both subject and experimenter, with lifelong responsibility for selecting and implementing new experiments, evaluating social effectiveness, and applying the results. The realization of this new experimental ethos will require far greater participation and public evaluation of social feedback than has ever occurred before in any democracy, including the personalized democracy of the city-states of ancient Greece. The information utility, linked to the public real time information base, could conceivably provide the leading instrumentality for the public to scan the social scene, identify problems, contribute to social control, and provide continuing corrective feedback on the interplay of pluralistic social experimentation.

The new experimental ethos will also require an infrastructure of continuing, lifelong education in real time, the acquisition of new knowledge when it is needed, in time to meet problems as they arise. When education occurs in real time, it is responsively adaptive and education becomes indistinguishable from on-the-spot human problem-solving. Real time education will then articulate with the tumultuous flow of social experience, and education will become an integral part of such experience. The dusty dogma of the academic creed may become a relic of the past.

There are those who argue that experimental method is good, true and beautiful, but only as long as it remains in the domain of natural science where it originated, and where, they claim, it belongs. As soon as experimental method is taken out of conventional scientific pursuits and is indiscriminately placed into such fields as democracy and education, into social affairs, then, these critics claim, you enter the never-never land of human values and transcendental metaphysics where statements become meaningless from a scientific point of view. Such is the position, for example, of the logical positivist and most behaviorists.

The pragmatist rejects this view since it would keep

scientific method confined within the scientific priesthood and deny it to the public. Experimental method is the most precious legacy of scientific endeavor, and it is too important and too valuable to entrust it to any aristocracy, scientific or otherwise. The crux of the pragmatic position is that values may be formulated as hypotheses, operationally defined, quantitatively measured, and empirically tested with results subject to further test and evaluation as in any other scientific activity. If values are treated as hypotheses, then democracy and education, social attitudes and social change, when they are operationally defined under empirically verifiable conditions, are correspondingly amenable to social experiment.

The early pragmatists were in a difficult position in defending their stand on the possibility and validity of social experimentation because the means for the universalization of experimental method were not at hand and they could not point to concrete, real-world social experiments. They could defend their position in principle but not in practice—a vulnerable position for one who calls himself a pragmatist. But now conditions have changed dramatically, particularly with the advent of systems science and the proliferation of the high-speed electronic computer. Social experiment is now possible on a scale undreamed of by the early pragmatists.

It is commonplace to point out that computers make it possible to collect, organize, and process vast amounts of data quickly and reliably in real time experiments that were beyond the ken of the precomputer era. The computeris, in fact, revolutionizing experimental method in the physical, biological and social sciences, and the end is nowhere in sight. Systematic experimental method is comparatively recent in human history, dating back only to the Western Renaissance (Sarton, 1948).[10] It has changed rapidly since its inception and has received an electronic jolt with the emergence of the general-purpose digital computer since World War II. Social experiments are now possible in a bewildering variety of forms, for a growing number of variables, with real time collection, reduction and analysis of social data. For many, perhaps most, the question is no longer whether to experiment on a social scale, but how to experiment in the best interests of the public.

The power of systems science in catalyzing and accelerating the extension of experimental method is probably not as well understood as the more obvious impact of computers. The concrete, tangible system, with its specified stages of definition, design, production, installation and operation, with test and evaluation occurring at all stages, is the organizational vehicle for the breakthrough into real-world experimentation. If a coherent social activity is organized into a formal systems framework, then the system serves both as subject and object of its own evolutionary series of system experiments, for continuing system test and evaluation. The integrated system is the crucible in which the real-world experiment is forged.

The combination of computers and system science, in a concrete system context, makes possible the universalization of experimental method in an unprecedented manner. The computer complex can and has served as a built-in laboratory for test and evaluation of system performance. This has occurred most notably and dramatically in the earliest large-scale real time systems, in SAGE air defense and in Mercury-Gemini-Apollo manned spaceflight. In each case there was an attempt to make a great leap forward into new knowledge and new control over uncharted domains. The only way to achieve system goals within planned timetables was to experiment rapidly and boldly with new techniques and new findings.

The system configuration served as its own test bed in measuring and assessing system performance. Simulation, training, testing and evaluation were indistinguishably intermixed in system development in a new experimental style—in computer-aided, interdisciplinary, mission-oriented, self-experimentation in real time, under common schedules and common system goals. With the advent of other and newer real time information systems in industry, science, education, medicine, and now, on the threshold of computer utilities, the experimental dance is improvised on new real time tempos.

To the extent that a systems approach is deliberately integrated into social organization, and to the extent that such systems are computerized, to that extent will potential experimental power grow for real-world social experiment. Saying that experimentation is good and noble is not enough, there must be the means and the explicit social configuration—the real time systems configuration—to make such experimentation feasible. The first step, then, in the evolution of real-world social experiment is the evolution from non-real time to real time systems. And the more advanced the computerization of such real time systems, the more potent are the possibilities for ongoing systems experimentation. Social experiment will spread as real time information systems spread and proliferate into interlocking networks, ultimately into ecological complexes of openly cooperative and competitive real time information systems.

The form that real time social control will take will depend on how real time information systems are implemented. If competitive social experiments are freely conducted in open forum, if many alternatives are explored, if real, not rigged choices are open to the public, if grassroots participation and feedback is built into the genes

and chromosomes of object systems at the system design stage, if adequate checks and balances are devised between the public, the managers, the operators, and users of such systems, if these and related conditions are met, then real time social control may effectively turn out to be of the people, by the people, and for the people, rather than for the old plutocracy or a new technological elite.

