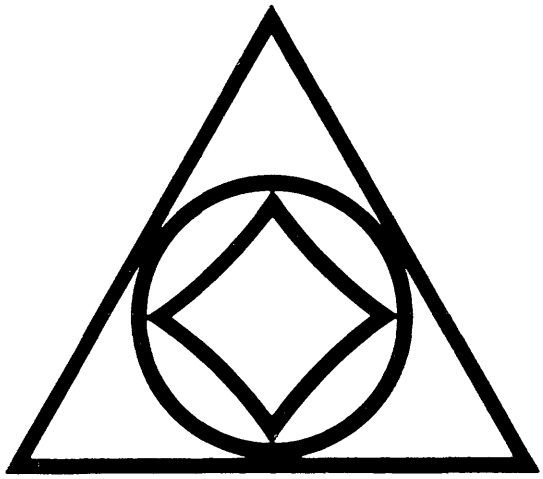
AFIPS

AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES



PROCEEDINGS 1962
SPRING JOINT COMPUTER CONFERENCE

SAN FRANCISCO, CALIFORNIA • MAY 1-3, 1962 • VOL. 21

ADDITIONAL COPIES

Additional copies of the Proceedings may be purchased from the printer at $6.00 per copy and will be sent postpaid if order is accompanied by remittance.

Orders should be sent to:

The National Press
850 Hansen Way
Palo Alto, California

DISCLAIMER

The ideas and opinions expressed herein are solely those of the authors, and are not necessarily representative of, or endorsed by, the 1962 Spring Joint Computer Conference Committee or the American Federation of Information Processing Societies.

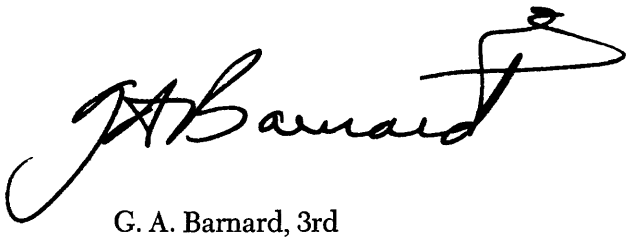
Manufactured in the U.S.A. by The National Press, Palo Alto, California

PREFACE

This volume embodies the substance of technical presentations made at the first of two major computer conferences to be held this year. Taking a name which designates when it took place, rather than its geographic location, the 1962 Spring Joint Computer Conference heralds recognition that these meetings are nationwide in scope, no longer mere regional conventions. AFIPS continues the tradition of the former Western Joint Computer Conferences by bringing together from all parts of the country men of talent who can provoke and inspire us to meet the challenges which face our industry today.

Each paper presented has been selected for the unique contribution which its author can make toward satisfying our industry's appetite for new ideas. Rather than catering to any one specific theme, this program touches the highlights of new developments, points out trends, summarizes progress, and orients the computer field with other disciplines.

We trust that this book will meet the need for which it was designed: to serve as a valuable source of reference to a collection of technical literature for the computer industry.



G. A. Barnard, 3rd
Chairman
1962 Spring Joint Computer Conference

AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES

AFIPS came into being as an unincorporated society of societies on May 10, 1961. AFIPS is a direct outgrowth of the National Joint Computer Committee (NJCC). The latter organization was formed in 1951 for the sole purpose of sponsoring and coordinating the Joint Computer Conferences, Eastern and Western. The same organizations which comprised the NJCC are the founding societies of AFIPS:

The American Institute of Electrical Engineers (AIEE)

The Association for Computing Machinery (ACM)

The Institute of Radio Engineers (IRE)

Officers, Directors, and Committee Chairmen of AFIPS are:

CHAIRMAN GOVERNING BOARD: Dr. Willis H. Ware, The RAND Corporation, 1700 Main Street, Santa Monica, California

EXECUTIVE DIRECTORS: R. A. Imm—AIEE; **Dr. H. D. Huskey**—ACM; **Dr. A. A. Cohen**—IRE

SECRETARY: Miss Margaret R. Fox, National Bureau of Standards, Data Processing Systems Div., Washington 25, D.C.

TREASURER: Frank E. Heart, Lincoln Laboratory, Room B-283, P.O. Box 73, Lexington 73, Massachusetts

AIEE DIRECTORS

R. A. Imm

IBM Corporation
Dept. 550, Bldg., 006-1
3605 Highway 52 North
Rochester, Minnesota

R. S. Gardner

American Institute of
Electrical Engineers
345 East 47th Street
New York 17, New York

Claude A. R. Kagan

Western Electric Company
P.O. Box 900
Princeton, New Jersey

Dr. Morris Rubinoff

517 Anthwyn Road
Merion Station, Pennsylvania

ACM DIRECTORS

Walter M. Carlson

Mechanical Research Laboratory
E. I. duPont DeNemours & Co.
101 Beech Street
Wilmington 98, Delaware

Dr. Bruce Gilchrist

IBM Corporation
590 Madison Avenue
New York 22, New York

Dr. Harry D. Huskey

Dept. of Mathematics
University of California
Berkeley 4, California

J. D. Madden

System Development
Corporation
2500 Colorado Avenue
Santa Monica, California

IRE DIRECTORS

Dr. Werner Buchholz
IBM Corporation
South Road Laboratory
Poughkeepsie, New York

Dr. Arnold A. Cohen

Remington Rand Univac
Univac Park
St. Paul 16, Minnesota

Frank E. Heart

Lincoln Laboratory, Room B-283
P.O. Box 73
Lexington 73, Massachusetts

Harry T. Larson

Aeronutronic, Division of
Ford Motor Company
P.O. Box 486
Newport Beach, California

AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES (Continued)

ADMISSIONS COMMITTEE

Dr. Bruce Gilchrist
IBM Corporation
590 Madison Avenue
New York 22, New York

AWARDS AND PRIZES COMMITTEE

Claude A. R. Kagan
Western Electric Company
P.O. Box 900
Princeton, New Jersey

BY-LAWS AND CONSTITUTION COMMITTEE

Walter M. Carlson
Mechanical Research Laboratory
E. I. duPont deNemours & Co.
101 Beech Street
Wilmington 98, Delaware

CONFERENCE COMMITTEE

Keith W. Uncapher
The RAND Corporation
1700 Main Street
Santa Monica, California

FINANCE COMMITTEE

Dr. Morton M. Astrahan
IBM Research Laboratory
Monterey and Cottle Roads
San Jose 14, California

COMMITTEE ON NEW ACTIVITIES

Dr. Morris Rubinoff
517 Anthwyn Road
Merion Station, Pennsylvania

PUBLIC RELATIONS COMMITTEE

J. D. Madden
System Development Corporation
2500 Colorado Avenue
Santa Monica, California

PUBLICATIONS COMMITTEE

Dr. Werner Buchholz
IBM Corporation
South Road Laboratory
Poughkeepsie, New York

SOCIAL IMPLICATIONS OF INFORMATION PROCESSING TECHNOLOGY COMMITTEE

Harry T. Larson
Aeronutronic, Division of
Ford Motor Company
P.O. Box 486
Newport Beach, California

AFIPS REPRESENTATIVE TO IFIPS

I. L. Auerbach
Auerbach Electronics Corporation
1634 Arch Street
Philadelphia 3, Pennsylvania

1962 SPRING JOINT COMPUTER CONFERENCE COMMITTEE

CHAIRMAN: G. A. Barnard, Philco WDL, Palo Alto, Calif.

VICE CHAIRMAN: Dr. H. D. Crane, SRI, Menlo Park, Calif.

SECRETARY-TREASURER: R. A. Isaacs, Philco WDL, Palo Alto, Calif.
L. J. Borrego, Philco WDL, Palo Alto, Calif.

TECHNICAL PROGRAM

Chairman: **Dr. R. I. Tanaka,** Lockheed, Palo Alto, Calif.
Vice Chairman: **R. C. Minnick,** SRI, Menlo Park, Calif.
Assoc. Chairman: **J. E. Sherman,** Lockheed, Palo Alto, Calif.

SESSION CHAIRMEN:

F. M. Tonge, Stanford University, Calif.
R. A. Kirsch, Nat'l Bureau of Standards, Wash., D.C.
J. I. Raffel, Lincoln Laboratory, Lexington, Mass.
D. C. Engelbart, SRI, Menlo Park, Calif.
B. G. Farley, Lincoln Laboratory, Lexington, Mass.
J. H. Pomerene, IBM, Yorktown Heights, N.Y.
V. L. Larrowe, University of Michigan
Jack Goldberg, SRI, Menlo Park, Calif.
B. A. Galler, University of Michigan
Louis Fein, Consultant, Palo Alto, Calif.
H. K. Skramstad, Naval Ordnance Lab, Corona, Calif.
R. M. Bennett, IBM, San Jose, Calif.
J. P. Runyon, Bell Telephone Lab, Murray Hill, N.J.

EXHIBITS

Chairman: **J. W. Ball,** Pacific Telephone, Sacramento, Calif.
Vice Chairman: **Art Scholar,** Pacific Telephone, San Francisco, Calif.

LOCAL ARRANGEMENTS

Chairman: **R. G. Glaser,** McKinsey & Co., San Francisco, Calif.
Vice Chairman: **R. D. Smith,** Smith-Corona Marchant, Oakland, Calif.
G. W. Mills, Burroughs, San Francisco, Calif.
H. G. Asmus, General Electric, Phoenix, Ariz.
D. C. Hays, General Electric, San Francisco, Calif.

REGISTRATION

Chairman: **D. E. Eliezer,** IBM, San Jose, Calif.
Vice Chairman: **Leo Rinsler,** IBM, San Jose, Calif.
R. A. Shaw, IBM, San Jose, Calif.

PRINTING AND MAILING

Chairman: **W. O. Hamlin,** Fairchild, Mountain View, Calif.
Vice Chairman: **R. J. Johnston,** Fairchild, Mountain View, Calif.

PUBLICATIONS

Chairman: **E. T. Lincoln,** IBM, San Jose, Calif.
Vice Chairman: **J. J. McNulty,** IBM, San Jose, Calif.

PUBLIC RELATIONS

Chairman: **N. S. Jones,** Friden, San Leandro, Calif.
Vice Chairman: **R. D. Painter,** Fairchild, Mountain View, Calif.

EXHIBITORS' PROGRAM

Chairman: **D. C. Lincicome,** SRI, Menlo Park, Calif.

LADIES' ACTIVITIES

Chairman: **Margaret G. Conley,** IBM, San Francisco, Calif.
Vice Chairman: **Anne Niemi,** IBM, San Francisco, Calif.
Joy Gallagher, IBM, San Francisco, Calif.
Barbara Taylor, IBM, San Francisco, Calif.

SPECIAL EDUCATION PROGRAM

Chairman: **R. J. Andrews,** IBM, San Jose, Calif.
Vice Chairman: **William Gerkin,** Rem. Rand, San Francisco, Calif.
Bryce Ells, IBM, San Jose, Calif.
A. J. Bodine, IBM, San Jose, Calif.
E. W. Means, Rem. Rand, San Francisco, Calif.

CONSULTANTS

Exhibits: J. L. Whitlock Associates, Oakton, Va.
Public Relations: W. C. Estler, Palo Alto, Calif.

REVIEWERS

AFIPS and the Conference sincerely appreciate the excellent job done by the Reviewers. Their careful study and comments have been vital to maintaining the high standards of quality for this collection of papers.

S. Amarel
W. L. Anderson
M. M. Astrahan
D. C. Augustin
J. A. Baker
R. M. Barnett
R. C. Baron
D. E. Beecher
G. A. Bekey
R. M. Bennett
D. F. Berg
J. E. Bertram
E. Bloch
E. K. Blum
E. V. Bohn
A. Bridgman
W. Buchholz
T. H. Bullock
F. Carr
A. B. Clymer
M. Cohn
L. J. Craig
T. H. Crowley
E. C. DeLand
J. F. Dirac
C. B. Dowling
J. Earle
R. D. Elbourn
G. F. Emch
D. C. Engelbart
G. W. Evans
B. G. Farley
I. Flores
S. Frankel
W. S. Fujitsubo
B. A. Galler
H. L. Garner
J. P. Gibbons
B. Gilchrist
M. C. Gilliland
E. L. Glaser
R. C. Gold
E. A. Goldberg
J. Goldberg
M. H. Goldstein
E. Goto
G. R. Grado
D. Greenberg
J. Griffith
D. Haagens
D. W. Hagelbarger
J. K. Hawkins
L. D. Healy
P. J. Hermann
T. R. Herndon
T. Higgins
W. H. Highleyman
M. E. Homan
R. D. Horwitz
R. M. Howe
K. Iverson
A. S. Jackson
G. T. Jacobi
S. Jamison
D. B. Jordan
J. A. Joseph
L. Kanal
W. J. Karplus
W. Kindle
R. A. Kirsch
R. H. Kohr
G. A. Korn
L. D. Kovach
R. A. Kudlich
V. L. Larrowe
R. S. Ledley
C. Y. Lee
C. T. Leondes
B. M. Levin
M. H. Lewin
J. D. Little
B. Loveman
J. Lyman
O. L. MacSorley
J. Macy, Jr.
J. Mangnall
W. Marggraf
F. N. Marzocco
M. S. Mason
M. S. Mayzner
E. J. McCluskey
M. E. McCoy
R. M. Meade
R. Meagher
R. E. Miller
R. C. Minnick
M. Morgan
T. H. Mott, Jr.
P. G. Neumann
L. W. Neustadt
C. J. Nisson
H. Nonken
A. B. Novikoff
M. Paskman
G. T. Paul
A. Pohm
J. H. Pomerene
C. R. Porter
T. F. Potts
J. I. Raffel
M. J. Relis
S. Rogers
H. Rosenberg
A. I. Rubin
M. Rubinoff
B. W. Rudin
J. P. Runyon
T. T. Sandel
C. M. Schoenberg
N. R. Scott
F. F. Sellers
W. L. Semon
N. Z. Shapiro
Q. W. Simkins
H. K. Skramstad
W. R. Slack
A. Slade
K. H. Speierman
R. I. Tanaka
H. M. Teager
D. Teichroew
L. E. Thibodeau
R. B. Thomas
R. M. Tillman
F. M. Tonge
K. B. Tuttle
G. Vandling
R. L. Van Horn
H. R. Van Zoeren
L. M. Warshawsky
G. P. Weeg
R. R. Wheeler
H. K. Wild
R. O. Winder
C. H. Wolff

This list was compiled of all reviewers registered as of the date this book went to press.

EXHIBITORS

This list was compiled as of the date this book went to press.

Aeronutronic Division of Ford Motor Co.
Newport Beach, Calif.

AMP, Inc.
Harrisburg, Penna.

Ampex Corp.
Redwood City, Calif.

ANelex Corp.
Boston, Mass.

Applied Dynamics, Inc.
Ann Arbor, Mich.

Automatic Electric Sales Corp.
Northlake, Ill.

Bell Telephone System, Pac. Tel. Co. and
Long Lines Dept.
San Francisco, Calif.

The Bendix Corp., Bendix Computer Div.
Los Angeles, Calif.

Berkeley Div. of Beckman Instruments
Richmond, Calif.

Brush Instruments Div. of Clevite Corp.
Cleveland, Ohio

Bryant Computer Products
Walled Lake, Mich.

The Bureau of National Affairs, Inc.
Washington, D.C.

Burroughs Corp.
Detroit, Mich.

California Computer Products, Inc.
Downey, Calif.

C-E-I-R, Inc.
New York, N.Y.

Collins Radio Co.
Dallas, Texas

Comcor, Inc.
Denver, Colo.

Computer Control Co., Inc.
Los Angeles, Calif.

Computer Systems, Inc.
Monmouth Junction, N.J.

Consolidated Electrodynamics Corp.
Pasadena, Calif.

Control Data Corp.
Minneapolis, Minn.

Datamation
New York, N.Y.

Data Products Corp.
Beverly Hills, Calif.

Datapulse, Inc.
Inglewood, Calif.

DI/AN Controls, Inc.
Dorchester, Mass.

Digital Equipment Corp.
Maynard, Mass.

Digitronics Corp.
New York, N.Y.

DYMEC Div. of Hewlett-Packard Co.
Palo Alto, Calif.

Electro Instruments, Inc.
Sunnyvale, Calif.

Electronic Associates, Inc.
Long Branch, N.J.

Electronic Engineering Co. of Calif.
Santa Ana, Calif.

Electronic Memories, Inc.
Los Angeles, Calif.

Engineered Electronics Co.
Santa Ana, Calif.

Epsco, Inc.
Cambridge, Mass.

Fabri-Tek, Inc.
Amery, Wis.

Fairchild Semiconductor
Mountain View, Calif.

Ferranti Electric, Inc.
Plainview, L.I., N.Y.

Friden, Inc.
San Leandro, Calif.

Gen. Dynamics/Electronics Information Tech. Div.
San Diego, Calif.

EXHIBITORS (Continued)

General Electric Co., Computer Dept.
Phoenix, Ariz.

The Hallicrafters Co.
Berwyn, Ill.

Indiana General Corp., Electronics Div.,
Memory Products Dept.
Keasbey, N.J.

Informatic, Inc.
Beverly Hills, Calif.

International Business Machines Corp.
New York, N.Y.

International Telephone & Telegraph Corp.
New York, N.Y.

Invac Corp.
Natick, Mass.

Kearfott Div., General Precision, Inc.
Little Falls, N.J.

Laboratory for Electronics, Inc.
Boston, Mass.

Litton Systems, Inc.
Beverly Hills, Calif.

McGraw-Hill Book Co., Inc.
Corte Madera, Calif.

Memorex Corp.
Santa Clara, Calif.

Moxon Electronics Corp.
Beverly Hills, Calif.

The National Cash Register Co.
Dayton, Ohio

North American Aviation, Inc.
Los Angeles, Calif.

Omnitronics, Inc.
Philadelphia, Penna.

Packard Bell Computer Corp.
Los Angeles, Calif.

Philco Corp. Computer Div.
Philadelphia, Penna.

Photocircuits Corp.
Glen Cove, N.Y.

Potter Instrument Company, Inc.
Plainview, L.I., N.Y.

Radio Corp. of America,
Semiconductor and Materials Div.
Somerville, N.Y.

Raytheon Co.
Waltham, Mass.

Remington Rand UNIVAC
New York, N.Y.

Rese Engineering, Inc.
Philadelphia, Penna.

Rheem Electronics Div. of Rheem Mfg. Co.
Los Angeles, Calif.

Rotron Manufacturing Co., Inc.
Woodstock, N.Y.

Royal McBee Corp.
New York, N.Y.

Scientific Data Systems, Inc.
Santa Monica, Calif.

The Service Bureau Corp.
New York, N.Y.

Soroban Engineering, Inc.
Melbourne, Fla.

Tally Register Corp.
Seattle, Wash.

Teletype Corp.
Skokie, Ill.

John Wiley & Sons, Inc.
New York, N.Y.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TOWARD A GENERAL SIMULATION CAPABILITY

Michael R. Lackner

System Development Corporation

Santa Monica, California

Summary

Simulation of a system by digital computer requires:

- A model of the system which is intelligible to the student of the system while compatible with the limitations of the computer.
- Translation of the model to computer code.
- Movement of the model through time.
- Recording the performance of the model.

SIMPAC, a "simulation package," incorporates coherent techniques and devices for the accomplishment of these objectives: modeling concepts for building a computer-compatible model, a vocabulary for encoding the model, a computer program for moving the model through time and recording its performance, and an output presentation program.

A model of a hypothetical business system has been implemented with the first version of SIMPAC for the purpose of studying management controls in a complex system.

This paper discusses digital simulation and SIMPAC and introduces modeling concepts which may lead to a set of simulation systems, called 'Muse', which would assemble models of varying complexity from descriptive statements and analyze the models prior to simulation.

Introduction

The fundamental attraction of digital simulation to the analyst or designer of systems is that a dynamic model can be constructed whose behavior can be studied. Using traditional techniques of analysis, particular questions are asked and particular answers obtained. With a simulation model, a student of a system may see much more than end results of individual calculations. He may examine how things happen as well as what things happen. Problems too difficult to phrase with traditional techniques may be studied with simulation. Simple relationships may be assembled to form a model which defies traditional statement but which runs on a computer and produces a time series for every variable.

Analysis and design have traditionally involved the use of models, such as architects plans and specifications. The first dynamic models were physical analogs; e.g., scale models. A simulation model must represent a dynamic situation. Actions as well as entities must be represented.

Digital computers permit the implementation of dynamic, logical models; system models may be made machine interpretable and may be implemented by machines. A simulation model for a digital computer may represent entities with descriptive data and actions with algorithms -- series of individual arithmetic-logical operations upon the data. A computer is well suited for the elaborate bookkeeping involved in moving such models through time.

Modeling and implementation have been time consuming and expensive. Several attempts have been and are being made to reduce time and expense of implementation while preserving the inherent versatility of the digital computer. This paper does not survey or evaluate these various simulation schemes but describes one such effort at System Development Corporation.

The paper is in three parts; an introductory discussion of digital simulation; second, a description of SIMPAC, short for 'simulation package'; third, an outline of Muse, a set of simulation systems based upon general modeling concepts.

Part 1: Discussion of Digital Simulation

In a digital simulation model, the activity of consuming gasoline is representable by a subtractive operation upon a variable measuring gasoline; spatial movement is representable by modification of three variables measuring placement in three dimensional space. The inputs to these algorithms are the influential factors in their environments, and the outputs of the algorithms are measures of their state or the state of their affected surroundings. The inputs to the gasoline consuming algorithm might be altitude, temperature, humidity, etc. Variables affecting movement would be input to the spatial movement algorithm.

A model is a caricature. Certain phenomena are grossly exaggerated in representation, while others are ignored. The modeler, his purpose of modeling in mind, attempts to characterize phenomena, including those things he judges important, excluding those he judges insignificant. Minutiae are everywhere aggregated. In a digital simulation model, atomic attributes of phenomena are summed in a single datum, atomic actions summed in a single operation.

Stochastic processes are examples of gross aggregation. A modeler may wish to exclude from the model, or may not fully understand, the intricacies giving rise to a particular distribution of values for some variable. For example, he may use

a random device to select a sample weight from an observed distribution of weights.

Stochastic processes, however, should be reserved for use when there is no traceable relationship between a specific condition in the model and the results of the stochastic process. For instance, if air temperature is known to be directly related to the time of day, and the latter is represented in the model, random selection of a value of air temperature is poor practice. On the other hand, if temperature varies and no one knows why or when, but simply within what limits, it may be proper to rely upon a stochastic process for selecting temperature.

The Time Problem

In a real system, interaction of components may be discrete or continuous. (Gasoline consumption and spatial movement are both examples of continuous activity.) A digital device precludes continuous activity; in a simulation model for a digital computer, all interaction must be discrete.

Thus, the dynamic character of the simulation model must fit the dynamics of the computer. To reflect simultaneous, continuous activities, provision must be made in the model for the computer, a machine incapable of such activity, to continually move these modeled activities to the same point in time by a series of discrete actions.

Moving the Model Through Time

A cycling routine, continually performing algorithm after algorithm, may move each component forward in time by an equal increment during each cycle. At the end of each cycle, then, all components of the simulation model are at the same place in time. Midway in any cycle, half of the components are a full time increment ahead of the other half. The choice of the time increment is obviously important. In some models, the choice is determined by the duration of the "fastest" activity because the nature of the model is such that once an activity is begun it must be completed. If the model provides for the existence of a "busy" or "in process" state for activities, however, the choice of the basic time increment may be independent of their particular durations. Such provision also avoids problems associated with nonmultiple durations. For example, if activity A requires three hours to perform (in the real world), and activity B requires two hours, the algorithms for these activities cannot be performed every one, two, or three hours to yield meaningful results. Resort must be made to some device such as expressing activities as pairs of algorithms (begin, complete--for example), so that upon the addition of any increment of time it may be shown that an activity has been performed the proper number of times and is left in a state of being performed or not.

The time increment may or may not be variable from run to run, or even from cycle to cycle. (This capability is again dependent upon the provision in the model of a "busy" state for activities.) In addition, all components of the model may be moved forward by the same increment, or some by different increments than others. This may be desirable in terms of computer time when points and times of interactions between major components are few and seldom.

Finally the time increment may be a function of the state of the model and may be computed each cycle. The time increment might be related to events. Each time an event occurs it may be possible to determine the time(s) of consequent event(s). These times may be tabled and the model, or portions thereof, moved to the time indicated by the earliest event in the table. This method can lead to a great many "cycles" in a complex model and may require extensive recomputation of event times and consequent correction of table entries.

Measures of Performance

The momentary state of a model may be determined from the states of its variables. Some of these may be final measures of performance; that is, some of the algorithms performed as the model moves forward through time may not be part of the representation of the real phenomena at all, but analytical routines designed to measure performance. Since these routines consume computer storage--at a premium in simulation--and their source information can be recorded for later analysis, they are usually run after simulation--as another phase of the program, for example.

The availability of particular measures of performance is determined by the availability of source information. What is specific in the model may be specific as output. To some extent, each variable in the model is a measure of performance. Measurement of performance becomes less dependent upon inference as the number of specific variables increases.

The normal form of output from a simulation model is a time series. Models normally provide a limited choice of format for a time series or statistics obtainable from time series.

Implementation

There are several means of implementing a simulation model; by developing completely new simulation techniques and methodology toward the construction of a system specific model, by specifying parameter values for a "general-purpose model", or by choosing an intermediate approach.

A general-purpose simulation model presents a modeler with a set of preprogrammed components. He specifies their configuration and parameter values to produce a model. The mechanics of computer implementation with general-purpose models may be simple. With some, no programming is involved. However, the system analysis precedent to choosing which of a limited set of formulae best describes activity X may be monumental or

practically impossible. A running computer program may be produced very quickly, but the model may represent the real phenomena in only general terms.

Simulation Languages

Another approach, providing much more flexibility, is to present a modeler with a simulation-oriented language for constructing model components. A simulation language is the result of an attempt to isolate common simulation functions, provide fitting capabilities, and so accelerate the implementation of a model. A Weltansicht must be at least implicitly established to permit the construction of a simulation language. The language, based upon this particular view of the world, contains expressions for the kinds of phenomena contemplated. Expressions must have the same meaning in algorithms expressive of quite diverse real activities. A comprehensive, system specific model may be obtained with a simulation language and the model may be as modular as its designers wish it to be. However, the language itself provides no means of moving the model through time, recording its performance, or other simulation-common chores.

The coupling of a program to a simulation language is a further step toward speeding implementation of a system specific model. The program provides a basis for the model to be built with the simulation language, and such services as movement through time, recording of performance, initialization, etc. By this means a sophisticated system specific model may be constructed without expenditure of effort or those problems common to the implementation of all such models.

Modular Construction and Implementation of a Simulation Model

Systems are subject to change, and system models must be easily modified. Ease of modification is obtainable only through modularity.

Fortunately, whatever is said about modularity is said in its favor. The requirements placed upon the structure of the model, system analysis, and programming are desirable.

Part 2: SIMPAC

Background

For the purpose of studying management controls in a complex system, the Management Control Systems Project at System Development Corporation hypothesized and described an entire business system, a manufacturing concern, to be simulated on a digital computer. The Mark I business system model has been implemented with the first version of SIMPAC, which was developed and checked out in parallel with the construction of the first modules of the model.

The problem of implementing a model of such a total system on a digital computer required the development of a method of system analysis capable of rendering the system in a form translatable to a digital simulation model and a means of implementing the model. These tools comprise a "simulation package" called SIMPAC.

This first version of SIMPAC[1] differs in several respects from the version described in this paper. Most important of these differences is the treatment of what was called "reference information" in the earlier version. It was recognized then that all information in the model could be handled as "object information," i.e., contained in specific transient records, but it was felt that a network might become needlessly intricate if all information were so exchanged. (Some savings in running time were also involved.) Therefore, a means of aggregating information exchange operations had been provided in the form of "reference files." Elimination of these reference files does away with several compromises and clarifies the operation of the system by requiring more explicit treatment of certain model components, such as activity performers, previously handled by the reference files.

SIMPAC

The implementation of a digital simulation study is typically achieved in four sequential steps:

- Model construction,
- Model translation to computer code,
- Model performance,
- Performance evaluation.

SIMPAC offers techniques and devices for the accomplishment of these objectives. They are described in the order in which they occur: modeling concepts; a vocabulary for encoding the model; a computer program for moving the model through time; and an output presentation program.

Modeling Concepts

The modeling concepts are derived from the problem of representing an aggregation of diverse activities as an information-processing system--something which can be implemented with a digital computer.

These observations can be made of many systems:

- The system contains recurring activities which interact through exchange of information.

---

[1] M. R. Lackner, SIMPAC: A research tool for simulation (SDC document SP-228). Santa Monica, Calif.: System Development Corporation, March 13, 1961. 22 pp.

- Activities are performed at different times or at different rates.
- Buffers capable of storing information exist between activities.
- A discrete performance of an activity takes place when:
  a) people or machines capable of performing the activity are assigned it, and
  b) information required for performance is accessible.
- The duration of a discrete performance of an activity is a function of the quality and quantity of people and machines performing.
- Records carrying transient information generated in the discrete performances of activities tend to form queues between activities performed at different times or at different rates.
- There are more activities performed than there are people or machines who perform them.

The controls to which these systems are sensitive are:

- Queue discipline: dictation of the order of transient records in queues, and the mode of selecting them;
- Resource allocation: assignment of activity performers to various activities they can perform; and
- Information routing: specification of the source of information input to an activity and the destination of output information.

The basic modeling concepts are those of activity, activity performer, transient information record, and queue. A caricature of the system drawn upon these concepts places the system in an information-processing context. Activities may be defined as series of arithmetic-logical operations--algorithms. Activity performers may be identified by quantitative and logical descriptors, and transient information records contain like descriptors by definition. Such a model is relatable to the phenomena under study and conveniently translated to computer code.

In conducting this type of modeling:

- Potential queues are identified,
- Transient information in the system is identified,
- Activities are separated from the people or machines who perform them,
- Control activity is identified and characterized,
- People and machines are measured in terms of their ability to perform activities.

An activity is performed if

- Activity performers are assigned it. These are the people or machines who can perform the activity. They are assigned by other activities (which are performed if..., etc.).
- The transient information records required for its performance are in its input buffers. These records represent the information or physical entities required for performance.

Activities in the system may exert direct or indirect control over other activities or merely exchange information with them.

They may assign activity performers to and from other activities, modify the queue disciplines used by other activities when securing or storing transient information, or they may direct the flow of information by specifying input and output buffers for other activities.

An Example of SIMPAC Modeling. Assume it is desired to simulate the operation of a small library. This may be begun by describing the operation of this system. As it is described, the activities, transient information, activity performers, and buffers where queues may form, will be underlined. The description may begin as a catalog of observations.

> Books in the return bin are stored by clerks in the stacks.
>
> Books from the ready bin are read by visitors and placed in the return bin.
>
> Visitors generate requests for books and place them in the request bin.
>
> Librarians examine requests for books from the request bin and place them in the find bin.
>
> Clerks match requests against books in stacks and either put book in ready bin or put requests in out bin.

A flow chart may be constructed at this point which fixes the relationships of the components thus far identified (see figure 1).

Thus far, the flow chart shows no control activities, that is no activities directing the flow of information, modifying queue disciplines, or allocating performers among activities. A two-dimensional flow chart is limited in application to fairly stable component relationships. At some point in modeling, the transient information flow

chart becomes incapable of rendering component relationships intelligibly, although flow charts continue to be of value where relationships are relatively stable; e.g., in the internal functions of an activity.

The transient information flow chart must be read as consisting of the routes of information intersecting the processes performed upon the information, and the buffers where the information reposes when not being acted upon. The fact that performance of a process is contingent upon allocation of activity performers and availability of required information must be borne in mind. If the routing of information is not fixed, the transient information flow chart may become burdened with contingencies to the extent that its utility disappears.

The following control activity might be one added to the library model:

> Librarian assigns clerks to storing books if queue in return bin is longer than queue in the find bin, or to matching requests if queue in find bin is longer than queue in return bin.

Note that here activities and activity performers, as well as queue content are treated as transient information.

A number of additional control activities are necessary to move the librarian from activity to activity, get her and the clerks to work in the morning, etc.

Each of the activities must be described by an algorithm. The composition of the algorithm will identify information exchanged with other activities which must be contained in the records passed between them. Information used in control activities may be obtained from control blocks associated with the various activities, activity performers, and queues (buffers).

The modeling of a system with SIMPAC requires the unambiguous statement of much information. Consider the activity, "Generate requests for books." How long does it take a visitor to generate a request? Is this time constant or is there a distribution of such times? Does each visitor generate one request, or more?

The model of generation activity may include sampling from Poisson, normal, or rectangular distributions for determining the number of requests to generate during any time period, or the content of any of the fields in the transient records generated; e.g., what book.

Very likely the process will be stochastic, but it need not be. The particular requests might be input to the model as exogenous events, if so, actual history might be utilized in defining these events.

There are no simple rules for going about this. Each of the activities is a highly aggregated representation of a portion of the system. The completed model is a caricature of the system, and each component of the model a caricaturistic sum of myriad phenomena. Modeling serves to clarify understanding of the system and indicate areas of insufficient knowledge. It is the purpose of SIMPAC to provide building blocks for expressing the model and a framework on which to rest them.

Translation of the Model to Computer Code

Once a model has been specified in terms of activities, performers, transient information records, and queues (or buffers), it is necessary to translate the model to machine language--instructions and data.

SIMPAC provides, in the form of macro definitions, a simulation-oriented extension of the normal symbolic instruction set. Expressions for entities like queues, and for operations upon them, form this extension. For example, 'QTAKD A,B' is an expression for: 'select a record from queue B in accordance with the queue discipline found at A'. 'GETBL A' for: 'secure a free block of storage from list A'. 'MCNRN A,B' for: 'generate a sample drawn from a normal distribution, mean A, standard deviation B'. 'X ACMAC A,B,C,D,E' for: 'there is an activity X, which services transient information records in queue A, selecting them according to the prevailing discipline found at B, and requiring the expenditure of an amount of service time equal to C times the content of record field D before placing the record in queue E'.

All activities are modeled as pairs of algorithms: A set-up phase in which information necessary for performance is gathered and a resource time of performance computed, and a completion phase in which the activity itself is performed. The activity is considered "in process" between the performance of set-up and completion, and the time between is a function of activity performer allocation.

Control Blocks. Each component of the model has a control block.

The activity control block contains:

- The addresses of the set-up and completion phases of the algorithm expressing the activity.
- Variables which indicate the performer time necessary for the current performance and the performer time thus far spent upon performance.
- A busy:not-busy indicator to show an in-process state of the activity.
- The minimum, maximum, and current performer allocations.

- Counters of the number of times the activity has been performed, of the amount of performer time expended upon performance, and of the amount of performer time spent on idle for lack of fulfillment of necessary conditions.

- The addresses of the activity's input and output queues.

The activity performer control block contains:

- A count of currently available performers. The unit of this count is a function of their separability.

- A count of the number of activities to which these performers are assignable.

- A list of assignable activities wherein a block for each activity indicates its activity control block address, the address of an algorithm for determining the effective value of assigned performers, and the current allocation.

- A count of performers assigned to 'other' activities than those in the list.

Transient information records may represent bearers of information or physical entities. The record is made up of a number of logical fields sufficient to identify the entity in the context of the system. A particular format may serve for many records, and occasionally the same record may appear in several queues. The control block identifies the format of the record.

The queue control block indicates:

- The current count of waiting records.

- A list of these records.

- An indication of the order in which the records are held.

The ordered mix of SIMPAC macro expressions and symbolic expressions is merged with the body of the SIMPAC program and compiled by a general-purpose compiler. The program produced is the SIMPAC program supplemented by the particular model. See figure 2.

Moving the Model Through Time

The basic SIMPAC program contains modules for effecting operations called out by various macro expressions and routines which handle the mechanics of moving the model through time and recording its performance.

The model is moved through time by a cycling routine which steps each activity by a fixed increment of time. The increment is variable from one run to another, however, and the scheme is adaptable to varying the increment from cycle to cycle as a function of predicted events.

Each cycle the time increment is multiplied by the performer allocation of the activity to produce an amount of performer-time. Performer-time is then applied to the activity, resulting in partial or multiple performances, or accumulation of idle performer-time.

The computer code necessary to perform certain common operations is contained in a coherent group of over sixty subroutines. The performance of a given service may require the operation of several subroutines.

Each of the SIMPAC subroutines performs some particular service in one of the following areas: initialization and production of exogenous events; clock and calendar timekeeping; delayed transmission of records; queue manipulation; associative storage control; raw data recording; data analysis and output presentation; distribution sampling; miscellaneous, including table-look-up, square root, etc.

Familiarity with the routines is not required of the user, merely familiarity with the macro instructions and their usage.

(Each subroutine in the SIMPAC program is consistent with conventions of modularity so that portions of the program, and the associated macro expressions, may be useful in the implementation of information-processing systems unrelated to simulation: e.g., the associative storage mechanisms.

It is conceivable that the simulation of a planned information-processing system might result in the development in the model of activity algorithms directly transferable to the actual system.)

SIMPAC Associative Storage. The queue control block contains the addresses of the first and last queue words, each of which contains a referent address of a transient record and the address of the following queue word. Records contain no link to other records in the same queue. This indirect catenation permits multiple concatenations of records without incurring redundancy of the information contained and avoids the physical movement of records (see figure 3).

Queue words are each one word of computer storage. Transient information records vary in length with the number and size of logical fields contained and are conjunctive.

SIMPAC contains a device to enable the sharing of available storage by queue words and transaction records, thus eliminating arbitrary limits upon queue lengths. The device is as follows:

- An initial associative allocation of 1, 2, 3,...n word blocks is made to push-down/pop-up lists; n is a multiple of all smaller blocks, and most available storage is allocated to n word blocks.

- Free blocks are made available and returned by macro instructions.
- Automatic communication with a storage banker routine is established when a control word of a push-down/pop-up list indicates no more blocks are available.
- The storage banker diverts n word blocks to 1, 2, 3,... word blocks as needed, or, in a second mode (when n word blocks are exhausted) builds larger blocks from available conjunctive smaller blocks.

This operation consumes little computer time (with the exception of mode 2 of the storage banker which involves tape movement) and provides great flexibility of storage utilization.

Although records in a queue are not conjunctive, they are recorded on tape as a continuous record.

Output Presentation

Before a simulation run is made, data recording parameters are fixed, which determine the raw data which will be recorded as the model is moved through time.

Available outputs are limited by the fund of raw data made available by the recording routines. The data which may be recorded are the content of the three control blocks (queue, performer, and activity) and the content of the transient records in the queues.

After the simulation run, output listings are produced in accordance with list requests phrased with a small vocabulary of macro expressions. The output phase of the program may be rerun as often as desired, producing whatever listings might be of interest.

Sample listings are shown in figure 4.[1] Time series listings of selected variables, functions of the series, and, as in the case of the listings illustrated the particular contents of queues - the discrete records, or summaries of the contents, may be obtained.

Part 3: Muse

SIMPAC is predicated upon a Weltansicht which recognizes systems as complexes of transient information, queues formed in the system's buffers by transient information records, algorithms which generate, use and destroy this information, and performers of these algorithms.

This is a fairly general "world view" and, as a result, SIMPAC is a fairly general simulation tool.

Muse is the name given to the project of developing a simulation capability more general and more efficient than SIMPAC. The name is appropriate in that simulation is a blend of art and science, and several - if not nine - simulation systems may be developed. These systems will be developed with more regard for pure theoretical considerations than was SIMPAC, whose development was greatly influenced by immediate, practical considerations. This is not to say that practical considerations will be ignored; they will be weighed, but the practical considerations will be tomorrow's rather than today's.

Before introducing the special "world view" upon which Muse is based, several terms which have been used throughout this paper are defined.

'Model' - 'A representation of an aggregate of phenomena'

'Symbol' - 'A representation of a phenomenon'

'Simulation' - 'Dynamic representation of dynamic phenomena'

'System' - 'A body of phenomena'

'Object system' - 'A system to be modeled; the system under study'

A symbol is a representation of a unity, a whole. A model is a representation of an entirety, an aggregate whole. A simulation model is a representation in time of an aggregate whole, a representation of a dynamic, aggregate whole. Mere aggregations of phenomena need not be systems but may be modeled. An aggregation of phenomena is a system if it forms an organic whole.

Weltansicht: there are things, and every thing may be from time to time, in several logical states. A logical situation is a particular pattern of certain things in certain states. Multiple occurrences of similar things in similar states may give rise to multiple occurrences of similar logical patterns and thus multiplicities of similar logical situations. Certain logical situations invite the exertion of forces which affect those things in the logical situations, destroying the situations. Thus things are created, destroyed, and placed in different states.