When pragmatism was thrust with a "barbaric yawp" into the world of philosophy, mainly through the efforts of William James, a hue and cry arose from many quarters on the crassness and narrowness of this new outgrowth of American materialism. Pragmatism was maligned as opportunistic, self-indulgent, unscientific, anarchistic, and as an ideal comedian's philosophy. The semantic storm over the pragmatic as the narrowly practical was overwhelming; repeated onslaughts from the ideal, the good, true, and beautiful, from the pure and theoretical were launched against this newborn philosophy from all sides.

Peirce, disagreeing with James' exposition of pragmatism, insisted that his theory be called pragmaticism. James turned to the more technical concept of radical empiricism to ward off the semantic pitfalls of pragmatism. Dewey lingered longest over pragmatism, and somewhat reluctantly turned to instrumentalism and experimentalism to defuse the relentless onslaught from critics.

But the temper of the times has changed and the horrendous connotations of pragmatism have become more respectable in a world that desperately needs intelligent, practical solutions to mounting problems. While pragmaticism, radical empiricism, and instrumentalism remain as distinctive historical hallmarks of their creators, American pragmatism persists as the designation of their common origin, continued growth, and diversification in contemporary affairs. After being drummed out of court for challenging the established, absolutistic order, American pragmatism is experiencing a renaissance.

With the emergence of real time information systems, the pragmatic temper of American science and technology has received a fresh impetus and powerful new thrust. Peirce's "knowledge of consequences" has become transmuted into the principle of real time feedback; James' "cash-value" has become mission payload and system payoff; Dewey's social inquiry and behavioral effectiveness has evolved into system and cost-benefit analyses; the early pragmatic focus on the regulation of future consequences has been transformed into human real time control. The philosophy of pragmatism has evolved into the philosophy of real time.

The intent of this essay has been suggestive and exploratory. It has suggested the power of the real time concept in contemporary social change and it has explored its links with the tradition of American pragmatism. The meaning and import of real time information systems has not yet been pursued and systematized by philosophers. The scope and diversity of pragmatism and competing philosophies has only been hinted at in these pages. Controversies and logical problems in the construction of a public philosophy of real time information systems have been bypassed or have been mentioned only in passing, and those that have been addressed have not been dealt with in detail. Some of these problems are treated elsewhere at greater length by the author (1967).[11] If this discussion has served to point up the vital need for a public philosophy of real time information systems, and if it has shed some light on the formulation of the problem and on key issues, then its purpose has been met.

REFERENCES

1 A TOCQUEVILLE
Democracy in America 1835

2 C S PEIRCE
Collected papers of Charles Sanders Pierce
Charles Hartshorne and Paul Weiss (eds) The Belknap Press of Harvard University Press Cambridge Massachusetts 1935

3 W JAMES
Pragmatism A new name for some old ways of thinking
Longmans Green & Company New York 1907

4 W JAMES
A pluralistic universe
Longmans Green & Company New York 1909

5 J DEWEY
Intelligence in the modern world
J Ratner (ed) Random House New York 1939

6 B RUSSELL
A history of western philosophy
Simon & Schuster New York 1945

7 J DEWEY
The public and its problems
Henry Holt New York 1927

8 G ORWELL
1984
Harcourt Brace New York 1949

9 N WIENER
God and Golem Inc
M I T Press Cambridge Massachusetts 1964

10 C SARTON
The life of science
Indiana University Press Bloomington 1948

11 H SACKMAN
Computers system science and evolving society
Wiley & Sons New York 1967

A special interest session on computer design automation

Computer design automation: What now and what next?

by JEROME M. KURTZBERG, Chairman

IBM Watson Research Center
Yorktown Heights, New York

This special interest session of computer design automation explores the current problems that face us, what we can do and are accomplishing, and what appears to be our objectives and possibilities in the future.

Historically, the design tasks that were first automated were those with the largest savings, in terms of cost and time, and ease of mechanization. Thus, manufacturers initially automated the clerical checking and documentation functions involved in computer design. When done manually these tasks consume a major portion of design time and are particularly onerous for humans. Concurrently, the task of establishing pin-to-pin interconnection chains for backplane wire routing was mechanized, at first for manual wiring, and later for input to automatic wire-wrap machines.

Success in these areas encouraged manufacturers to try to automate more of the tasks involved in the physical design area using as input the logic equations or diagrams. This called for development of algorithms to handle the interrelated problems of partitioning and assignment of logic equations to circuits on modules (card-packaging), placement of the modules in a backplane, and determination of pin interconnection and wire routing patterns. Although these tasks are interrelated, it was considered necessary, because of their complexity, to treat them separately. Furthermore, the exact definitions of these physical design tasks are strongly dependent upon the particular circuitry employed, so as to encompass an entire class of problems. Today, a number of the major manufacturers have developed a set of techniques to handle these problems.

Attention also turned to the automatic generation of logic equations from the register information flow for the various computer operations. Gorman and Anderson[1] in a paper given at the 1962 FJCC demonstrated that the production of logic equations can be treated as a formal translation process in the manner of compiler construction.

The problem of logic equation simplification has been addressed and extensively treated from the theoretical viewpoint.[2] The same has been true for techniques for detection of single faults in combinational logic.[2] Also, there are many production problems in the manufacturing interface area that have received considerable attention such as automatic testing of hardware.

Today, the problems in design automation are more difficult than in the past because of the increasingly complex demands imposed on the computer design process. These demands arise from the trend toward more sophisticated computer organizations and the numerous and interrelated constraints occurring in the new technologies. These problems are discussed by the panelists with audience participation after the speakers explore specific areas of interest.

Gorman deals with the areas of functional computer design and evaluation of the design. He treats the problems of securing an adequate language for expressing the system specification, the problems of "design translation" and subsequent evaluation of the resulting detailed design. Next, Russo discusses the interface between logic and hardware. The problems posed by the LSIt echnology are examined. Donath then elaborates on some of the problems in the physical design area. He stresses the need for certain analytic studies in order to gain a better understanding of the algorithms necessary to handle the network design tasks. Breuer speaks last on the problems in the automatic generation of component failure detection and diagnostic test sequences. The various approaches that have been used and their limitations are discussed along with the effect of hardware fault detection in the actual design of equipment.

The interested reader is referred to a survey paper by Breuer[2] for a comprehensive bibliography of previous work in computer design automation.

REFERENCES

1 D F GORMAN J P ANDERSON
A logic design translator
1962 Proc FJCC pp 86–96

2 M A BREUER
General survey of design automation of digital computer
Proc IEEE Vol 54 No 12 December 1966 pp 1708–1821

PANELISTS

Donald F. Gorman
RCA
Cherry Hill, New Jersey
Roy L. Russo
IBM Watson Research Center
Yorktown Heights, New York
Wilm E. Donath
IBM Watson Research Center
Yorktown Heights, New York
Melvin A. Breuer
University of Southern California
Los Angeles, California

Functional design and evaluation

by DONALD F. GORMAN

Radio Corporation of America
Cherry Hill, New Jersey

The *design* of information processing systems consists of the development of (1) the initial marketing specifications, (2) a functional design to meet these specifications including a detailed logic design to achieve these functions, and (3) an implementation of this logic design, in either hardware or stored microprogram, using the latest technology.

Functional design encompasses all activities involved in producing the functional design from the behavioral specifications and the conversion of this design into a logic design.

Evaluation of a design is the process of determining if the functional design, the logic design, and the implementation satisfy the initial behavioral specifications.

What now?

What techniques are currently available for aiding in the design and evaluation tasks? To assist in the design of systems, a technique called "design translation" is used to convert from one level of design to another. It treats design as the translation of a description of a product from one language to another.

The conversion from a marketing specification to a functional design is, at the present time, not a well-defined process and, therefore, not amenable to automation. Although translation from a functional design to a logic design has been shown to be feasible (1962) the development of these techniques has not progressed very rapidly, due to an apparent lack of interest. Nevertheless, several such systems are currently being implemented; emphasis is on achieving a practical automation system that will produce practical designs.

The evaluation of information processing systems, including both hardware and software, has proven to be quite intractable. The method of evaluation most widely used today is simulation; however, its use is limited and hardly qualifies as automation. Current work on simulation centers on the development of the concepts required to model such a complex product as an information processing system and the development of simulation languages to embody these concepts and make them easy to use.

What next?

Some areas in which progress must be made in order to bring the present efforts to fruition will now be considered.

Provided that a fixed control philosophy is assumed, design translation techniques can be applied to a functional design to produce a logic design. However, in order to optimize a design, it must be possible to merge several control methods, such as synchronous, asynchronous, a centralized clock, distributed control, etc., into a single design. Techniques must be developed to determine the control method best suited for each portion of the design, and translation techniques are needed to produce a single coordinated design utilizing several different control philosophies.

Design translation techniques will provide, for the first time, sufficient detailed information about alternate designs that tradeoff studies may be effectively undertaken with respect to hardware vs. software, synchronous vs. asynchronous, etc. When criteria to be used in these tradeoffs have been established, the tradeoffs themselves may be incorporated into the translation procedure and designs may then be optimized over a wider range of criteria.

Translation from a behavioral specification to a functional design is the truly creative part of the design activity and is currently considered to be an art rather than a science. Design experience using translation techniques and the ability, via automation, to analyze

numerous designs will eventually lead to a methodology for automating aspects of this phase of design.

Evaluation of information processing systems encounters even more problems than design. One of the fundamental problems today is that there is general disagreement over what constitutes a "good" design. Assuming that satisfactory criteria can be found, techniques for rapidly evaluating systems with respect to these criteria are needed.

Analytic procedures are currently being developed, but are, as yet, far from being general enough for adaptation to automation.

Simulation, which is the only evaluation technique available today, suffers from a severe handicap—inefficiency—in both the writing and running of simulation programs. Efficiency will be achieved and simulation will become a practical design aid through the following steps: (1) the use of simulation program generators that will facilitate the creation of customized simulation programs, (2) the generation of simulation models that allow a detailed view of the area of immediate interest to be combined with a very general view of other portions of the system, and (3) the generation of integrated simulation programs based on such models.

Summary

Design automation, which was first used in the manufacturing and physical engineering areas, has made considerable progress in logical design and is now making inroads into functional design, previously considered to be an entirely creative process. The use of simulation and design translation is the first step in transforming the functional design of both the hardware and the software of information processing systems from an art to a science.

The logic-to-hardware interface area of design automation

by ROY L. RUSSO

IBM Watson Research Center
Yorktown Heights, New York

The logic-to-hardware interface area is concerned with the problems of translating a logic description into a high level physical description. The logic description is an interconnection of the primitive circuits, such as And's, Or's, Nor's, Flip-Flops, etc., that are to be used in the physical packages. A high level physical description takes into account, for example, partitioning of the logic into physical packages, but does not include relative placement of the primitive circuits within a package, or pin assignment, wire routing, etc.

This interface area is extremely important at the present time for two basic reasons. First, the present integrated circuit technology imposes critical and interrelated constraints which must be satisfied during this phase of the design process if the technology is to be used effectively. These constraints include minimizing the number of integrated circuit chips or chip types to reduce cost and specifying chips that can be tested adequately and economically. Second, decisions made in specifying the high level physical description determine to a great extent, whether or not severe machine design goals, such as fast cycle times, will be met.

The problems that design automation programs must help solve for the logic-to hardware interface area are best categorized by the class of machine being designed: either a small low-performance one or a large, high-performance machine.