The following definitions are adopted from this "world view."

'Logical situation' - 'A logical pattern formed of certain things in certain states'

'Total logical situation' - 'The logical pattern formed of all things'

'Event' - 'A change in a logical situation'

---

[1] From tests of the Management Control Systems Mark I model. See Patricia McKeever, SIMPAC output phase (SDC document FN-5256). Santa Monica, Calif.: System Development Corporation, April 20, 1961. 18 pp.

'Événement' - 'A change in total logical situation'

'Unstable logical situation' - 'A l.s. which invites the exertion of a force'

'Stable logical situation' - 'A l.s. which does not invite the exertion of a force'

This definition of 'événement'[1] does not admit of simultaneous événements. There may be simultaneous events, but they produce but one change in total logical situation.

'F' - 'The class of forces'

'Q' - 'The class of logical situations'

'T' - 'The class of things'

Several simulation systems will be developed, based upon these concepts. In all models constructed within these systems, things and thing/states are represented by symbols, forces are represented by algorithms, and logical situations are represented by patterns of thing/state symbols.

S1

A loose description of the first, and basic system follows. The system, S1, is predicated on a simplified interpretation of the Weltansicht, and is inchoate at this writing.

S1 does not contemplate thing/states, only things. Furthermore, multiple occurrences of similar things are precluded. A thing is, or is not.

In later systems, multiple occurrences, thing/states, and individual characteristics of similar things will be contemplated.

In S1, the following rules and definitions hold for any object system.

1.0 $F = \{f_1, f_2, f_3, \ldots, f_\varphi\}$

'φ' - 'the number of forces; the cardinal of F'

1.1 $Q = \{q_1, q_2, q_3, \ldots, q_\chi\}$

'χ' - 'the possible number of total logical situations; the cardinal of Q'

1.2 $T = \{t_1, t_2, t_3, \ldots, t_\tau\}$

'τ' - 'the number of things; the cardinal of T'

1.3 $F \cap Q = \phi$

1.4 $Q \cap T = \phi$

1.5 $T \cap F = \phi$

1.6 $\chi = 2^T$

Explanation: A thing is, or it is not.

1.7 $\varphi \leq \chi$

Explanation: The number of forces corresponds to the number of unstable total logical situations and the number of possible événements, at most equal to the number of total logical situations.

Abbreviation: For 'object system n', '$O.S._n$'

An object system must be stated in simple terms in S1, due to 1.6.

Example. In $O.S._1$ there are 4 things, and 3 unstable logical situations. (Recall there are no thing/states in S1.)

In $O.S._1$:

$$\chi = 16 \qquad 1.6$$

A number from 0 to 15 may be assigned to each possible logical situation; thus:

$$Q = \{q_0, q_1, \ldots, q_{15}\}$$

There are 3 unstable logical situations, and so three forces:

$$F = \{f_1, f_2, f_3\}$$

Since the antecedent (unstable) and consequent (stable or unstable) logical situation for each f_i completely define each f_i, it is convenient to identify f_1, f_2, and f_3 with numeric subscripts identical to their respective antecedent and consequent logical situations; thus:

$$F = \{f_{1-3}, f_{3-8}, f_{4-6}\}$$

Inspection of this statement of F makes possible identification of a set of sequences of total logical situations, or states, of $O.S._1$.

$$\{(1, 3, 8), (4, 6)\}$$

It is clear that if $O.S._1$ is ever in state 1 it will proceed to state 8 and remain there; and it is equally clear that if $O.S._1$ is ever in a state other than 1, 3, or 4, it will remain there.

S1, upon examination, is a very simple system. Analysis of an object system is an almost trivial task.

[1] 'Événement', French for 'event', is used to connote something more comprehensive than an event.

This simplicity is its virtue. For if more sophisticated systems (Sn) are developed whose statement may be transformed to S1 statement, the problem of system analysis will be greatly simplified. Algorithms can be programmed which perform model analysis.

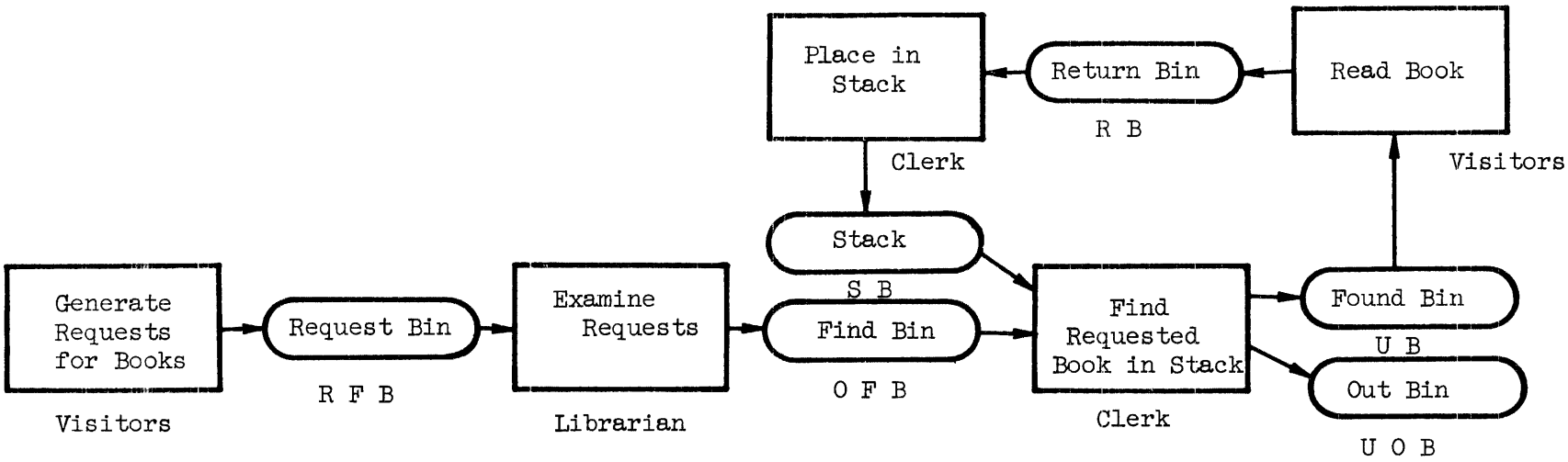
The eventual scheme of operation might be as follows:

1. Statements concerning $O.S._i$ in Sn. By a modeler.
2. Synthesis of a model of $O.S._i$ in Sn.
3. Transformation of the Sn model to an S1 model.
4. Analysis of the S1 model.
5. Translation of the S1 model analysis to statements about the Sn model.
6. Presentation of the analysis to the modeler.
7. Possible revision of the Sn model and repetition of steps 1 through 6.
8. Simulation runs of the Sn model.
9. Presentation of simulation results to the modeler.

Figure 5 is an example of what a simulation model in S2 or S3 might look like. Upon transformation to S1, ambiguities and inconsistencies in such a model become identifiable. The possible terminal states, as functions of initial states, may be deducible.

Within the syntax exemplified in figure 5, quite complex statements may be written. These statements may refer to individual characteristics of things, states, and logical situations. For example, duration time may be determined by a formula containing occurrences of variables which are functions of individual characteristics of things occurring in the antecedent logical situation formula.

Summary. Muse is intended to provide a system of analysis, modeling, and simulation which will reserve to the student of systems the functions he alone may fulfill and which will delegate as much as possible to machine. By delegating deduction to machines and reserving premise and hypothesis statement to humans, we expect to be able to study the simulated behavior of very large, very complex systems.



Transient Information Records

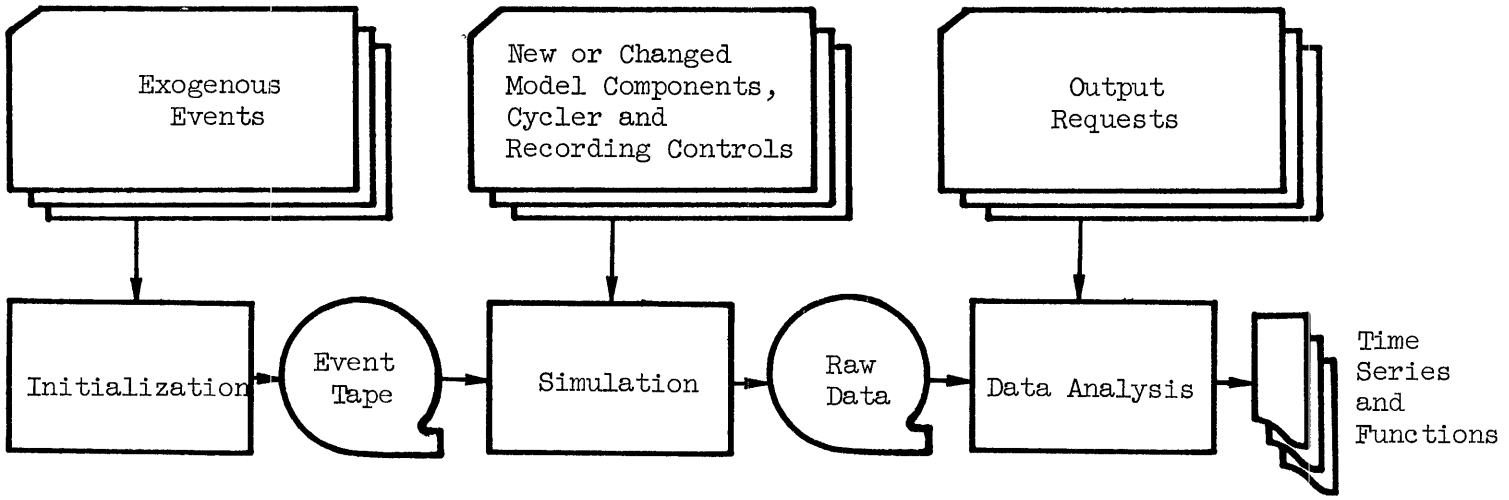
O F B Order for Book
R B Read Book
R F B Request for Book
S B Stacked Book
U B Unread Book
U O B Unfilled Order for Book

Activity Performers

Clerks
Librarian
Visitors

LIBRARY SYSTEM

FIGURE 1



OPERATION OF THE SIMPAC PROGRAM

FIGURE 2

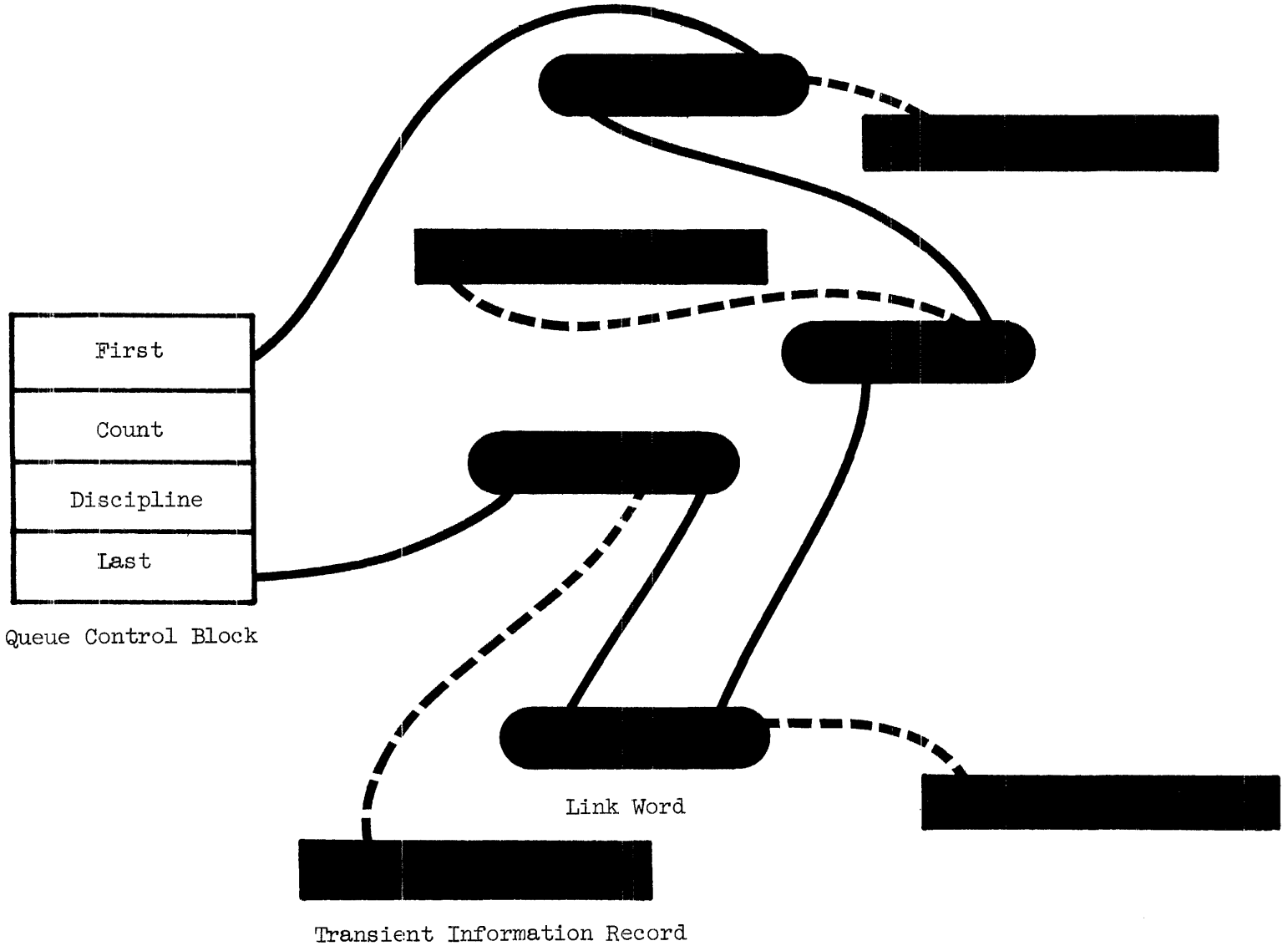
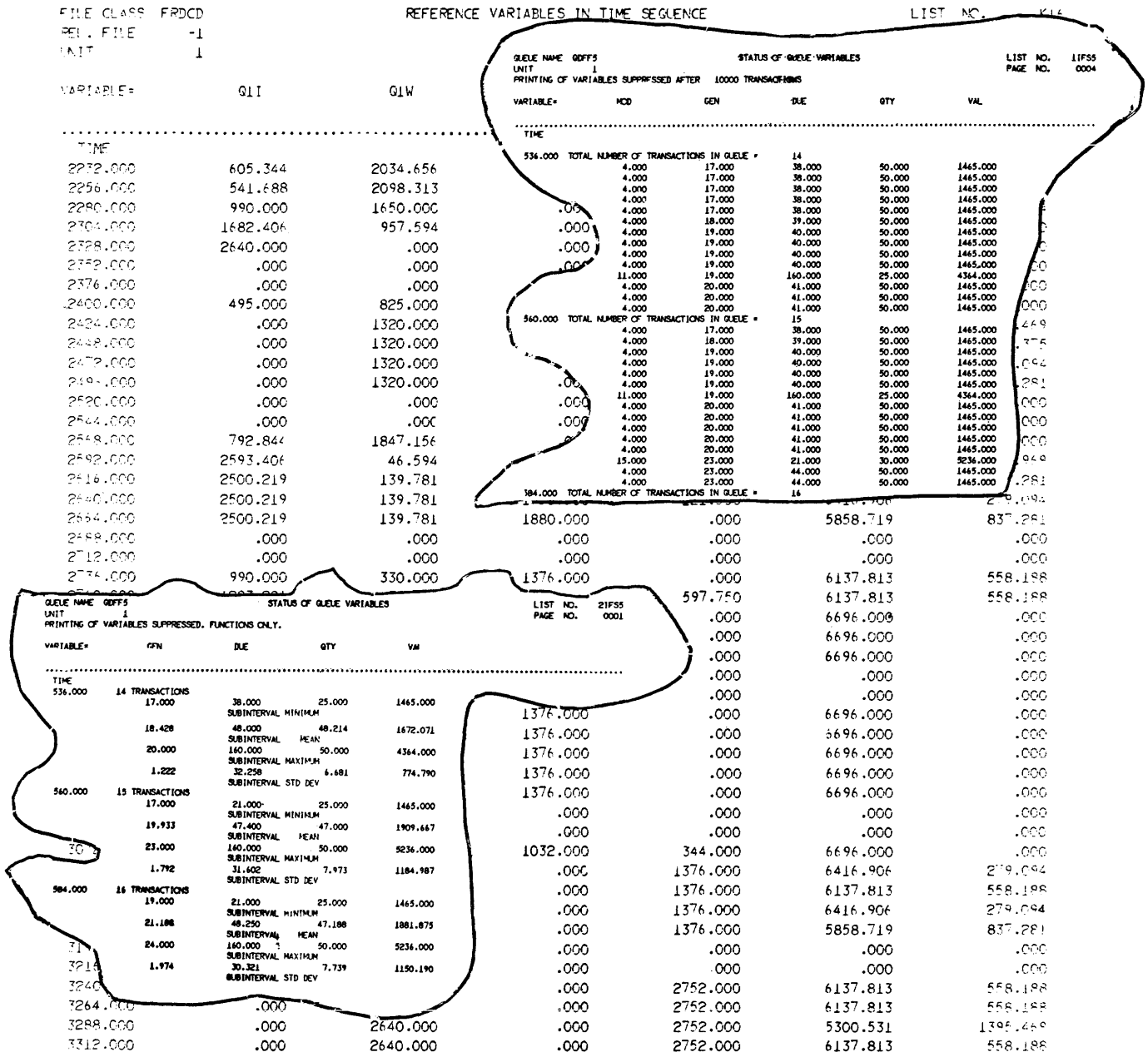


FIGURE 3 SIMPAC Associative Storage



FILE CLASS FRDCD
REL. FILE -1
UNIT 1

REFERENCE VARIABLES IN TIME SEQUENCE LIST NO. K14

VARIABLE=	Q1I	Q1W				
TIME						
2232.000	605.344	2034.656				
2256.000	541.688	2098.313				
2280.000	990.000	1650.000	.000			
2304.000	1682.406	957.594	.000			
2328.000	2640.000	.000	.000			
2352.000	.000	.000	.000			
2376.000	.000	.000				
2400.000	495.000	825.000				
2424.000	.000	1320.000				
2448.000	.000	1320.000				
2472.000	.000	1320.000				
2496.000	.000	1320.000	.000			
2520.000	.000	.000	.000			
2544.000	.000	.000	.000			
2568.000	792.844	1847.156				
2592.000	2593.406	46.594				
2616.000	2500.219	139.781				
2640.000	2500.219	139.781				
2664.000	2500.219	139.781	1880.000	.000	5858.719	837.281
2688.000	.000	.000	.000	.000	.000	.000
2712.000	.000	.000	.000	.000	.000	.000
2736.000	990.000	330.000	1376.000	.000	6137.813	558.188

QUEUE NAME GDFF5 STATUS OF QUEUE VARIABLES LIST NO. 1FS5
UNIT 1 PAGE NO. 0004
PRINTING OF VARIABLES SUPPRESSED AFTER 10000 TRANSACTIONS

VARIABLE=	MOD	GEN	DUE	QTY	VAL
TIME					
536.000 TOTAL NUMBER OF TRANSACTIONS IN QUEUE =		14			
	4.000	17.000	38.000	50.000	1465.000
	4.000	17.000	38.000	50.000	1465.000
	4.000	17.000	38.000	50.000	1465.000
	4.000	17.000	38.000	50.000	1465.000
	4.000	17.000	38.000	50.000	1465.000
	4.000	18.000	39.000	50.000	1465.000
	4.000	19.000	40.000	50.000	1465.000
	4.000	19.000	40.000	50.000	1465.000
	4.000	19.000	40.000	50.000	1465.000
	4.000	19.000	40.000	50.000	1465.000
	11.000	19.000	160.000	25.000	4364.000
	4.000	20.000	41.000	50.000	1465.000
	4.000	20.000	41.000	50.000	1465.000
	4.000	20.000	41.000	50.000	1465.000
560.000 TOTAL NUMBER OF TRANSACTIONS IN QUEUE =		15			
	4.000	17.000	38.000	50.000	1465.000
	4.000	18.000	39.000	50.000	1465.000
	4.000	19.000	40.000	50.000	1465.000
	4.000	19.000	40.000	50.000	1465.000
	4.000	19.000	40.000	50.000	1465.000
	4.000	19.000	40.000	50.000	1465.000
	11.000	19.000	160.000	25.000	4364.000
	4.000	20.000	41.000	50.000	1465.000
	4.000	20.000	41.000	50.000	1465.000
	4.000	20.000	41.000	50.000	1465.000
	4.000	20.000	41.000	50.000	1465.000
	4.000	20.000	41.000	50.000	1465.000
	15.000	23.000	21.000	30.000	5236.000
	4.000	23.000	44.000	50.000	1465.000
	4.000	23.000	44.000	50.000	1465.000
584.000 TOTAL NUMBER OF TRANSACTIONS IN QUEUE =		16			

1376.000	.000	6696.000	.000			
1376.000	.000	6696.000	.000			
1376.000	.000	6696.000	.000			
1376.000	.000	6696.000	.000			
1376.000	.000	6696.000	.000			
.000	.000	.000	.000			
.000	.000	.000	.000			
1032.000	344.000	6696.000	.000			
.000	1376.000	6416.906	279.094			
.000	1376.000	6137.813	558.188			
.000	1376.000	6416.906	279.094			
.000	1376.000	5858.719	837.281			
.000	.000	.000	.000			
.000	.000	.000	.000			
.000	2752.000	6137.813	558.188			
.000	2752.000	6137.813	558.188			
.000	2752.000	5300.531	1395.469			
.000	2752.000	6137.813	558.188			

QUEUE NAME GDFF5 STATUS OF QUEUE VARIABLES LIST NO. 2IFS5
UNIT 1 PAGE NO. 0001
PRINTING OF VARIABLES SUPPRESSED. FUNCTIONS ONLY.

VARIABLE=	GEN	DUE	QTY	VAL
TIME				
536.000 14 TRANSACTIONS				
	17.000	38.000	25.000	1465.000
		SUBINTERVAL MINIMUM		
	18.428	48.000	48.214	1672.071
		SUBINTERVAL MEAN		
	20.000	160.000	50.000	4364.000
		SUBINTERVAL MAXIMUM		
	1.222	32.258	6.681	774.790
		SUBINTERVAL STD DEV		
560.000 15 TRANSACTIONS				
	17.000	21.000	25.000	1465.000
		SUBINTERVAL MINIMUM		
	19.933	47.400	47.000	1909.667
		SUBINTERVAL MEAN		
	23.000	160.000	50.000	5236.000
		SUBINTERVAL MAXIMUM		
	1.792	31.602	7.973	1184.987
		SUBINTERVAL STD DEV		
584.000 16 TRANSACTIONS				
	19.000	21.000	25.000	1465.000
		SUBINTERVAL MINIMUM		
	21.188	48.250	47.188	1881.875
		SUBINTERVAL MEAN		
	24.000	160.000	50.000	5236.000
		SUBINTERVAL MAXIMUM		
	1.974	30.321	7.739	1150.190
		SUBINTERVAL STD DEV		

3264.000	.000	
3288.000	.000	2640.000
3312.000	.000	2640.000

SAMPLE OUTPUTS
FIGURE 4

THING	SYMBOL	STATE		
		On Tree	On Ground	Rotten
Apple	T1	1	2	3
		Hungry	Sated	Leaves the Scene
Boy	T2	1	2	3
		Producing Apples	Not Producing Apples	
Tree	T3	1	2	

Force/Algor.	Statement	Comments
1	T1/2T2/1:T2/2	Given an apple on the ground and a hungry boy, the apple disappears and the boy is sated.
2	T2/2:T2/1;N6,1.5	A sated boy becomes hungry in about 6 time units.
3	T2/1:T2/3;N40,15	A hungry boy leaves the scene in about 40 time units.
4	T2/A1,2:T2/3;P1000	A boy leaves the scene in about 1000 time units, whether he has been hungry or sated.
5	T1/1:T1/2;P20	An apple on the tree falls to the ground about 20 time units from first appearance.
6	T1/2:T1/3;P3	An apple rots if it is on the ground about 3 time units.
7	T3/1:T1/1T3/1;P5	A tree producing apples does so at a rate of about 1 every 5 time units.
8	T3/1:T3/2;N18,6	After a tree has been producing apples for about 18 time units, it stops producing apples.
9	T3/2:T3/1;N18,4	When a tree hasn't been producing apples for about 18 time units, it starts producing apples.

FIGURE 5

The example in figure 5 is in accordance with the following:

Notation

'Ti/j' for: 'Thing i in state j'

'Ni,j' for: 'Duration time normally distributed, mean i time units, standard deviation j time units'

'Pi' for: 'Duration time Poisson distributed, mean i time units'

'Ai,j' for: 'Either i or j; inclusive disjunction'

Syntax

(Antecedent logical situation):(Consequent logical situation);(Required duration of antecedent)

Rules

Given the occurrence, and specified endurance, of an antecedent logical situation, the antecedent l.s. yields to the consequent l.s.

Destruction of the antecedent l.s. before the required endurance precludes the occurrence of the consequent l.s.

If a duration time is not stated, duration time is 0. All duration time distributions are truncated: $0 < t \leq \infty$.

Things and states mentioned in the antecedent logical situation and not in the consequent are destroyed.

Things and states mentioned in the consequent l.s. and not in the antecedent l.s. are created.

A NONLINEAR DIGITAL OPTIMIZING PROGRAM FOR PROCESS CONTROL SYSTEMS

Raymond A. Mugele

Control Systems, General Products Division

International Business Machines Corporation

San Jose, California

Summary

This paper deals with nonlinear programming. In particular, it summarizes a newly developed program suitable for optimization of a computer-controlled process. The program applies probing and constraint-following algorithms which permit solving the optimization problem in difficult cases. These cases include nonlinear or discontinuous objective functions, constraint functions, and nonconvex domains.

The program has become known as the "Poor Man's Optimizer," as it requires relatively little storage for program and data. It is applicable to the relatively small digital computers now popular in process control.

Introduction

We review briefly the purpose of nonlinear programming.

In designing or controlling some large unit, for example, a chemical plant or a petroleum refinery, we may devise a mathematical model. This model describes at least the internal material balances and heat balances and chemical reactions that are involved. It is already useful in determining requirements for operation or construction of the unit. Next, the model may be rendered more flexible by allowing for variability in operating conditions. This added flexibility may lead to an insight which suggests better operating procedures, shows the response of the system to economic changes, or provides other useful information.

But this is not yet programming.

The remaining step is to set up a procedure which calculates optimum operating conditions, according to some preset criterion. This is programming.

In general, some sort of approximation to the industrial system may be obtained with a linear model. In the linear model, things change at a constant rate relative to the basic variables. For example, if each ton per day increase in throughput at a plant produces $100 per day increase in profit, we have a linear situation. But there are cases in which the linear model is inadequate (not precise enough, or misleading as to trends). Perhaps, in the above example, each additional ton per day increase in throughput produces a smaller improvement than the preceding increase. Then we have a nonlinear situation. In such cases, the programming cannot be accomplished using the relatively direct and simple methods of linear programming. Instead, the methods we use must be capable of accepting nonlinear objective functions and constraints and proceeding to an optimum, perhaps constrained by a stringent set of specifications. This is nonlinear programming.

Several workers have devised or evaluated methods of nonlinear programming, particularly in the field of operations research.[1, 2, 8, 9, 14] Such methods have been applied to such diverse problems as gasoline blending,[7] chemical plant problems,[12] and algebraic test problems.[2, 13]

However, these methods do not apply to cases in which several of the constraints have complicated nonlinear forms. Such cases become important when we consider something as complex as a chemical plant or a petroleum refinery. Here the interaction of physical, chemical, and economic factors leads to an involved objective function. Also, practical considerations such as product specification and component availability lead to a number of constraint inequalities, some of them highly nonlinear. Thus, we face the problem of nonlinear programming with nonlinear constraints. To handle such programs, extensions of existing nonlinear methods have been proposed.[8] In the following sections we shall describe such proposals and also some effective methods which have already been

developed and applied to full-scale problems.

Before entering a discussion of some of the techniques which have been studied, we offer some definitions. Although some variations from these definitions may be found in other publications, in this paper the following definitions are used consistently. This does not by any means imply that our usage is more valid than others, but rather that ours is directed toward a specific field.

Definitions

Programming. Setting up a method for solving an optimization problem. (This of course restricts us to a special field, part of the broader field of computer programming.)

Optimization. Determining a maximum for an objective function. (If a minimum is required for F, then maximizing $-$ F solves the same problem.)

Objective or Value Function. A quantity F that measures the desirability of some product, process, or method. In simple cases, only directly measurable quantities are involved (e. g., the daily volume of gasoline produced by a refinery). In more complex cases, intangibles from technology and economics also enter (e. g., safety factors, tax structure, amortization policy).

Independent or Basic or Decision Variables. Quantities y_u needed for calculating F and for expressing the optimal solution. The y_u may be subject to bounds and constraints, but they are not fixed until the problem is solved.

Tolerance. A set of quantities (δ_{1e}, δ_{2e}, . . . , δ_{ke}) predetermined with the intent that each y_u in the calculated solution shall differ from that in the true solution by no more than δ_{ue}.

Bounds. Functions of form $B_u^- \equiv y_u - Y_u^-$ and $B_u^+ \equiv Y_u^+ - y_u$ which are required to be nonnegative at the optimum: $Y_u^- \le y_u \le Y_u^+$.

Constraints. Quantities C_i that are functions of the y_u and are required to be nonnegative at the optimum. To avoid duplication, we require that the C_i have forms other than $k(y_u - Y_u^-)$ and $k(Y_u^+ - y_u^-)$, i. e., that they differ from bounds. The nonnegative character of C_i usually arises from practical limitations. The constraint may at first be expressed in some other form, but it is always transformable to the form $C_i \ge 0$. For example, if $y_1^2 + y_2^2$ is required to be no greater than 10, we take $C_i = 10 - y_1^2 - y_2^2$. Some caution must be exercised in dealing with constraints, to avoid thinking of properties of a particular constraint function as properties of the optimization problem. For example, the functions $C_1 \equiv y_1 - y_2$ and $C_2 \equiv y_1^3 - y_1^2 y_2 + y_1 y_2^2 - y_2^3$ are interchangeable as constraints, since they are nonnegative in exactly the same region; yet their gradients are very different.

Feasible. Having all bounds and constraints nonnegative. An operating vector $\bar{y}$ is feasible if $B_u^{\pm}(Y_u) \ge 0$ for all u, and $C_i(\bar{y}) \ge 0$ for all i. Nonfeasible, of course, implies having at least one bound or constraint negative.

Proper Maximum. A maximum of an objective function which has only positive bounds and constraints.

Constrained Maximum. A feasible maximum of an objective function which has at least one zero bound or constraint.

Local Maximum. A point (condition of system or operating vector) such that any feasible small change will cause a decrease in the objective function.

Global Maximum. The local maximum having the highest value of the objective function.

Vector. An ordered set of quantities (components, or major variables) which determines the state of a system (operating vector). It also determines a change in the state (gradient or step vector). As an example, consider the condition of a reactor to be specified by feed temperature T, catalyst concentration C, and pressure P. Then we may write $\bar{y} \equiv (T, C, P)$ to represent the operating vector. A function of this vector is evaluated according to the separate values of the components; for example, if F is the daily product value for the reactor, we may write $F \equiv f(\bar{y})$ to indicate that F is some known function of T, C, and P.

Gradient. A vector which expresses the local variation of a function. In the case of $F(y_1, y_2, \ldots, y_k)$, the gradient vector is the set $(\partial F/\partial y_1, \partial F/\partial y_2, \ldots, \partial F/\partial y_k)$. This may be expressed by grad F or ∇F ("del" F). It determines the direction of most rapid change in F. This direction is sometimes described as "normal" to F = constant.

Linear. Having a constant gradient. To emphasize the far-reaching consequences of this simple restriction, we note the rapid and widespread success already achieved by linear programming. This success is in large measure due to the advantage of having a constant gradient, i. e., one which does not have to be recalculated at each step toward the optimum.

Projection. A vector which expresses the local variation of a function in a special direction. When the "projection of $\overline{z}$ on C_j" is specified, this vector has the form $\overline{p} = \overline{z} + \mu \nabla C_j$. It is also normal to ∇C_j. Hence we have $\Sigma^J Z_u \partial C_i/\partial y_u + \mu \Sigma (\partial C_i / \partial y_u)^2 = 0$, from which μ may be calculated. When the "projection of $\overline{z}$ on C_j and C_m" is specified, $\overline{p}$ will have the form $\overline{z} + \mu_1 \nabla C_j + \mu_2 \nabla C_k$ and will be normal to both ∇C_j and ∇C_k, with the coefficients μ_1 and μ_2 then determined by a pair of simultaneous equations. This procedure may be extended to projections of higher order, but is necessarily limited to projection on k - 1 simultaneous constraints, where k is the number of basic variables (dimensionality, or degrees of freedom) of the problem. If, in particular, $\overline{z} = \nabla F$, then $\overline{p}$ is the "gradient projection of F." In these calculations, bounds are handled in the same manner as constraints, with the calculations somewhat simplified due to the simplicity of ∇B.

Magnitude. A positive quantity which indicates "size" without regard to "direction." In the case of vector $(y_1, \ldots, y_k)$, the magnitude is defined by $|\overline{y}| = (y_1^2 + y_2^2 + \ldots + y_k^2)^{1/2}$.

Norm. A measure which is not strictly the vector magnitude as defined above, but is ordinarily easier to calculate. In the case of vector $(y_1, y_2, \ldots, y_k)$ the norm is defined by $\{\overline{y}\} \equiv |y_1| + |y_2| + \ldots + |y_k|$. In any case, $1 \leq \{\overline{y}\}/|\overline{y}| \leq k^{1/2}$.

Unitize. To change the magnitude of a vector to unity without changing the ratios of components. If vector magnitude is used for this purpose, the unitized components are defined by $\hat{y}_u \equiv y_u / |\overline{y}|$. If the vector norm is used, the unitized components are defined by $y_u \equiv y_u \equiv \{\overline{y}\}$.

Some Methods of Nonlinear Programming

With the help of the foregoing nomenclature, we are in a position to describe some methods used in nonlinear programming. Allusions to some of these techniques are already in the literature of optimization, and useful bases for the development of general nonlinear methods have long been available. However, only two basic methods have been developed sufficiently for wide use in systems that have nonlinear constraints as well as nonlinear objective functions.

In order not to lengthen this paper, we offer only a brief description of these methods. The literature references should be useful to those desiring more information about particular techniques.

Analytical Method

Use elementary calculus and algebra to determine all proper maxima possessed by the objective function. The highest of these maxima may be the solution, if feasible. Next determine constrained local maxima subject to each of the constraints and bounds (Lagrange's method of indeterminate multipliers is convenient for this). The highest of these may be the solution, if feasible and not surpassed in the earlier calculations. Next, do the same for local maxima subject to pairs of constraints or bounds. Note that there are $(2K + q) \cdot (2K + q-1)/2$ such pairs. Continue in this way until all local maxima on groups of constraints or bounds have been isolated. Then select the global maximum. Some of the mathematics required in this method may be found in the literature.[9, 11]

Grid Method

Space y_u values at reasonable intervals for each variable (e.g., divide the entire range of each variable into ten parts). Calculate F at each feasible member of the $(j_1 + 1)(j_2 + 1)\ldots(j_k + 1)$ resulting grid points (where j_u is the number of intervals for variable y_u). Designate the point with the highest F value as the "first solution." If more precision is required, explore the vicinity of the first solution on a finer mesh than before (or else apply some interpolation device, such as a second-degree "hypersurface"). Continue this until satisfactory precision is obtained. Check the solution by making gradient or gradient-projection excursions. To reduce computing time when using this method, eliminate protions of the grid whenever possible by considering analytical properties of the constraints. This is sometimes called the "case study" method in design work.

Monte Carlo Methods

Use a random number table (or generator) to select a point in the bounded region. Retain this point if feasible, and calculate its F. Repeat for another random point. Retain the new point if its F is higher than that of the old point. Continue in this way for a predetermined number of points (calculated from statistical considerations to give a preset confidence level). The last retained point is taken as an approximate solution. Check this solution by making gradient or gradient-projection excursions.

A variation of this method uses the retained point as a start for the next move. Then the direction of the move is determined at random, but the magnitude is fixed. If a certain number of moves does not produce an improvement, the magnitude of the move is decreased. Brooks[2]

discusses some of the possibilities of these methods.

Cross-Section or Univariate Method

With fixed starting values of $y_2, y_3, \ldots, y_R$, vary y_1 by preselected steps, spanning its range. At each point, calculate F if feasible. Select the best F so far and set the corresponding value of y_1. Then vary y_2 in the same way, and set y_2 according to the best feasible point so far. Continue in this way until a complete cycle produces no further change in F. As with the grid method, a finer mesh may be introduced after the final approximation is achieved. As usual, the solution must be checked by making gradient or gradient-projection excursions.[6]

Blocking Methods

Consider the possibility of reducing the region to be examined, in a systematic rather than a random manner (cf. Monte Carlo Methods). The general idea is to eliminate about half of the bounded region of coordinate space at each step. These methods may be satisfactory for a small number of variables, but they become unmanageable for a large number of variables because the procedure requires a knowledge of the objective function and of the constraint functions for at least the "corners" of the region to be eliminated. In a region with k major variables, there are 2^k such corners. For k = 20, this number is more than one million.

Gradient Projection Method

From a feasible starting point, follow the gradient until stopped by a constraint. Follow this constraint, via gradient projections on tangent hyperplanes, until stopped by another constraint. Then follow the two constraints, via gradient projections on intersections of tangent hyperplanes, until stopped by a constraint. Follow this constraint, via gradient projections on tangent hyperplanes, until stopped by another constraint. Continue in this way until "cornered" (at a constrained maximum) or stopped by a zero gradient or gradient projection (at a proper maximum or partially constrained maximum). Check the final calculated point to see whether it is a solution. Due to the use of tangents to approximate constraints, it will sometimes be necessary to return to the feasible region when a calculated step has led to a nonfeasible point. This is done by an interpolation procedure.[8]

Probe Methods

With preselected step sizes for all variables, make excursions from a feasible point, in directions of increasing and decreasing y_u. The first successful excursion (feasible point with improved F) determines the next point on the course, and this point becomes a nucleus for further probing. Continue the course until no improved feasible point is achieved, due perhaps to approaching a constraint. A technique for following constraints is described in the section on Subprogram EDGE, later in this paper. When a course terminates away from all bounds and constraints, a nearby solution is indicated. When the appropriate procedures have failed to produce any advance, it is time to decrease the step size and repeat the entire procedure. After repeated reductions, the step size falls below some tolerance, and the calculation stops, indicating a solution. The calculated solution is to be checked via gradient or gradient-projection calculations, provided that it is possible and practical to calculate derivatives of the objective and constraint functions.

A number of practical variations of the method just outlined are possible. One of these involves resetting the probe point to a bound whenever such a bound is crossed in probing. Another involves persisting with probes in a favorable direction, once such a direction is established. Another consists in designating certain variables as "discrete," in which case their increments will be integer multiples of specified values (usually unity), and the variables themselves will take on only specified values (usually integers).

Figure 1 illustrates the general operations involved in a Probe program. The details are discussed later.

Other Methods

Of course we should not, and do not, claim that all methods of nonlinear programming have been included in this summary. However, some published methods[15, 17] are so closely related to linear programming that it is better to treat them in publications on that subject. Other methods[16] relate more specifically to special classes of problems, for which the ultimate solution depends on such methods as we have already considered.

Comparisons. Experience with actual programming and problem solving, using the methods described above, has indicated that either the Gradient Projection method or the Probe method, or special combinations of these, will prove most effective in attacking optimization problems which relate to evaluation, design, or control of large systems.

Note at this point that, although some analogies do exist between Gradient methods and Probe methods, essential differences also exist. For

example, if the calculation of a derivative is not possible or practical, the Gradient Projection method cannot be used. On the other hand, if a situation arises which demands only the most rapid feasible advance, the Probe method is ruled out.