In the small machine category, chip type minimization and machine cycle time constraints can be set aside to make the problem more amenable to attack and the cost of the computer programs more reasonable. Since manufacturing costs will depend almost directly upon the number of unique chips in the machine, the major problem in this case will be to partition so as to reduce that number. The relatively small number of terminals on a chip compared to the large number of circuits on the chip makes this partitioning extremely difficult. It also increases the number of tests which must be applied to the chip and this raises the cost of testing. Hence, the effect of the implementation on testing must also be considered at this level. If the packaging hierarchy consists of logic on chips, chips on carriers, etc., then the problems become more difficult and the techniques must be extended.

The problems increase enormously for large, high-performance machines because the number of constraints and their complexity increase sharply. In particular, chip type minimization and proposed machine cycle time constraints must be considered. (The low volume of machines implies that the same chip type will have to be used repeatedly in the machine to reduce costs.) Whereas in the small machine case, the specific placement of chips in the packaging hierarchy was not important, it must now be considered because of its effect on the cycle time. Hence, for this class of machines, there are problems of partitioning logic to chips in such away as to minimize chip types, and of placing the individual chips in the packaging hierachy in such a way as to satisfy a proposed short cycle time with its implied constraints.

At the present time, it is only by examining hardware packaging alternatives that designers will be able to obtain a high level physical description that best satisfies or compromises among the many constraints. Trying alternatives depends upon three elements. First, the designer must decide on the alternative he wants to try. Second, a means must be available to implement the alternative rapidly and economically. Third, the designer must examine the effect of the implementation and decide what he wants to do next. Design automation programs provide a rapid and economical means for trying a sufficient number of alternatives with appropriate output information to the designer.

We in design automation have been for a long time using the computer as a sort of "passive" partner in the design process. We've been using it to keep files, to do checking and to interface with manufacturing. In so doing we have lost sight of the real problem of design, that of evaluating alternatives, and of the real potential of the computer, to be an "active" partner in the design process by helping the designer to make his design decisions. The logic-to-hardware interface area offers the potential of allowing the art of design automation to advance to this new level.

Hardware implementation

by W. E. DONATH

IBM Watson Research Center
Yorktown Heights, New. York.

The area of hardware implementation shall be (somewhat arbitrarily) defined to include placement, wire routing, terminal assignment, and the interface to hardware fabrication devices. At this stage, many companies have automated at least some of these steps. In the first three areas, we encounter problems which must be solved by analysis into sub-problems and by development of algorithms; in the last area, we are faced with the problem of developing a convenient means of man-machine communication. LSI Mask Design, [1] for which various systems were developed over the last few years, falls into the latter area.

In the first three areas, one may have to solve problems related to placement and wiring rules. However, at this stage it is most important to note that few, if any, DA programs can generate as good a design as a good engineer could. They need either human intervention or much more interconnection space than is really necessary. This is proportionately more serious now since the ratio of interconnection cost to circuit cost is increasing with the new technologies.

Two lines of investigation suggest themselves. For one, we need to develop a better understanding of the capabilities of heuristic alogrithms. This implies mathematical analysis and systematic studies. However, DA problems are much too complex to yield to such analysis or studies. It seems best to abstract out of the general range of DA problems mathematical problems that contain the essential difficulties of the DA problems. At that point mathematical analysis and systematic studies become feasible.[2]

A second area of investigation arises from the structure inherent in computer logic complexes. E. F. Rent, in an unpublished work in 1960, stated that for a manual partition of large logic complexes into subcomplexes of c circuits, the terminal count is given by Ac^p, where $p=2/3$ and A is a constant. Understanding of the phenomenon might well lead to methods for taking advantage of such structure in DA programs. Furthermore, we can feel surer of our mathematical analysis if our model of logic structure is as close to reality as possible.

In conclusion, I feel that a necessary step toward better DA methods is the systematic investigation of the mathematical problems basic to Design Automation procedures where both analytic and experimental procedures are employed.

REFERENCES

1 P W COOK W E DONATH G A LAMKE A E BRENNEMAN
Automatic network generation for large scale integration
IEEE of Solid State Circuits SC 2 No 4 pp 190–196 Dec 1967

2 W E DONATH
JSIAM 16 pp 439–457 1968

Hardware fault detection

by M. A. BREUER

University of Southern California
Los Angeles, California

One of the most pressing problems currently confronting the design automation specialist is that of the automatic generation of component failure detection and diagnostic test sequences. This problem is made more difficult due to the new LSI circuits where we may

have hundreds of components on a chip with only a few accessible input/output terminals.

These chips contain both combinational and sequential logic which must be tested. It is now feasible to generate tests for combinational logic.[1,2] However, a computationally feasible solution for the general class of sequential circuits is still not known. The trend may be to only carry out a partial test and specify that a chip is component failure free with probability p. Most computer companies are currently investigating this problem, and some encouraging results have been reported.[3]

There are a number of interesting approaches dealing with generating test sequences. One technique, called the behavioral approach,[4,5] is based purely on the abstract next state and output functions defining the sequential machine M to be tested. Given a black box containing a machine M′, this approach tries to answer the question of whether or not M′ is equivalent to M. This approach leads to very long testing sequences, and is only practical for extremely small sequential circuits. One reason these sequences are so very long is that this procedure is actually doing machine identification rather than simply fault detection.

A second approach to this problem of generating test sequences is to take into consideration the actual physical implementation of M, where one obtains the machine M_j from M by considering how M operates under the jth component failure.[3,6] This procedure leads to shorter test sequences, and appears to be the most promising approach presently available.

Most approaches make many assumptions on the type and number of faults which can occur. For example, usually only one fault is considered at a time, and the number of states in the machine remains the same. Fortunately, the resulting checking sequences are usually still effective even if these assumptions do not hold. Unfortunately, the stuck-at-zero and stuck-at-one static fault model most commonly used is not always sufficient to test a machine, since it neglects time delay and interconnection shorts. The specification of an adequate fault model for an LSI circuit chip is still an open question.