Analytical procedures are rejected because of the complications which would be encountered in the usual control problem. Grid methods require examination of too many "cases"; e. g., a million cases might be examined in solving a three-variable problem with a solution tolerance of 1% of the range. Blocking methods are similarly cumbersome. Monte Carlo methods may be useful for finding a solution, but in the usual form they do not give information about the nature of conditions close to the solution: such conditions may be of considerable interest in a control problem. Cross-section methods and variations of Gradient Projection, now under study by others,[8, 12] may be found to be interchangeable with the methods that we have stressed.

In this paper, details of structure and program are presented for Probe methods only.

Optimizing Subprograms

In the preceding section, general features of Probe techniques were outlined. We shall now describe in detail the principal subroutines which are incorporated in the final program to be considered later. This program is simple in concept and application, but gives indication of being suitable for use with large and complicated problems. In the sections that follow, the entire program will be elaborated in enough detail to guide coding for a stored-program computer.

Subprogram PROBE, shown in flow form in Figure 2, contains the basic logic of the "probe" and is operative as long as special local conditions (e. g., regions where the gradient changes sharply within one step, or regions near constraints) do not intervene. The basic procedure is as follows:

With preselected step-sizes for all variables, probe, in a preset order, from a feasible point (nucleus) in the (positive and negative) directions of the basic variables. Desist as soon as a higher-valued feasible point is found; use this point as a new nucleus. When the probes on all variables fail to produce an improvement, go into Subprogram RIDGE, unless some probe point was nonfeasible; in that case go into Subprogram EDGE. Subprograms RIDGE and EDGE are discussed later.

If progress is made for several (e. g., 4) steps with a fixed step size, this operation is under the control of Subprogram ORDER. But if the return to Subprogram PROBE from RIDGE or EDGE shows no improvement, it is time to halve the step size. When the step size falls below some preassigned tolerance, the calculation will be halted under direction of the main program.

In Figure 3, which shows a simple two-dimensional (schematic) problem with two constraints, this subprogram is illustrated by the course SABDEF. Note that for this course the preferred direction is changed at B (under control of Subprogram ORDER), where it is not necessary to go into Subprogram EDGE; and again at F, where it is necessary to go into EDGE.

The use of a flag, Key 23, for this branching, is illustrated in Figure 2. Another function of Subprogram PROBE is to select the critical points (high and low feasible and nonfeasible alternates) to be used in Subprograms RIDGE and EDGE if needed.

Subprogram RIDGE, shown in flow form in Figure 4, is designed to maintain progress with large steps even in regions where the gradient changes sharply within one step. Suppose that probes from the nucleus (at full step size) have failed to indicate an impending constraint, yet have also failed to yield an improved point. Then before reducing the step size, proceed as follows: between the two highest-valued probe points, evaluate the midpoint. Break out of the subprogram if the midpoint is nonfeasible (failure) or if it is feasible and higher than the nucleus (success). Otherwise, use the evaluation together with a quadratic approximation to estimate the maximum F between the critical probe points. The equations for this estimate are:

$$y_{1R} = (1 - r)\, y_{1A} + r y_{1B} \ ;$$

$$y_{2R} = (1 - r)\, y_{2A} + r y_{2B} \ ;$$

$$r = (F_A - F_B)\ /2\ (2F_M - F_A - F_B).$$

Here y_1 and y_2 represent the variables involved in probes that produce the highest feasible point (A) and the next-highest feasible point (B); F is the objective function; M represents the midpoint between A and B; and r is a parameter used in calculating the approximate maximum or "ridge" point between A and B. This calculation is in part based on a quadratic approximation of the objective function on the ridge.

Figure 5 illustrates a course in which Subprogram RIDGE is effective in using the midpoint for

several steps (SCFJ), then fails at the midpoint (N) but succeeds at the quadratic approximation (P).

Subprogram EDGE, shown in flow form in Figure 6, directs procedure after a constraint (or a confluence of constraints) is sensed (Key 23 = 3). It directs the course of computation so as to "follow" the "nearest" constraint, unless improvement can be made in a direction away from the constraint. An outline of procedure follows.

Since at least one of the probe points is nonfeasible and all probe points one step-size away have been evaluated (otherwise we would not be in EDGE), information is available to classify highest-valued and lowest-valued feasible and nonfeasible points. The classification may be represented as follows:

	Feasible	Nonfeasible
Highest	HF	HN
Lowest	LF	LN

Now, first test the midpoint between HF and HN. If feasible, evaluate. If not better than the nucleus, replace HF by the new point. Then proceed similarly with LF and LN. Repeat the procedure using HF and HN, as well as the procedure using LF and LN, at least four times before concluding that a revised choice of critical points is required. Then combine the first- and second-choice points according to the schedule shown at the upper right in Figure 6, under control of flag K3. As usual, break out of the subprogram as soon as a point better than the nucleus is discovered.

In Figure 3, the combined use of Subprograms PROBE and EDGE is illustrated by the course FKLPWX. In this schematic example, the need for step-size reduction does not occur until the last step (which also involves resetting to a bound).

For a single nearly linear constraint, as in the diagram at the right of Figure 6, this procedure must produce a point (such as L or H) which is better than the nucleus and feasible, provided that the value function is also nearly linear and that we are not already at the optimum. (As a matter of theoretical interest, this statement can be proved for a number of variables and for more general conditions than those implied here.) Although these conditions may seem very restrictive, they commonly occur in practice, with the result that Subprogram EDGE is very useful. The approximation to linearity, of course, improves at small step-size.

Subprogram ORDER, shown in flow form in Figure 7, reorders the probe array JORD and the constraint-calculating order array JCON when needed in the interests of efficiency. The procedure used in probing is to bring the successful probe to the head of the list, and to remove its complement to the end. For example, if the probe order has been −3, +2, +1, −1, −2, +3, and a successful probe is made on the fourth trial (−1), the new probe order becomes −1, −3, +2, −2, +3, +1. The array JCON is adjusted according to a tally of the number of times that individual constraints have been calculated negative during the course of a major cycle. The largest frequency NC(K) determines the first C(K) to be calculated in sequence, the next largest NC(K) determines the next C(K), and so on. This eliminates superfluous calculations of constraints likely to be positive, when a negative one is sought. For a feasible point, the complete set of constraints must be calculated in any case; then the order is unimportant and will be left unchanged.

Main Probe Program and Subroutines

The main program is shown in flow form in Figure 8. In addition to connecting the subprograms and doing the necessary initializing, finalizing, and branching, the main program also calls the subroutines which define the problem. Such subroutines are defined in the following paragraphs. General restrictions on formulation of problems are also presented following the description of the subprograms.

Problem Subroutines. The subroutines that define the problem are briefly described as follows:

1. ENTER. The optimization program will call ENTER only at the start of the calculation. Accordingly, this subroutine should include all the one-time operations required to set up the calculation. It will set the values of KY, Y, YL, YH, and KC, normally by means of a data-load. It may set an initial value for STEP, and values for STEND and D. Although these are not essential (being calculable as fixed fractions of the ranges of variables), a proper selection may shorten the program running time. If discrete variables are to be used, ENTER must set the MQD values accordingly. ENTER can also be used to load other data or parameters, perform initial calculations, or provide an initial printout.

2. SAVE. This subroutine is the first one called by the optimization program after it changes the Y(I) values. To eliminate duplicate programs, calculations should be included here that are common to several constraint functions, or

common to the objective function and to some constraint functions. Calculations which pertain to the objective function alone, or to only one or two constraint functions, ordinarily are omitted from SAVE.

3. OBJECT. This subroutine sets VAL equal to the value of the objective function at the specified set of Y(I). The subroutine can assume that SAVE results are consistent with the assigned Y(I)'s, but must not use any results from FENCE, since they may be inconsistent. Results reported as VAL at successive points should represent a single-valued function of the variables, and should not vary randomly because of convergence and roundoff errors. If such variations occur in the last few digits of a calculated objective function, they should be dropped before reporting the value as VAL.

4. FENCE. This subroutine must set C(JC) equal to the value of the constraint function for the stored index JC, at the specified set of Y(I). The subroutine can assume that SAVE results are consistent with the assigned Y(I)'s, but must not use any results from OBJECT, nor those from FENCE for any other value of JC.

5. FOUL. If all C(I) values are positive or zero at the initial set of Y(I) values, this subroutine will not be used. However, if any C(I) is negative at the start, it is recognized as nonfeasible. The optimization program will attempt to find a feasible start, without regard to the objective function. After each unsuccessful try, it will call FOUL. If the programmer wishes to follow the results of these trials, he can program FOUL to print out pertinent results calculated by SAVE and FENCE. Once a feasible point is reached, OBJECT will be called. Thereafter, nonfeasible points are treated as intermediate results with no special provision for publishing. If the optimization program concludes without finding a feasible point, it will set KERR = −1 and call FINISH to terminate the calculations.

6. FAIR. This subroutine is called each time a new nucleus is established. Thus the programmer can use FAIR to publish results of interest for moves (points along the path to the optimum). When desired, FAIR can include machine options to bypass the publication. Moreover, a short testing routine at the start of FAIR can limit the points published to insure that each published VAL exceeds the previously published VAL by some specified increment. Since FAIR may be called several hundred times in a typical optimization, some limitation on either the points published or the length of publication per point is desirable.

In some calculations, the desired optimum may actually be set in advance. For example, in using an optimization technique to solve simultaneous nonlinear equations, the objective function starts as a large negative number and gradually approaches zero. In such calculations, the programmer may wish to conclude the run as soon as VAL approaches the known optimum closely enough.

7. FINISH. This subroutine is called at the conclusion of the optimization, for any of the following reasons:

a. The optimization program or one of its subroutines declares an answer;
b. an error is indicated;
c. no feasible start is found; or
d. the machine is required for another problem.

The programmer can use FINISH to perform any final calculations at the optimum and to publish detailed results. If desired, the error flag can be used to switch to a diagnostic routine within FINISH.

General Restrictions. The programmer should keep in mind the following restrictions.

1. The objective function, all constraint functions, and any intermediate variables used in their calculations should be single-valued, real, and calculable for any set of Y's with $YL(I) \leq Y(I) \leq YH(I)$ for all I. Discontinuous functions are not ruled out, but should be subject to careful scrutiny during the course of the calculation. In some cases it will be advisable to introduce an arbitrary function for the slope of a discontinuous variable, or to provide arbitrary (negative) values for constraint functions and an arbitrary (large negative) value for the objective function at the discontinuity.

2. The subroutines must provide easy access to the values of Y, YL, YH, C, KY, KC, JC, VAL, KERR, STEP, STEND, D, MQD. (In FORTRAN coding, this can be done by carrying these in the first COMMON statement.

3. Subroutines SAVE, OBJECT, FENCE, FOUL, and FAIR must not alter any of the above listed values, except that OBJECT must set VAL, FENCE must set C(JC), and any subroutine may set KERR to an appropriate value if an error is detected.

4. Adequate permanent storage must be provided for all of the above quantities so that they are always present, even though the subroutines may be moved about or altered. (In FORTRAN coding,

this can be achieved with suitable COMMON and DIMENSION statements.)

5. Limitations of the Probe Method. As indicated before, this Probe method does not purport to be a "universal" optimizer, although it has important advantages. As is true of all optimization programs known to the author, it is possible to construct problems which will defeat the program. It is even expected that such problems will arise in a practical setting, although none such has yet come to our attention. For this reason, a careful examination of calculated results is always required, from a mathematical as well as an engineering standpoint.

In case the result obtained is a local maximum and not a global maximum, even the careful terminal examination is of no avail. As Rosen[8] indicates, no better way of achieving the global maximum is yet known than that of proceeding from several different starting points. These starting points may be selected by a Monte Carlo procedure, or by more rational means if the problem is familiar to the programmer.

We note that there may be "pathological" features of value functions and constraint functions capable of defeating optimizers. Analysis of such features is outside the scope of this report. They are generally associated with discontinuities in derivatives of the value function or constraint functions. These give rise to "essential nonlinearities" which render the reduction of step-size futile as a mode of escaping from an unsatisfactory point in the calculation. An important exception is the case of a discrete independent variable; even though such a variable introduces essential nonlinearities, it does not defeat the Probe method.

6. Examples and Criteria for Numerical Testing. Problems of different degrees of complexity have been used to test the Probe program. The list includes synthetic test problems, practical design problems, and the solution of algebraic systems (simultaneous equations, curve fitting, and rational approximation).

For the present report, three test problems are selected and tabulated (Tables 1, 2, and 3) merely to give the reader some indication of the relations between problem data and program parameters.

A satisfactory criterion of "successful" solution is easily established. The criterion (as suggested earlier) is as follows: Within the preset tolerance, an excursion along the gradient or gradient projection should produce no better solution than that already obtained. This, of course, does not test for a global maximum but only for a local maximum. In a problem which cannot be solved analytically, the probability of achieving a global optimum might be increased by solving the problem a number of times, using a different starting point each time.

A satisfactory criterion of "efficient" solution is not so easily established. However, in order to have some kind of indication, we suggest that a solution be judged inefficient, if, in the course of it, some constraint, or the objective function, is calculated more than $n_e = k^s \log_2 R_{max}$ times, where k is the number of major variables, R_u is the range of a basic variable divided by the final tolerance on that variable and R_{max} is the greatest of the R_u.

Study of Tables 1, 2, and 3 shows that the program achieved "successful" solutions of each problem. Except in two extreme cases (B-2 and B-3), the method is "efficient" by our criterion (with s = 3, an arbitrary but probably reasonable value). Of course, these results are merely illustrative of the possibilities of the PROBE program. Peculiarities of a given problem, selection of a starting point, and choice of initial step sizes and final tolerances may alter the efficiency even on the same problem. Nevertheless, the PROBE program has performed well on a wide variety of nonlinear problems, including a "refinery simulation" with ten variables and seventeen constraints[10]. Therefore, this method is recommended for further application to nonlinear problems (including control problems), especially when complicated nonlinear constraints are present.

Table 1

Summary of Problem A

$$F = y_4 - {}^2y_1y_4 + {}^3y_1 - y_1^2 - y_2^2 + 4y_2 + y_3 - y_2y_3 - y_1y_2y_3$$

$$0 \le y_i \le 3 \quad (i = 1, 2, 3, 4)$$

$$C_1 = 3 - y_1 - y_2 - y_3 - y_4 \ge 0$$

$$C_2 = 1.10 - y_1y_2 \ge 0$$

	Starting Values			Solution			Efficiency Criterion		Tallies	
Test	$\bar{y}_o$	F_o	δ_o	$\bar{y}_e$	F_e	$\bar{\delta}_e$	R	n_e	n_{ci}	n_f
A-1	0.50	3.125	0.10	0.6586	5.433	0.002	1500	675	318	360
	0.50		0.10	1.6703		0.002	1500		465	
	0.50		0.10	0.0000		0.002	1500			
	0.50		0.10	0.0000		0.002	1500			
A-2	0.50	3.750	0.10	0.6586	5.433	0.002	1500	675	320	360
	0.80		0.10	1.6703		0.002	1500		460	
	0.30		0.10	0.0000		0.002	1500			
	0.20		0.10	0.0000		0.002	1500			
A-3	0.6594	5.427	0.10	0.6586	5.433	0.002	1500	675	240	260
	1.6594		0.10	1.6703		0.002	1500		340	
	0.0000		0.10	0.0000		0.002	1500			
	0.0000		0.10	0.0000		0.002	1500			

Table 2

Summary of Problem B

$$F = y_1^2 + (y_2/100)^2 + (y_3/300)^2$$

$$0 \le y_1 \le 1;\ 10 \le y_2 \le 95;\ 50 \le y_3 \le 200$$

$$C_1 = 1 - y_1 - y_2/100 \ge 0$$

$$C_2 = 1 - [(y_2 - 1)/100]^2 - (y_3/200)^2 \ge 0$$

$$C_3 = -1 + y_1 + y_3/200 \ge 0$$

	Starting Values			Solution			Efficiency Criterion		Tallies	
Test	$\bar{y}_o$	F_o	$\bar{\delta}_o$	$\bar{y}_e$	F_e	$\bar{\delta}_e$	R	n_e	n_{ci}	n_f
B-1	00.60	0.540	0.10	0.8999	1.2605	0.0002	5000	332	358	219
	30.00		3.00	10.01		0.02	4250		212	
	90.00		9.00	199.15		0.06	2500		212	
B-2[a]	0.60	0.540	24.85	0.8998	1.2604	0.0002	500000	511	640	323
	30.00		2492.5	10.01		0.02	474000		330	
	90.00		7477.5	199.15		0.06	332500		413	
B-3[b]	0.60	0.540	24.85	0.9000	1.2608	0.0002	500000	511	734	397
	30.00		242.5	10.00		0.0020	470000		380	
	90.00		727.5	199.18		0.0059	330500		435	

[a] Upper bound of each variable increased 100-fold for this test

[b] Upper bound of y_1 increased 100-fold; y_2 and y_3 increased 10-fold

Table 3

Summary of Problem C

$$F = 52.909653 - (y_1 - 0.5\,y_2 - 1.0)^2 - 2\,(y_2 - 2.04\,y_1 + 2.072)^2 - 3(y_3 - 2\,y_1 + 2.2)^2 - 4\,(y_1 - 0.13\,y_3 - 1.618)^2 - 5(y_2 - 1.5\,y_3 + 0.5)^2 - 6\,(y_3 - 0.196\,y_2 - 1.0864)^2.$$

$$0 \le y_i \le 2 \quad (i = 1, 2, 3).$$

$$C_1 = 5.80 - (y_1^2 + y_2^2 + y_3^2) \ge 0.$$

$$C_2 = (y_1 - 1.6)^2 + (y_2 - 1.5)^2 - 0.07 \ge 0.$$

$$C_3 = 2.445 - y_2 - y_3 \ge 0.$$

	Starting Values			Solution			Efficiency Criterion		Tallies	
Test	$\bar{y}_o$	F_o	$\bar{\delta}_o$	$\bar{y}_e$	F_e	$\bar{\delta}_e$	R	n_e	n_{ci}	n_f
C-0[c]	0.00	10.00	0.10	1.800	52.910	0.001	2000	296	-	230
	0.00		0.10	1.600		0.001	2000		-	
	0.00		0.10	1.400		0.001	2000		-	
C-1[d]	0.00	10.00	0.10	1.673	52.725	0.001	2000	296	350	325
	0.00		0.10	1.246		0.001	2000		-	
	0.00		0.10	1.200		0.001	2000		-	
C-2[e]	0.00	10.00	0.10	1.675	52.724	0.001	2000	296	250	207
	0.00		0.10	1.246		0.001	2000		220	
	0.00		0.10	1.200		0.001	2000		-	
C-3[f]	0.00	10.00	0.10	1.677	52.723	0.001	2000	296	290	264
	0.00		0.10	1.247		0.001	2000		280	
	0.00		0.10	1.196		0.001	2000		260	

[c] No constraints in effect.

[d] Constraint C_1 in effect.

[e] Constraints C_1 and C_2 in effect.

[f] All constraints in effect.

Nomenclature, Part 1

Note: The first part of this table defines symbols used in the descriptive part of the text. The second part defines FORTRAN expressions which have been used on the flow charts and also in coding the program.

B_u^+, B_u^- = Upper and lower bound functions, $y_u^+ - y_u$ and $y_u - y_u^-$.

C_i = Constraint function.

F = Objective or value function.

HF, LF, HN, LN = Designations of highest, lowest, feasible, nonfeasible points in "constraint" subprogram.

K' = Proportionality constant.

k = Number of basic variables.

n_{ci} = Tally on constraint function calculations.

n_e = Efficiency criterion, $k^s \log_2 R_{max}$.

n_f = Tally on value function calculations.

p = Gradient projection vector.

q = Number of constraint functions.

r = Parameter used in subprogram RIDGE.

R_u = Ratio of range of y_u to tolerance on y_u.

R_{max} = Largest R_u.

y_u = Upper bound for y_u.

y_u = Lower bound for y_u.

y_u = Independent or basic variable.

y = Operating vector $(y_1, y_2, \ldots, y_k)$.

δ_u = Basic step length for y_u.

$\bar{\delta}$ = Set of all δ_u.

δ_{1e} = Tolerance on first basic variable.

Subscripts

o = Initial conditions.

e = Final conditions.

Nomenclature, Part 2

BIG	=	Number beyond expected range of any variable (as 10^{30}).
C(K)	=	Constraint (function of Y).
D(K)	=	Initial increments for Y(K).
DIN(K)	=	Array for history of step sizes.
DORD(K)	=	Signed probe length, K ranging from 1 to KY2.
EDGE	=	Subroutine for "following" a constraint or confluence of constraints.
ENTER	=	Subroutine for publishing problem data and making initial calculations.
FAIR	=	Subroutine for publishing feasible advance.
FENCE	=	Subroutine for calculating constraints.
FINISH	=	Subroutine for publishing solution and making terminal calculations.
FOUL	=	Subroutine for publishing nonfeasible advance when seeking a feasible point.
I	=	Index number of basic variable y(I).
J	=	JORD(I).
JA	=	Variable affected in present probe.
JAR(K)	=	Array of the eight subscripts for YAR.
JC	=	Number of specified constraint.
JCON(K)	=	Order array for constraint calculations.
JORD(K)	=	Order array for probes.

Nomenclature, Part 2

KC	=	Number of constraints.
KERR	=	Error flag, numbered for statement which detects error, or for selected types of errors.
KEYC	=	1 when seeking feasible point, otherwise 2.
KEY23	=	2 to select Subprogram RIDGE, 3 to select Subprogram EDGE.
KORD	=	J, 200 or 300 in Subprogram PROBE, RIDGE, or EDGE on successful probe (otherwise 0).
KVIOL	=	1 for nonfeasible, 2 for feasible point.
KY	=	Number of basic variables.
KY2	=	KY + KY = range of JORD.
MQD(K)	=	Array for discrete variables (MQD(K) = 0 if y(K) is a continuous variable).
NC(K)	=	Tallies of constraint violations during major cycle.
NCT(K)	=	Tallies of calculations for individual constraints.
NLIST	=	Tally of listings (one per feasible advance).
NP	=	Problem identification number.
NVAL	=	Tally of VAL calculations during major cycle.
NVALT	=	Tally of VAL calculations during complete problem.
OBJECT	=	Subroutine for calculating VAL.
ORDER	=	Subprogram to set orders of probes and constraint calculations.
PROBE	=	Principal probing subprogram.
RIDGE	=	Subprogram for "following" a "ridge".
SAVE	=	Subroutine for calculating and storing quantities which will be needed for VAL and/or C(K).
STEND	=	Final tolerance on step size for first variable.
VAL	=	Value function or objective function.
VALHF1, etc.	=	VAL at Y(JAR(1)), etc.
VALS	=	Reference value of VAL
Y(K)	=	Basic variable.
YA	=	Y(JA).
YAR(K)	=	Array of the eight probes Y(JAR (1)), etc. See Figure 2 for classification of critical probes.
YH(K)	=	Upper limit for Y(K).
YL(K)	=	Lower limit for Y(K).
YS	=	Reference value of Y(JA).

References

1. N. R. Amundson, American Institute of Chemical Engineers, Tenth Annual Institute Lecture, Cincinnati, December, 1958.
2. S. H. Brooks, "A Discussion of Random Methods for Seeking Maxima," Operations Research, vol. 6, p. 244; 1958.
3. R. Dorfman, P. A. Samuelson, R. M. Solow, "Linear Programming and Economic Analysis," McGraw-Hill, New York; 1958.
4. R. O. Ferguson, L. F. Sargent, "Linear Programming Fundamentals and Applications," McGraw-Hill, New York; 1958.
5. S. I. Gass, "Linear Programming Methods and Applications," McGraw-Hill, New York; 1958.
6. Clifford Hildreth, "A Quadratic Programming Procedure," Naval Research Logistics Quarterly, vol. 4, p. 79; 1957.
7. A. S. Manne, "Scheduling of Petroleum Refinery Operations," Harvard University Press, Cambridge; 1956.
8. J. B. Rosen, "The Gradient Projection Method for Nonlinear Programming; Part I, Linear Constraints," J. Soc. Industrial and Applied Math, vol. 8, p. 181, March, 1960; "Part II, Nonlinear Constraints," to be published.
9. T. L. Saaty, "Mathematical Methods of Operations Research," McGraw-Hill, New York; 1959.
10. E. Singer, "Simulation and Optimization of Oil Refinery Design," Shell Development Co., Publication P-839, Emeryville; 1959
11. I. S. Sokolinkoff, and E. S. Sokolinkoff, "Higher Mathematics for Engineers and Physicists" (Second Ed.), McGraw-Hill, New York; 1941.
12. Henry Stone, Personal communication.
13. Philip Wolfe, "Computational Techniques for Nonlinear Programs," Princeton Conference on Linear Programming, March 13-15, 1957.
14. W. S. Dorn, "Non-Linear Programming," Research Report RC-266, IBM Corp., Yorktown Heights, N. Y., May 24, 1960.
15. R. E. Gomory, "Outline of an Algorithm for Integer Solutions to Linear Programs," Bulletin of the American Mathematical Society, vol. 64, p. 275; 1958.
16. R. Bellman, "Dynamic Programming," Princeton University Press; 1957.
17. R. E. Griffith, R. A. Stewart, "A Non-linear Programming Technique for the Optimization of Continuous Processing Systems," Management Science, vol. 7, p. 379; 1961.