One assumption which is usually not valid is that one has available the state transition table for the machine. For any reasonable size circuit, this table is much too large to either generate or store. In practice all one actually has available is the set of equations or diagrams specifying the logic of the machine.

A special case of the fault detecting problem is that associated with cascaded machine structures.[7] A simple example of such a structure is an n bit accumulator, which should be considered as n identical 4 state machines rather than as one 2^{an} state machine. Efficient procedures for testing such structures are not known except for some specific cases.

Finally, one can take a probabilistic approach to the problem of generating a good checking sequence. That is, for a fixed value C, how would one construct a checking sequence of length C which will maximize the probability of detecting a fault, should a fault be present.

Fault diagnosability deals with specifying, down to the component or circuit level, what fault has occurred. Usually, a fault detection test gives some information on what fault is present. More detailed information can be obtained by increasing the length of the test sequence and by a judicious sequencing of the test cases. Usually, the computer user is only interested in locating a fault down to the level of a replaceable unit.

In summary, there are many problems still unsolved in the area of completely automating the generation of fault detecting experiments. It is important that these problems be solved so that our new technological advances can be utilized to their fullest potential. The solution to this problem could be made somewhat easier if the concept of fault detection was added as a new criteria in the design of circuits.

REFERENCES

1 J P ROTH
Diagnosis of automata failures: a calculus and a method
IBM Journal of Research aud Development 278–291 1966

2 D B ARMSTRONG
On finding a nearly minimal set of fault detection tests for combinational logic nets
IEEE Trans on Electronic Computers vol EC–15 no 1 66–73 1966

3 E R JONES C H MAYS
Automatic test generation methods for large scale integrated logic
IEEE J Solid-State Circuits vol SC–2 221–226 1967

4 F C HENNIE
Fault detecting experiments for sequential circuits
IEEE-Switching Circuit Theory and Logical Design 5th Annual Symposium—Special Publ 5–164 95–110 1964

5 Z KOHAVI P LAVALLE
Design of sequential machines with fault-detecting capabilities
IEEE Trans. on Electronic Computers vol EC–16 no 4 473–485 1967

6 S SESHU D N FREEMAN
The diagnosis of asynchronous sequential switching systems
IEEE Trans on Electronic Computer vol EC–11 457–465 1962

7 M A BREUER
Fault detection in a linear cascade of identical sequential machines
Proceedings 9th Annual Switching and Automata Theory Symposium 1968

1968 FALL JOINT COMPUTER CONFERENCE COMMITTEE

Chairman

Dr. William H. Davidow
Palo Alto Division
Hewlett-Packard Company

Vice Chairmen

Thomas R. Dines—Administrative
NASA
Ames Research Center

Donn B. Parker—Technical
Control Data Corporation

Conference Administrator

Albert C. Porter
California Public Utilities Commission.

Education Program

R. J. Andrews—Chairman
IBM Corporation

N. O. Salberg—Vice Chairman
IBM Corporation

R. C. Haseltine
R. C. A.
Instructional Systems

Exhibits

Thomas C. Bieg—Chairman
IBM Corporation

Jeffery Stein—Vice Chairman
Greyhound Computer Center

Ray Telford McLaury, Jr.
Crocker Citizens Bank

Ladies Program

Mrs. Nancy Thoman—Chairman
Hewlett—Packard Company

Mrs. Mary Ann Maloney—Vice Chairman
Computer Usage Development Corp.

Miss Bernice Doebler
Hewlett—Packard Company

Mrs. Janet Shane
Bechtel Corporation

Miss Jan Swanson
IBM Corporation

Mrs. Jackie Wheeler

Local Arrangements

Ralph R. Wheeler—Chairman
Lockheed Missiles & Space Co.

Norm E. Pobanz—Vice Chairman
Electronic Associates, Inc.

Joe Brashear
Control Data Corporation

Sandy Caipo
Control Data Corporation

Robert L. Colvin
Lockheed Missiles & Space Co.

Everett Eiselen
IBM Corporation

Cuthburt C. Hurd
Computer Usage Company, Inc.

Printing and Mailing

Gordon Pelton—Chairman
Mobility Systems, Inc.

James M. Weldon—Vice Chairman
Hewlett—Packard Co.
Microwave Division

Public Relations

Russell H. Berg—Chairman
Hewlett—Packard Company

George F. Caulfield—Vice Chairman
URS Systems Corporation

William C. Estler—Consultant

Publications

Joseph Horner III—Chairman
Control Data Corporation

James B. Dolkas—Vice Chairman
Control Data Corporation

Donald M. Lytle
URS Systems Corporation
Atlantic Systems Center

Registration

David Katch—Chairman
Boole & Babbage, Inc.

Kent Gould—Vice Chairman
Boole & Babbage, Inc.

James L. Mueller
U.S. Geological Survey

Science Theater

Nels Winkless—Chairman
Communications Contact, Inc.

Technical Program

Robert H. Glaser—Chairman
Compata, Inc.

Rex Rice—Vice Chairman
Fairchild Semiconductor

Professor James B. Angell
Stanford University

Robert Bond
Hewlett—Packard Company

Dr. Sidney Fernbach
University of California

Miss Marjorie Hill
Control Data Corporation

Paul Hodge
Memorex Corporation

Warner King
Computer Usage Development Corp.

Mrs. Maria Robertson
Compata, Inc.

Marty Silberberg
IBM Corporation

Roger Simons
IBM Corporation

Treasurer

Edward Dodge—Chairman
R. C. A.
Instructional Systems

Liaison

Dr. Morton M. Astrahan—AFIPS Conference Committee
IBM Corporation

Howard Bromberg—ACM
Information Management Incorporated

David R. Brown—IEEE
Stanford Research Institute

D. R. Cruzen—AFIPS Headquarters
American Federation of Information Processing Societies

James L. Dolby—AMTCL
R & D Consultants

John E. Sherman—Simulation Councils
Lockheed Missiles & Space Co.