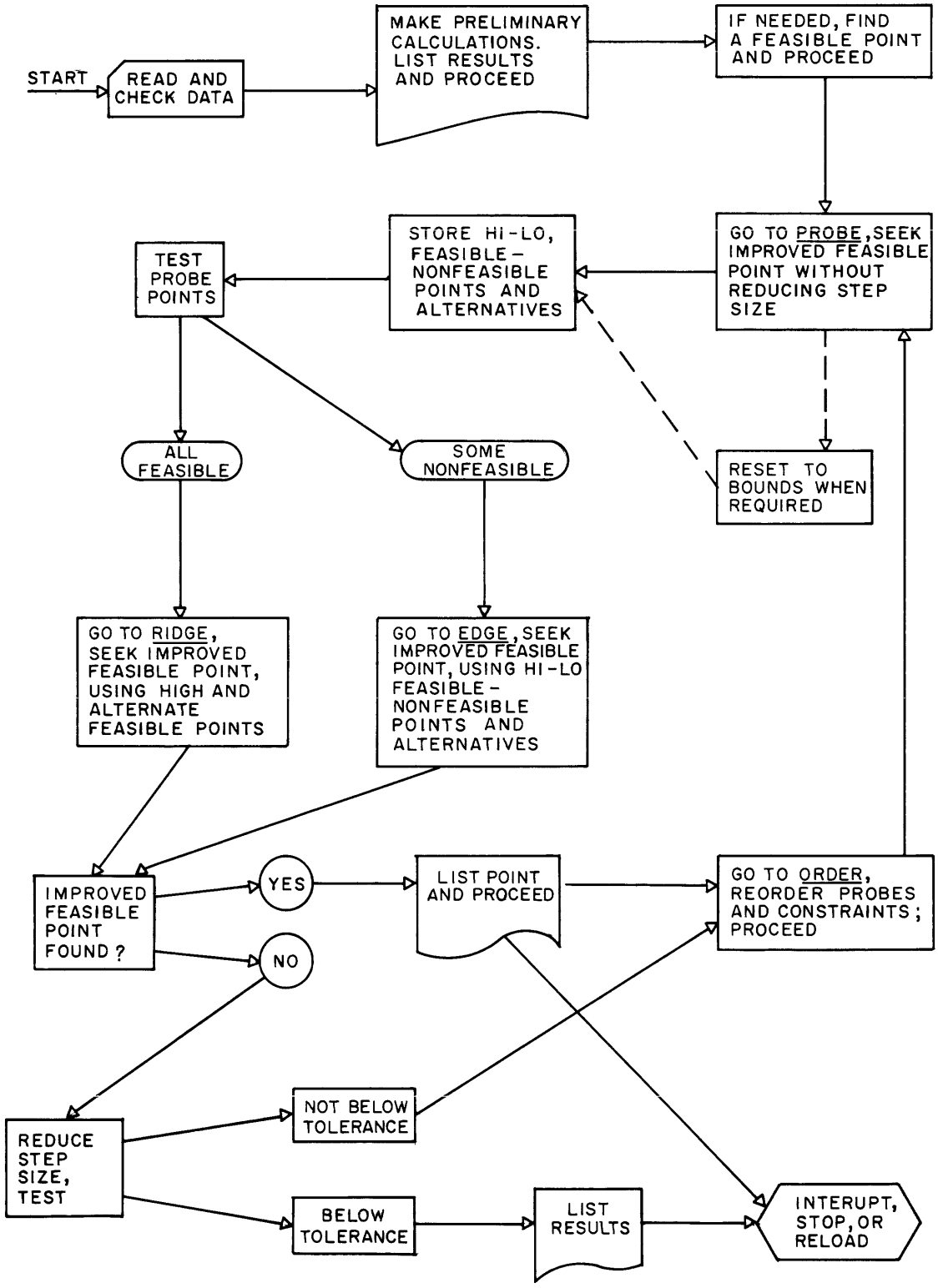


Fig. 1 Probe method for nonlinear constrained optimization

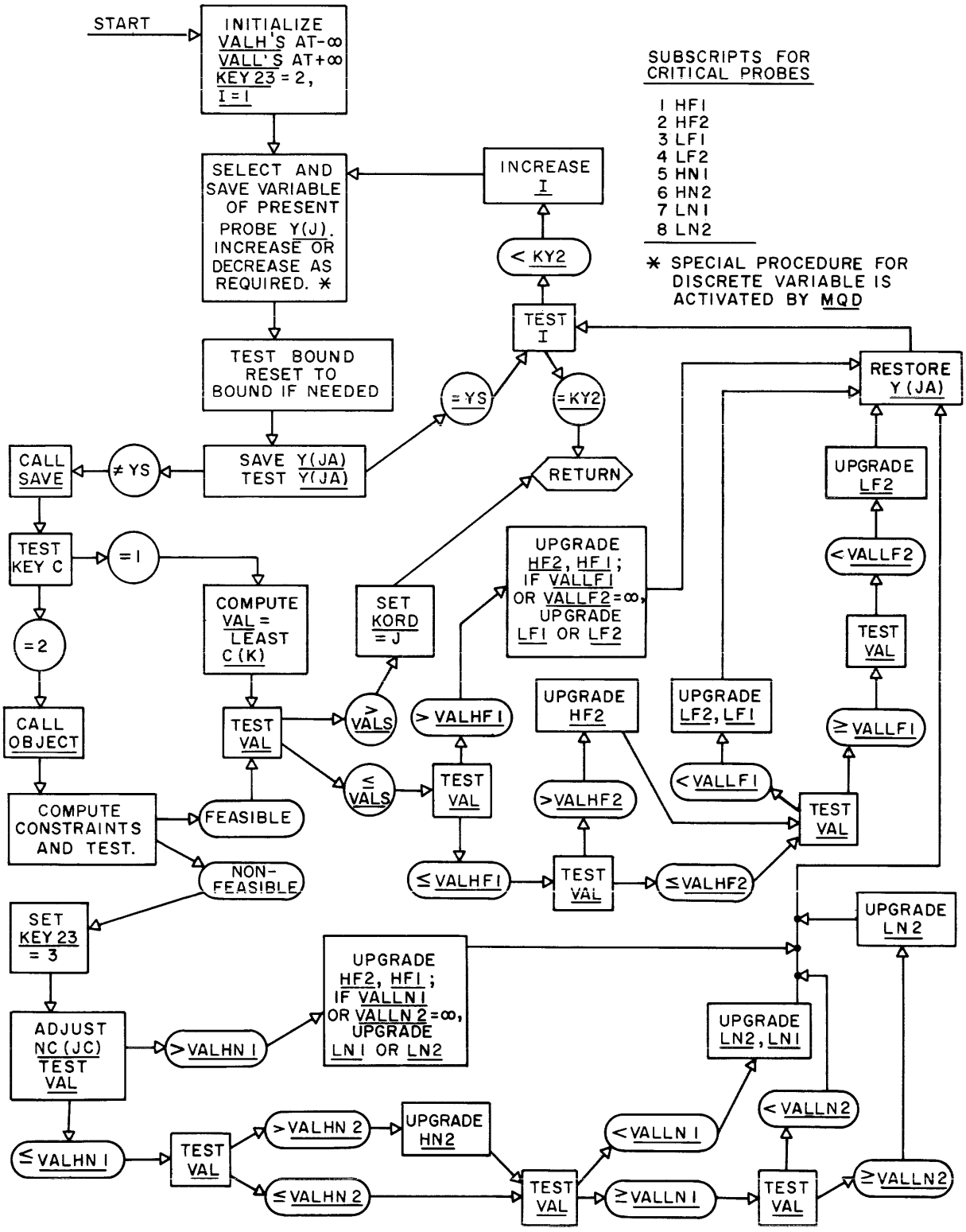


Fig. 2 Subprogram PROBE

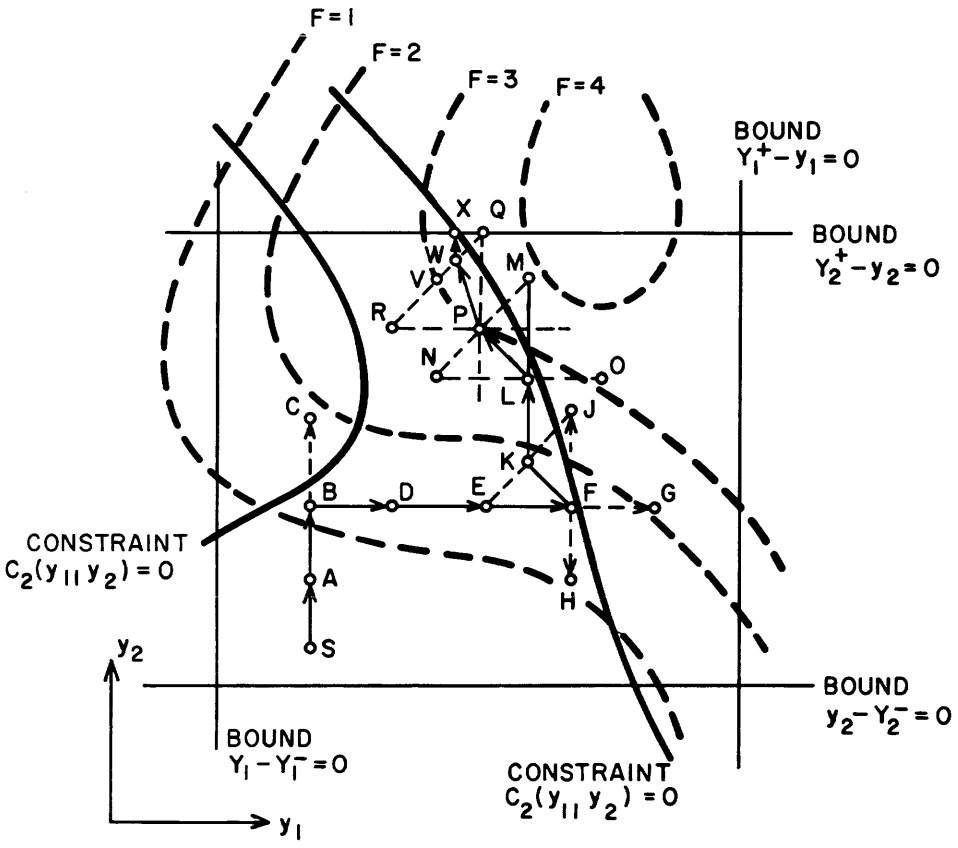


Fig. 3 Operation of subprograms PROBE and EDGE

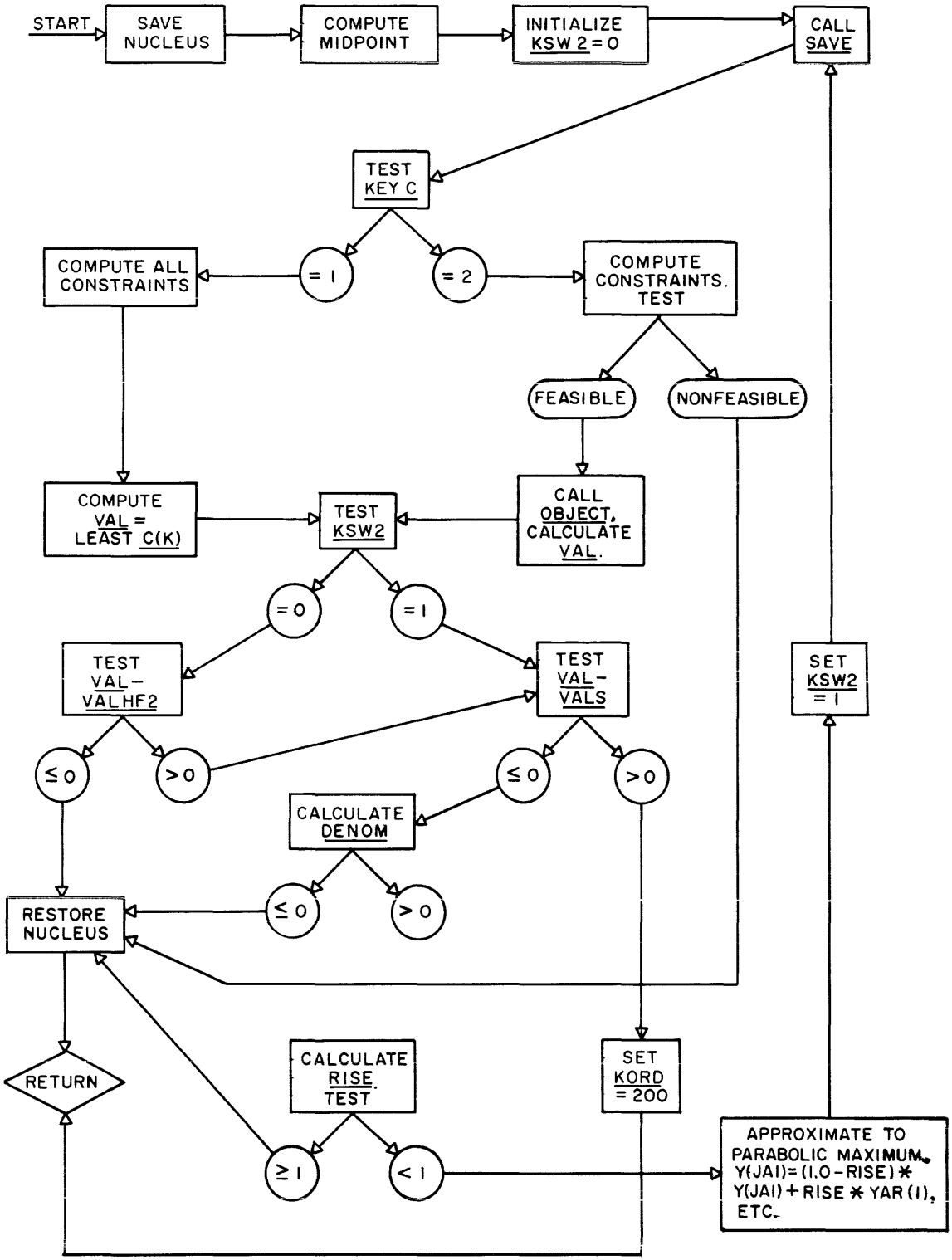


Fig. 4 Subprogram RIDGE

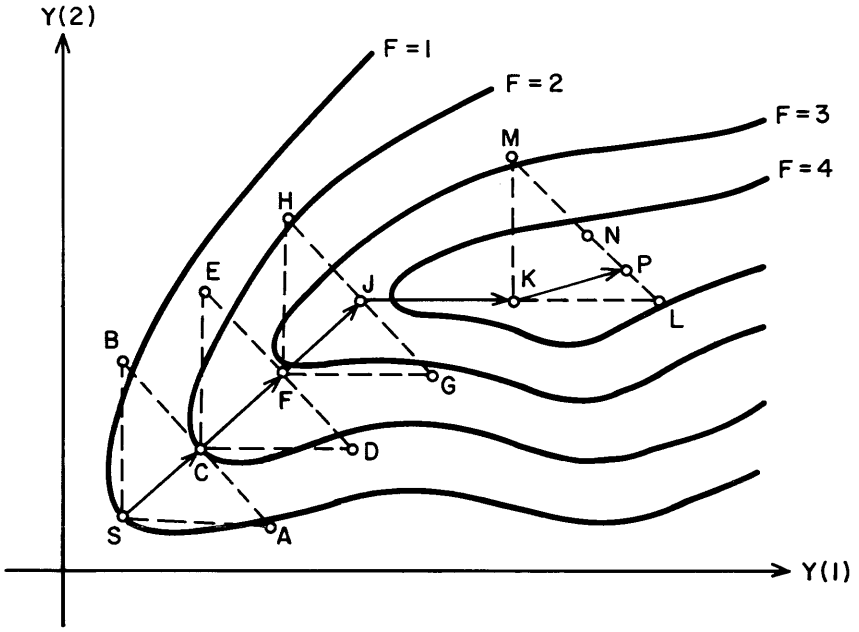


Fig. 5 Operation of subprograms PROBE and RIDGE

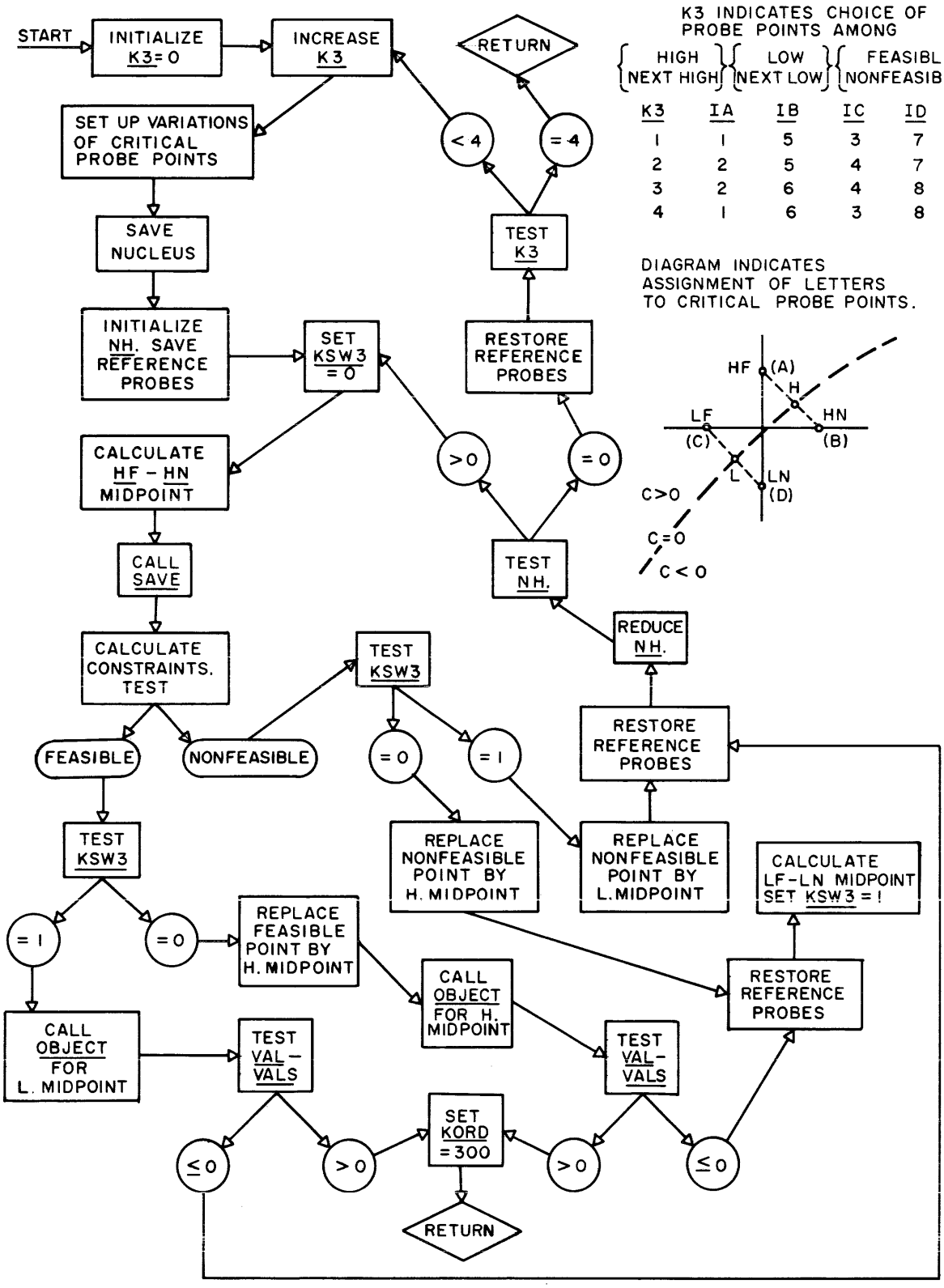


Fig. 6 Subprogram EDGE

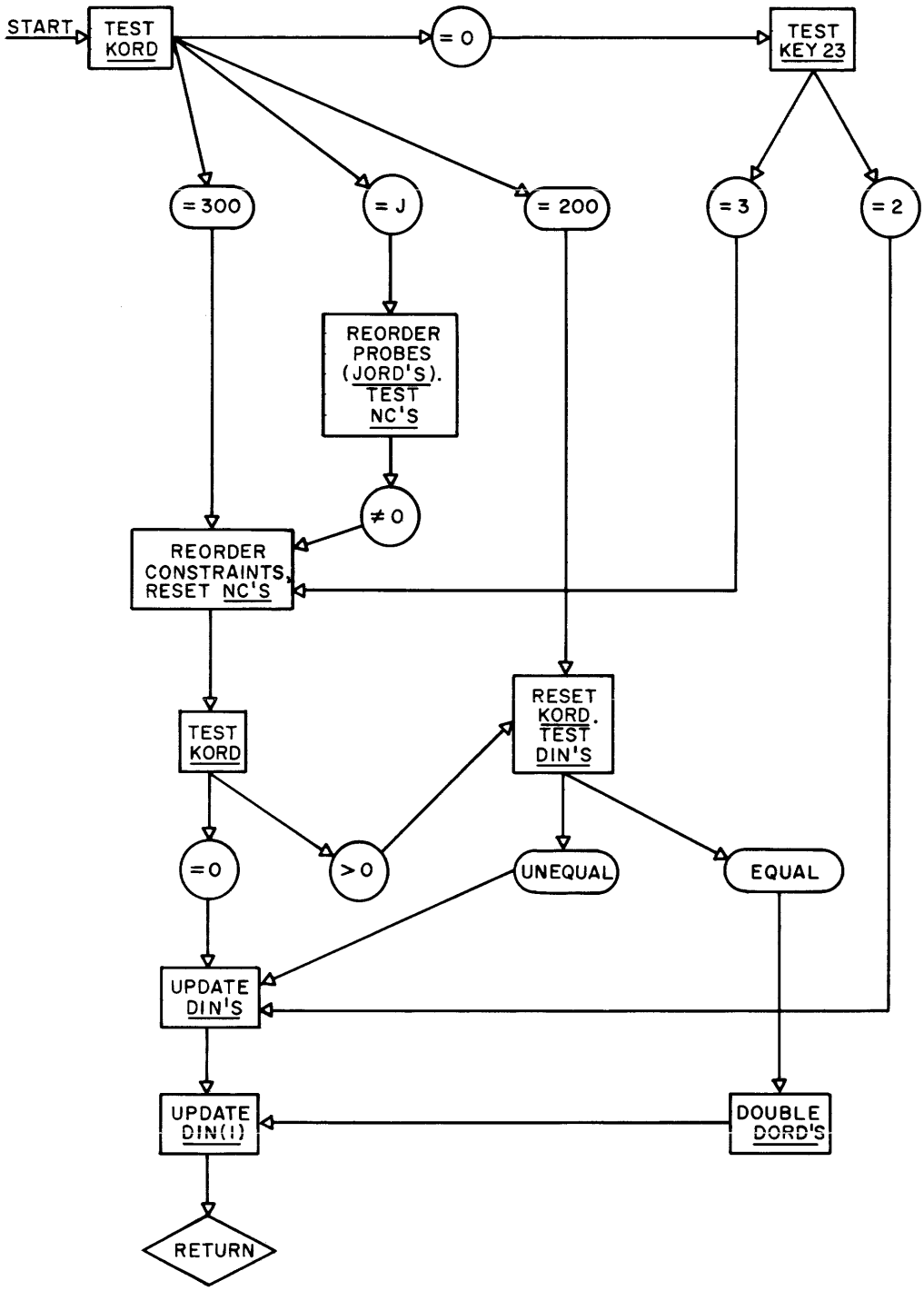
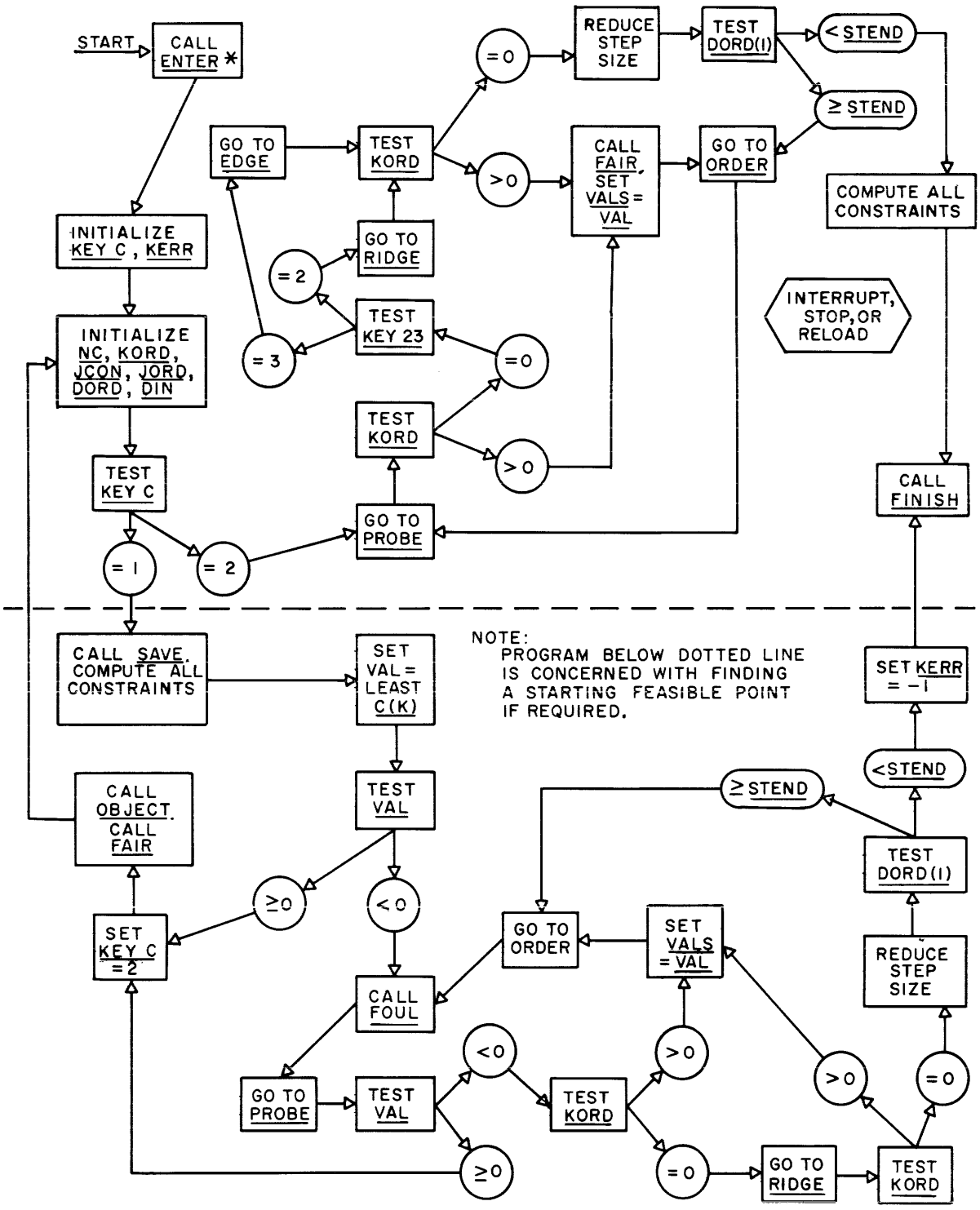


Fig. 7 Subprogram ORDER



* SUBROUTINE ENTER SHOULD CONTAIN ALL PROCEDURES REQUIRED FOR LOADING AND CHECKING DATA, LISTING DATA, AND COMPUTING QUANTITIES THAT WILL BE CONSTANT DURING THE OPTIMIZATION. IT SHOULD ALSO INITIALIZE TALLIES CONTROLLED BY FAIR, FOUL, ETC.

Fig. 8 Probe method--main program

A SIMULATION OF A BUSINESS FIRM

Charles P. Bonini

Graduate School of Business

Stanford University

Summary

This paper describes a simulation model of a hypothetical business firm. The model was constructed to include not only the accounting and economic factors of costs, profits, sales, units produced, etc., but also psychological and behavioral concepts. Individuals in the firm have aspiration levels, feel pressure, and react in accordance with behavioral theory.

The purpose of the model is to study the effects of informational and organizational factors upon the decisions of a business firm. We have had limited knowledge of such variables as: the effects of tardy information, the effects of different distributions of information within the firm, the effects of differing degrees of centralization or decentralization, etc. A comprehensive model, such as the one proposed, is necessary to answer such questions.

Eight specific hypotheses involving changes in the organization and information system of the firm were formulated and tested using a factorial experimental design. The results of this experiment demonstrate the usefulness of this model as a research tool.

Introduction

In studying information systems in business firms, management accountants and data processing students have been rightly concerned about the effects of informational variables upon decision-making. It is generally agreed that factors such as: kinds of information received; amounts of information received; distribution of information; and the timing of said information are important in designing a data system which will enable management to make good or "best" decisions. But little evidence, either theoretical or empirical, has been advanced about these factors.

On the one hand, the theoretical models that have been constructed (such as the Marschak-Radner Theory of Teams[1]) have been too simple to be of use in studying information flows in the large complex business concerns of today. The experimental work, too, has been largely limited in scope, dealing with simple information in simple network structures.

On the other hand, much progress has been made in recent years in the data useful for what might be called micro decision-making. That is, decision-making at the individual level. Part of this progress has been the result of the development of Management Accounting, especially budgeting and cost accounting. Another part has gone hand in hand with the rise of Management Science and the development of mathematical and statistical models for decision-making. But the area which could be called macro decision-making has been largely ignored. Here we are concerned not only with information's effects upon a particular decision-maker, but also its effects upon the business organization as a whole.

The need, then, is for a model or framework that the theorist can use to study the effects of information and related organizational factors upon decision-making in the whole firm. Such a model must be:

1. A complete systems model, so that effects can be traced throughout the firm

2. A model embodying psychological and behavioral principles--since the manner in which information is transformed into action should be included

3. A model complex enough to fairly represent the real world.

The model described in this paper is an attempt to meet these criteria.

Note that the information theorist is not the only one to benefit from such a model. Organization theory, also, needs a formal model within which to study the effects of changes such as: changed emphasis of authority, or the question of centralization versus decentralization. Also, in economics, we see numerous areas where a complex model of the firm could be of much use. The economic theories concerning behavior of business firms in oligopoly market situations has been seriously handicapped because they lack the inclusion of factors internal to the firm. (In oligopoly markets, the firms have some influence over the market. This influence is often an offshoot of the organizational structure of the firms, the information systems of the firms, etc.) However, economists are beginning to realize the need for more complex models.[2] The sub-discipline of the Behavioral Theory of the Firm is becoming an accepted part of economics.[3] It is within this framework that the present model is to be viewed.

The Model

We have described, in part, the function that an adequate model of the firm can fill. In this section, we shall discuss the particular

model that we have constructed. The method of experimentation with the model is to change its components. For example, we could change the formal organization or a particular link in the information system. Thus, for our study, it is not the specific parts of this particular model that are important, but the general form by which the model is constructed.

Ideally, the model of the firm is an attempt to build a formal structure, using as much of the inherited bodies of theories from Economics, the Behavioral Sciences, Organization Theory, and Management Accounting as are relevant. Such a model is so complicated that we could not manipulate it analytically. Thus, we must be content with a simulation study, intended to represent the behavior of the firm over a simulated period of time.

The formal organizational structure of our model is shown in the chart. Essentially, the company contains (a) an executive committee that does the planning, the co-ordinating, and acts as the "top" controlling function of the firm. The (b) sales branch contains a general sales manager, seven sales districts (each headed by a local manager), and some forty salesmen. The (c) production division contains a vice-president for production, a plant manager, five foremen, and an industrial engineering department. The decisions made by each are described in general terms in the chart.

Each "decision-maker" (person or group) in the organization behaves according to a specific predetermined procedure, which we shall call a decision rule. Information, gleaned from the information system of the firm, is fed to the decision-maker and enables him to use the decision rule. (For example, a vice-president may abide by the decision-rule--"Set price 10% above cost." To utilize the rule, he must know what figure represents cost.)

How are behavioral concepts built into the model? As an example, the executive committee has a set of decision rules by which it controls the firm. The decision rules blend psychological theories on aspiration level and observable business practice. The specific profit goal of the firm is a function of past profits. Now, if by estimates from various departments, the projected profit does not measure up to the executive committee's goal, modifications in their plans are tried. (I.e., if a gap between aspiration level and reality exists, search is initiated.) They may: (a) cut costs, (b) try to increase sales, (c) change price, or (d)--if all else fails--lower the profit goal. This process continues until a satisfactory projected profit is reached.

Another behavioral concept built into the model is pressure. Each individual has an "index of felt pressure." This index is intended to represent the sum total of all the various pressures exerted upon the decision maker. As a specific example, the factors making up the index for a salesman are listed below, together with the weights attached to each factor.

Index of Pressure for a Salesman

Factor	Weight
1. Index of pressure of his superior (the district sales manager)	25
2. His quota relative to his sales in the past month	40
3. Sales of the average salesman in his district relative to his sales	10
4. .75 + (percentage of his products less than 75% of quota)	10
5. His total quota for the past quarter relative to his total sales for the last quarter	15
	100

These represent, more or less, the formal factors affecting the salesman. Of course, a large number of informal factors could theoretically be added, such as personality conflicts with his superior, etc. Note how the index provides a connection between information flows in the firm and a behavioral variable, namely pressure. As pressure mounts, individuals react (although not to the same degree or in the same way). As pressure diminishes, a phenomenon called "organizational slack" gradually develops and results in inefficiency within the firm. An example of organizational slack would be a tendency toward "empire building": (i.e. when business is profitable, the occurrence of excessive advertising, over-staffing, etc.) Also, information is transformed into action via pressure. Thus, a salesman feels pressure depending upon reported sales, costs, and profits (information); the salesman may then expend more sales effort (action to conform with what is expected from him).

In addition to psychological concepts, a variety of business rules are inculcated into the model. Thus, the decision rules by which budgets are set, production scheduled, and standards determined are inherited from business practice. Market behavior is represented in accordance with standard economic demand theory.

Computation and Experimental Design

Our model was programmed in FORTRAN to run on the IBM 7090. (Each run of 108 periods took about 2/3 minute.)

Eight hypotheses (representing changes in the organization or information system of the firm) were selected for study. Space limitations

prohibit our explaining the eight major hypotheses of the model in detail. We shall enumerate them, however:

Changes in the External Environment of the Firm:

1. External World Variability. (Stable costs and sales versus fluctuating costs and sales.)

2. Market Growth Trend. (Slow cyclical pattern of market growth versus fast irregular growth.)

Changes in the Decision System:

3. Loose vs. Tight Industrial Engineering Department. (Loose or tight standard costs.)

4. Amount of Contagion of Pressure in the Firm. (Degree to which an individual transmits his "felt pressure" to his subordinate.)

5. Sensitivity of Individuals to Pressure.

Changes in the Information System:

6. LIFO vs. Average Cost Method of Inventory Valuation.

7. Knowledge by the General Sales Manager of the Company Inventory Position.

8. Emphasis Upon Current Period Information vs. Past Information for Control Purposes.

A 2^8 factorial experimental design was utilized to evaluate the results.

When testing the outcome of a change in our model, the question arises, "Is this result applicable to only this specific type of firm?" By use of the factorial design we have built in eight hypothetical firm "differences" the combinations of which can give us a large number of firms. Now we can test our result against a "many-faceted" background of firms to see if the result is consistent.

In essence, then, our model is similar to a laboratory experiment. We can control all the factors of a "real-life" business firm, make changes, and record the effects of these changes. Again, while our model is still a simplification of reality, it is more complex and can answer a broader range of questions than before.

Some Significant Results

The effects of the experiment as they relate to information theory are:

Effect of LIFO

The major effect of the use of LIFO versus the average cost method of inventory valuation was a significant increase in sales, pressure, and profits within the firm. And since the model did not include taxes, this increase in profits was not attributable to tax benefits.

How was this result achieved? LIFO, when used for internal reporting purposes, increased the variability of profits from period to period.* The average cost method, on the other hand, tended to smooth out costs (by averaging) and hence produced more stable profits from period to period. The fluctuating profits under LIFO tended to keep the firm "on its toes" and quick to take advantage of cost saving and profit making opportunities. The firm with the more constant profit picture was more sluggish.

The importance of this result is not that it proves LIFO as the best policy, but that some surprising unintended results came from what appeared to be a simple accounting procedure. The result was achieved because the information affected others in the firm than those intended--those external to the firm. Tax savings and external reporting should not be the only considerations in deciding a policy of inventory valuation.

Knowledge by the General Sales Manager of the Company Inventory Position

There was no significant difference in the sales or profits of the firm depending on whether or not the general sales manager knew of the inventory position of the company. The conclusion is that this information (about inventory) was not useful to sales in this case and may not be useful in others. Future research can specify when such information is useful and when it is not.

Loose vs. Tight Production Standards

We had the Industrial Engineering Department tighten up its standards with a resultant cost cut. However, profits did not rise. Why? By producing cheaper, one need sell less to make the same profit. And the sales force felt the ease in pressure and equaled out the cost cut by lower sales. The implication for organizational theory? Perhaps a "cost-cut type" campaign is not effective when viewing the firm as a whole. As then, alternatively, pressure on the sales force may cause a simultaneous "ease-up" for the production people.

---

*LIFO is often used for external reporting only, whereas we used it for internal reports also. In addition, it is possible that the same result achieved by LIFO, could be obtained by FIFO or by a standard cost system. Future research will investigate these possibilities.

This result poses an interesting question for information theorists. On the surface it would seem desirable to give individuals information about the progress of the whole organization. They could thus see their contribution to the over-all effort and be motivated to perform better.

But the result here is exactly the opposite. The effect of a cost reduction is balanced by an "ease-up" in the sales department. Knowledge of the over-all profit level caused poorer performance, rather than the reverse.

Conclusion

On a small scale we have set up a reasonable facsimile of how individuals behave within a firm. We have used this model to study changes in the information and decision system of the firm, with some surprising results. These results are at least suggestive of phenomenon that can happen. They do open up new fields of inquiry. And they show that this type of model is quite valuable as a research tool.

Acknowledgements

This research has been sponsored in part by the Office of Naval Research and in part by the Western Management Science Institute. The computation was performed at the Western Data Processing Center, University of California at Los Angeles.

The author is deeply indebted to R. M. Cyert and W. W. Cooper for guidance and innumerable suggestions during the course of the research. In addition, Fred M. Tonge and Daniel Teichroew offered helpful comments on this paper.

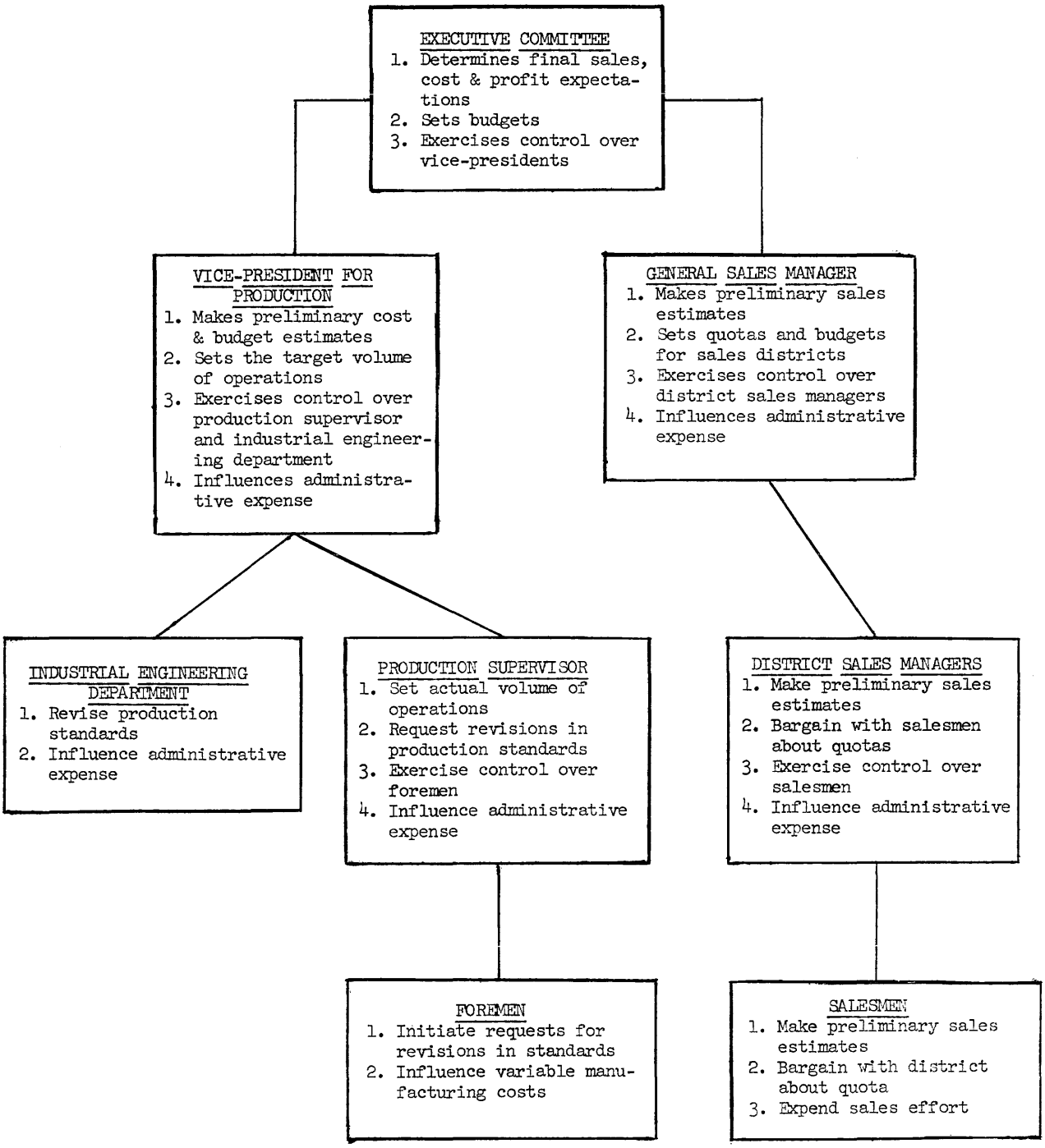
References

1. Marschak, Jacob, "Elements for a Theory of Teams," Management Science, Vol. 1, No. 2, January 1955

2. Cyert, R.M. and March, J.G., "Organizational Factors in the Theory of Oligopoly," Quarterly Journal of Economics, Vol. 70, February 1956

3. Cyert, R.M., Feigenbaum, E.A., and March, J.G., "Models of a Behavioral Theory of the Firm," Behavioral Science, April 1959

ORGANIZATION OF THE SIMULATED FIRM
(including types of decisions at each level)



MH-1, A COMPUTER-OPERATED MECHANICAL HAND*

Heinrich A. Ernst

International Business Machines Corporation

San Jose Research Laboratory

San Jose, California

SUMMARY

MH-1 is a motorized and sensitized servomanipulator operated by the TX-O computer at the Massachusetts Institute of Technology. It serves as an experimental vehicle to explore the feasibility of direct relations between a digital computer and the physical world with which this computer is concerned. Usually, a human interpreter stands between the computer and the physical world. Instead, the TX-O computer in the MH-1 system is programmed to perform by itself some of the functions normally assigned to the human intermediary; namely, to perceive the world, to appreciate it, and to determine a reasonable course of action after a goal has been specified for the hand. The data processing tools used are, rather than numerical operations on quantitative signals, pattern recognition and simulation of higher cognitive processes such as awareness and understanding. This paper describes some of the experiments performed with MH-1 and the mechanisms upon which the capabilities of MH-1 are based.

INTRODUCTION

The idea of building a mechanical hand to be operated by a digital computer was originally presented by Shannon and Minsky during a seminar at the Massachusetts Institute of Technology in the fall of 1958. This paper is a report on the work on that project carried out during 1960 and 1961, and on the results of it.

Our first goal was simply to build such a hand-computer system, but we soon realized that a great many problems were involved, more than we could pay attention to at one time. A closer study of some of them revealed several specific subgoals of particular interest, and other areas that could be neglected for a while. We chose to concentrate on the information-processing problem of the system. One aspect that we decided to neglect as far as possible was the instrumentation problem, although many rewarding studies can be made in this area. What, for instance, is a really good mechanical substitute for a muscle? We bypassed this and similar questions by purchasing a mechanical servomanipulator which we then provided with sense organs and motors.

We shall now look at the information-processing problem arising in connection with the mechanical hand, first from the point of view of the designer of automata; then from the point of view of the control engineer; and finally from the point of view of the computer engineer. This analysis will enable us to understand why we thought that the information-processing problems would be of particular interest, and what specifically these problems are.

AUTOMATA AND MECHANICAL HANDS

The first mechanical arms that we know of were built by the ancient Egyptians, as parts of statues of Gods, and they were operated by priests who were assumed to be acting under divine inspiration.[1] It is hard to say now what exactly all that meant to the Egyptians, but it does point out the existence of some kind of primordial interest in machines resembling human beings and imitating human actions. I can testify to the fact that this interest still exists, at least in most of the people who have met my mechanical hand.

Many centuries later, in the Hellenistic period, we find several schools of creators of hydraulically operated statues. These statues were originally (especially under Heron of Alexandria) built to illustrate the science of hydraulics. Their description was patterned after the language used in geometrical proofs, involving axioms and theorems. It was not the original intention of the designer to imitate either the internal mechanisms nor the external appearance of human life. Revolving doors were as interesting as walking statues. The design of these automata was an intellectual exercise, the models served as teaching aids. Since most of these machines were stored in temples, eventually they did not fail to fascinate and entertain the worshipers, and during the decline of classical Greek culture, some might even have been built for that sole purpose.

---

* This paper is based on a thesis submitted in partial fulfillment of the requirements of the degree of Doctor of Science in the Department of Electrical Engineering at the Massachusetts Institute of Technology on December 23, 1961. The work was supported in part by the U. S. Army Signal Corps, the Air Force Office of Scientific Research, and the Office of Naval Research.

The most intricate mechanical puppets were built in the eighteenth century in France, Switzerland, and Germany.[1] Some of them had moving forearms permitting them to write and draw sketches, some even had moving fingers with which they played small musical instruments such as organs or flutes. The actuating mechanisms were hidden from the spectators and were of no interest to anybody except the designer. These mechanisms consisted of intricate arrangements of cams and levers, or even stacks of cams chosen by a selector, much like a computer disc-memory.

All of these automata were built with the intention of presenting a convincingly human performance. The designers succeeded marvelously in their objective. People traveled for days just to see these charming mechanical wonders. The presentation of one of these machines to a monarch or prince would practically guarantee a lucrative existence to the builder for the rest of his life.

Nobody, however, was interested in investigating the significance of these mechanisms; nobody was dissatisfied with the fact that these automata were completely insensitive to their environment. All of the control action went from the inside, from the hidden control gear, toward the outside, to the limbs or the fingers. There was no information flowing in the opposite direction from the outside world into the control mechanism. There was, to use the terminology of our time, no capacity of discernment, although Droz's puppet could write: "Je ne pense pas, ne serais-je donc pas?", in imitation of Descartes' "Cogito, ergo sum."[1]

NATURAL VERSUS TECHNICAL FEEDBACK MECHANISMS

The next step in the development of automata was the discovery of the possibility of using signals that monitor the performance during the control process. This gave rise to what is now called feedback control systems.

If we were to state the purpose of such a system as generally as possible, we could say that it is to achieve a desired output state despite the interference of unwanted disturbances. The purpose of a positioning device, for instance, is to reach a certain location despite varying friction, noise entering the control signal, variations in many components of the system, external forces tending to drive the system away from the desired position, and so forth.

Recent neurological studies have shown that similar mechanisms operate in human beings and animals, for instance, in the pupillary system. It is therefore possible to build useful mechanical arms and hands based upon such control principles.[3,5] All that one has to do is to preprogram a sequence of desired states or goals which the control system will reach one after the other, and thus eventually build up a complicated motion. Examples of such machines are "Noman,"[2] or the tape-controlled servomanipulator built by Borg-Warner.[6] They are actually automatically controlled machine tools with claws instead of metal-cutting devices.[4]

Instead of the mechanical cams found in the automata of the eighteenth century (or, for that matter, in early automatic machine tools) these new "mechanical hands" use an electrically prerecorded trajectory. However, neither the introduction of electrical signals nor the introduction of feedback, changes anything in the fundamentally preprogrammed nature of these devices, although the programming becomes much more flexible and can even be made to depend on external factors, as far as these can be foreseen. Feedback improves the performance in the framework of the program, but it does not improve the program itself.

Humans and animals do not seem to operate in such a preprogrammed, step-by-step mode. In addition to this difference in operation, the automatic devices that we have explained do not possess on a higher level the property required of a feedback control system; namely, to resist all kinds of disturbances. There is nothing in these preprogrammed devices that resists disturbances unforeseen during the selection of a specific step-by-step procedure. If such a machine, for instance, is to fill boxes with nuts, the interruption of the supply of boxes will probably completely upset the operation of the machine, or, at best, switch off if the programmer was sufficiently circumspect. But a squirrel filling a hole with nuts will not be confused in this situation, it will dig a new hole. A human being can get to work on a rather broad description of what he is to do, instead of requiring a detailed step-by-step program. He fills in the details by himself, as he goes along. He will be able to cope with minor or major deficiencies in this job description.

In other words, besides being to some extent machines containing classical feedback control mechanisms, animals and humans have other, more advanced information-processing capabilities they can bring to bear upon the problem of controlling their actions. They can perform decisions, remember earlier, similar situations, use trial and error procedures, and, in general, perform "intelligent" processes that greatly enlarge their capabilities of getting around more complicated difficulties that arise while they pursue a certain goal.

One purpose of this work is to learn how we can build some of these higher level control facilities into our hand-computer system. Is it possible to design a system that behaves sensibly with respect to the overall goal, even if it should turn out that the preprogrammed sequence of operations is inadequate or incomplete? Could the computer do some of the detailed sequencing by itself, as the activity is pursued, interpreting the original broad job description in accordance with the current state of its environment?

THE POINT OF VIEW OF THE COMPUTER ENGINEER

To the computer engineer, the same problem appears in quite a different light. Digital computers are almost exclusively operated in a human environment. An engineer or a scientist formulates a mathematical model of the physical world, the numerical analyst chooses the proper numerical procedures to solve the equations, the programmer is responsible for converting them into machine language, and, after the results appear, the user applies them and interprets them in relation to the real world.

Since MH-1 works in a real-world environment, the computer will have direct access to the real world. We want to build a system that works in the real world without the help of human intermediaries. Why? If the digital computer is used, not just as a processor of quantitative signals, as usual in conventional digital control systems, but to perform all the higher control tasks, beginning with the perception and the appreciation of the real world and concluding with the performance of a purposeful task in the real world, the computer theorist stands to gain some insight from this study himself. Computers have, as shown, relied exclusively on their human operators for most of the applications. If one couples a computer to something other than a human environment, for instance, to the real world directly, it is conceivable that, if difficulties arise in the execution of a program, the computer could gather the necessary additional information from the medium to which it is coupled rather than call for human assistance, and thus get around the difficulty by itself.

In other words, it is now commonplace to complain about the "stupidity" of digital computers - that they don't realize it if what they do makes no sense at all, and commit the most nonsensical errors if there is any mistake in the program. It is our thesis that this is not caused by an essential shortcoming of digital computers, but by the lack of necessary information that has not been given to the computer. No computer has yet any access to abstract mathematics, and therefore it cannot possibly understand whether or not what it does makes any sense. However, the hand-computer system has access to the real world, it can therefore be programmed to sense by itself whether its actions make sense in the real world or not. It should therefore be possible to program the computer in a way which makes it suffer less from that "stupidity" in the case of a small programming error or of some unexpected situation.

After all of these explanations, let us now summarize the problem with which we are confronted. We showed that both the computer and the mechanical hand can profit quite a bit from being coupled together in an intelligent way. To achieve this, we have to:

1. Design a hand that is, first, an efficient motor organ in the real world, adapted to the nature of the commands coming from the computer, and, second, a sense organ for the kind of information which the computer needs for its control activities.

2. Design a programming system that takes care of the classical control requirements of the system, as well as of the simple discernment, processing, and decision tasks associated with the control problem of the level of complexity between, say, stating a task in a normal language, and setting the parameters in the control system in accordance with the situation in the real world and the requirements of the specified task.

As compared with the designers of the clockwork automata, we are more concerned with the internal structure of the automata and their significance in terms of the study of cognitive processes, and much less with their external appearance. The latter will only be a criterion of the successfulness of our procedures, not an aim in itself.

THE EXPERIMENTAL EQUIPMENT

There are no analytical tools that show us how to build that hand, but there are two other methods; armchair speculation and experimentation. We find that there is already enough of the first and hardly any of the second; and, besides, experimentation is a good deal safer for reaching conclusions.

Most spectators suspect that there is a large element of play in our experiments, but they fail to see the justification of it. Usually, if one conducts an experiment, one has some definite idea in mind which he wants to test. Once the experimental apparatus exists, the main task of the experimenter is taking data. In our case, the most important function of the experiments is to get a feeling for such a system, for how it works, and what its possibilities and limitations are. Once the experimental apparatus exists, the main task of the experimenter is to play with it. From the experiences gained thereby, he will be able to form specific ideas which he then tries out by building a new system. By playing around with the new system, he will get new information about what is

wrong with it and again try to improve it. This procedure not only tests any ideas, but its results provide a guide line for the next step, and both of these contributions, the testing of ideas and the catalytic action of generating new ones, are acutely needed in this work.

With these ideas in mind, we proceeded to build the necessary apparatus. The MH-1 system consists of the TX-O computer, a control unit and a servomanipulator. The TX-O was chosen because of the flexible in-out equipment and because of its availability (the total computer sign-up time was in excess of 500 hours). Since for several reasons interpretive programming techniques were used throughout the work (the speed of operation is determined by the external equipment and not by the speed of the computer), the relatively small memory capacity (8000 words, 18 bits each) of the TX-O never even came close to exhaustion.

The control unit selects the motors and sense organs according to the computer instructions, and generates the servomotor voltages. A block diagram of it is shown in Fig. 1. The unit contains 150 transistors, 300 diodes, and 28 thyratrons, and occupies almost two-thirds of a relay rack.

The mechanical arm consists of a motorized American Machine and Foundry Servomanipulator, shown in Fig. 2. Each of the seven servomotors drives one degree of freedom, and one potentiometer is coupled somewhere to each degree of freedom. Low-quality low-resolution potentiometers are used because it is felt that organisms do not have a very accurate sense of position. Whenever possible, we tried to imitate the properties of living systems when similar design decisions had to be made, since after all the animal system is an existence proof of the solvability of our problem.

More important than the sense organs for position, are the sense organs for touch which are spread all over the hand. The hand is shown in Fig. 3 and explained by Fig. 4, as far as the nature of the sense elements goes. A binary sense of touch is satisfactory, since the applied forces are better determined in a quantitative way by determining the motor input instead of the pressures exerted on the sense elements. Notice that all the possible feedback loops go through the computer.

The job of building the equipment took the better part of the year 1960. The mechanical part and the motor action of the hand did not present any novel problems, and neither did the control unit. But considerable experimentation and modification was needed in the construction of the sense organs, as our philosophy of their operation and function changed gradually during the experimental phase, and major changes would have had to be made in the sensory apparatus if the work had been continued beyond that reported here.

DESCRIPTION OF SOME EXPERIMENTS

MH-1 has gone through several states of increasing program sophistication. We shall describe some of the experiments with the last programming system, and follow it up with the explanation of the underlying principles.

In order to be able to evaluate the progress made with the different programming systems, we chose one standard activity for MH-1: To play with wooden blocks and boxes. MH-1 builds small structures out of these blocks, or puts them away into the boxes. The following program, for instance, will locate a box on the working table and then grope for the blocks and put them into the box.

Program No. 1

Instructions	Description
1. goto ml,mL	locate box (mL)
2. scan ml, md	determine size and location of ml
3. goto m2, mL	grope for blocks (m2)
4. scan m2, md	determine size and position of m2
5. take m2, hld	take the block and hold it
6. goto ml, mL	go to ml with m2
7. put ml, mL+ xb+ yh+ zd	put m2 into ml
8. goto m2, 0	go to the former position of m2 and grope for new block
9. transfer to 4	

In more detail, the following events occur during the 9 phases:

1. MH-1 starts a searching motion for an object (the box) which it will call ml after it has found it. If ml should have been scanned earlier, it would go there directly.

2. MH-1 determines the absolute position of ml so it can find it again later, it determines its position with respect to the hand and its size with respect to the maximal finger opening, so it could grasp it if this were demanded.

3. MH-1 goes back into a search routine, groping for m2 (the blocks). If it should run into m2 on its way back to the initial search position, it would skip the search phase and proceed directly with 4). Generally, if an obvious short-cut exists, MH-1 will take it. On the other hand, if something has not been done properly, MH-1 will repeat it. For instance, during the search procedure MH-1 makes sure periodically that it does not search too far above the table, since the blocks are only two inches high. Should it ever bump into the table during the search action, it will interrupt the progress, repeat the check for proper heights,

and then proceed.

4. MH-1 determines the position and the size of m2. Scan routines can be avoided if the corresponding information is given to the computer by the programmer in a "setmodel" instruction.

5. MH-1 grasps the block and holds it. The method used will depend on size and location of the block, facts which have been determined in 4.

6. During that period, MH-1 will move into the vicinity of the box ml, and feel around for it until it bumps into it.

7. It will deposit the block into the box.

8. It will start to look for a new block, beginning about where it found the previous block which is now in the box.

What will happen if more severe trouble arises than the kind which can be eliminated by simple short-cuts and repetitions? Notice that Program No. 1 contains no explicit information whatsoever about what to do if difficulties arise. Should MH-1, for instance, run into the box while looking for blocks, it will grasp it and will get fouled up trying to put the box into itself. It does not realize that the box it takes is the same object as the box it is supposed to deposit its findings in. The reason is that m2 can be any object, not only blocks. If we want to limit m2 to blocks, we have to say that at the beginning, and the corresponding program will then look like this:

Program No. 2

```
setmodel m2,hor          this tells that m2 is a
                         small object
goto ml,mL
scan ml,md
goto m2,mL
take m2,hld              no scanning of m2 re-
                         quired now, since its
                         size has been established
                         by the setmodel instruc-
                         tion
goto ml,mL
put ml,mL+xb+yh+zd
goto m2,mL
transfer to (take m2,hld)
```

With only this program modification, MH-1 would not grasp the box if it runs into it a second time, even if the box had been removed from its original place and deliberately put into the path of the block-search program. MH-1 would instead remember the new position of the box and start the search for blocks all over again. In still another, slightly different program, the hand deposits the box at some standard location after it has found it, somewhere on the side of the table so as not to be in the way during the search for the blocks. If one now takes the box away and puts it back into the path of the block-search, the hand will put the box back where it belongs and then continue. It is evident that in some cases the hand really does the reasonable thing if unexpected disturbances of a considerable logical complexity arise, without having to be told so explicitly by the programmer. How successful it is depends, as we have seen, somewhat on the form of the program.

Once in a while, MH-1 will, however, do something that does not make any sense at all, and, if it does not check that itself, it may have to be interrupted manually. This happens about once in five successful operations, such as putting a block into a box. The reason for this is that the computer has made a wrong decision when it was confronted with an unexpected situation. Typically, it might then shove away the box, kick over a block, or search at a wrong place. However, the general performance is quite satisfactory, and there is usually quite some suspence built up among the spectators which releases itself in applause when the first block is dropped into the box.

Another program builds a tower out of the blocks by piling them on top of each other. This is quite interesting to watch because the growing tower sooner or later collapses. All of these programs are quite simple to write, as Program No. 1 shows. A small input routine permits it to write them directly into the machine with the on-line flexowriter.

PRINCIPLES OF THE PROGRAMMING SYSTEM

We have seen that MH-1 has made considerable headway towards coping with difficulties by itself without burdening the programmer. How is this achieved?

Several levels of interpretive programs are used. The computer contains a description of the objects in the real world (model), indicating their absolute and relative position and their size. These descriptions are generated by the programmer by "setmodel" instructions, or automatically by the "scan" program and always when the hand gropes for an object. The interpreter for the highest program level (such as described in Program No. 1) has to find a routine in memory which corresponds to:

1. the type of action (go to, find, etc.) demanded

2. the type of object concerned, as determined by the model of this object

3. the goal specified in the program (hld=hold, mL+ xb+ yh+ zd=on top of)

A typical routine would be one which grasps small objects to the right of the hand. Each routine is described in a list that the interpretive program scans in its search for appropriate routines. These routines, in turn, are sequences of statements indicating desired states of sense organs. One statement may concern many sense elements. A typical statement might be that the outside of the left finger should touch something. The interpreter of these routines determines symbolically which motors will have to be energized and how much in order to bring about the desired state of these sense elements. This motor assignment may depend on the position of the hand. The symbolic motor commands thus generated are then interpreted on the lowest level by routines involved in the position and speed-control loops, by stop routines, timing routines, and several error-checking and interlocking routines.

The interpreter of the desired sensory state level in addition to generating control commands for the motors, creates a list of sense organs that are expected to change during that phase. Periodically, all remaining sense organs are checked for changes and if such an unexpected change should occur, the normal program execution is interrupted. A special routine then begins to determine what should be done about that unexpected occurrence. There are three alternatives that this routine can take.

First, it finds out whether these unexpected changes are intended changes in the near future (by looking ahead in the program) or if they did occur in the near past (by looking back in the program). If either one of these conditions is fulfilled, control will be transferred to that program location and the program will continue from there on. This is the basis for the mentioned behavior of MH-1 with respect to obvious shortcuts and repetitions of unsatisfactorily performed actions. But one can easily see that there is a chance that the interrupting unexpected change occurs as an intended change in a completely different connection, and then it is unlikely that the corrective action taken, namely, transferring control to that program location, will be successful. The occurrence of this can be minimized by not letting the interrupt program search in too distant parts of the routines, and certainly not in different routines. Sometimes, however, the corrective action taken will be wrong, and in most cases there will then be another interruption. But that is usually too late because by that time MH-1 has lost track of the original goal, since there is no return to the original program once the interrupt program has found a spot that seems to be appropriate to the type of interruption. This is what has happened when MH-1 starts to do something that appears illogical from the outside.

The second alternative is adopted if no such intentional occurrence of an unexpected sense change can be found. If the program is in a "take" phase, this phase will be interrupted, and the scan program will be started. When the nature of the disturbing object is discovered, another special program will look up the highest level program and find out whether a similar object is ever mentioned there. If so, control will be transferred to that location of the program. This is the explanation of the phenomenon that in the block and box program the box can often be encountered a second time without any disastrous consequences if MH-1 realizes that it is running into the box again and that it is not supposed to. It will simply switch back to that part of the program in which it dealt with that box.

Third, if no appropriate instruction can be found in the highest program level, and whenever an interrupt search is unsuccessful in a phase other than "take", the hand will withdraw until the disturbance disappears, and then continue what it tried to do before, repeating this several times if necessary.

It would be nice if it were possible to have a program generate a corrective program itself in the case of interruptions, but the present sense organs make this impossible except for simple motions like withdrawing. The present sense organs are satisfactory if used together with a program written by somebody who knows what the hand is supposed to do, but they are not sufficient to determine a reasonable sequence of subgoals, expressed in terms of sense elements, with respect to some over-all goal. To give an example of this, it is possible to program MH-1 in such a way that it will detect the edge of an object, by performing a few test motions with the fingers. But the sense output makes sense only in connection with the knowledge of the idea behind those test motions. If the hand were to move by itself and you had only access to the sense output, and not to the intention of the programmer, you would not be able to detect the fact that the hand has just run over the edge of an object. This is why the automatic generation of routines is not possible with the present sense organs.

So much for the design of the programs which achieve the present degree of independence of disturbances and which permit the present, conveniently vague way of specifying a program. In the following discussion we shall take a second look at all this, but from a somewhat more fundamental point of view, using less of the terminology of computer programming.

AWARENESS AND UNDERSTANDING

We have to distinguish between the computer and the real world. The computer-internal world consists

of the models (descriptions) of some of the real-world objects, the higher level programs, and the registers containing the present sense input. The models are an abstract image of the real world, the programs represent the intentions of action of the hand in the real world, and the registers containing the sense input are that part of the sensory apparatus to which the computer has access. It is important to understand how these things are interrelated.

The internal world is the only thing upon which the computer can operate directly. Mere existence of objects, relations, and conditions in the real world is not sufficient. Unless these can be projected into the computer-internal world, they are neglected by the computer. Therefore, if we want to couple a system with mechanical intelligence to the real world, we have to make sure that the computer contains an internal representation of the external world which is sufficiently complete. The task of the sense organs is to transfer information between these two worlds. To give an example of this, the internal world of MH-1 contains a complete representation of everything touching the fingers, and it can, therefore, be programmed to interrupt its activity and decide on a better course of action if something touches the fingers unexpectedly. However, MH-1 has nothing in its internal world which indicates progress. MH-1 will, therefore, not complain if a poorly written program takes half a dozen attempts to achieve its purpose, grabbing a block for instance.

A good internal world and good sense organs will permit the mechanical imitation of what in psychology is called "awareness". The design of a sufficiently complete internal world is probably the most difficult problem in this computer-real world symbiosis. A computer cannot be aware of phenomena that are not reflected in its internal world, and cannot bring its "intelligence" to bear upon them.

Awareness, however, is not all that is required for the coupling of an intelligent machine with the real world. The machine must be able to correlate the processes in the real world with the operations performed in the internal world and vice versa. We shall call this ability "understanding the two worlds". Let us now see what form this understanding takes in MH-1. Let us assume that the sense organs have detected an unexpected change, and that the machine is now aware of it. Understanding is required if the machine is to find a way out of this situation by means of processes in its internal world because it must be able to relate the processes performed in the internal world and those in the outside world. MH-1, for instance, assumes that hand actions are responsible for any unexpected change. It, therefore, looks up the representation of its actions in the internal world, namely, its program. It determines which part of that program might bring about such a change in the environment as the one which caused the interruption. Without performing the action, MH-1 is able to understand what its program means with respect to the outside world, and, therefore, to the sense input. To find an appropriate passage in its programs, and to transfer control to that part of the program, will often solve the difficulty.

Comparing the action plan (program) with the real world is only one form of understanding, comparing the models with the real world is another, and this form is used in the second of the described types of interrupt action. However, it follows from the structure of both of these processes of understanding that MH-1 would be at a loss if it was confronted with a living world, where things change of their own accord, not only because of actions of MH-1. A different form of understanding would be required to cope with the problems of such a world, a form that takes into account the dynamics of the real world.

MH-1 is not very advanced in the development of its power of understanding, but I do think that it is at least a step in the right direction.

THE SENSORY APPARATUS

We have not said much about the sense organs, although it became evident during the experimental phase that we had much to learn about them. We became aware of their importance concerning the programming language. In an earlier system, the routines were written in terms of the motors to be used at each element of time. Since the coordination of the sense of touch with the motor organs depends on the position of the hand, it is quite easy to see that it would be rather difficult to scan such a program for an interrupt condition.

In classical control systems, the sense elements translate the stimuli originating in the real world into the system-internal signal form, an electrical current, a mechanical displacement, and so forth. In MH-1, the programming language corresponds to the signal form in a classical system. On the mechanical hand, the task of the sense organs is therefore to match the external stimuli to the programming language. This difference in tasks results in sense organs that are different from those usually found in control systems. There, sense organs, or transducers, generate one specific signal with one specific purpose, for instance, they detect the temperature at one place. For the hand, we need distributed sense organs, something like our skin. The output of each microelement is only a small part of the pattern composed of the output of all of the microelements. This pattern forms a word in one of the interpretive languages, and it is this whole word which is of importance, which conveys information about the real world and upon which

as a whole the computer works. Each single sense element has a much less specific task of its own; what it really does is determined by the process the computer applies to the received words, and not by the sense element.

The construction of this type of sense organs, or rather ensemble of sense elements, is a novel problem of which we became aware late in the project, and for which we do not have any solutions yet. The sense elements in Fig. 4 are only an unsatisfactory substitute. This new approach to the construction of the sensory apparatus would have been the next step if the project had been continued beyond the work reported here.

APPLICATIONS

This work is of interest with respect to two possible applications: The construction of mechanical hands as useful tools, and the exploitation of the philosophy behind the control problem.

Mechanical Hands for Industrial Applications

There has been a considerable interest in general-purpose programmable tools of the form of a human arm and hand.[2,8] The difficulty seems mainly one of cost, since the demand falls off rapidly if the price rises above $15,000. This is an order of magnitude less than the cost of a computer alone which would be required for a system similar to MH-1. This difficulty could be partly relieved by having one computer control several mechanical hands at a time, and by having functions, which are now performed in a routine fashion by the TX-0, delegated to less expensive analog equipment that would go with each hand. The computer would then enter the picture only when something has to be decided which is beyond the power of the equipment that goes with each hand; instead of really controlling all of the hands all of the time, the computer would only supervise them.

In general, it seems that the more universal a tool becomes - and a mechanical hand is meant to be universal - the less qualified is the work the tool is supposed to do, at least in production applications. This is probably the reason behind the mentioned price squeeze, and besides it raises serious doubts about the desirability of such machines in the present labor market, which consists of a surplus of unqualified labor and a shortage of qualified forces.

Mechanical Hands in Hostile Environments

We are talking about the very problem that gave rise to the appearance of servomanipulators: Manipulation in radioactive areas. Other adverse environments may be found in deep-sea exploration and space exploration. This situation is radically different from the described production situation because under these circumstances even very simple actions become highly qualified ones. However, there is still no economical or technical advantage to be gained from replacing the human operator of such a manipulator by a computer, except in one case. The manipulator may be so far removed from the human operator that communication with it becomes difficult, either because of delay or because of the power required to ensure a reasonably good transmission. This situation arises in space exploration. One way to solve this difficulty is to send some mechanical intelligence with the manipulator, in the form of a computer. One can easily see that the transmission of a program like the one in our example is much easier than the establishment of a complete, real-time telecontrol system, besides solving the nasty delay problem. Including the apparative delays (scanning rate of the TV system) it is estimated that the delay in a control loop extending to the moon would be approximately 3 seconds. Assume that the manipulator should pick up a rock on the moon. The operator will see that the hand is in the appropriate position only 1-1/2 seconds after this event occurred. Another 1-1/2 seconds will pass until the signal to stop reaches the manipulator. Unless the manipulator works very slowly, it will have moved beyond the rock by that time. But, if there were sufficient intelligence incorporated into the system on the moon, we would simply line up the manipulator and then tell it to go forward until it touches the rock, similar to the actions that are taken by MH-1 when it searches for objects.

The cost of providing that mechanical intelligence is irrelevant in similar applications. It might even be that the accompanying simplification of some of the other communication equipment more than makes up for the cost of the computer.

APPLICATION OF THE PROGRAMMING PRINCIPLES

I am mainly thinking now of the accessibility of some non-human environment to the computer in order to enable it to use the described forms of awareness and understanding. We have only begun to scratch the surface of this problem, and it will probably remain in the research stage for some time to come. This research will have to proceed in two directions:

1. Development and construction of the required sense organs. It would help greatly if we could build an eye for something like the MH-1 system.

2. Development of the necessary internal worlds, for instance, better models and better ways to generate them. This research will be interesting both from a human and from a technical point of view.

Although the exploitation of this research may be farther off than the construction of programmable mechanical hands, I think that ultimately it is far more interesting. It seems difficult to build a mechanical hand that will surpass our own human hand. We can easily surpass some of its properties, but that is easier accomplished with special-purpose equipment than with something as general as a hand. However, I am optimistic enough to think that a better fundamental understanding of our human information processing will open all kinds of doors for interesting technical applications. I, therefore, consider any contribution made by the study of MH-1 in this area to be more important.

REFERENCES

1. Alfred Chapuis, Edmond Droz, "Les Automates. Figures Artificielles d'Hommes et d'Animaux", Editions du Griffon, Neuchâtel, Switzerland; 1958

2. H. W. Nidenberg, "Programmed Manipulation: Noman", Technical Report, Advanced Manufacturing Engineering Service, March 15, 1960

3. John W. Clark, "The Mobot, a Fully Remote, Mobile Handling Device", Technical Memorandum 627, Hughes Aircraft Company, November 27, 1959

4. "Automatic Programming of Numerically Controlled Machine Tools", Final Report 6873-FR-3, Electronic Systems Laboratory, MIT.

5. R. Tomovic, "The Human Hand as a Feedback System", International Federation of Automatic Control, Moscow, 1960, vol. 2, pp. 1119. (Translation by Butterworth's Scientific Publications, London)

6. "Electro-Mechanical Manipulator for Remote Handling", Automation Engineering Department, Borg Warner Corporation, 1956

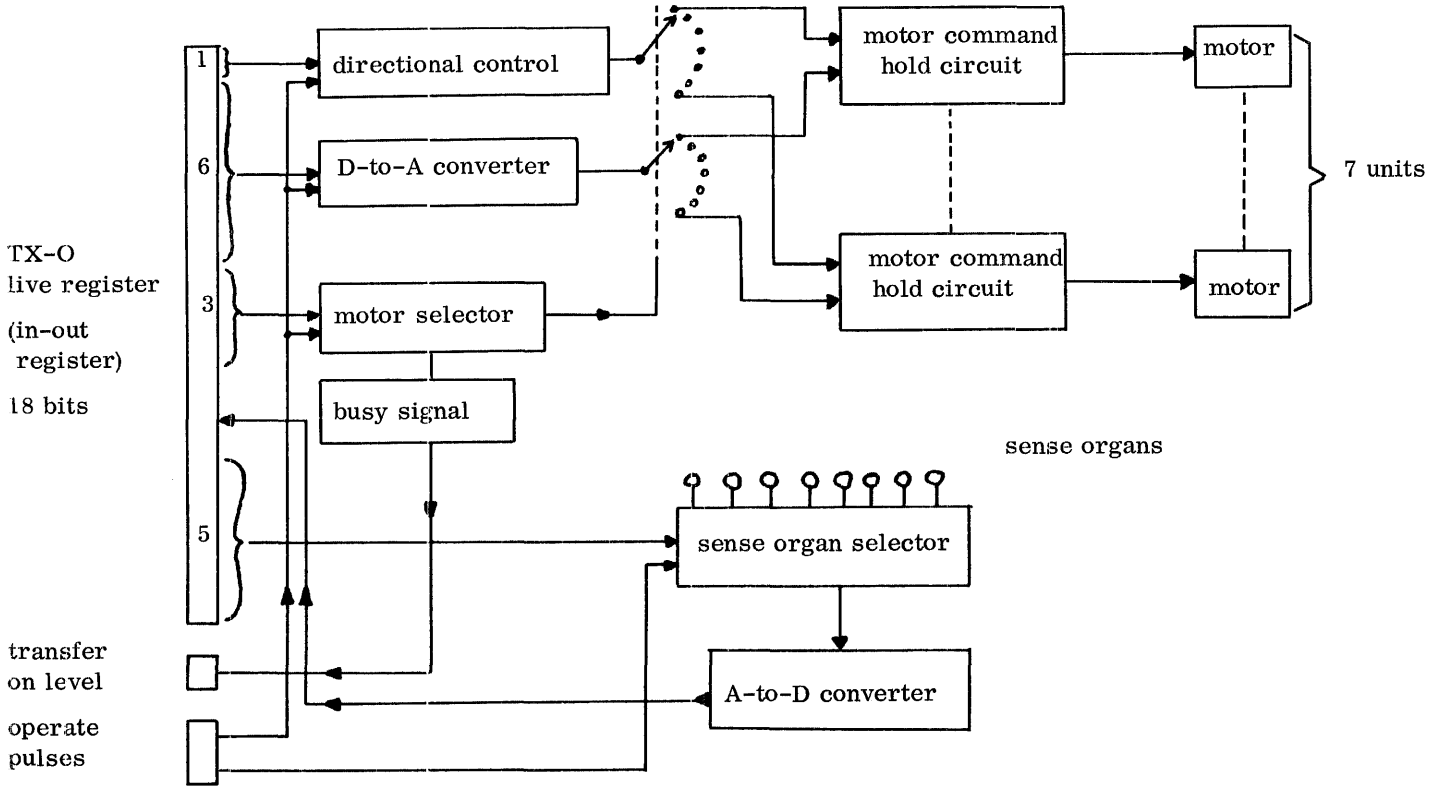


Fig. 1. Block diagram of the MH-1 control unit.

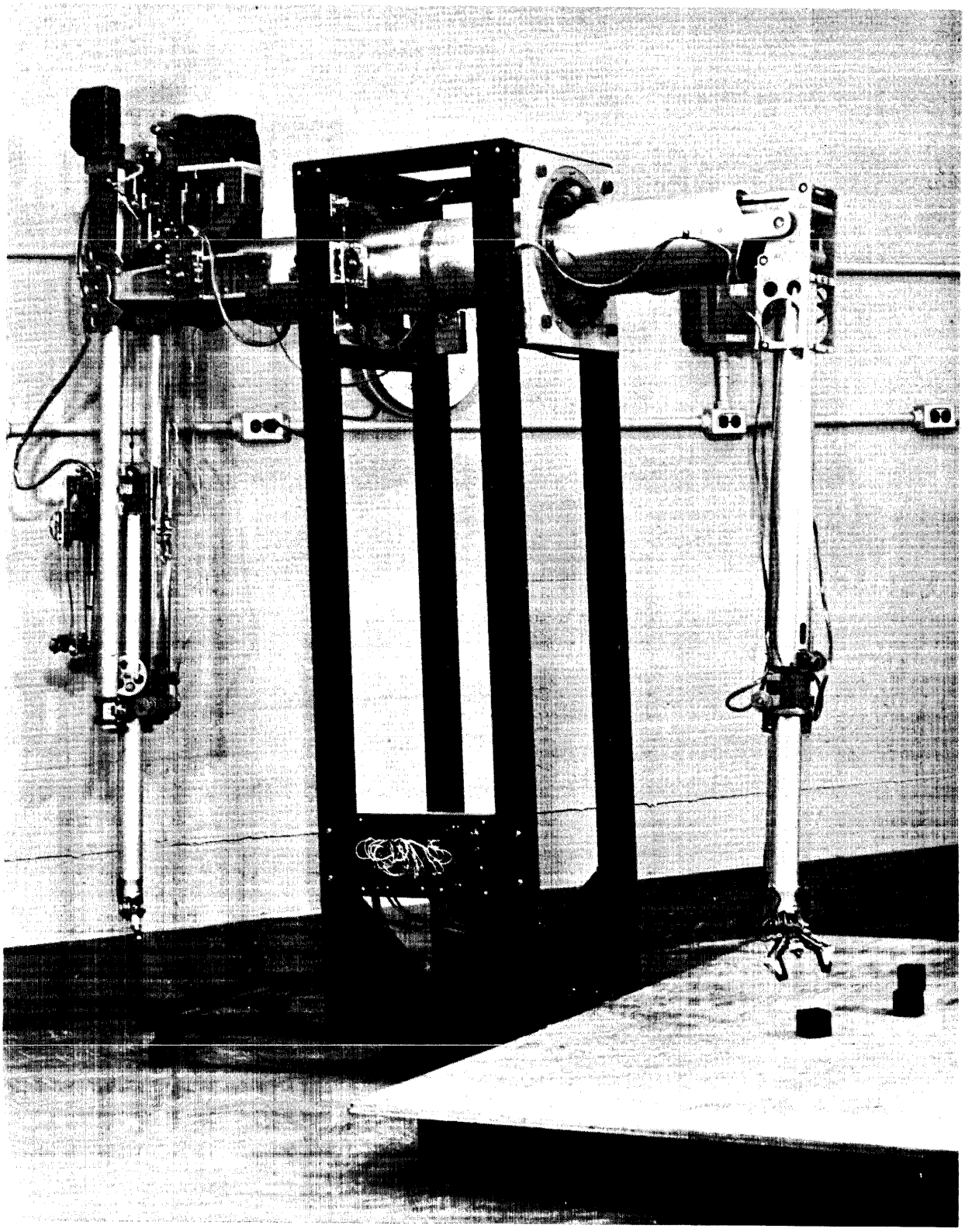


Fig. 2. The servomanipulator. The part toward the reader is the active arm, the second arm serves only for balancing purposes and carries the servomotors.

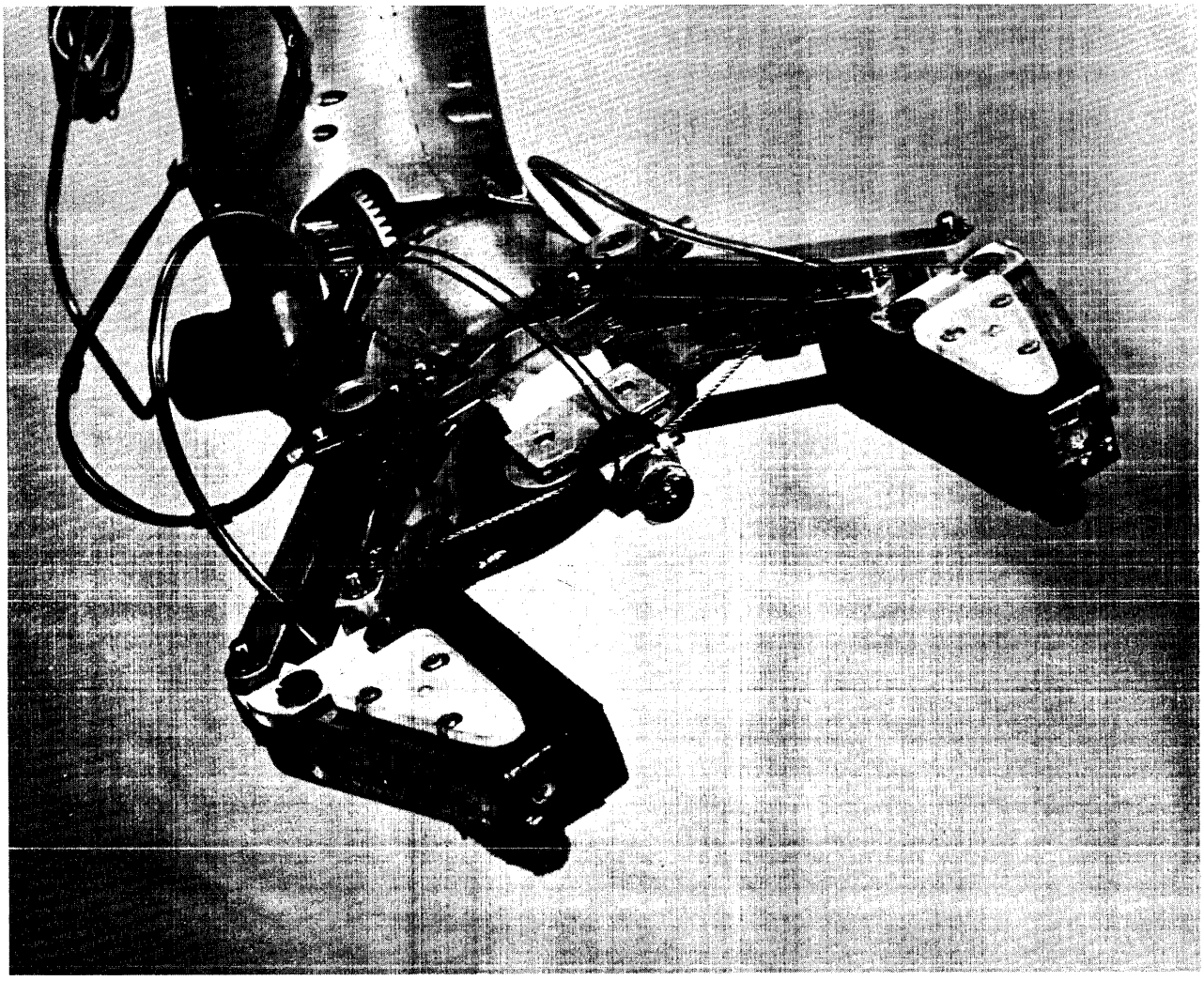


Fig. 3. The mechanical hand.

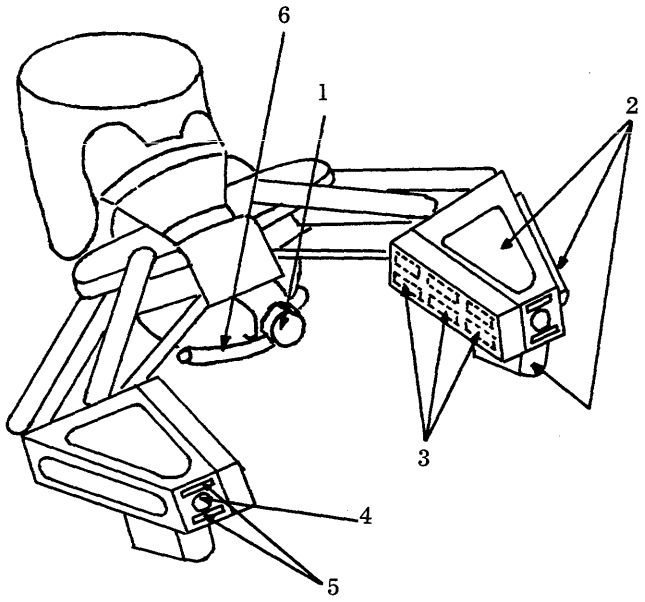


Fig. 4. The sense elements on the hand.

(1) switch closing if touched; detects position of objects between fingers; binary output.

(2) a total of six contacts (2 per plate) closing if touched; indicate contact with finger surface; binary output.

(3) 6 pressure elements in rubber pad; detect firmness and location of grip; continuous output (variable resistance).

(4) photodiode; reacts on shadows cast by black objects; continuous output.

(5) 2 pressure pads as in (3).

(6) pressure element on bottom of wrist; closes when hand rests on table; binary output.

Elements (2), (3), (4), and (5) appear on each finger.

AN ABSTRACT MACHINE BASED ON CLASSICAL ASSOCIATION PSYCHOLOGY

Richard F. Reiss

Librascope Division

General Precision, Inc.

Glendale, California

Summary

The theories of classical association psychology (circa 1750-1900) attempted to explain human thought processes in terms of certain mechanistic forces operating on discrete entities called "sensations," "images," and "ideas." Although these theories have become unfashionable since the turn of the century, due primarily to their ambiguity and the difficulty of experimental verification, and whereas they may never prove adequate for human psychology, it is possible that they may, nevertheless, provide a fruitful basis for some types of artificial intelligence. One method of exploring ramifications of the classical theories is the formulation of an abstract "machine" which constitutes an interpretation of the theories and whose behavior can be examined in any desired detail. In this paper such a machine is partially constructed, and some of its behavioral features and problems are discussed.

Introduction

The general problem of synthesizing intelligent artifacts is sufficiently broad and difficult to justify a variety of research strategies. One such strategy is to select a psychological theory, however inadequate (as they all are), and examine it for useful ideas and insights. This strategy has been used in the studies on which this paper is based.

Classical association psychology was developed primarily in England and enjoyed considerable popularity for more than a century (circa 1750-1900). Space does not permit even a brief recount of its history, but some of the key works will be mentioned for purposes of reference. Although Aristotle vaguely recognized the minimal association principles (see Hamilton[8]), the modern development of association theory began with some hypotheses laid down by the British philosophers Hobbes[10], Locke[13], Berkeley[5], and Hume[11]. They were, however, concerned primarily with epistemology rather than psychology. Hartley[9] is generally considered to be the founder of associationism as a psychological doctrine, and it flowered during the 19th century due largely to the efforts of Brown[7], James and John Stuart Mill[14,15] and, above all, Bain[1,2,3,4] and Spencer[16]. For a detailed history, see Warren[19].

Association theory proposed to explain the highest levels of conscious thought processes, perhaps the most elusive of all experimental subjects. By 1900, the revolutionary development of physiological and experimental psychology had created an atmosphere which was very hostile to any theory that could not quantify its concepts and be subjected to precise experimental testing. Associationists were unable to meet these criteria successfully, and the classical theories in their general forms were rapidly abandoned. We say theories, rather than theory, because associationists rarely agreed on more than a few basic concepts. Later we shall simply refer to "the classical theory," by which we mean those concepts that did receive widespread acceptance. Some of these concepts are to be found, frequently disguised by new terminology, in many contemporary schools of psychology. But the old theories with their sweeping assumptions have been effectively dead for 50 years.

The problem of artificial intelligence provides a distinctively new framework for evaluating psychological theories. If one is concerned with the subproblem of mechanizing human thought processes as they actually occur, then of course the validity of a psychological theory is of great importance. The general problem, however, does not require such stringent criteria; the search is for <u>suggestive</u> material, in whatever form.

The association theories are of interest on three counts: they were produced by several brilliant minds struggling through introspection with the problem of mind; they involve the action of a few mechanistic forces operating on discrete entities; and they aim at explaining the higher thought processes. Unfortunately, the associationists made no sustained attempts to quantify their assumptions and search out the detailed consequences. This must be done, then, for the first time (to my knowledge), and the resulting abstract machine represents just one possible interpretation of classical theory. Another interpreter might well produce a machine that differs in many respects. Some alternatives will be noted in the discussion below, but in general the postulates will be introduced without apologies or historical justifications.

From Hartley on, associationism was generally tied to parallel theories of the nervous system. In constructing the abstract machine, this facet of association theory will be almost entirely ignored. Except in the concluding remarks, the problem of physically realizing the abstract machine will likewise be ignored.

The objectives here are simply to show what a machine based on association theory might look like, to demonstrate some of its behavioral characteristics, and to raise problems that appear important to further research along these lines. Our strategy in this project has been to strive

for a reasonably complete machine without regard for our ability to analyze its behavior; we hope thereby to avoid the premature selection of problems that appear interesting but are actually trivial. The nine postulates (together with subordinate assumptions) which are discussed here fall far short of a complete machine. But combined with remarks on further postulates in the concluding section, they should be adequate for the purposes of this paper.

The postulates will be introduced in three groups. After each group has been stated, some behavioral consequences of that group will be examined in detail and a few generalizations ventured.

The Four-Postulate Machine

The first four postulates form a group which produces a basic machine without sensory inputs or motor outputs. The behavioral potentialities of this machine must be examined in some detail before proceeding to further postulates.

Postulate P1: There exists a finite set M of "memory tokens" $m_1, m_2, \ldots, m_\mu$.

Initially we make no assumptions whatever regarding the nature, structure, or properties of these tokens although they obviously represent hypothetical entities--such as "images," "ideas," "impressions," "concepts" with which the associationists furnished the mind. The number of memory tokens, μ, is assumed constant; and we do not inquire into the origin of these tokens at present.

Postulate P2: Between each pair of memory tokens, m_i and m_j, in M there are two "bonds" whose "strengths" are given by "coupling coefficients." The bond from m_i to m_j is represented by the coupling coefficient c_{ij}; and the bond from m_j to m_i, by c_{ji}.

The "bonds" introduced by P2 are first approximations to the associationists' "connections" and "attractions." Although the associationists seem to have generally assumed that connections are symmetrical in the human mind, it was and is a point of controversy; and we here take the more general case where the bonds from m_i to m_j and from m_j to m_i are treated as independent entities. It will be seen later that these bonds must, in turn, be split into two components.

The coupling coefficients may be organized to form a square ($\mu \times \mu$) matrix where c_{ij} is the element in the i th row and j th column. This will be called the coupling matrix and will be denoted by "(c_{ij})." Digraphs will be used to illustrate small token groups. Figure 1a shows a system of five memory tokens, and Figure 1b is the equivalent coupling matrix. It will be noted, in the graph, that bonds with zero coefficients have been left out and that symmetrical bonds (as between m_2 and m_4) are represented by a single two-headed arrow. The statement that "m_i is not bonded to m_j" will mean that $c_{ij}=0$. All coupling coefficients are assumed to range over the positive real numbers, with a lower limit of zero and no upper limit specified at present.

Postulate P3: There is an "attention register" which can hold a set A of memory tokens. The maximum number of tokens in A is λ, and this integer will be called the "length" of the attention register.

The introduction of the attention register completes the hardware of the four-postulate machine. It will be assumed that $\lambda \ll \mu$ and that A is, therefore, a proper subset of M. The intent of the adjective "attention," if not immediately apparent, will become clear in the course of discussion. In this paper, λ will be considered an independent variable, or parameter, of the machine.

Before stating the fourth postulate, it will be necessary to introduce a special function and a related set. For every memory token m_i we define the adduction function f_i. If m_i is a member of A, i.e., is in the attention register, then $f_i=0$. If m_i is not a member of A, then the value of f_i is given as

$$f_i = \sum_j c_{ji} \qquad (1)$$

where j ranges over the subscripts of tokens in A. Thus if m_i is not in A, f_i is the sum of the coupling coefficients of bonds from all tokens in A to token m_i. In terms of the coupling matrix, f_i is the sum of all those elements in the j th column, which are also in rows corresponding to the tokens in A.

The adduction set F is defined as follows: memory token m_i is a member of F if, and only if, there exists no memory token m_j for which $f_j > f_i$. In other words, the adduction set is composed of the token(s) having the largest adduction function value(s). Be it noted that F cannot be empty; indeed, if all coupling coefficients are zero, F≡M.

Postulate P4: Every Δt seconds a memory token is chosen at random from the adduction set F and entered into the attention register, i.e., set A. If the attention register is already filled, then the "oldest resident" will be simultaneously ejected.

This postulate introduces action into the abstract machine. The state of the machine at any moment is defined by the coupling matrix and a list of memory tokens in the attention register. Every Δt seconds the machine changes state by entering a memory token into the attention register and ejecting the token that has been longest resident in the register. It will be assumed that all members of the adduction set F, at any moment, have equal chances of being selected for entry into set A; i.e., if there are n members of F, the probability that any particular member will be selected is 1/n. If F contains only one member,

then the next token to enter A is uniquely determined.

The sequencing of tokens through the attention register may or may not be a stochastic process, depending upon the nature of the coupling matrix. For example, if there is exactly one nonzero element in each row and column, then the machine's behavior for all future times is precisely determined by the tokens initially placed in the attention register. On the other hand, it is easy to formulate any number of coupling matrices which produce stochastic behavior. It should be noted that the coupling matrix is not a Markov matrix; the coupling coefficients are not transition probabilities.

Discussion of the Four-Postulate Machine

The first four postulates produce a class of machines which clarifies certain problems and potentialities inherent in classical association theory. The sequencing of memory tokens through the attention register might be interpreted as a first, crude approximation to the flux of ideas, images, etc. into and out of the conscious state. The associationists generally believed that thought processes are conscious processes and that introspection is, therefore, a powerful tool for the analysis of thought. As Bain[4] put it, "...Introspection is still our main resort--the alpha and omega of psychological inquiry: it is alone supreme, everything else subsidiary. Its compass is ten times all the other methods put together, and fifty times the utmost range of Psycho-physics alone." By the time Bain died (1903), evidence was rapidly accumulating in favor of the thesis that subconscious processes influence thinking to a remarkable extent. This development, compounded by failing attempts to make introspection a precise analytical tool and various other trends (see Boring's history[6]) led to a widespread rejection of introspection. It seems the associationists' theory had become so closely connected to the introspective technique that, when the technique was discredited, the theory was discredited too. Actually the failures of introspective technique show only that the association theory is difficult to test, not that it is invalid. It is by no means necessary to assume that all, or even most, association processes occur at a conscious level. In fact association theory has survived (in a simpler form) in theories of conditioning and learning which ignore consciousness altogether.

Since we are not concerned here with the validation of a psychological theory, and since consciousness is not in any event a necessary constituent of association processes (although the classical associationists failed to recognize this fact), we shall not postulate any relations between consciousness and the states of our abstract machine. As a consequence of P4, a memory token must be in the attention register in order to influence the behavior of the machine. Nothing more is to be read into that postulate unless it amuses the reader to do so.