REVIEWERS, PANELISTS, AND SESSION CHAIRMAN

REVIEWERS

C. T. Abraham
R. L. Alonso
E. B. Altman
A. Arakawa
P. Armer
G. N. Arnovik
M. M. Astrahan
D. C. Augustin
A. Avizienis
P. R. Bagley
J. A. Baker
A. E. Barlow
R. Barnett
A. L. Bastian
A. Batenburg
J. A. Bayless
G. S. Beckwith
G. A. Bekey
S. S. Biglione
C. D. Birkhead
H. D. Block
J. A. Bloomfield
H. E. M. Blowey
G. Boer
G. R. Bolton
H. Borko
A. M. Bradley
J. R. Brown, Jr.
R. E. Brown
L. L. Burns
P. Calingaert
E. D. Callender
A. V. Campi
R. Carlson
R. L. Carmichael
R. Chapman
T. E. Cheatham, Jr.
B. F. Cheydleur
C. Chow
W. F. Chow
A. Clark
L. J. Clingman
A. B. Clymer
R. W. Coleman
T. W. Connolly
M. Connors
R. N. Constant
A. E. Corduan
R. K. Cralle
J. F. Cunningham
D. R. Dawdy
K. Detzer
S. M. Drezner
T. J. Dylewski
L. Earnest
R. Eggelston
R. F. Elfant
P. England
D. Evans
G. A. Fedde
D. Finn
R. Fitzgerald
J. L. Flanagan
A. M. Fleishman
C. S. Fluke
F. H. Fowler
M. Fox
D. Frazer
J. Friedman
C. V. Freiman
R. T. Fulep
R. H. Fuller
L. Gainen
G. H. Gales
L. F. Garrett
T. J. Gilligan
M. Gilliland
E. Glaser
J. A. Gosden
M. H. Gotterer
A. J. Gradwohl
G. Groner
O. Gutwin
R. G. Hamlet
A. G. Hanlon
H. P. Hartkeweier
R. D. Hartwick
C. Haspel
K. Haughton
A. Hauser
M. Heilweil
P. J. Hermann
T. C. Hogan
G. P. Hyatt
E. L. Jacks
A. S. Jackson
B. Johnson
T. G. Jones
T. Kallner
W. J. Karplus
N. M. Kendall
C. W. Kessler
E. S. Kinney
R. C. Knepper
M. Kochen
J. Koford
G. A. Korn
R. L. Kuehn
J. L. Kuhns
J. H. Kuney
B. Lampson
D. J. Lasser
J. Lathrop
C. E. Leith
D. C. Lincicome
C. R. Lindholm
A. T. Ling
R. Linsley
H. A. Long
C. W. Malstrom
M. Mantalband
W. A. Marggraf
J. Markus
T. W. Martin
E. Martinelli
A. J. Mauceri
C. H. Mays
M. McCarthy
M. E. McCoy, Jr.
J. McLeod
J. E. Meyer
J. Michener
S. W. Miller
R. G. Mills
M. Minsky
E. E. L. Mitchell
B. Mittman
W. Moore

G. J. Moshos
J. Murphy
F. W. Murray
R. P. Myers
J. A. Narud
D. Nee
J. L. Neece
I. D. Nehama
R. A. Nesbitt
P. Neumann
A. Newell
N. Nilsson
J. Noe
A. Obrien
D. Olson
P. R. Orman
E. Osborne
T. Pendergast
J. S. Perry
S. R. Petrick
W. J. Plath
N. Pobanz
A. V. Pohm
D. A. Pope
A. W. Potts
R. Prather
R. J. Preiss
J. P. Pritchard, Jr.
T. P. Plyer
I. C. Pyle
K. H. Rash
L. C. Ray
S. G. Reed
F. C. Reiman
J. W. Rigney
L. Roberts
J. J. Robinson
A. E. Rogers
J. Roseman
C. Rosen
M. Rosenberg
S. Rosenthal
J. D. Sable
E. Salbu
G. Salton
J. L. Sandborn
W. B. Saner
F. J. Sansom
L. Sashkin
H. N. Sassenfeld
E. Savas
J. D. Schmidt
A. J. Schnieder
E. Schubert
S. T. Sedelow
T. K. Seehuus
D. Shansky
W. Sharpe
G. T. Shuster
A. Shaw
R. Silver
L. C. Silvern
Q. W. Simkins
R. F. Simmons
T. A. Smay
B. L. Smith
L. Smith
P.C. Smith
R. V. Smith
S. Smith
W. R. Smith
E. W. Snyder
C. Spitzer
W. C. Spring
H. H. Steenbergen
K. Stevens
A. J. Stone
J. C. Strauss
W. A. Sturn
R. K. Summit
I. Sutherland
R. I. Tanaka
A. S. Tauber
W. P. Timlake
F. M. Tonge
D. Toombs
L. Travis
D. Truitt
G. Tyson
R. L. VanHorn
H. R. Van Zueren
E. G. Vesely
R. Vichnevetsky
D. E. Walker
D. Walker
R. K. Wakerling
C. Walton
B. Wang
W. M. Washington
G. A. Watson
M. N. Weindling
R. White
R. L. Wigington
M. Wildman
J. E. Wolle
H. Wolpe
R. E. Wyllys
J. W. Young, Jr.
P. De Young
N. S. Zimbel