Postulates P2, P3, and P4, taken together, are the result of a difficult decision that must be made in any interpretation of the classical theory. At times the associationists spoke of the connections between mental entities as though they were little more than indicators of possible, or preferred, sequences among which some higher agency chooses a particular path. Thus Brown[7], for example, insisted on using "suggestion" rather than "association." On the other hand, the physiological bias initiated by Hartley[9] and particularly noticeable in Bain[2] and Spencer[16], often resulted in associative connections being treated as physical forces or neural links competing in a completely mechanical way. Reading between the lines, the associationists were apparently torn between a desire to create a mechanistic, deterministic theory, resembling contemporary physical and chemical theories, and a desire to preserve the "free will" of an immanent "soul." (It might be noted that a theist engaged in artificial intelligence research today is confronted by a similar dilemma.)

I have been unable to find in the literature a clear, unambiguous discussion of the selection problem inherent in any system of competing associations. But this problem cannot be avoided in the formulation of our abstract machine. The introduction of a deus ex machina, some free will demon which selects tokens, would merely beg the question as to how much might be accomplished by the relatively simple associationist machinery. There seem to me to be two main alternatives. Coupling coefficients might be introduced as transition probabilities with P4 replaced by some rule governing joint probabilities, in which case the machine would be heavily biased toward stochastic processes. The other alternative is exemplified by the postulates used here, with the machine biased toward deterministic behavior. This bias will be magnified by postulates P5 and P6 so that the machine will, in the absence of sensory inputs, tend toward cyclic, stereotyped behavior patterns with few if any random fluctuations. We proceed to examine briefly a few behavioral characteristics of the four-postulate machine.

Assume the machine contains just five tokens, bonded as in Figure 1, and that the attention register can hold only one token, i.e., $\lambda = 1$. Suppose that m_1 is initially in the attention register. The sequence of tokens entering A will then be m_2, m_4, m_2, m_4, ..., an endlessly repeating cycle. Let m_3 be initially in A. Now the adduction function for all tokens has the value zero, and the adduction set F contains all tokens. A token from F is chosen at random. If M_3 is selected, then the process is repeated until some other token enters A. The selection of m_5 will result in m_3 returning to A. Eventually m_1, m_2, or m_4 will enter A, and the endless sequence above will follow.

This example shows two characteristics of the abstract machine. First, a token that has no bonds to other tokens, such as m_3 (and corresponding to a "terminal node" in graph theory), cannot stop the machine. It simply provides an equal

opportunity for all tokens to enter A. Secondly, the machine can be easily trapped into cyclic behavior by rings of two (such as m_2 and m_4), or more tokens. It should be noted that if the reflexive bond from m_4 to m_4 had a coupling coefficient greater than 5, it would not succeed itself since its adduction function is zero when it is in A.

Consider the case where $\lambda = 2$, and let $A = \{m_1\}$ initially. The sequence of tokens entering the attention register will be m_2, m_4, m_1, m_2, m_4, ..., another endless cycle. No matter which token, or pair of tokens, is initially in A, the machine will eventually settle down to this three-token cycle. This little example demonstrates that the machine can be trapped into cyclic behavior by networks other than rings, in this case the chain m_1, m_2, m_4. It is also clear that at least $\lambda + 1$ tokens are required for cyclic sequencing.

Before examining further examples of simple coupling patterns, it will be convenient to define "strings," "rings," "chains," and "trees" in terms of the "predecessor-successor" relation. If $c_{ij} > 0$, then m_i is a "predecessor" of m_j and m_j is a "successor" of m_i. A "string" is a group of tokens containing a unique "origin" token which has no predecessors and exactly one successor in the group, a unique "terminal" token having no successors and exactly one predecessor in the group, and all other tokens in the group having exactly one predecessor and one successor in the group. A "ring" is simply a closed string, i.e., a string without origin or terminal tokens. A "chain," as the name suggests, has two "end" tokens, each of which is connected to one token in the group, which token is both a successor and predecessor; all other tokens in the chain are connected to exactly two others which are also both successors and predecessors. Thus two strings are "embedded" in a chain. A "tree" will be understood to be a group with one origin token having no predecessors and two or more successors, several terminal elements having one predecessor and no successors, and all other tokens in the group having one predecessor and two or more successors. These inelegant definitions will suffice for the rather informal discussions of token structures in this paper.

As a consequence of the adduction function, all of the tokens in the attention register influence the selection of the next token to be entered into A. This corresponds to the process Bain[1] called "compound association." Unfortunately, the associationists did not pursue in detail the implications of such a process. They tended to think in terms of simple strings and trees of tokens (using our terminology), and compound association is not very important in such cases. Consider, for example, the case where all memory tokens are organized into a single string and any one of them is initially in A. Even if λ is large, once A is filled the adduction function will be zero for all tokens except the successor of the last token to enter A. Consequently, the sequencing of tokens through A will correspond to the ordering of tokens in the string regardless of the value of λ. Thought experiments of this sort lead to the general conclusion that the fewer the bonds between memory tokens, the less important is λ. In terms of association psychology, this conclusion could be characterized by the general rule that a mind in which ideas are sparsely associated cannot benefit from an increase in attention span. The behavior of such a mind will tend to be stereotyped in spite of efforts to increase the attention span. Certainly the behavior of the abstract machine becomes independent of λ if memory tokens are organized into simple strings.

However, even simple coupling patterns produce some interesting sequencing phenomena, and we shall briefly examine a few of these. Let us first consider the effects of symmetrical bonds, i.e., cases where $c_{ij} = c_{ji}$. The four tokens in Figure 2 are connected by symmetrical bonds having the indicated strengths. Suppose that $\lambda = 1$ and that $A = \{m_1\}$ initially. The sequence of tokens through A, which will henceforth be called the A-sequence, is m_1, m_2, m_3, m_2, m_3, ... If $A = \{m_3\}$ initially, the A-sequence will be m_3, m_2, m_3, m_2, m_3, ... Thus we see immediately that symmetric bonds do not necessarily lead to symmetric sequences, i.e., m_1, m_2, m_3 versus m_3, m_2, m_1. Furthermore, an increase in λ can have marked effects. Let $\lambda = 2$ and initially $A = \{m_1\}$. The A-sequence will be m_1, m_2, m_3, m_4, m_2, m_3, m_4, ..., a type of behavior seen in small groups without symmetric bonds.

In brief, experiments on a variety of small coupling matrices have failed to show any outstanding effects uniquely correlated with the presence of symmetric bonds or that such effects might emerge in large ensembles of memory tokens.

For the sake of simplicity, symmetric bonds are assumed in the five-token group of Figure 3, which illustrates a "hub and spoke" pattern of coupling (a special type of tree). The bonds between the hub m_1 and the outlying tokens have progressively smaller coupling coefficients. The effect of increasing λ is quite regular for such a cluster of memory tokens. Let $A = \{m_1\}$ initially, and suppose $\lambda = 1$. The A-sequence will be m_1, m_2, m_1, m_2, ... For $\lambda = 2$, the A-sequence is m_1, m_2, m_3, m_1, m_2, m_3, m_1, ...; and for $\lambda = 3$, it is m_1, m_2, m_3, m_4, m_1, m_2, ... Thus the "hub and spoke" configuration provides a basic mechanism for cyclic scanning of memory tokens, the scope of the scanning process being controlled by λ. A simple ring will also produce cyclic scanning, but the scope of scanning cannot be controlled by λ.

This is an example of sparsely bonded token clusters which produce marked changes in sequencing as a function of λ; it is a counterexample for the general rule stated above.

Although Gestalt psychology has been a major opponent of association theory, it turns out that gestalt phenomena can appear in the abstract machine. Consider the group of memory tokens in Figure 4. Suppose that $\lambda = 1$ and that m_1 is in A.

The "response" of the machine is the entry of m_2 into A. If m_3 is in A, then the response will be the entry of m_4 into A. However, if $\lambda = 2$ and both m_1 and m_2 are inserted in A, then the immediate response will be the entry of m_5 into A. Thus the machine's response to the joint presentation of m_1 and m_3 in A is quite different from the "sum" of its responses to m_1 and m_3 presented separately. This fundamental characteristic of gestalt processes is therefore obtainable with associationistic mechanisms; there is no necessary conflict between association theory and gestalt phenomena. This case will become more significant after sensory inputs to the attention register are introduced below.

The length λ of the attention register can influence the sequencing through tree structures. Consider the simple tree in Figure 5. Let $\lambda = 1$ and $A = \{m_1\}$ initially. The A-sequence will be $m_1, m_2, m_3, m_4, m_5, m_6, \ldots$. For $\lambda = 2$, the A-sequence will be the same; but if $\lambda = 3$, a new type of behavior suddenly appears. It might be called "branch jumping." For $\lambda = 3$, the A-sequence will be $m_1, m_2, m_3, m_4, m_5, m_7, m_8, m_9, \ldots$. Instead of continuing out the upper branch, the A-sequence jumps down to the lower branch and continues on that branch. The crucial factor is the retention of m_3 in A when the weak bond between m_5 and m_6 is reached. At this point, $f_6 = 1$ while $f_7 = 2$; and therefore m_7 enters A instead of m_6. Thus if $\lambda > 1$, the A-sequence is not determined simply by the relative strengths of coupling coefficients at branch points in a tree. Clearly, small changes in λ could effect great changes in the machine's behavior if memory tokens are organized into certain types of trees, another exception to the general rule that λ is an ineffective control parameter for sparsely bonded token systems.

A case of interest is the basic configuration of "intersecting" strings, i.e., where a memory token is the member of two strings. Two simple strings are embedded in the system of Figure 6, m_1 through m_5 and m_6 through m_9 (including m_3). Of course several other strings can be abstracted from this group, but there is no need to identify them. Let us assume that all coupling coefficients have the same value, say, 1.0. Consider first the case where $\lambda = 1$ and $A = \{m_1\}$ initially. The adduction set $F = \{m_2, m_3\}$, and either token may be chosen to succeed m_1 in A. In fact, set F always contains two tokens, regardless of which token is in A. Thus the machine operates in a stochastic mode over this group of tokens, and many A-sequences are possible, e.g., $m_1, m_2, m_4, \ldots$, or $m_1, m_3, m_8, m_9, \ldots$. If $A = m_6$ initially, the situation is essentially the same.

Now let $\lambda = 2$. The behavior of the machine shifts radically from a purely stochastic to a purely deterministic mode! Furthermore the intersection of the several strings does not provide opportunities for "turning corners." To see this, assume that initially $A = \{m_1, m_2\}$, with m_1 the "oldest" token (it will therefore be ejected first). The A-sequence is guaranteed to be $m_1, m_2, m_3, m_4, m_5, \ldots$ and on out the horizontal strings. Similarly, if the machine is moving down the vertical strings, the A-sequence will be $\ldots m_6, m_7, m_3, m_8, m_9, \ldots$ and so on.

The bonds from a token to the next but one token, e.g., from m_1 to m_3, m_2 to m_4, etc., correspond to what are sometimes called "remote forward associations" in serial learning experiments. While we will not venture a direct interpretation of this abstract machine behavior in terms of serial learning, we will adapt the psychological phraseology to our purposes and refer to these bonds (in such overlapping string structures) as "remote forward bonds."

The example above shows that strings accompanied by remote forward bonds may intersect at many tokens without producing ambiguity in the sequencing. When $\lambda = 2$ (or more), it is as if the sequencing process acquires "inertia" which carries it right through intersections without deflection. Thus a certain economy of memory tokens is possible in the abstract machine; a particular token may be coupled in numerous strings without destroying the independence of corresponding A-sequences.

This example also shows why it is dangerous to characterize the machine as either stochastic or deterministic. Given even this simple coupling configuration, the small extension of the attention register from $\lambda = 1$ to $\lambda = 2$ is sufficient to shift the machine's behavior from one extreme to the other. From the standpoint of classical association psychology, this phenomenon is, of course, very suggestive. The strings with remote forward bonds might well represent habitual sequences of "thoughts" and "ideas." If, then, the attention span of an individual is reduced to a minimum, a condition approached perhaps in "free-association" experiments and dreams, the behavior of the abstract machine suggests that the normal, habitual thought sequences might be broken up and replaced by random, novel sequences.

Having examined a few fundamental behavioral capabilities of the four-postulate machine and engaged in some speculative psychological interpretation, we shall proceed to introduce two more postulates.

The Six-Postulate Machine

Among the numerous types or "laws" of association that have been proposed there are two that gained almost universal acceptance. They were most commonly called "Similarity" and "Contiguity." The "law of Similarity" may be summarized as the tendency of ideas or images to become associated in the mind if they have certain "similarities" in quality, structure, function, etc. For example, the concept of a cat may be succeeded in consciousness by that of a rat because of visual or auditory similarities in the names. The "law of Contiguity" draws attention to the fact that ideas or images which have appeared together or in close succession in the mind tend to become associated.

This principle, translated into a behaviorist framework, underlies most theories of conditioning.

Some associationists argued that the law of Similarity could be derived from the law of Contiguity, while others argued the opposite. There were debates concerning the nature of the relation of "similarity" (germane to the modern problems of pattern recognition) and so on. See William James[12] for an excellent critical discussion of such issues; in this paper they will be generally avoided. We shall simply postulate two kinds of bonds suggested by the "laws" of Similarity and Contiguity, ignore all other types of associative connection, and examine the consequences.

Postulate P5: Every coupling coefficient c_{ij} is the sum of an "external coefficient" a_{ij} and an "internal coefficient" b_{ij}; that is, $c_{ij} = a_{ij} + b_{ij}$ for all m_i and m_j in M.

This is an extension of P2 wherein the bonds previously discussed are split into two subtypes which will be called "external bonds" and "internal bonds." As a consequence, the coupling matrix is the sum of an "external" matrix and an "internal" matrix. This postulate does not require alteration of the definitions of adduction functions or sets, nor of postulates P3 and P4.

Postulate P6: Initially, the external coefficient $a_{ij} = 0$ for all m_i and m_j in M. Thereafter, whenever both m_i and m_j appear in the attention register, a_{ij} is incremented by an amount δ_{ij}. The internal coefficient b_{ij} is constant for all time for all m_i and m_j in M.

Thus, at "birth," the machine's external coefficients are zero; and the internal coefficients have values that will be retained throughout the life of the machine. Initially, $c_{ij} = b_{ij}$. Then, if $\lambda > 1$ so that two or more memory tokens may reside simultaneously in the attention register, external coefficients are increased in value as a function of the particular sequencing of tokens through the attention register. The only way to suppress the growth of external bonds is to maintain $\lambda = 1$. Otherwise at least one element of the external matrix (a_{ij}), and therefore of the coupling matrix (c_{ij}), will be increased each Δt during the lifetime of the machine.

At any time the state of the external matrix reflects the "experience" of the machine up to that time, i.e., the past behavior of the machine, the sequencing of memory tokens through A. The internal matrix, on the other hand, reflects certain constant relations between the tokens. The strength of an internal bond, as given by the corresponding internal coefficient, is intended to represent the degree of "similarity" in the internal "structures" of the two tokens involved. In effect, postulates P5 and P6 state that memory tokens have distinguishable internal structures and that these structures remain forever unchanged. But nothing further is presumed regarding the structures of tokens. In particular it should be noted that these postulates do not imply any transitive relations between internal structures; given three tokens m_1, m_2, m_3 and the statement that $b_{12} = b_{23}$, there are no grounds for predicting the value of b_{13}.

Although it is tempting to assume that the relations of similarity, however defined, must be symmetrical, we leave that an open question here because a satisfactory discussion would carry us too far from the main line of development. Thus it is assumed that $b_{ij} \neq b_{ji}$ is possible. Similarly, postulate P6 does not entail the symmetric growth of external bonds. If m_i and m_j are both members of A over a particular time interval, then both a_{ij} and a_{ji} will be increased in value. But it is possible that $\delta_{ij} \neq \delta_{ji}$. The question of symmetric versus asymmetric growth of external bonds will be taken up in the discussion below.

Discussion of the Six-Postulate Machine

As more postulates are introduced, the possible varieties of behavior become ever more numerous and their analysis more difficult. Since we wish to add several more postulates in this paper, it will be necessary to hold the discussion of specific examples to a minimum. In this section two main behavioral issues will be considered.

First, the most apparent effect of P5 and P6 is the progressive elimination of stochastic behavior in the abstract machine as time passes. Consider the fragmentary network of couplings in Figure 7. Assume that the machine has just been born so that $c_{ij} = b_{ij}$ and the bond coefficients indicated are therefore equal to the internal coefficients. Let $\lambda = 2$ and $A = \{m_1, m_2\}$ with m_1 being the oldest. At this time the adduction set $F = \{m_3, m_4\}$ because $f_3 = f_4 = 1$, and there are no other more strongly coupled successors of m_2. The next token to enter A will be chosen at random from F; suppose m_4 is selected. After Δt, m_4 enters A and the external coefficients a_{24} and a_{42} will be increased by some finite increments.

Suppose that at some later time m_1 and m_2 again appear in A as a result of the coupling pattern for tokens not shown. Now the adduction set F contains but one member, m_4. As a result of the increase in a_{24} on the previous pass, $f_4 > f_3$; and there will be no random choice this time. The A-sequence will again be $m_1, m_2, m_4, \ldots$, and a_{24} will be incremented again, giving m_4 a still greater advantage over m_3. Thus the selection of a token to follow m_2 into A was initially a stochastic process but has quickly become deterministic. The immediate and obvious conclusion we draw from this little experiment is that the growth of external bonds eliminates stochastic behavior that may have been implicit in the initial coupling matrix.

But suppose that at a still later time m_5 and m_6 appear in A. If the growth of a_{24} has been sufficiently small, so that $c_{24} < 4$, then the

A-sequence will be m_5, m_6, m_2, m_3, ... and a_{23} will be increased. A repetition of this sequence would result in $a_{23} = a_{24}$ and therefore $c_{23} = c_{24}$. Now if m_1 and m_2 again appear in A, $f_3 = f_4$ and a random choice between m_3 and m_4 is required. Thus it is clear that the sequencing of memory tokens in a particular group may be stochastic, then deterministic, then stochastic, and so on indefinitely. A change in λ is not required (as in the four-postulate machine) to shift the machine's behavior from stochastic to deterministic behavior or vice versa. The rule that stochastic behavior is progressively eliminated by the growth of external bonds must hold only for a class of initial coupling matrices. Unfortunately, I don't know what criteria define this class of matrices although small-scale experiments indicate that most simple strings, rings, and trees are members. This problem is interesting but may not be very important; for when sensory inputs are introduced in even a rudimentary form, the situation becomes immensely more complicated. For one thing, it becomes both possible and desirable to assume that initially there are no memory tokens in the machine (and hence no initial coupling matrix), and that the continuous creation of memory tokens via sensory inputs produces a growing coupling matrix.

A second behavioral problem of considerable importance is the general effect of varying λ. In the example of Figure 6 it was seen that "remote forward bonds" can produce remarkable changes in behavior as a function of λ. As a direct consequence of P6, such bonds (external type) are regularly produced when $\lambda > 2$. In order to examine this effect, it will be useful to introduce some auxilliary, temporary postulates governing the increment δ_{ij} in P6.

First we note that by P4, memory tokens enter the attention register one at a time, every Δt seconds. Thus the members of A can always be ordered by the times of entry; and given any two members, one is necessarily "older" than the other. We have been assuming all along that once the attention register has become filled, the entry of a new memory token causes the "oldest" token to be ejected. Now let us postulate that a_{ij} is increased by δ_{ij} every Δt seconds for which both m_i and m_j are both in the attention register. Similarly for a_{ji} and δ_{ji}. We also postulate (1) that if m_i is older than m_j, then $\delta_{ij} > \delta_{ji}$ (which results in asymmetric external bonds) and (2) that the increments δ_{ij} and δ_{ji} are the same for all pairs of tokens in the machine; let's say δ_1 and δ_2, respectively.

Consider now three tokens m_1, m_2, m_3 which, as a result of initial coupling coefficients, enter the attention register in that order. Case 1: $\lambda = 1$. Since no two tokens can exist in the attention register simultaneously, the external bonds between these tokens are not altered by sequencing through A. Case 2: $\lambda = 2$. Tokens m_1 and m_2 will be co-residents for Δt seconds with the result that a_{12} will be increased by δ_1 and a_{21} by δ_2. Similarly for a_{23} and a_{32}. Case 3: $\lambda = 3$. Now m_1 and m_2 will be co-residents for $2\,\Delta t$ seconds, and therefore a_{12} is increased by $2\,\delta_1$ and a_{21} by $2\,\delta_2$. Likewise for a_{23} and a_{32}. But m_1 and m_3 will also be co-residents for Δt seconds so that a_{13} will be increased by δ_1 and a_{31} by δ_2. Here we have the creation of remote forward and backward (external) bonds, the former being stronger than the latter. If λ is raised to larger values, still more remote bonds are formed.

The coupling pattern illustrated by Figure 6 now assumes a peculiar importance: such strings of external bonds are the natural result of permitting the machine to operate with $\lambda > 2$, regardless of the initial coupling pattern. Continued operation of the machine with large λ's will simply reinforce these strings because, as demonstrated earlier, under such circumstances the sequencing process acquires a certain "inertia" which carries it through intersections without deflection. Such strings produce deterministic behavior for $\lambda > 1$ except perhaps at intersections of the type in Figure 7. Of course predominantly stochastic sequencing may still be obtained at any time by setting $\lambda = 1$, which also terminates further reinforcement of external bonds.

Discussion of the six-postulate machine cannot be concluded without a few remarks on the classical "frequency" and "recency" laws. The former stated, in effect, that the more frequently two mental entities appear together in consciousness (for whatever reasons), the stronger the association between them. Thus repetition is a major basis of learning. Clearly, the growth of external bonds, as postulated in P6, produces such an effect. The "law of recency" is somewhat more subtle in its implications. It states that the more recently two mental entities have appeared together, the stronger the association between them. At first glance this "law" appears to require the postulation that associations decay in strength with the passage of time, in which case the law would be fundamental rather than derived from other postulates. However, it is not necessary to postulate decay of associations, at least in some cases, in order to achieve the effects described by the recency law. The example of Figure 7 shows that the growth of competing bonds can, in effect, reduce the strength of a given bond. The postulation of temporal decay in coupling coefficients remains an interesting possibility and would result in a distinct species of abstract machine. Such machines may exhibit unique properties, but we will not pursue the question here.

In closing, we note that P6 ignores an important issue, namely, the stipulation of limits on the growth of external coefficients. The associationists certainly assumed that there were upper limits on the strengths of bonds, limits imposed by the physical properties of the nervous system. The same assumption is necessary in the case of any physically realizable machine. If no temporal decay is present, then eventually all bonds that are capable of growth would approach the same strength; and stochastic processes (or

some substitute decision agent) would tend to dominate the machine's behavior. At least this is an obvious theorem. Of course if the growth rate of coupling coefficients is very small relative to the upper limits, then this problem might be ignored by assuming that the lifetime of the machine is too short to permit many coupling coefficients to reach their limit. However, we will not pursue the problem here for lack of space, evidence, and opinions. We are presently concerned primarily with the early growth phases of the abstract machine's life, rather than old age and senility phases.

The Nine-Postulate Machine

For logical reasons we have constructed the abstract machine from the inside. The six-postulate machine still has no contact with its environment. This is the reverse of the theory construction procedures utilized by the associationists. In developing his epistemology, Locke[13] began with his "tabula rasa" theory which, in opposition to the then popular doctrine of innate ideas, argued that the mind is initially like a blank tablet upon which sensory inputs wrote their record. The mind gradually forms by accumulations and interactions of sensations, images, and whatnot derived from sensory inputs. In their theoretical expositions the associationists typically followed this order of development, beginning with postulates concerning sensation and concluding with a description of the mind's internal machinery which ultimately results from sensory inputs. But this order is not necessary, and I find it more satisfying to begin with the internal machinery. It is largely a matter of taste, and the development of the abstract machine in this paper reflects, perhaps, prejudices gained in the computer field where one so often begins with the central processing units and leaves the question of inputs and outputs to the last (sometimes with unfortunate results).

In any event, the next group of postulates finally put the abstract machine in contact with its environment.

Postulate P7: There is a sensory register which at all times contains a set S of ψ sensory tokens s_1, s_2, s_3, ...s_ψ.

Postulate P8: At any moment, each sensory token s_i in S is bonded to each memory token m_j in A with a strength given by the "discrimination coefficient" d_{ij}; and each sensory token s_i in S also has a "vivacity coefficient" v_i.

Before introducing the ninth postulate it will be necessary to define a special function and related set. They are similar to the adduction function and set defined in connection with postulate P4.

For every sensory token s_i in S there is an admission function g_i whose value is given by

$$g_i = \sum_j d_{ij} + v_i \qquad (2)$$

where j ranges over the subscripts of memory tokens in A. Thus the admission function is the sum of discrimination coefficients between the sensory token and all memory tokens in A, added to the "vivacity" of the sensory token.

The admission set G is defined as follows: sensory token s_i is a member of G if, and only if, there is no other sensory token s_k such that $g_k > g_i$, and $g_i > \theta_s$ where "θ_s" is the admission threshold. Unlike the adduction set F, the admission set G may be empty; for the sensory token(s) with the largest admission function may fail to satisfy the criterion imposed by the admission threshold θ_s. We are now prepared to state the ninth postulate.

Postulate P9: Every Δt seconds a sensory token is chosen at random from the admission set G (unless G is empty) and entered into the attention register, causing the oldest memory token in A to be ejected from the attention register. As the sensory token enters A, it becomes a memory token (member of M).

It will be assumed that although sensory and memory tokens may enter the attention register at the same rate, namely, one per Δt seconds, these processes are not in phase. Thus a sensory and a memory token will never enter A simultaneously, and there is no need to modify the ejection rules. It is important to note that nothing has been said about the nature of sensory tokens or their period of residence in the sensory register. We will assume that a particular sensory token may enter S and, if not admitted to A, remain in S for a long period of time. And if it is admitted to A finally, it may be immediately replaced in S by another sensory token identical in structure and properties. This represents the presence of a constant stimulus condition in the environment.

Thus the machine cannot influence the appearance of sensory tokens in S, and its only control over the ejection of sensory tokens is by way of admitting them to A where they become memory tokens, i.e., permanent parts of the machine. It is the environment which determines the nature and sequencing of sensory tokens in the sensory register; the machine is free only to select, from those presented, the individuals that will be admitted. If the machine was given motor apparatus for acting on the environment or reorienting itself relative to the environment, then of course it would gain some measure of control over the appearance of sensory tokens in S. There is insufficient space in this paper to introduce an efferent system by formal postulate, but the subject receives some discussion in the concluding section.

The discrimination coefficients represent bonds from sensory tokens to memory tokens. Their values reflect relations between the structures of tokens, not past experience, and are therefore similar to the internal coefficients previously

introduced. The discrimination coefficients constitute a first approximation to the effects of a pattern-recognition process, a measure of the "similarity" between sensory and memory tokens. Since the discrimination bonds are presumed to exist only between sensory tokens in S and memory tokens in A, a memory token not in A cannot influence the admission of sensory tokens. Thus the attention register still has the pivotal role in the nine-postulate machine.

The vivacity coefficients represent an attempt to reproduce the effects of "vivacity" as postulated by associationists. They assumed that the vivacity of a sensation or percept is a function of stimulus intensity and other factors, such as object-field contrasts. Further refinements of the abstract machine would require postulates governing variables which determine the value of a vivacity coefficient (stimulus intensity, for example), but in this paper the coefficient will be treated as a parameter.

At this point in the machine's development we might well postulate a vivacity coefficient for each memory token and redefine the adduction function accordingly, for the associationists universally assumed that the objects of memory also had varying degrees of "vivacity." To be consistent with the classical theory such a postulate must eventually be introduced, but again the resulting complications are too numerous for treatment within the confines of this paper.

The introduction of the admission threshold θ_s provides a distinctly new type of system parameter comparable in importance to λ. Large values of θ_s will tend to isolate the machine from its environment, while low values will permit a steady influx of sensory tokens. If, in further refinements of the abstract machine, θ_s is converted to a system state variable determined by events within and without the machine, then there will be rich opportunities for subtle interactions between the machine and its environment. Eventually λ and ψ (the length of the sensory register) must also be converted to system-state variables with similar consequences, as touched upon in the concluding section.

Discussion of the Nine-Postulate Machine

With the addition of the last three postulates we have two "registers," side by side as it were, with a procession of sensory tokens through one and memory tokens through the other. Depending upon discrimination and vivacity coefficients, there may be periodic transfers from S to A so that the sensory procession may influence the attention procession, but not vice versa. The events in the abstract machine now represent a crude approximation to the picture of sensory and mental processes as painted by many associationists. For example, Spencer[16] imagines himself driving along the seashore and describes the psychological events as follows (his "vivid states" correspond to our sensory tokens, the "faint states" to our memory tokens): "In broad procession the vivid states--sounds from the breakers, the wind, the vehicles behind me; changing patches of colour from the waves; pressures, odors, and the rest--move on abreast, unceasing and unbroken, wholly without regard to anything else in my consciousness. Their independence of the faint states is such that the procession of these, in whatever way it moves, produces no effect whatever on them. Massed together by ties of their own, the vivid states slide by resistlessly. The procession of the faint states, however, while it has a considerable degree of independence, cannot maintain complete independence. The vivid states sweeping past always affect it in a greater or less degree--drag part of it with them by lateral cohesion."

This rather poetic passage is, incidentally, a good example of the theoretical statements from which one must squeeze the postulates for an abstract machine. The associationists assumed quantifiable forces but rarely bothered to quantify specific parameters and variables.

Spencer's "lateral cohesion" is represented by the discrimination coefficients in our abstract machine. Rather than using his ambiguous verb "drag," we speak of the transition from S to A, the transfer and conversion of sensory tokens into A where they can influence succeeding A-sequences. These transitions correspond to the process of "copying" in association theory; i.e., an associationist would call our memory tokens "copies" of "sensations" or "percepts," although we shall think here in terms of the physical transfer of tokens from one register to the other.

The most important consequence of P9 is that the machine can now "grow." There is no need to assume the existence of memory tokens at the birth of the machine. As memory tokens are created, the coupling matrix expands in size. A newly created memory token will have no external bonds to or from other memory tokens, but will immediately begin to form such bonds with whatever other tokens are in A at the time. Since internal bonds represent inherent relations between memory tokens, when a token m_i is created, b_{ij} and b_{ji} are immediately fixed for all j for all time.

What sort of memory token structures will grow up as a consequence of sensory inputs to the abstract machine? We shall direct our attention primarily to this basic problem.

As a first example, consider the fragmentary network of memory tokens in Figure 8. The tokens are represented by two kinds of symbols--squares and circles--to suggest similarities and dissimilarities of token structure, hence the internal bonds. Suppose that at some time in the past m_2 and m_4 were created from sensory tokens in that order. They are dissimilar in nature but were co-residents of the attention register, and an external bond has formed between them. We shall assume in the following discussion that external bonds are highly asymmetric, i.e., that $\delta_1 \gg \delta_2$, so that for all practical purposes we can assume a single external bond from the older to the

younger token. Here m_2 was created first, followed by the creation of m_4 after Δt seconds.

Now suppose that at some later time another sensory token is admitted to A, becoming m_1, and due to a great similarity in structure (detected perhaps by a pattern-recognition system) there are large internal coefficients for the pair m_1, m_2. As a consequence, m_2 gains entry to the attention register. Let $\lambda = 4$, and let m_3 be created Δt seconds after m_1. Now we have m_1, m_2, m_3 in the attention register. Due to previous co-residence $a_{24} > 0$ and due to similarity of structure $b_{34} > 0$. Acting together it is likely that m_4 will next be entered into A. Now the attention register contains m_1, m_2, m_3, m_4. In addition to the internal bonds between m_1, m_2 and between m_3, m_4, external bonds from m_1 to m_3 and m_4, and from m_2 to m_3 will be formed, while the external bond from m_1 to m_4 is reinforced; and external bonds between m_1, m_2 and m_3, m_4 will form to give extra support to the internal bonds. As a result, the coupling pattern of Figure 8 will appear.

First we note that if, as we have supposed above, newly created memory tokens are able to dominate the selection of memory tokens in the normal A-sequence, then internal coefficients will be acting decisively; and there will be a tendency to form "clusters" of tokens. The pairs m_1, m_2 and m_3, m_4 are examples of the simplest possible clusters. Furthermore, if events in the environment are such that two or more types of memory token are repeatedly created in a certain order via the sensory register, e.g., m_1 followed by m_3, and m_2 followed by m_4, then strings of clusters will be created. The coupling within a cluster, being a combination of both external and internal bonds, will be stronger than the coupling between clusters, which we assume will be almost entirely the result of external bond formation.

A string of clusters might also be viewed as a bundle of parallel strings with numerous cross-bonds. But, in any event, it is clear that the sort of strings and chains assumed in our analysis of the more primitive four and six-postulate machines can be created in profusion by sensory inputs. If λ is increased well above 4, then remote forward and backward bonds between new and old memory tokens will also form. A reduction of λ to, say, $\lambda = 2$ will result in the creation of only the simplest sort of string.

If sensory tokens gain entry to A mainly on account of high vivacity coefficients and are unable to materially influence the normal sequencing of memory tokens, they will be merely "tacked on" to pre-existing strings by relatively weak external bonds. Although they have become weakly coupled satellites of a strong string, they may at future times provide a basis for formation of cross bonds, via strong internal coefficients, between widely separated tokens in the main string, a process that might be called "looping."

Returning to Figure 8, suppose again that the creation of m_1 results in the entry of m_2 into A, but that the sensory token which became m_3 (call it s_3) is this time unable to enter S by reason of a high admission threshold. However, the previously formed external bond from m_2 to m_4 is sufficient, let us assume, to draw m_4 into A. We now have m_1, m_2, and m_4 in A, with s_3 sitting in S. Since s_3 is presumed similar in structure to m_4, the discrimination coefficient d_{34} is large; and we will assume that now $g_3 > \theta_s$ and s_3 enters A, becoming m_3.

This little example shows an extremely important facet of the machine's behavior stemming from postulated discrimination coefficients. The contents of the attention register at any moment may determine which, if any, of the sensory tokens will be admitted. The machine need not passively ingest whatever sensory tokens are presented; rather, under our current postulates, those sensory tokens which have certain special relations with memory tokens in A are given an advantage. In general, the machine will tend to select those sensory tokens whose nature and order of appearance match the memory tokens concurrently passing through the attention register. We might say that the machine "pays attention" to some sensory inputs and "ignores" others, depending upon the state of the machine at the time. By now it should be abundantly clear why the attention register is so named. Of course this selectivety may be abolished at any moment by the appearance of sensory tokens with high vivacity coefficients and/or a marked decrease in the system parameter θ_s. A very vivacious sensory token can "force itself" upon the machine and, by way of internal bonds, disrupt the current A-sequence and initiate a distinctly different series.

It should be noted that if λ is large, say, $\lambda > 5$, then it will be less likely that sensory tokens disrupt the normal sequencing of memory tokens unless the machine's memory tokens are sparsely bonded into simple strings with few remote forward or backward bonds. A very youthful machine will, in fact, tend to have only sparsely bonded strings and hence will be strongly influenced by sensory inputs even if λ is large. But as the machine grows older and the memory tokens become densely bonded, it will be decreasingly influenced by its environment; indeed whenever λ is reasonably large, an old machine will tend to select only those sensory inputs which fit in with its experience (internal and environmental). Thus it appears that as the machine grows older it will display a type of rigidity in its behavior (relative to the environment) which is quite distinct from the tendency to deterministic sequencing discussed earlier.

In Figure 9 we suppose that at some time in the past m_2 and m_4 were created in sequence and are externally bonded. Now m_1 is created and due to a strong internal bond draws m_2 into A. But the creation of m_1 is followed by the creation of m_3, which is quite different from m_4. An external bond is formed from m_2 to m_3, and it may be as strong as the old bond from m_2 to m_4. We have in this process a basic mechanism for the creation of

"tree" structures in the coupling matrix. If in the future m_2 enters A, then there may follow a random choice between m_3 and m_4 unless, of course, a sensory token is meanwhile admitted and turns out to have a strong internal bond with m_3 or m_4, thereby determining the A-sequence. On the other hand if, say, $c_{24} > c_{23}$, the A-sequence is deterministic unless a sensory token is meanwhile admitted and creates a memory token with a strong internal bond to m_3. It may be then that $f_3 = f_4$, and the A-sequence becomes stochastic rather than deterministic.

In brief, it is clear that not only can tree patterns of coupling be easily produced, but that further sensory inputs may either cause or prevent stochastic sequencing in such tree patterns. Having shown how branches are formed, the equally important formation of merging strings must be demonstrated.

Consider Figure 10. At some time in the past, m_2 and m_4 were created in sequence and are accordingly bonded externally. At a later time, m_1 is created, followed by the creation of m_3, which has a strong internal bond to m_4. Assuming $\lambda > 2$ and that m_3 causes m_4 to enter A, an external bond from m_1 to m_4 is formed. Hence m_4 becomes a point at which strings merge. In the future, both m_1 and m_2 may be followed by the same token, m_4.

Since the machine can produce strings, branching strings, and merging strings, it appears that any conceivable finite coupling matrix can also be created given appropriate sensory inputs in the past. But this is not a proven theorem. It is an extremely important theorem because, if true, then whenever one finds a coupling matrix which produces a desirable or interesting behavior pattern (with or without further sensory inputs assumed), it is guaranteed that there exists at least one possible history of the machine which would lead to the coupling matrix in question. My intuition tells me that the theorem must be true, but it also seems quite likely that new behavioral principles will emerge if machines with large numbers of memory tokens (say 100,000) can be examined in detail. It is conceivable that really large token ensembles, given the postulates introduced here, have a class of "forbidden" states, i.e., forbidden coupling matrices. I have no proof, and of course the classical associationists never pursued their theory in sufficient detail to even recognize the existence of this problem.

A less serious problem arises from the likely formation of clusters of memory tokens connected by strong internal bonds (as in the first example above, Figure 8). It appears that if the number of tokens in a cluster exceeds λ, the machine would tend to be trapped, to cycle endlessly within the cluster. This is another problem not recognized by the associationists. It would be desirable to have just one or two tokens from a cluster enter A, "stand for" the whole cluster as it were, and then have the A-sequence move on to the next cluster. This seems to have been assumed by the associationists; but it is clear, within the framework of our present machine, that a special kind of postulate must be added in order to produce the desired behavior. It will be touched upon in the concluding section.

There are numerous other effects, such as conditioning and gestalt responses, which can be demonstrated, at least in a rudimentary form, to be possible in the nine-postulate machine. Some of these were discussed briefly in connection with the behavior of the four- and six-postulate machines and can be easily extended to the nine-postulate case.

To summarize, the introduction of sensory inputs in even the crude form of P7, P8, and P9 yields a machine which grows with the passage of time. It is not "preprogrammed" in the usual sense, having just two "registers," the ability to create permanent memory tokens from transient sensory inputs, and two kinds of coupling between tokens --- one that suggests physical links (external bonds) and one that suggests the existence of a pattern-recognition apparatus or "resonance" phenomenon (internal bonds). The coupling patterns which grow within the machine reflect the pattern of events in the environment. As the machine grows older, these patterns may become self-reinforcing by selecting from the sensory input only those tokens which match themselves. This tendency may be disrupted by the appearance of sensory tokens with relatively large vivacity coefficients (e.g., due to strong stimuli) or by a drastic reduction in the length of the attention register (in psychological terms, a state approximating that achieved in "free association").

If the sequencing of memory tokens into the attention register proceeds at a somewhat faster rate than the admission of sensory tokens, it will, in effect, be continually "predicting" future events in the environment on the basis of past experience. By this I mean that a sequence of the sort obtainable from the group in Figure 8, for example, can lead to a prediction: the A-sequence m_1, m_2 can bring in m_4, which is a "prediction" of the type of sensory token that will appear in S. But of course the machine has the nasty tendency to select for admission those sensory tokens which satisfy its own predictions --- a habit not entirely unknown in humans and perhaps exemplified by this paper.

Conclusion and Glimpses of Further Postulates

It must be emphasized that the nine-postulate machine constitutes neither a complete nor unique interpretation of classical association psychology. The vagueness of the associationists' doctrine and the variations from one author to the next suggest a variety of interpretations. We have presented here but one partial interpretation.

The analysis of the abstract machine's behavior has been based on small groups of tokens. However, the association theories were intended to

explain the behavior of the human mind, a very large collection of objects to say the least. The major problem posed by the abstract machine is, similarly, its behavior when large numbers of memory tokens --- say 10^5 to 10^7 --- have been accumulated and are interacting with each other and with environmental inputs and outputs. Small-group analysis can at best suggest general conclusions regarding large group effects, and the arguments in this paper are only tentative. We are in dire need of techniques for analyzing massive collections of tokens.

From the standpoint of artificial intelligence, the most interesting aspect of association theory is its purported ability to account for the growth of mechanisms that can "think," a growth that begins with nothing more than a few forces operating on the products of sensory inputs --- Locke's "tabula rasa" concept. If the associationists' claims are accurate, then they have, in fact, invented a machine which structures itself without the aid of "pregrogramming." But their claims have never been confirmed experimentally (in humans) nor have the logical consequences of their axioms been explored in any detail. Abstract machines of the sort discussed here provide a vehicle for such explorations providing that sufficiently powerful analytical techniques can also be evolved.

Computer simulation seems to offer, at present, the most promising approach to analyzing the abstract machine, although mathematical analysis has not yet been given a fair testing. Whether or not the complex combinatorial problems, which arise when a reasonably complete machine has grown beyond several hundred tokens, can be successfully attacked by conventional mathematical techniques remains an open question awaiting the attention of suitably talented mathematicians. However, the abstract machine has been constructed in such a way that it can be simulated --- though the cost might be forbidding --- on a general-purpose digital computer.

In constructing the abstract machine, the most difficult decision is confronted in formulating a mechanism for choosing among tokens which have equal claims to succession. This problem is inherent in any quantified associational system where objects are competing for attention. The associationists never faced this problem directly, preferring to leave a certain loophole through which a free-will soul might be slipped should the theory threaten to become too successful.

We have here postulated a selection mechanism which utilizes random choices only when two or more tokens have exactly equal claims on the machine's attention. Such random selections are presumably based on some "noise" mechanism. Depending upon assumptions regarding the growth of the coupling matrix, random selections may or may not be frequently required. We have shown by example how stochastic processes appear or disappear as a function of coupling patterns and variations in λ. I believe this is one of the most interesting aspects of the association problem. Of course there are promising alternative mechanisms, e.g., a machine in which coupling coefficients are treated as transition probabilities; but space is insufficient to discuss them adequately so they have been ignored altogether.

The forces operating in the machine, as represented primarily by external, internal, and discrimination coefficients, are few but by no means simple in their physical implications. They have been assumed without regard to the difficulty of physically realizing them in any real machine (although, again, they can be simulated by brute force methods). The objective is to explore the logical consequences and promise --- if any --- of associational mechanisms. If they are sufficiently promising, attention can then be directed to the problems of physical realization.

The results of research to date, only part of which is reported here, is inadequate for any firm conclusions as to the ultimate capabilities of the abstract machine. That the machine is capable of "learning" in at least several senses is obvious, although we have avoided the use of that term. The machine does produce an internal "image" or "model" of both regularities and irregularities in the external world, and this internal structure can clearly be used to predict the future in a rudimentary way. Assumptions regarding the possible species of memory and sensory tokens have been avoided but remain a complex problem of great importance that must be bypassed in this introductory paper. As noted in the introduction, the aim of this paper is to be suggestive rather than definitive, a necessity in view of the limited data in hand and restricted space available.

The nine-postulate machine falls so far short of the implications of association theory that it would be misleading to conclude without a few brief remarks on other postulates that should be and are under investigation.

First, a group of postulates creating a motor apparatus is required. An "efferent register" containing a set E of tokens seems the most direct approach. We distinguish two species of memory tokens: those that can be entered into E and those that cannot. In order to deal with the most primitive types of reflex behavior, it may be desirable to permit the direct transfer of sensory tokens from S to E. In any event, the set E causes the machine to act on its environment in specified ways. This makes it possible to consider the dynamics of feedbacks through the environment to the sensory register. In particular, the machine is then capable of focusing on some part of the environment and thereby stabilizing certain subsets of S. More importantly, it must be postulated that the scope of S can be expanded or contracted, relative to a sensory manifold, by the members of E.

Second, the sensory register must be broken into several parts representing different sensory modes, and coupling coefficients between sensory

tokens within a mode be introduced. These coefficients should be partially responsible for determining vivacity coefficients, and for admission functions connected with the multilevel attention register discussed below.

Third, all memory tokens must be assigned vivacity coefficients whose values are functions of those accompanying the sensory tokens from which these tokens were created. The adduction function must be appropriately redefined.

Fourth, an entirely new type of coupling must be introduced. In this paper we have considered only token-token coupling, but token-system coupling is required to represent the influence of general "system states" upon the machine's behavior. For example, we postulate a system state parameter P which ranges between positive and negative limits: call it the "Pleasure-Pain" scale. When memory token m_1 is created, its pleasure-pain coefficient p_1 is set equal to the current value of P. The adduction function is redefined to $f_i = \sum_j c_{ji} + Pp_i$ When P and p_i are of the same sign and relatively large, the state of the system as a whole can be responsible for the selection of m_i for entry into A. The next step is to convert P to a state variable whose value is determined, partially at least, by certain sensory tokens. Further refinements to this token-system coupling scheme carry one well beyond the classical theory.

Fifth, several levels of attention registers must be postulated. As Bain[3] put it in a poetic outburst: "No associating link can be forged.... except in the fire of consciousness; and the rapidity of the operation depends on the intensity of the glow." He was referring here only to those types of bond which can be reinforced; in our machine, external bonds. It was generally recognized that there are degrees of consciousness, and we interpret this as discrete attention registers. Let us say that there are n attention registers of lengths $\lambda_1, \lambda_2, \ldots \lambda_n$ which hold sets $A_1, A_2, \ldots A_n$. Each attention register is assigned a "level coefficient" k with $k_1 < k_2 < \ldots < k_n$. These may be thought of as "energy levels," and the adduction and admission functions as measures of energy available to the corresponding tokens. The level coefficient of the set M is assumed zero, and the admission threshold becomes the difference between the levels of S and A_1. Figure 11 shows an example of the postulated relations between levels of M, S, and the various A sets. The adduction set F must be redefined to include a comparison of adduction functions with a set of adduction thresholds similar in effect to θ_s. In Figure 11 there are four sets of transitions labeled a, b, c, and d. The first shows the types of transition involved in the simple transfer of a sensory token from S to A_1, where it becomes a memory token, and after brief residence in A_1 ejection. These are the kinds of transition discussed in connection with the nine-postulate machine. Set b shows the entry of a memory token directly into A_2, its drop to A_1, after a brief period, then finally its ejection. Transition c is forbidden by postulate. Set d shows how a token may move among the various attention registers for a considerable period of time before ejection.

As suggested by Bain, the increment in external bonds between co-residents within any given attention register is an increasing function of the level of that register. Thus external coefficients for co-residents in A_2 would increase at a greater rate than those for co-residents in A_1. Clearly multilevel attention registers create rich opportunities for multiple sequencing and the formation of complex coupling patterns.

Having once connected the notion of "energy level" with the level of a register, it is natural to consider conservation postulates. Stout's theorizing[17] leads to some rather exciting possibilities in this regard. A conservation postulate that suggests itself here is $\Sigma k_i \lambda_i$ = constant. This in turn suggests a postulate, not unreasonable in a physiological interpretation of association theory, that similar tokens are stored in similar places (to put it crudely) and that the energy supplied to any particular neighborhood is limited. Thus when a token is once raised to a particular energy level (i.e., attention register), the adduction functions of similar tokens (as measured by internal coefficients) are correspondingly reduced, thereby lessening the tendency for endless cycling within clusters of similar tokens --- a problem mentioned earlier.

These examples of further postulates give some indication of the incompleteness of the nine-postulate machine. And if one attempts to extend the abstract machine much beyond the explicit and implicit assumptions of classical theory, the nine-postulate machine constitutes only the barest beginning. Consider the problem posed by the creation and use of linguistic symbols in goal-directed thinking. The postulates presented or suggested here appear quite inadequate to produce such behavior. If linguistic symbols are always assumed to be constructed from stimulus patterns (mainly visual and auditory), then of course the nine-postulate machine, not to mention the more complex machines produced by the suggested additional postulates, would be capable of learning to associate such symbols with clusters of tokens which constitute the internal denotations. But this capability is still far removed from goal-directed thinking.

There are hints in the literature, particularly in Taine[18], which when followed up may lead to sufficiently powerful postulates that are not too question-begging and are consistent with the general approach of association theory. However, we can only conclude here that the postulates so far considered could produce at best a machine having a level of intelligence comparable perhaps to that of the lower vertebrates. And one might understandably take issue with this tentative conclusion, for the nature of the sensory and memory tokens have not been stipulated in sufficient

detail for comparisons with animal behavior, nor have we carried out methodical experiments on reasonably large abstract machines.

Acknowledgments

We are pleased to acknowledge the support of the U.S. Air Force Office of Scientific Research (Contract No. AF 49(638)-1039) in carrying out the research upon which this paper is based.

References

1. Bain, A.: "The Senses and the Intellect", London, 1855.

2. Bain, A.: "Mind and Body", London, 1876.

3. Bain, A.: "Association Controversies"; reprinted from Mind xii in "Dissertations on Leading Philosophical Topics", Longon, 1903.

4. Bain, A.: "The Respective Spheres and Mutual Helps of Introspection and Psycho-Physical Experiment in Psychology"; reprinted from Mind N.S.ii in "Dissertations on Leading Philosophical Topics", London, 1903.

5. Berkeley, G.: "A Treatise Concerning the Principles of Human Knowledge", 1710.

6. Boring, E.G.: "A History of Introspection" Psychological Bulletin, 50, pp 169-189, 1953.

7. Brown, T.: "The Philosophy of the Human Mind", 1820.

8. Hamilton, W.: Appendices D** and D*** in "The Works of Thomas Reid," 1846.

9. Hartley, D.: "Observations on Man," 1749.

10. Hobbes, T.: "Leviathan," 1651.

11. Hume, D.: "A Treatise of Human Nature", 1739.

12. James, W.: "The Principles of Psychology", 1890.

13. Locke, J.: "An Essay Concerning Human Understanding", 1690.

14. Mill, James: "Analysis of the Phenomena of the Human Mind", 1829.

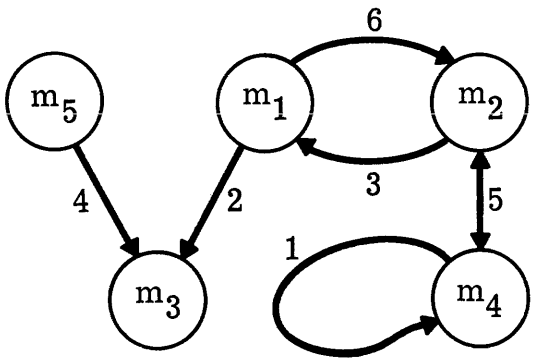
15. Mill, J.S.: "System of Logic", 1843.

16. Spencer, H.: "The Principles of Psychology", 1872.

17. Stout, G.F.: "Analytic Psychology", London, 1896.

18. Taine, H.: "On Intelligence", New York, 1875.

19. Warren, H.: "A History of the Association Psychology", New York, 1921.



a.

0	6	2	0	0
3	0	0	5	0
0	0	0	0	0
0	5	0	1	0
0	0	4	0	0

b.

Figure 1

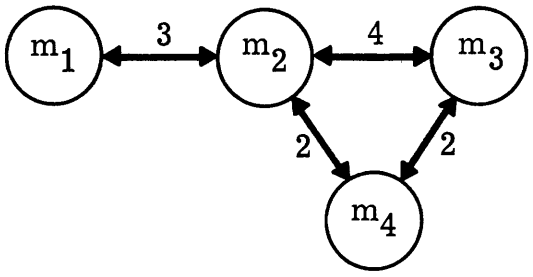


Figure 2

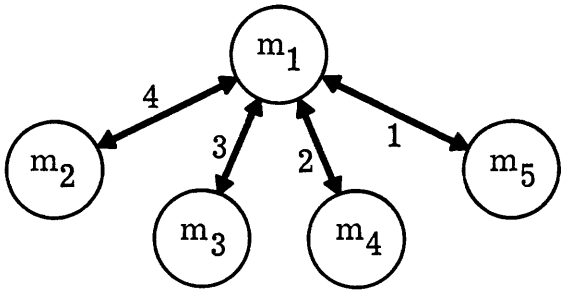


Figure 3

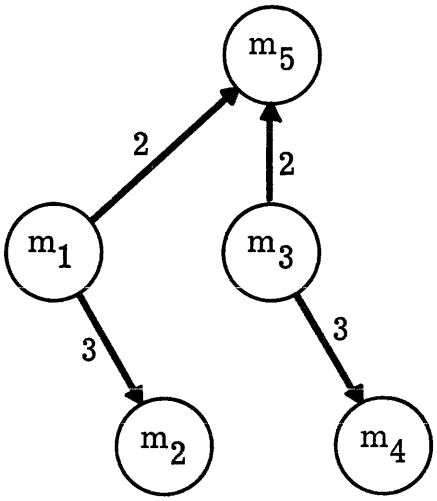


Figure 4

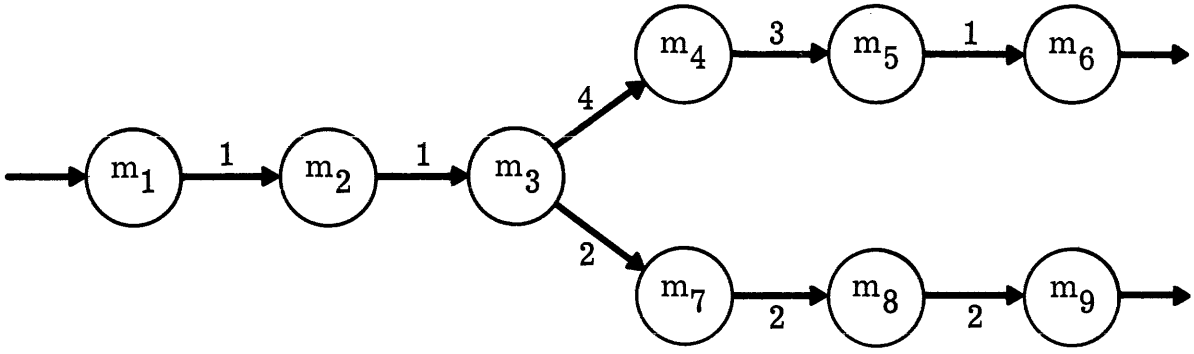


Figure 5

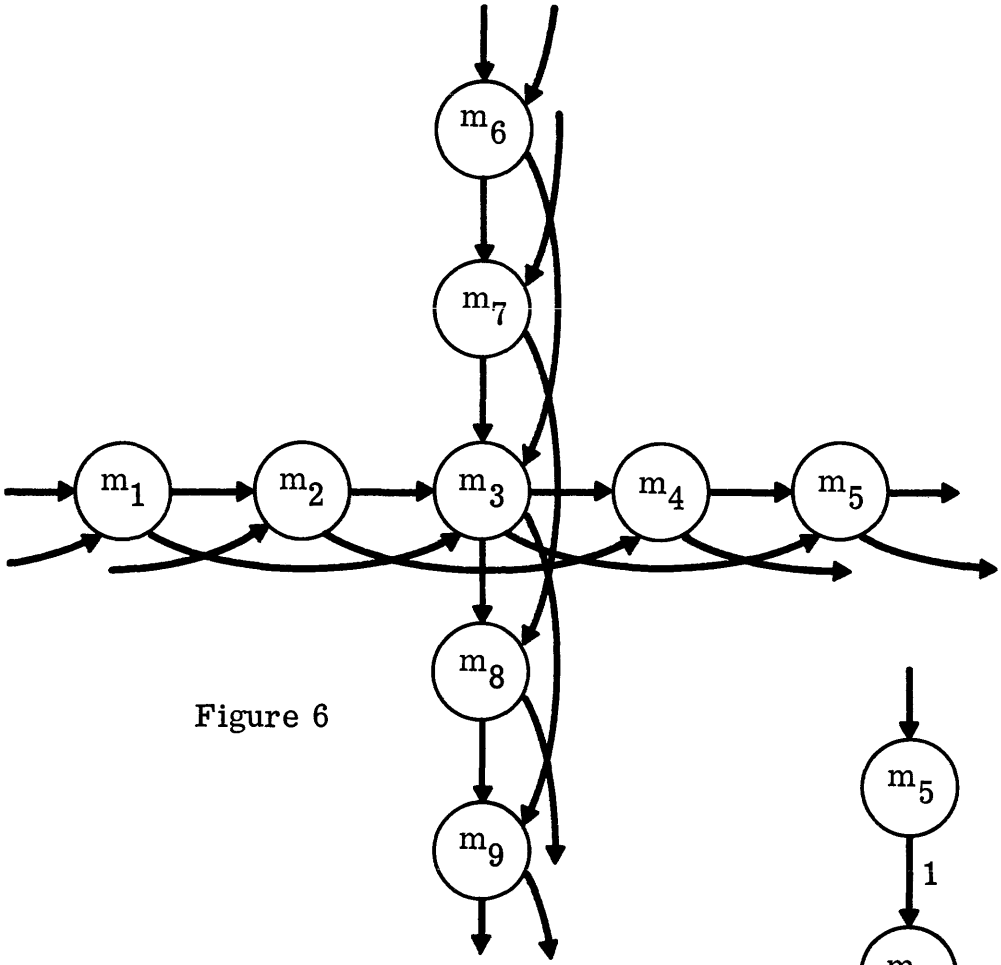


Figure 6

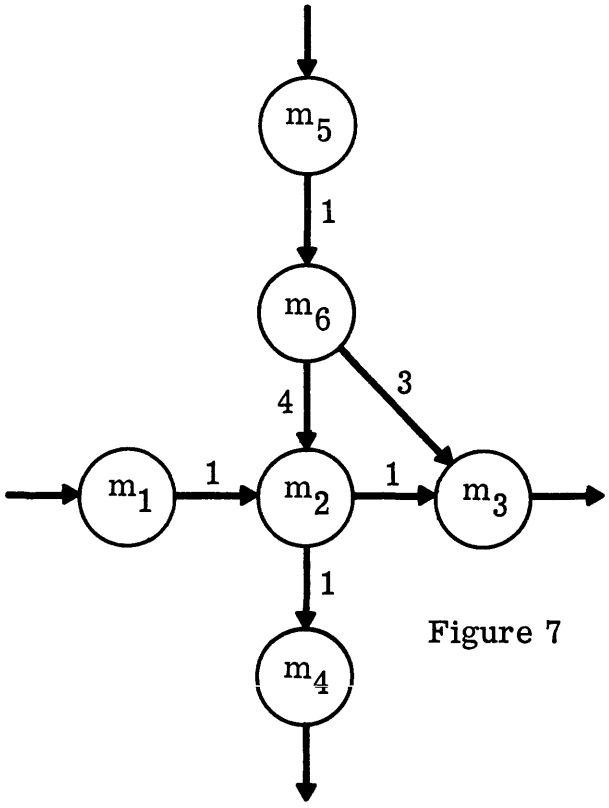


Figure 7

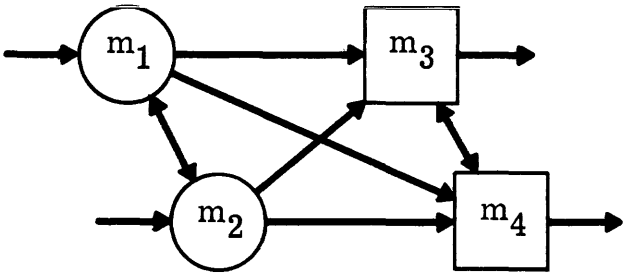


Figure 8

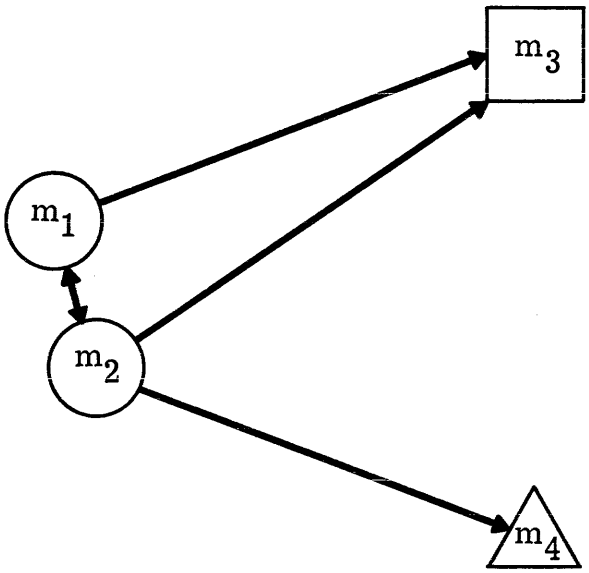


Figure 9

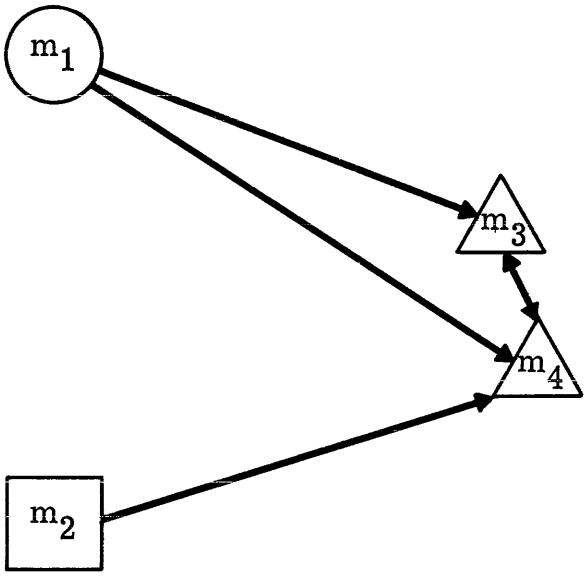


Figure 10

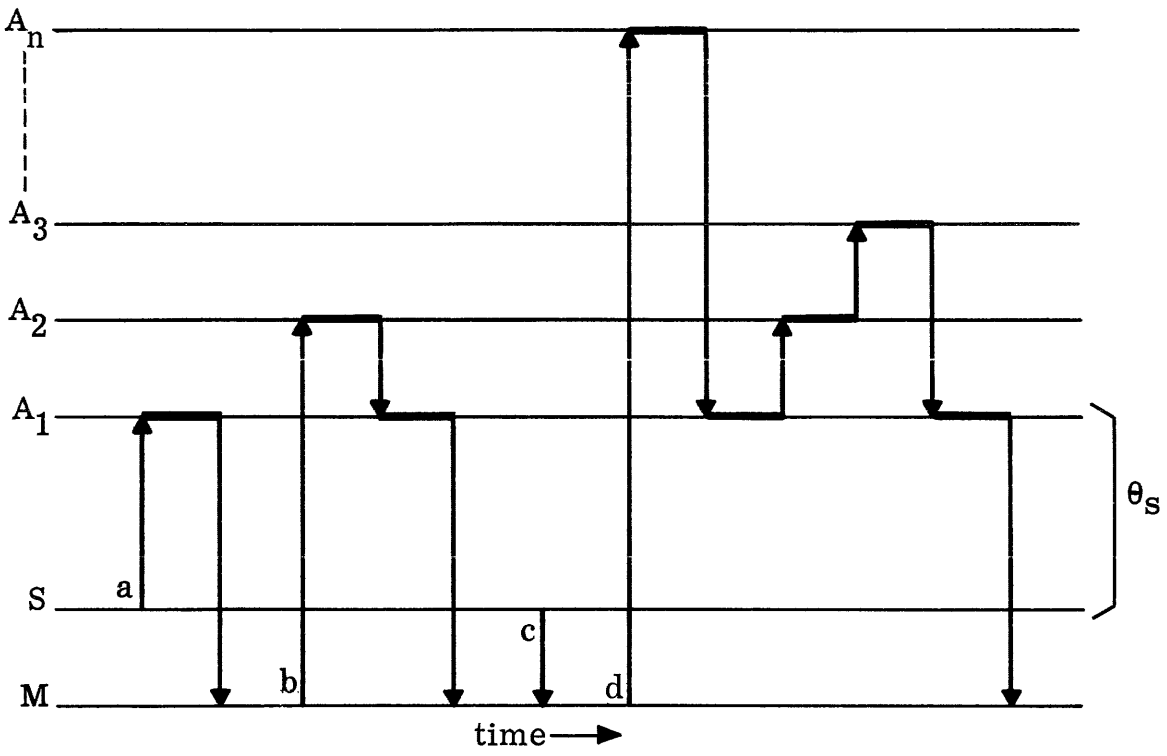


Figure 11

THE GÖDEL INCOMPLETENESS THEOREM AND INTELLIGENT MACHINES

Frank B. Cannonito

Grumman Aircraft Engineering Corp.

Bethpage, New York

There is a belief in some quarters that Gödel's incompleteness theorem expresses the existence of an intrinsic property of computing machinery which limits their use as creative robots and renders them unsuitable for the simulation of intelligent behavior[1]. We do not subscribe to this view[2], and it will be the purpose of this paper to indicate why not. To do this, we shall develop in Part I, the concepts of recursive function theory necessary to state Gödel's theorem so that an intelligent argument as to its consequences may be inferred. The method we have chosen - programs - seems to us to be that with which the reader will be most familiar and which has the greatest intuitive appeal.[3] The main result, the Gödel incompleteness theorem, will then appear as a statement to the effect that a certain set of integers can not be generated by a program. In Part II we show how in certain cases, sets of integers having similar properties may be generated by a modified program, and draw some conclusions vis-à-vis machine intelligence. We wish to emphasize that while our presentation is very informal, it is possible to give rigorous demonstrations of all theorems stated, and we shall henceforth regard this as implicit.

Part I

The specific concept we want to obtain is that of a recursively enumerable set of integers. We will see that this means simply that one can write a program which generates the set in a manner to be described below. We note that all integers will be assumed to be nonnegative without express mention to the contrary, and all functions will be integer-valued.

We wish to define what we will understand to be a 'program'. Let us suppose that we have a computer with an (unlimited) supply of storage locations A, B, C, . . . which, if otherwise is not known to be the case, are set to zero. If A is any storage location, $|A|$ will denote its contents. By a program, we mean a finite nonempty ordered sequence of instructions from the following list:

(i) STORE $|B|$ IN A

(ii) STORE $|A| + 1$ IN A

(iii) STORE $|A| - 1$ IN A

(iv) IF $|A| = 0$ GO TO m,
OTHERWISE GO TO n

(v) WRITE $|A|$

(vi) HALT

where m and n are integers. For convenience, we will identify any location A with its contents $|A|$, and use

(i*) A = B

as an abbreviation of (i), and similarly,

(iv*) IF A m, n

will abbreviate (iv).

Let $f(x_1, \ldots, x_n)$ be a function of n integer variables, and let $a_1, \ldots, a_n$ be n integers. If there is a program P, such that whenever P is started off with $a_1, \ldots, a_n$ stored in locations $A_1, \ldots, A_n$, then P halts after writing $f(a_1, \ldots, a_n)$, we say P computes the function $f(x_1, \ldots, x_n)$ at the point $(a_1, \ldots, a_n)$. If P computes the function $f(x_1, \ldots, x_n)$ at all points $(a_1, \ldots, a_n)$ where $f(a_1, \ldots, a_n)$ is defined, and computes something at only those points, we say that P computes $f(x_1, \ldots, x_n)$. A function $f(x_1, \ldots, x_n)$ is said to be computable if there is some program which computes it. Functions $f(x_1, \ldots, x_n)$ which are defined for all $(a_1, \ldots, a_n)$ will be called total.

For example, the function $x_1 + x_2$ is computed by the following program:

1. IF A 2, 4
2. WRITE B
3. HALT
4. A = A - 1
5. B = B + 1
6. GO TO 1 [4]

The appearance of subprograms can be conveniently abbreviated by introducing the following convention. When the subprogram computes a function $f(x_1, \ldots, x_n)$ and stores it in location A, we write

$$A = f(x_1, \ldots, x_n)$$

understanding by this that the actual sequence of instructions which compute $f(x_1, \ldots, x_n)$ may be written down if necessary.

Some functions are not total, a simple example is given by this program which computes the smallest value of y such that 1 + y = 0. Since we are dealing with nonnegative integers only, no y satisfies this relation; the program never halts.

1. A = A + 1
2. C = A + B
3. IF C 4, 6
4. WRITE B
5. HALT
6. B = B + 1
7. GO TO 2

We have the necessary notions at out disposal to define the concept of a recursively enumerable set of integers. Let M be a set of integers. We say M is recursively enumerable if there is a program which prints out each member of M and no other integers. If there is a program which prints out 0 for inputs belonging to M and 1 otherwise, we say M is recursive.

It is easy to enumerate a recursive set; for example, if M is recursive and f(X) is the function

$$f(X) = \begin{cases} 0, & X \text{ in } M \\ 1, & \text{otherwise} \end{cases}$$

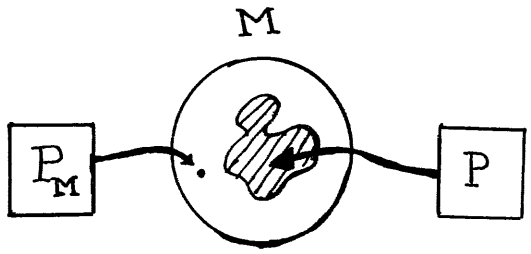
then we enumerate M by the program

1. A = f(X)
2. IF A 3, 4
3. WRITE X
4. X = X + 1
5. GO TO 1

which prints out all the members of M and only those.

We can cite some properties of sets of integers. These results are well known and can conveniently be found in reference 1. If M is recursive, so is its complement $\overline{M}$. Also, if M and its complement $\overline{M}$ are both recursively enumerable, then M is recursive. It is well known that there exist sets of integers that are neither recursive nor recursively enumerable. In fact, as we have already noted, Gödel's theorem will appear in terms of a particular type of nonrecursively enumerable set which we shall describe now.

A non empty set of integers M, see figure, is called productive, if there exists a program P_M (called the productive program for M), such that given any other program P which enumerates a subset of M, P_M gives a member of M not enumerable by P[5]. Since no recursively enumerable subset of a productive set exhausts the set, it is clear that productive sets are not recursively enumerable.



Possessing, at last, all of the machinery of recursive function theory we shall need in the sequel, we want now to use these methods in discussing formal languages so that we may exhibit Gödel's theorem and draw our first conclusion.

We therefore will consider the possibility of producing programs which can recognize, generate and prove formulas in some formal language, and see what lies implicit therein. We mention that to a large degree this is precisely what is done by the Arithmetic Translator-Compiler of the IBM FORTRAN Automatic Coding System, or any other similar system for that matter. In the practice with which we are accustomed, the

FORTRAN coding is rendered into numerical data which is punched into cards and fed into the machinery doing the compiling. Thus it is not unreasonable to think of a formal language as being a collection of integers. That is, each word in the language, each sentence, each paragraph, is represented in the language by a unique integer. Traditionally, formal languages did not arise in this way. They more or less resembled the formal symbolic language of mathematics. However, it was shown by Gödel that with each word W in the formal language, one is able to associate a unique integer gn(W) called its Gödel number, with similar properties holding for the formal analogs of larger expressions such as sentences, including programs! Accordingly, we shall identify a formal language with its associated set of Gödel numbers, and so the notions of recursiveness and recursive enumerability go over unchanged for words, formulas, and theorems.

Hence, let us pinpoint the properties that the formal language L we have in mind, possesses. First, the set of words of L, its dictionary if you will, is recursive. Next, L is to be an axiomatic language, in that there is a recursive set A of words of L, called the axioms of L. Furthermore, all theorems of L are obtainable by operating in a manner to be described on the axioms by means of a finite recursive set of programs of two or more storage locations called the rules of inferences of L.

Briefly, if $R_i(x_1, x_2, \ldots, x_r)$, $r > 1$, is a rule of inference of L and $W_1, W_2, \ldots, W_r$ are words of L, we will say W_1 is a consequence in L of $W_2, \ldots, W_r$ (by rule of inference R_i) whenever $R_i(W_1, W_2, \ldots, W_r)$ is true. Then the formal definition of a proof in L is given as follows: A finite non empty sequence of words $W_1, W_2, \ldots, W_n$ of L is a proof in L of a word W if and only if $W_n = W$ and for each $1 \leq i \leq n$, either

(1) W_i is an axiom, or

(2) There are $j_1, \ldots, j_k$ all less than i, such that W_i is a consequence in L of $W_{j_1}, \ldots, W_{j_k}$ by one of the rules of inference of L.

A Theorem is a word for which there is a proof. Then we have:

Theorem 1 The set of theorems of an axiomatic language L is recursively enumerable.[6]

This means, of course, that the theorems of L can be produced by a program.

Finally, the language L is assumed to be arithmetical, in that any sentence in ordinary arithmetic, using the symbols '+', '$\cdot$', '=', and parentheses, numerals, variables, and quantifiers, has a formal counterpart in the language L; also, whenever one is able to prove both A, and A implies B, then one is able to prove B. Under these conditions, we give the language the standard interpretation over the integers, and regard a statement of L as true, if and only if its interpretation represents a true property of the integers. One further requirement on L is that every function $f(x_1, \ldots, x_n)$ computed by a program be representable in L, where by 'representable' we mean that if $F(X_1, \ldots, X_n)$ is the formal expression in L whose interpretation is $f(x_1, \ldots, x_n)$, then one may formally prove in L either $F(a_1, \ldots, a_n) = y$ or its negation, according as $f(a_1, \ldots, a_n) = y$ or not.

Now it may happen that there is some formula f(x) in L with the property that one can prove in L that $f(x) \neq 0$ although each substitution instance of $x = 0, 1, 2, \ldots$ in $f(x) = 0$ is provable. That is, it is possible to prove each of $f(0) = 0$, $f(1) = 0$, $f(2) = 0, \ldots$, and $f(x) \neq 0$. In this event, L is called ω-inconsistent. Evidently, ω-inconsistent arithmetical languages are unsuitable formalizations for arithmetic.

We are now able to state the famous theorem of Gödel referred to above.

Theorem 2 (Gödel's Incompleteness Theorem) Either the formal axiomatic arithmetic language L is ω-inconsistent, or the set of its true sentences is productive.

Notice that Gödel's incompleteness theorem says that the system L is either unsuitable for arithmetic, or, there exist true sentences that are forever unprovable, and that we are able to write a program which can produce the unprovable true sentences in question!

You may wonder why this presents any difficulty, since if one can effectively produce a

sentence which is true and yet unprovable one is able to add it to the set of axioms, and, a fortiori, it is provable! The negation of this procedure is given by the following observation.

By Theorem 1, the set of theorems or provable sentences of L is recursively enumerable. On the other hand the set of true sentences is productive and hence not recursively enumerable. It follows that the set of true sentences can never coincide with the set of provable sentences!

The connection between this result and the problem of simulating creative intelligence on machines can now be made. Briefly, it is this: The operation of each large scale digital computer can be represented in a formal arithmetic axiomatic language since in supplying answers to problems it proceeds in a step by step manner in response to built-in directives which may be regarded as rules of inference. Since these machines are suitable for arithmetic, the representing language may be assumed to be ω -consistent. Thus there exists an infinitude of true sentences which are forever inacessible to the machine. This is regarded in some quarters as implying the impossibility of ever employing machines as creative robots to assist man in his mathematical chores in more than a trivial capacity since all machines employed in such tasks are assumed to be susceptible to this argument.

Let us remark first of all, that Gödel's theorem does not say that the true sentences produced by the productive program are unprovable by any conceivable means, but rather, that they are unprovable by the specific means already formalized within the language. That is, once all the methods of proof to be available are formalized within the language, or, equivalently, once the deductive system the language represents can be regarded as closed with respect to proof methods, then inaccessible true sentences arise. Of course, no one has ever exhibited a programmable proof procedure which is not formalizable in a language such as we have been considering. It is a fact however, that Godel's theorem does not inhibit the possibility of the existence of such a proof procedure, and so until all the precincts have reported in, there is just no reason to prejudge the question.[8] Which leads us to

Part II

In this part we will consider the following question: Can the concept of 'program' be modified so as to retain as much as possible of the old formulation, and somehow permit the generation of sets which formerly were not recursively enumerable? Naturally, for a particular non-recursively enumerable set, we have in mind the set of true sentences of an ω -consistent language L. The answer to this question was given in a remarkable theorem by deLeeuw, Moore, Shannon and Shapiro, and is 'yes'. We will attempt to sketch this result within the framework of the definitions and results in Part I. But first let us preface some heuristic remarks.

Of course every finite set of integers is recursively enumerable. One has for this case, the program

1. WRITE A_1
2. WRITE A_2

.

.

.

n WRITE A_n

On the otherhand, 'program' means 'finite list of instructions', so given an infinite non-recursively enumerable set $A = a_1, a_2, a_3, \ldots, a_n, \ldots$ simply storing a_i in B_i for $i = 1, 2, 3, \ldots$ and listing the infinite sequence of instructions

1. WRITE B_1
2. WRITE B_2

.

.

.

n. WRITE B_n

.

.

.

will not result in a program.

Next, suppose we construct a program consisting of simply

1. WRITE B ,

and then successively store a_i, $i = 1, 2, 3, \ldots$ in B. It is clear, that in a finite amount of time we can output each member of A. Have we recursively enumerated A? The answer to this question is obtained by recalling the definition of recursive enumerability. We see that the definition requires the program to output the set and only the set. The program above will output anything stored in B. We suspect that we shall have to look somewhere between the two extreme methods for enumerating A we have given above, and that the method we seek will somehow be concerned with some nonfinite concept.

We consider now, programs possessing a countably infinite number of fixed storage locations $A_1, A_2, A_3, \ldots$ such that each A_i can contain either 0 or 1. We will call such a program a concatenating program (CP) if for each $n = 0, 1, 2, 3, \ldots$, and each n-tuple of 0's and 1's $(a_1, a_2, \ldots, a_n)$, stored in $A_1, \ldots, A_n$ the program writes a finite sequence of integers $f(A_1, \ldots, A_n) = (n_{j_1}, n_{j_2}, \ldots, n_{j_r})$ such that $f(A_1, \ldots A_m)$ is an initial segment of $f(A_1, \ldots, A_{m+n})$ if $(a_1, \ldots, a_m)$ is an initial segment of $(a_1, \ldots, a_{m+n})$. When $A = (a_n, \ldots, a_n, \ldots)$ is an infinite sequence, $f(A_1, \ldots, A_n, \ldots)$ is the sequence of symbols consisting of all the initial segments $f(A_1, \ldots, A_n)$.

From an intuitive point of view, a CP operates as follows: It sees the contents of A_1 and writes $f(A_1)$ and halts. Then it sees the contents of A_2 and writes $f(A_2)$ immediately after $f(A_1)$ obtaining thereby $f(A_1, A_2)$ and halts, and so on. Let's give some examples of CP's. Here B, C $\neq A_i$ for any $i = 1, 2, \ldots$

CP_1:
1. B = B + 1
2. IF (A_B) 3 , 4
3. $A_B = A_B + 1$
4. WRITE A_B
5. GO TO 1

CP_2:
1. WRITE A_1

CP_3:
1. B = B + 1
2. WRITE B
3. WRITE A_B
4. GO TO 1

CP_4:
1. C = 2·B (we assume a subprogram for multiplication is available)
2. WRITE C
3. B = B + 1
4. GO TO 1

For the above programs 1-4, we have the following functions 1-4 computed

$$f_1(A_1, A_2, \ldots, A_n) = (\overbrace{1, 1, 1, \ldots, 1}^{n \text{ times}})$$

$$f_2(A_1, A_2, \ldots, A_n) = A_1$$

$$f_3(A_1, \ldots, A_n) = ((1, A_1), (2, A_2), \ldots, (n, A_n))$$

$$f_4(A_1, \ldots, A_n) = (0, 2, 4, \ldots, 2(n-1))$$

Note that the sets of integers written by programs 1-4 are respectively the following:

CP_1: $\{1\}$

CP_2: $\{A_1\}$

CP_3: The set of all ordered pairs (n, A_n)

CP_4: The set of even integers,

and that with the exception of CP_3, all programs are oblivious to A_i for all i.

Given a sequence $A = (a_1, a_2, \ldots, a_n, \ldots)$ and a CP with a_i stored in A_i for $i = 1, 2, \ldots, n, \ldots$, we say the set of integers written by the program is A-enumerable. It can be shown that when $a_i = 1$ for all i, any A-enumerable set is recursively enumerable.

Consider now, any real number $0 < p < 1$. If we expand p as a binary decimal $p = a_1 \cdot 2^{-1} + a_2 \cdot 2^{-2} + \ldots + a_n \cdot 2^{-n} + \ldots$, where $a_i = 0$ or 1 for all i, we may associate with p, the infinite sequence $A_p = a_1, a_2, a_3, \ldots, a_n, \ldots$ where the a_i's are the coefficients in the expansion. The result proven by deLeeuw, Moore, Shannon and Shapiro is

Theorem 3 Let $0 < p < 1$, and let $A_p = a_1, a_2, \ldots, a_n \ldots$ be as above.