PANELISTS

J. Arachtingi
M. H. Ballot
G. R. Bolton
D. Comer
A. J. Critchlow
J. Dengell
M. Duggan
F. R. Field, Jr.
J. K. Hawkins
J. H. Hiestand
G. Hollander
M. Irwin
A. S. Jackson
E. G. Johnson
J. Karber
R. V. Kelly
K. E. Knight
A. J. Mauceri
H. W. Mergler
E. Shapiro
T. Sheridan
B. Strassburg
P. D. Walker
E. A. Weiss
T. J. Williams

SESSION CHAIRMEN

J. B. Angell
K. Beisty
J. Bouvard
R. Conn
R. J. Creasy
A. J. Critchlow
A. van Dam
S. Elkin
J. M. Engel
D. C. Englebart
L. Fein
L. Feldner
H. R. Gillettee
A. Greeman
W. A. Gross
O. Gutwin
I. N. Hooton
M. E. Hopkins
T. Kehl
J. D. Kennedy
J. M. Kurtzberg
G. Lewis
W. Lichtenberger
R. N. Linebarger
R. Little
P. R. Low
G. Martins
M. S. Mason
G. Michael
D. E. Mulvihill
J. H. Munson
G. L. Murray
G. W. McClary
N. R. Nielsen
A. Opler
A. V. Pohm
B. Raphael
D. R. Reddy
M. Rotenberg
H. Sackman
I. R. Schwartz
A. Seelenfreund
J. E. Sherman
J. C. Strass
E. A. Weiss

AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES (AFIPS)

OFFICERS and BOARD of DIRECTORS of AFIPS

President:
MR. PAUL ARMER
The RAND Corporation
1700 Main Street
Santa Monica, California 90406

Secretary
MR. ARTHUR I. RUBIN, MP 170
Martin Marietta Corporation
P. O. Box 5837
Orlando, Florida 32805

Vice President:
DR. RICHARD I. TANAKA
California Computer Products, Inc.
305 North Muller Street
Anaheim, California 92803

Treasurer
DR. WALTER HOFFMAN
Computing Center
Wayne State University
Detroit, Michigan 48202

ACM Directors

DR. B. A. GALLER
University of Michigan
1056 Ferdon Road
Ann Arbor, Michigan 48104

MR. R. G. CANNING
Canning Publications Inc.
134 Escondido Avenue
Vista, California 92083

DR. WALTER HOFFMAN
Computing Center
Wayne State University
Detroit, Michigan 48202

MR. J. D. MADDEN
ACM Headquarters
211 East 43rd Street
New York, New York 10017

IEEE Directors

MR. L. C. HOBBS
Hobbs Associates, Inc.
P. O. Box 686
Corona del Mar, California 92625

DR. RICHARD I. TANAKA
California Computer Products, Inc.
305 North Muller Street
Anaheim, California 92803

MR. KEITH W. UNCAPHER
The RAND Corporation
1700 Main Street
Santa Monica, California 90406

MR. SAMUEL LEVINE
Bunker-Ramo Corporation
445 Fairfield Avenue
Stamford, Connecticut 06902

Simulation Councils Director
MR. JOHN E. SHERMAN
Lockheed Missiles & Space Corp.
D59-10; B-151
P. O. Box 504
Sunnyvale, California 94088

American Society For Information Science Director
MR. HAROLD BORKO
School of Library Service
UCLA
Los Angeles, California 90064

Association for Computational Linguistics Observer
DR. DONALD E. WALKER
Head, Language & Text Processing
The Mitre Corporation
Bedford, Massachusetts 01730

Special Libraries Association Observer
MR. BURTON E. LAMKIN, CHIEF
Library & Information Retrieval Staff
Federal Aviation Agency
800 Independence Avenue, S. E.
Washington, D. C. 20003

Society for Information Display Observer
MR. WILLIAM BETHKE
RADC (EME, W. Bethke)
Griffiss AFB NY 13440

American Institute of Certified Puolic Accountants Observer
MR. NOEL ZAKIN
Manager, Computer Technical Services
AICPA
666 Fifth Avenue
New York, New York 10019

Executive Director
DR. BRUCE GILCHRIST
AFIPS Headquarters
345 East 47th Street
New York, New York 10017

Executive Secretary
MR. H. G. ASMUS
AFIPS Headquarters
345 East 47th Street
New York, New York 10017

Assistant Executive Secretary
MR. D. R. CRUZEN
AFIPS Headquarters
345 East 47th Street
New York, New York 10017

AFIPS Committee Chairmen

Abstracting
DR. VINCENT E. GUILIANO
School of Information & Library Studies
Hayes C, Room 5
State University of New York
3435 Main Street
Buffalo, New York 14214

Admissions
DR. ROBERT W. RECTOR
Informatics, Inc.
5430 Van Nuys Boulevard
Sherman Oaks, California 91401

Awards
DR. ARNOLD A. COHEN
UNIVAC
2276 Highcrest Drive
Roseville, Minnesota 55113

Conference
DR. MORTON M. ASTRAHAN
IBM Corporation—ASDD
P. O. Box 66
Los Gatos, California 95030

Constitution & Bylaws
MR. ARTHUR I. RUBIN, MP 170
Martin Marietta Corporation
P. O. Box 5837
Orlando, Florida 32805

Education
DR. MELVIN A. SHADER
Manager of New Markets
IBM Corporation—SDD
1000 Westchester Avenue
White Plains, New York 10604

Finance
MR. WALTER L. ANDERSON
General Kinetics Inc.
11425 Isaac Newton Square South
Reston, Virginia 22070

Harry Goode Memorial Award
DR. GEORGE E. FORSYTHE
Computer Science Department
Stanford University
Stanford, California 94305

IFIP Congress 71
DR. HERBERT FREEMAN
Professor of Electrical Engineering
New York University
School of Engineering and Science
University Heights
New York, New York 10453

International Relations
DR. EDWIN L. HARDER
1204 Milton Avenue
Pittsburgh, Pennsylvania 15218

Public Relations
MR. CARL E. DIESEN
Chief, Computer Center Division
U.S. Geological Survey
Washington, D. C. 20242

Publications
MR. STANLEY ROGERS
P. O. Box R
Del Mar, California 92014

Social Implication of Information Processing Technology
MR. STANLEY ROTHMAN
TRW Systems, R3/2086
1 Space Park
Redondo Beach, California 90278

Technical Program
DR. DAVID R. BROWN
Stanford Research Institute
333 Ravenswood Avenue
Menlo Park, California 94025

Information Dissemination
MR. GERHARD L. HOLLANDER
Hollander Associates
P. O. Box 2276
Fullerton, California 92633

Consultant
MR. HARLAN E. ANDERSON
Time, Inc.
Time & Life Building
New York, New York 10020

U.S. Committee for IFIP ADP Group
MR. ROBERT C. CHEEK
Director of Management Systems
Westinghouse Electric Corp.
3 Gateway Center
Pittsburgh, Pennsylvania 15230

JCC General Chairmen

1968 FJCC
DR. WILLIAM H. DAVIDOW
Dymec Division
Hewlett-Packard Company
395 Page Mill Road
Palo Alto, California 94306

1969 FJCC
MR. JERRY KOORY
Programmatics
12011 San Vicente
Los Angeles, California 90049

1969 SJCC
DR. HARRISON W. FULLER
Sanders Associates, Inc.
95 Canal Street
Nashua, New Hampshire 03060

1968 FJCC LIST OF EXHIBITORS

Adage, Inc.
Addison-Wesley Publishing Company, Inc.
Addressograph Multigraph Corp.
American Telephone & Telegraph
Amp, Inc.
Ampex Corporation
Anderson Jacobson Inc.
Applied Data Research, Inc.
Applied Dynamics, Inc.
Applied Magnetics Corporation
Association for Computing Machinery
Astrodata/Comcor
Athana Corp.
Audio Devices, Inc.
Auerbach Corporation
Auto-trol Corporation

Beta Instrument Corporation
Boole & Babbage, Inc.
Bryant Computer Products
Burroughs Corporation

Caelus Memories Inc.
California Computer Products, Inc.
Calma Company
Certex Inc.
Collins Radio Company
Communitype Corp.
Computek, Inc.
Computer Applications Inc.
Computer Communications, Inc.
Computer Design Publishing Corp.
Computer Displays Inc.
Computer Industries
Computer Peripherals Corporation
Computer Sciences Corporation
Computer Transceiver Systems, Inc.
Computer Update
Computerworld
Computron Inc.
Com-Share
Conrac
Control Data Corporation
Cybetronics, Inc.

Data Communications, Systems Inc.
Data Disc, Inc., Display Division
Data General Corporation
Datamation
Data Processing Magazine
Data Products Corporation
Datascan
Datel Corp.
Decade Computer Corporation

Di/An Controls, Inc.
Digi-Data Corporation
Digital Development Corporation
Digital Devices, Inc.
Digital Equipment Corporation
Digitronics Corporation
Dura, Division Intercontinental Systems, Inc.
Dynamic System Electronics

Eastman Kodak Company
Edwin Industries
E-H Research Laboratories, Inc.
Electro-Mechanical Research Inc.
Electronic Associates Inc.
Electronic Design

Fabri Tek
Factsystem Inc.
Ferroxcube Corporation
Frieden, Inc.

Gamco Industries, Inc., A Subsidiary of Siboney Corp.
General Automation Inc.
General Computers, Inc.
General Design, Inc.
General Electric Company
General Kinetics Incorporated
Gerber Scientific Instrument Company

Hendrix Electronics
Hewlett Packard Company
HF Image Systems, Inc.
Honeywell
Houston Instrument Div. Bausch & Lomb
Hybrid Systems Inc.

Indiana General Corporation
Information Control Corporation
Information Displays, Inc.
Information Technology, Inc.
Infotechnics, Inc.
IEEE
Interdata, Inc.
IBM Corporation
ITT/Industrial Products

Kennedy Company

Lenkurt Electric Company
Litton Automated Business Systems
Litton/Datalog Division
Lockheed Electronics Company

Macmillan Company
MAC Panel Company

Magne Head, A Div. of General Instrument Corp.
McGraw Hill Book Company
Memorex Corporation
Memory Technology Inc.
Micro Switch, A Div. of Honeywell
Milgo Electronic Corporation
3M Company
Modern Data Systems
Mohawk Data Science Corp.
Motorola Instrumentation & Control Inc.

The National Cash Register Co.
Nissei Sangyo Co., Ltd.

Olivetti Underwood Corp.
Omnitec Corp., A Subsidiary of Nytronics, Inc.

Peripheral Equipment Corp.
Potter Instrument Company, Inc.
Prentice Hall, Inc.

RCA Electronic Components
RCA Information Systems
Raytheon Computer
Rixon Electronics, Inc.

Sanders Associates, Inc.
Sangamo Information Systems
Scientific Control Corp.
Scientific Data Systems
Semicon Computer Systems
Soroban Engineering, Inc.
Spartan Books
Stromberg Datagraphics, Inc.
Systems Engineering Laboratories, Inc.
Systron-Donner Corporation
Systronics, Inc.

Tektronix, Inc.
Teletype Corporation
Telex Computer Products Division
Texas Instruments Incorporated/Components Group
Transistor Electronics Corporation
Tri-Data Corporation
Tymshare, Inc.

UNIVAC, Division, Sperry Rand Corp.
URS Systems Corporation
U.S. Magnetic Tape

Varian Data Machines
Vermont Research Corporation
Viatron Computer Systems Corporation

Wang Laboratories, Inc.
John Wiley & Sons, Inc.
Wyle Laboratories, Inc.

Xerox Corporation

AUTHOR INDEX