Then if there is a program P which computes a function f(B) such that $f(B) = a_B$ for all B, then every A_p-enumerable set is recursively enumerable: otherwise there are A_p-enumerable sets which are not recursively enumerable.

The numbers $p = \sum_{i=1}^{\infty} a_i 2^{-i}$ having the property that there is a program which computes the coefficients a_i are called computable. It is known that almost all (in the sense of the Lebesque theory) real numbers $0 \leq x \leq 1$ are not computable. That is, the probability of picking such a noncomputable real number at random is 1.[9]

The answer to the question raised at the beginning of Part II "can programs be modified so as to obtain recursively nonenumerable sets" has been answered affirmatively. What does this bode for the prospects of developing creatively intelligent machines? We think it offers much encouragement in that it assures us that some sets of integers formally thought to be inaccessible to machines, may now be obtainable via some concatenating program. Of course we do not know at this juncture whether the nonrecursively enumerable set of true sentences of arithmetic can ever be A_p-enumerated by some program, the problems involved are extremely difficult, perhaps impossible (for reasons as yet unknown). It is clear, however, that the concept of concatenating programs enhances the possibility of the emergence of computing machinery from its current chrysalis to a higher species of resourcefulness.

Apropos the possibility of actually building concatenating programs, let us remark that deLeeuw, Moore, Shannon and Shapiro have shown that A_p-enumerability is equivalent to being enumerated by a program which is equipped with a device which stores 0's and 1's in some specified location at discrete intervals of time $t = 0, 1, 2, \ldots$ with probability p, $0 < p < 1$ of storing 1 and probability 1-p of storing zero, each storing being an independent event. The program then runs as usual. We see no reason why this kind of machine can not be built. Of course the problems of producing the correct program to link the device to still persists, but we are still hopeful. A bon chat, bon rat.

References

1. Davis, M., Computability and Unsolvability, McGraw-Hill, 1958.
2. Kleene, S.C., Introduction to Metamathematics, Van Nostrand, 1952.
3. deLeeuw, K., Moore, E.F., Shannon, C.E., and Shapiro, N., "Computability by Probabalistic Machines", Automata Studies, ed. Shannon and McCarthy, Princeton, 1956.
4. Nagel, E., and Newman, J.R., Gödel's Proof, New York University Press, 1958.
5. Myhill, John, Lecture Notes, Univ. of Mich. Eng'g. Summer Conference, 1961.
6. Fraenkel, A.A., and Bar-Hillel, Y., Foundations of Set Theory, North-Holland, Amsterdam, 1958.

Notes

1. Our use of "intelligent" with respect to machines refers to their performing of tasks such as the deciding of mathematical problems, and does not necessarily coincide with any other current usage of the word. We note that many mathematicians believe machines may ultimately contribute solutions to nontrivial problems. For example, Tarski's decision procedure for elementary algebra and geometry is regarded by Fraenkel and Bar-Hillel [6] as a method of great potential in this respect.

Notes (Cont)

2. Judging from discussion at the recent AMS Convention neither does Professor Marvin Minsky.

3. This method was first used, as far as we know, by Professor John Myhill [5] .

4. Instruction #6 is simply: IF A 1, 1.

5. Because of the convention that the empty set (of integers) is recursively enumerable, we require a productive set to nonempty. The precise connection between the productive program P_M of a productive set M and any program P enumerating a subset of M is given by means of the so called Post enumeration of programs which defines an injection (1-1 map) of the set of programs into the integers. Thus if the set of enumerated programs is P_{i_1}, P_{i_2}, P_{i_3}, . . ., the exact relationship is then, whenever P_{i_k} enumerates a subset of a productive set M, $P_M(i_k)$ is an element of M that fails to be enumerated by P_{i_k}. This property provides an additional argument for the assertion that a productive set is never recursively enumerable, for if a productive set M were recursively enumerated by some program P_{i_k} in the Post enumeration, then $P_M(i_k)$ would be an element of M which fails to be enumerated, which is absurd. See Davis [1] , Sect. 5.4 and Theorem 5.1.4.

6. See Davis [1] , Theorem 8.1.1.

7. See Kleene [2] , p 204 ff, and Davis [1] , Theorem 8.3.7.

8. See chap. VIII of Nagel and Newman [4] for discussion and commentary.

9. This is, of course, mathematical probability only; so far as we know there is no practical procedure for producing these numbers by, say, machines.

A SUPERCONDUCTIVE ASSOCIATIVE MEMORY

Paul M. Davies

Abacus, Inc.

Santa Monica, California

Consultant to Space Technology Laboratories Inc.

Redondo Beach, California

Summary

The general properties of an associative memory are explained, and their advantages relative to a random access memory discussed. Then, a superconductive mechanization of such a memory is described which is based upon the cross film cryotron. The memory requires 5 cryotrons per bit and 9 cryotrons for a control module associated with each word. Any combination of bits of the word can be used as the key, and any number of records in the memory can be identified and read out as the result of a single association. The speed of various circuits in the memory is approximated and some applications are suggested.

Introduction

Within the last year, considerable attention has been focused upon a new memory organization which possesses certain properties which make it more powerful in many applications than the random access memory. This memory is called the associative memory because of its ability to retrieve records from storage on the basis of an association between a key transmitted simultaneously to all words of the memory and selected data portions of the stored records.[1]

A more complete definition of the type of memories I shall discuss would prescribe the following properties:

1. A record is written into memory without the specification of an address. The record is transmitted to all memory words and stored in the first empty one.

2. A set of records in the memory can be tagged as the result of an association between a set of bits called a key transmitted to all memory words and the corresponding bits of the stored records. In the memory to be discussed, there are no restrictions on the set of bits of the record chosen as the key. There is also no restriction on the number of records which can be tagged by a single association. In general, the association can be any relationship between two numbers: equality, less than, greater than. However, in this talk, only equality association will be discussed.

3. Records which have been tagged as the result of an association can be read from the memory one at a time.

4. Tagged records can also be cleared from the memory.

Organizational Advantages

Before proceeding to the description of mechanization, I should like to address myself to two preliminary questions. First, "What are the advantages of the associative memory organization?"

Consider the problem of maintaining vehicle registration records. Suppose that each record contains the owner's name and address, the license number, and the engine number. Assuming that a random access memory of large enough capacity were available to store such a file, there would still be considerable problems in organizing the file and retrieving required records from it. In the first place, there is no simple way to relate any item of the record to a memory address. This is because of the redundancy of the items. For example, if the maximum length of an owner's name is 20 letters, then there are 26^{20} possible names. Obviously, it is impractical to have a memory with that many words so that the names can not be used to address the words directly. Similar conclusions are drawn from a consideration of the other items of the record.

If the records are stored in the memory at random, then a sequential search must be made through half the memory words, on the average, to find a desired record. This situation can be improved by maintaining the file in a sorted order. In that case, a record could be located in approximately $\log_2 N$ look-ups where N is the number of records in the file. Thus, 10 look-ups would be required to pick one record out of a thousand. This could be

done as follows: If there are N words in the memory, first read word N/2. A comparison of its key with the one being sought will indicate whether the desired record is in the first or second half of the memory. Next, read record N/4 or 3N/4 according to the previous decision, and so forth.

It is true that such a method would greatly reduce the number of look-ups. However, it has many disadvantages. First, the file must be maintained in a sorted sequence. Therefore, every time a record is to be added to or deleted from the file, a word must be made available or closed up at the proper spot. This requires re-recording about half the records in the memory. Secondly, it is often desirable to look-up records on the basis of different keys at different times. In the case of the vehicle registration records, any of the items listed above might be used as such a key. But, the memory can only be sorted on the basis of one key. Therefore, several files would have to be maintained. Finally, additional time and logic are required to perform the process described.

An associative memory solves all of these problems. Each record is stored only once and no particular ordering of the records is maintained. When a record is to be retrieved, an appropriate key is transmitted to all memory words. This may be the man's name, the license number, etc. Simultaneous comparisons are made in all words to find matching keys and the selected record or records are read out. Any combination of bits of the record may be used as the key and this combination may be changed from one retrieval to the next. For instance, the license number may be used for one search and the engine number for the next. It is also possible to use the owner's last name and the first three digits of the license number. Furthermore, if more than one record responds to an interrogation, all responding records may be read out, one at a time. There is no limit on the number of records that may be read out in this way. Numerous applications of these techniques suggest themselves in such areas as the analysis of military intelligence.

Another important characteristic of the associative memory is the manner in which records are stored in it. Since an address need not be specified to store a record, it is not necessary to keep track, externally, of which words are empty and which are full. This avoids the, sometimes unpleasant, alternatives available with a random access memory of either searching for an empty word or keeping track of the empty words by means of a program, an additional list or a special arrangement of the data in the memory.

The Suitability of Cryotron Mechanization

The second preliminary question I should like to consider is, "Why use cryotrons?" They are not the only possible means for mechanizing an associative memory. However, they possess the following desirable attributes:

1. They are small. Densities of $100/in^2$ leaving room for interconnecting wires can be achieved at present, with much higher densities forecast.

2. They consume little power. This is important since a basic requirement of the associative memory is that logic be performed simultaneously in all words. Present cryotrons consume as little as 2×10^{-6} watts of power when switching at maximum speed. This is sufficiently low that present cryostats can handle the power dissipated by presently planned memories. Furthermore, as the circuits are made smaller, and more numerous, the power per bit goes down as the square of the linear scale so as to exactly compensate for the increase in number of circuits. Therefore, a densely packed plane of microscopic circuits should dissipate no more heat than present configurations.

3. The cryotron provides logical functions as well as memory functions. Furthermore, the output characteristics are compatible with the input requirements so that memory devices and logical devices can communicate with each other with no buffering.

4. Cryotrons do not impose severe power requirements on external drive circuits nor severe gain requirements on output circuits. In addition, these requirements are not particularly increased by increasing the size of the memory.

The Structure in General

Figure 1 is a block diagram of an associative memory. The memory consists of a cryogenic data block which communicates in parallel with a transistorized M register. The data block consists of m words of n bits each. Each word consists of a control module which controls writing, comparing and reading in that word and a data module which stores one record or word of data. The M register which is outside the cryostat is similarly composed of a data module and a control module which communicate with the data block by means of lines entering the cryostat. In the diagram, the data lines vertically connect each bit of the M register with the corresponding bits of all the memory words. Similarly, the control module

of the M register is connected by vertical lines to all control modules of the data block.

To store a record in the memory, the record is placed in the M register. It is then transmitted simultaneously to all memory words along the vertical data lines. At the same time, control signals on the vertical control lines select the first empty word counting from the M register. In response, the control module of the first available word sends a signal along a horizontal line joining all data bits of its word. This causes the transmitted data bits to be recorded in the selected word. Ordinarily, one of the transmitted bits is a "busy" indicator which identifies the word as being occupied so that later data will be stored elsewhere.

To select a record or set of records to be read or erased from the memory, a key which identifies the desired records is placed in the M register. There are no restrictions on the choice of this key. It may consist of any set of bits of the record whether they be contiguous or not. However, ordinarily, one of the bits will be the "busy" indicator so that only busy words will be interrogated.

The key is transmitted simultaneously to all memory words along with certain "enabling" signals that indicate which bits constitute the key. The other bits are ignored in the comparison process which follows. At the same time, control signals are sent to all control modules. In each word of the data block, a comparison is performed between the transmitted key bits and the corresponding stored key bits. Wherever the two compare, this fact is recorded in the associated control module. In this way, all records in the memory which are identified by the transmitted key are tagged for future readout or erasure.

To read a set of records which have been tagged in this way, read control signals are sent to all control modules. The first tagged word responds by producing a read control signal on a horizontal line. This signal causes all bits in the selected data word to be transmitted along vertical lines to the M register. After the first record has been read out, an additional control signal turns off the tag associated with that word. Then, the process is repeated, reading the second tagged word, and so forth. After the last tagged record has been read, the memory responds with an "end of memory" signal which terminates readout.

Ordinarily the reading process is nondestructive. However, a destructive readout is also possible. It amounts to no more than writing zeroes into a selected word after its original contents have been read. If a "busy indicator" is used, a zero in it indicates that the word is un-occupied. On the other hand, the "busy indicator" may be eliminated and the "all zeroes" record used to identify an empty word.

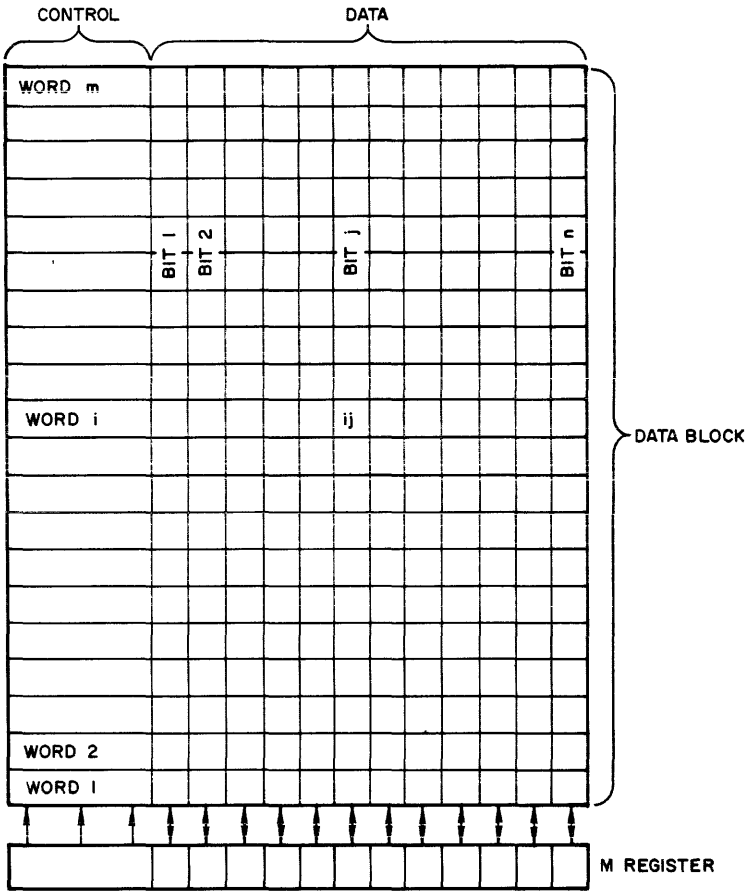
Superconductor Preliminaries

Before proceeding to a detailed discussion of the circuits, it will be convenient to review the basic superconductor phenomena on which the circuits are based. Certain metals such as lead and tin exhibit superconductivity, i.e. as their temperature is reduced to a certain critical value a few degrees above absolute zero, they abruptly lose all electrical resistance. The resistance can be restored by the application of a magnetic field whose required strength is a function of temperature. This is illustrated by the phase state diagram of Figure 2 which shows the relationship between temperature and critical field for a typical superconductor. The region below the curve corresponds to the superconductive state and the region above the curve to the normal state.

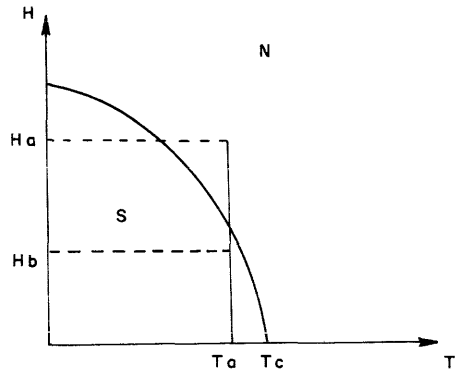
If the metal is maintained at a temperature, such as Ta, lower than the critical temperature, Tc, then it can be switched between the normal and superconductive states by changing the magnetic field between Ha and Hb. This can be done by changing the level of current in the superconductor itself or by changing the current in an adjacent lead as is done in the cryotron. These phenomena provide the basis for a variety of superconductive switching networks.

Figure 3 is an example of such a network, called a gated persistor. The lines represent thin film superconductive leads. The rectangles represent thin film gate elements which can be switched from the superconductive to the normal state by the application of a magnetic field. Crossing each gate and separated from it by a thin insulating film is a control element. This control is very narrow compared to the gate so that current through the control produces higher field intensities than an equal current through the gate. Whenever a current pulse of appropriate amplitude is applied to the control, the gate is switched. The control remains superconductive because it is made of a material with a higher critical field.

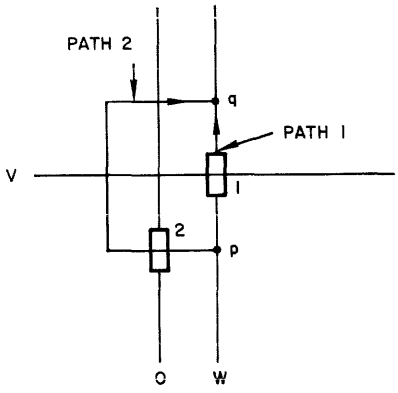
The gated persistor is capable of storing one bit of information in the form of a circulating current as follows: To write into the persistor, current is applied to line W. At point p, the current divides between paths 1 and 2, which comprise the persistor, inversely as the inductances of these paths. Thus,



Block Diagram of an Associative Memory
Figure 1



Phase State Diagram for a Typical Superconductor
Figure 2



A Gated Persistor
Figure 3

$$I_1 / I_2 = L_2 / L_1$$

Next, a current is applied to line V making gate 1 resistive. The IR voltage across gate 1 exponentially transfers I_1 current to path 2. After most of the current has been transferred to path 2, leaving $I_1 + I_2$ in path 2 and 0 current in path 1, the current in V is turned off. The distribution of currents in paths 1 and 2 then remains constant since there are no voltages in the loop. Finally, the W current is turned off. This causes the currents in paths 1 and 2 to be reduced by I_1 and I_2 respectively. The final current values are:

$$\text{Path 1: } 0 - I_1 = -I_1$$

$$\text{Path 2: } I_1 + I_2 - I_2 = + I_1$$

Thus, a persisting current of I_1 (clockwise) has been established in the persistor. This will remain until gate 1 is made resistive again.

The process described results in a stored one. A stored zero is represented by the absence of persisting current. The process for writing a zero is identical to that for writing a one, except that no current is applied to line W. The current on line V then erases any previously stored current in the persistor by making gate 1 resistive.

To read from the persistor, current is applied to line 0. If the persistor contains a one, then the persisting current will cause gate 2 to become resistive and a voltage can be detected on line 0. If the persistor contains zero, gate 2 remains superconductive and no voltage will be detected. Note that the readout is non-destructive.

Memory Structure in Detail

The Memory Bit Cell

Consider next the typical memory bit cell i - j as shown in Figure 4. Vertical lines 0j, Wj and Ej connect the j-th bit of the M register with the j-th bit of every memory word. Horizontal lines Vi, Ci and Ri connect the i-th control module with all bit cells of the i-th memory word. The bit cell consists of a gated persistor made up of Vi, Wj, paths 1 and 2 and gate 1, a comparison circuit comprised of Ci, Ej and gates 2 and 3 and a readout circuit consisting of 0j, Ri and gates 4 and 5.

To write information into this cell, a word select signal is impressed on the Vi line by the i-th control module and a current signal corresponding to a one or zero is impressed upon Wj by the j-th bit of the M register. These two signals, properly sequenced, store the transmitted bit exactly as in the simpler circuit just described.

To compare a key in the M register with corresponding keys in the memory words, the M register key is transmitted along the Wj lines just as in writing: current for a one, no current for a zero. At the same time, enabling currents are transmitted on all Ej lines corresponding to key bit positions. In bit cell i - j, the current in Wj divides between paths 1 and 2 just as in writing. The stored and transmitted currents combine in path 1 of the persistor as summarized in Table 1, where I_t represents the transmitted current in path 1 and I_s the stored current in path 1. It can be shown that regardless of the inductances of the two branches, $L_t = I_s$, provided the same Wj current is used in both writing and comparing. I_r is the resultant current in path 1. Interpreting $I_t = I$ as a transmitted 1 and $I_s = -I$ as a stored one, Table 1 becomes the truth table for exclusive OR: $T\,\bar{S} + \bar{T}\,S = T \longleftrightarrow S$. Thus, whenever the stored and transmitted bits match, there is no net current and gate 2 remains superconductive. When they mismatch, a net current exists and gate 2 becomes resistive. Note that only if the j-th bit is chosen as a key is there current on Ej and is gate 3 resistive. In all columns where Ej = 0, gate 2 is shunted by a superconductive gate 3. Since Ci contains a serial sequence of gate pairs 2 and 3, one in each bit cell, Ci will remain entirely superconductive if and only if a match occurs in every enabled bit cell of the i-th word. Otherwise, Ci will be resistive in at least one column. The superconductive status of Ci in cells with matching keys is the tag used to identify those cells selected by the comparison process. In the description of the control module it will become clear how this information is used in writing, reading and clearing.

After a set of records have been tagged by a comparison operation, they may be read out sequentially. Reading is accomplished by pulsing Ri, making all gates 4 in the i-th word resistive. Gates 5 will be resistive wherever a one is stored. If a current is applied to 0j, then a DC voltage will be observed on 0j just in case a one is stored in the j-th bit cell of the selected word. This voltage can be detected by a read amplifier associated with the M register. After the first selected record has been read, the corresponding Ci line is turned off and the next record is read.

The Control Module

Consider next a mechanization of the control module which will interpret Ci signals

I_f	I_s	I_r
0	0	0
0	-I	-I
I	0	I
I	-I	0

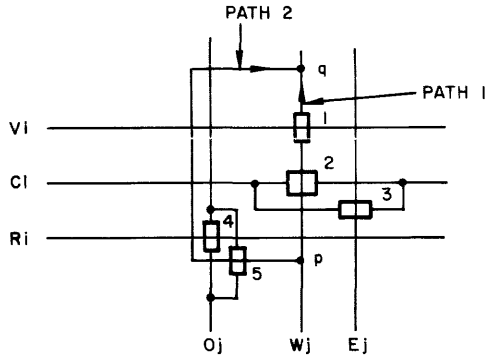
PATH I CURRENT STATES

T	S	R
0	0	0
0	I	I
I	0	I
I	I	0

TRUTH TABLE

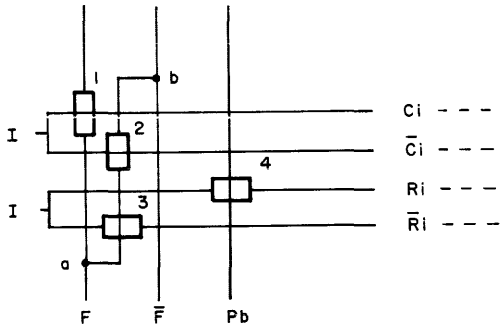
Truth Table for Comparison Circuit

Table 1



Typical Memory Bit Cell i - j

Figure 4



One Stage of the F (Ci) Network

Figure 5

from the data modules and produce Vi and Ri signals appropriately.

The F (Ci) Network

One of the crucial operations to be performed by the control modules is the selection of one out of many signals. For instance, in writing, it is necessary to write into the first empty word. In reading it is necessary to read the first selected word. This operation is performed by the F (Ci) network, one stage of which is illustrated in Figure 5. The vertical lines continue through all control modules and the gating shown is repeated in each. The function of the F (Ci) network is to turn on only that Ri which corresponds to the first Ci which is on. This it does as follows: First Pb is pulsed to reset all Ri's. Then, F is pulsed. The current travels along F until it reaches point a in the first word counting from the M register. If C_1 is off, gate 2 will be resistive and gate 1 superconductive, so the current will pass through gate 1 and continue until it comes to the first word with Ci = 1. At this point, gate 1 will be resistive and gate 2 superconductive so that the current will be diverted through gate 2 to $\bar{F}$. Once there, it never returns to F, since in each stage either gate 1 or gate 2 is resistive. As the current passes from a to b, gate 3 is made resistive and the I current is switched from Ri to $\bar{R}i$. Thus, Ri has been turned on in the first word with Ci = 1.

Comparison

Figure 6 shows a control module connected to one typical bit cell of the data module. The module controls the comparison process as follows: First, Pa is turned on making all gates 5 resistive. At this time Pc and all Ej's are off so that the Ci's are superconductive. Hence, the DC current I is switched into the Ci branches. At the same time, the key is transmitted along the Wj lines so that all gates 11 reach a steady state that reflects the state of agreement between the transmitted and stored bits. Now, the Ej lines are turned on in all key bit positions, and Pa is turned off. Wherever a match occurs in all key bit positions of a word, Ci remains superconductive, and the current remains in Ci, but, wherever a mismatch occurs in a key bit position, both of gates 11 and 12 become resistive and the current is switched out of Ci to $\bar{C}i$. Thus, in the final state, Ci is on in just those words with matching keys.

Writing

To write into the memory, all empty words are first tagged by comparing for non-busy words. This is accomplished by keying with a zero on the "busy" column if a busy bit is used or by keying with all zeroes in case that pattern is used to designate an empty word.

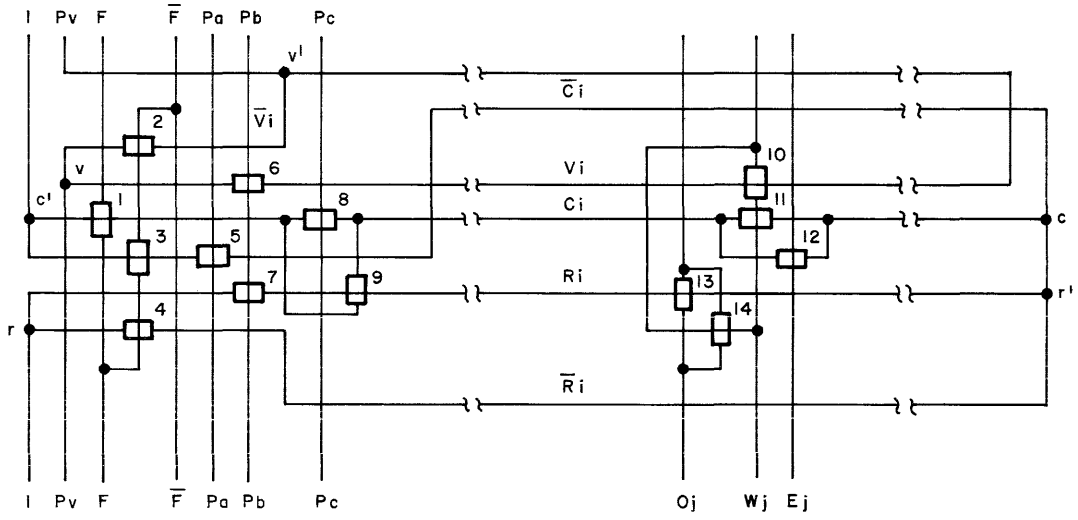
Now, with the Ci's on in just the empty words, Pb, F and Pv are turned on in that sequence. Since gate 6 is resistive, Pv current takes the Vi path from point v to point v' in all words except the first empty one. There, gate 2 is also resistive because the F current is diverted from F to $\bar{F}$ in that word, so the Pv current divides between the Vi and $\bar{V}i$ paths. Next, Pb is turned off and gate 2 transfers all the Pv current to Vi in the first empty word. Current in that Vi causes information transmitted along the Wj lines to be written into the i-th word in the manner described previously. Part of the information that is written is a "busy" indication so that the word will not be written into again. The writing process is terminated by turning off F and Pv and then the Wj's. Pb should be pulsed to prevent circulating currents in the Vi, $\bar{V}i$ loops.

Reading

To read a set of records from the memory, the set is first tagged by comparing all records with an appropriate key. Ordinarily this key will contain a one in the busy bit so that only occupied words can respond. After a set of Ci's have been turned on, Pb is pulsed to turn off all Ri's. Then the F (Ci) network is used to turn on Ri in the first selected word by making gate 4 resistive. Current in Ri causes readout of the data in the i-th word as described previously.

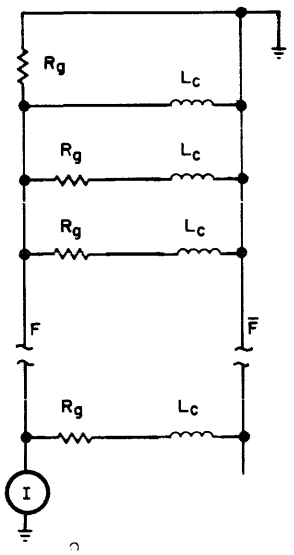
After the first word has been read, Pc is pulsed. This makes all gates 8 resistive and turns off Ci in the first selected word since that is the only word where Ri is on, and hence where gate 9 is resistive. Now, the F (Ci) network can be used again to select the second tagged record for readout. The process is repeated until all selected words have been read and all Ci's have been turned off. Then, the next time F is turned on, the current will remain in F to the top of the network where it will turn on a gate which indicates that there are no more words to be read.

It should be observed that in writing, the F (Ci) network turns on Ri in unison with Vi and that Pb turns off Ri and Vi in unison. Therefore, Pc can be used to sequentially write into a set of selected words just as it is used to sequentially read from a set of selected words. After each record is written, Pc is pulsed. Gates 8 and 9 are then resistive in the selected word and its Ci is turned off. Then, the next record is written into the next word.



Control Module and Typical Bit Cell of Memory

Figure 6



F (Ci) Equivalent Network

Figure 7

The reading process, as described, is non-destructive. However, a destructive read-out can be accomplished by following the normal read-out by a write cycle in which zeroes are written into the selected word, or at least into its "busy" indicator.

Operating Speeds

The speed of cross film cryotron circuits is intimately associated with the gain of the cryotron.[2] Ordinarily, the cryotron controls must be kept significantly narrower than the gates and the interconnecting leads in order to achieve desired gain. In a practical geometry, the leads may be 10 times as wide as the controls. Furthermore, the separation between the control and the ground plane may be 3 times that between the leads and the ground plane. This results in a ratio of inductances per unit length of 30. In many circuits this implies that the interconnecting leads contribute about as much inductance as the controls. In such cases, the L/R time constant of a network is given by:

$$\frac{1}{G_1 G_2}\ \frac{L}{R} = 2\ n\ m\ f(t)$$

where m is the number of parallel cryotrons which switch current from one branch of a loop to another and n is the number of series controls in the loop. G_1 = the gain of cryotrons whose gates are in the loop, G_2 = the gain of cryotrons whose controls are in the loop, and f(t) is a function of gate thickness only. For a tin gate thickness the order of 4000 Å, this reduces approximately to:

$$\frac{1}{G_1 G_2}\ \frac{L}{R} = 4\ n\ m \text{ nanoseconds}$$

The relationship between gain and time constant allows us to trade gain for speed. This is, of course, true only if use can be made of devices outside the cryostat (e.g., semiconductors in the M register) to maintain an overall gain larger than one.

Let us consider the Ri loop of Figure 6 as an example. For a memory with 49 bits per cell, n = 50, since each bit contains a gate 10 with a control in Vi. m = 1 so that the optimum time constant defined above becomes $4 \cdot 50\ G_1 G_2$ nanoseconds. Gates 4 and 7 may have less than unity gain by applying larger F and Pb pulses to compensate or by using in-line cryotrons. On the basis that gates 13 are also fractional gain cryotrons, we can assume that $G_1 = G_2 = .4$. This gives a time constant of 32 nanoseconds for the loop.

A similar time constant could be achieved for the Vi loop. The Ci loop could be made even faster since it contains only 2 controls. However, the F (Ci) network would undoubtedly be slower since it contains a number of resistive paths in parallel (the gates 3). In order to optimize the speed of the F (Ci) network, the F and $\bar{F}$ lines which contain no controls may be made very wide. In this case, their inductance may be ignored, and the F (Ci) network may be represented by the network shown in Figure 7. The L/R time constant for a worst case condition in the F (Ci) network (switching from the first word in memory to the last) is given by:

$$\frac{1}{G_1 G_2}\ \frac{L}{R} = 2(N + 1)\ f(t)$$

where G_1 = the gain of cryotrons 1 and 3, G_2 = the gain of cryotrons 2 and 4, N = number of cells in the memory, and f(t) is a function of gate thickness only. This reduces to:

$$\frac{1}{G_1 G_2}\ \frac{L}{R} = 4(N + 1) \text{ nanoseconds}$$

Here, we can assume that gates 1, 2, 3 and 4 have fractional gain. Taking $G_1 = G_2 = .4$, N = 499, we have:

$$\frac{L}{R} = 300 \text{ nanoseconds}$$

It is clear that as the number of words in the memory increases, the speed of the F (Ci) network becomes an important factor. For very large memories, it would be desirable to reduce this limitation. Lower gain cryotrons could be considered for the F (Ci) network or the memory could be divided into functional blocks.

Auxiliary Circuits and Equipment

In the reading process, a voltage is developed on line Oj. However, this voltage is too small ($\sim 300\mu$ v) to be read directly. It is therefore essential to raise the signal level before attempting to take it out of the cryogenic circuit. One way to accomplish this is to have the Oj line control a high gain cross film cryotron in an output flip-flop. The current levels in the flip-flop would be much larger than in the Oj line and the resistance of the output gate of the flip-flop could also be larger than normal. Hence, the output voltage would be large enough to detect by external circuitry. A flip-flop of this kind would have a relatively large reversal time but this would not add appreciably to the time constant for data retrieval which depends upon the operation of other loops which include many gates.

The external circuitry of the M register includes flip-flops, line drivers and read amplifiers, all of which can be mechanized

using standard transistor techniques.

Applications

Many applications suggest themselves in such areas as:

1. Military intelligence analysis
2. Inventory file maintenance, and
3. Artificial intelligence

Most of these applications require rather large capacities. However, there is at least one class of problems for which a relatively small associative memory should be very useful. These are problems in which a large file of data must be searched for records which match any of a set of keys. In this case, the keys are stored in an associative memory and the records of the main file are stored in some large capacity, serial memory such as tape. As each record is read from the tape, it is compared simultaneously with all keys in the associative memory. The trick employed here is to reverse the usual operating procedure by storing the keys in the associative memory and channeling the large mass of data through the M register.

A specific application for this technique occurs in the air traffic control problem. Here, the main file of flight plans would be read from a tape store and compared simultaneously with a set of new flight plans in the associative memory to determine whether a conflict exists. Since a number of conflicts are searched for at once, it is permissible that the search be serial and hence slow.

Acknowledgements

This paper is based upon work in which the author participated as a consultant at the Physical Research Division of Space Technology Laboratories, Inc. The author thanks Dr. John L. Rogers and Dr. Arthur J. Learn for their help and criticism.

References

1. Seeber, Jr., R. R., "Cryogenic Associative Memory", National Conference of the A. C. M., Milwaukee, 23 August 1960
2. Cohen, M. L. and Miles, J. L., "Characteristics of Film Cryotrons", Solid State Electronics, Volume 1 No. 4, September 1960.
3. Slade, A. E. and McMahon, H. O., "A Cryotron Catalog Memory System", Proceedings of the E. J. C. C., December 1956
4. Kayes, M. K., "Cryotron Storage, Arithmetic and Logical Circuits," Solid State Electronics, Volume 1 No. 4, September 1960
5. Goldberg, J. and Green, M. W., "Large Files for Information Retrieval Based on Simultaneous Interrogation of All Items," Symposium on Large Capacity Memory Techniques for Computing Systems, May 1961 (to be published by ONR)

A CRYOGENIC DATA ADDRESSED MEMORY

V. L. Newhouse
General Electric
Research Laboratory
Schenectady, New York

R. E. Fruin
General Electric Heavy Military
Electronic Dept.
Syracuse, New York

Abstract

A computer storage system which is addressed by content rather than location is described. The design has been verified by constructing and successfully operating a three-word module consisting of 81 crossed-film cryotrons on a 6-inch by 3-inch substrate.

Introduction

A typical, word organized, random access memory system, as used in existing digital computers, is shown in Fig. 1. Each storage bit position of such a memory uses a magnetic core threaded by two or more selection and sense lines. To enter an item of information into such a memory, a location has to be assigned and selected, using the address register and selection matrix shown in the figure. Hence this type of memory may be characterized as "location addressed." All conventional core memories fall into this category.

For certain problems, it is desirable to address the memory by content rather than location. The Data Addressed Memory (DAM), shown in Fig. 2, has this capability. Every filled location can be addressed by any or all of its contents. Because all addressing functions are performed in this manner, neither an address register nor a selection matrix is required.

For applications where the contents of the entire memory must be compared with a specific item of input information, a DAM has a considerable speed advantage over a conventional location addressed system.* Such applications occur in problems involving pattern recognition, inventory control, document retrieval, etc. In all such problems the DAM can compare the contents of all its locations simultaneously with the input information, whereas the conventional memory must search through all or part of its locations in sequence.

The large number of components with non-destructive read-out facilities required for each location of a DAM renders the use of thin film cryotrons advantageous. Such a system has been suggested by P.H. Boucheron.[3] Similar systems have been proposed by Seeber[4], Learn[5], and Davies[6] under the name associative memory. Experimental data addressed cryotron memories which can only deal with a single matching location for any category have been described by Slade and coworkers.[7,8] A small model of a magnetic data addressed memory using a modification of the domain wall viscosity effect for non-destructive read-out is described by Kiseda, et al.[10,11]

*Except in certain search problems dealing with only a few categories where "list" programming can be used.[2]

System Description

The Data Addressed Memory which is the subject of this paper is shown schematically in Fig. 2 and in detail in Fig. 4. It used shielded Crossed Film Cryotrons[1] for switching and storage. The cryotrons are interconnected by lead films which stay superconducting at all times during operation. Storage is performed by setting up circulating currents in various storage cells as described in greater detail below. Since any current distribution established in a completely superconducting network is stable, positive feedback circuits analogous to Eccles-Jordan flip-flops need not be used in superconductive systems.

The functions of a DAM will be described in terms of a specific problem, that of cataloging books in a library. For instance, each memory word location might contain the title, author's name, subject and shelf location of a particular book. Each category of information would be restricted to a specific portion of the word. If the book is to be listed under several subjects, it would be entered in several locations. In each location, the title, author's name and shelf location would be duplicated.

To gain access to information on a specific book, its title is entered into the appropriate positions of the input-output register of Fig. 2. The system is then switched into its comparison mode. In this mode the title position of the input-output register is simultaneously compared with the corresponding position of every storage location. Wherever a "matching" location is found, i.e., wherever the title stored in the location matches that in the input-output register, a current is switched into the upper branch of the so-called comparison line connected with that specific location (see Fig. 4).

In many cases of practical importance the contents of several locations will match, so that several links of the comparison line will have been "set". To deal with this problem sequencing circuits are included in the control logic of Fig. 2 which inhibit access to all matching locations except that one nearest to the input-output register.

Once the presence of a matching location has been established, the system can be switched to its

read-out mode in which the contents of the whole of the matching location are duplicated in the input-output register. Alternatively, the system can be switched to its write mode. In this case, new information is written on top of the information in the selected location. A word is erased by rewriting it as all zeros. The control logic ensures that access to the selected location is not lost even if the information in it is changed.

When the nearest matching location has been dealt with, and before looking for the next nearest one, persistent current is induced in the so-called comparison bypass cell attached to the first selected location. This allows the location to be "bypassed" during subsequent matching cycles. Without changing the contents of the input-output register, the system is now put through another comparison cycle. The sequencing logic allows access to the second nearest matching location. When this location has been dealt with, its bypass cell is activated thus allowing access to the third nearest matching location. When no further matching location exists a signal is produced by the control logic. The comparison and bypass cells of all the previously selected locations can now be reset simultaneously, thus releasing the system for interrogation with a new title.

If new information is to be entered into the memory the system is first interrogated with a title consisting solely of zeros. Access is thus automatically provided to the first empty location if one exists.

System Operation

The Storage and Comparison Cell

The principle of the circulating current storage cell[1] used throughout the memory is illustrated by the digit loop in Fig. 3. The right and left legs of the loop are of equal inductance. A stored "1" is defined as a clockwise circulating current of I/2. To establish this, a current of magnitude I is applied to the digit line in a downward direction. Simultaneously, a write line current is applied driving the write cryotron resistive and forcing the digit current to the right leg. The write current is then removed, followed by the digit current. Since the legs are of equal inductance, this leaves a current of I/2 circulating in the clockwise direction in the loop. A "0" is defined as a counterclockwise circulating current of I/2. It can be established in a similar manner using a digit current of magnitude I in the upwards direction.

Comparison of Stored and Applied Information. If a current corresponding to a "1" (I downwards) is applied to a cell in which a "1" is stored (I/2 cw) or a "0" is applied to a cell in which a "0" is stored, the applied and stored currents will add in the right leg and cancel in the left leg. A "1" applied and a "0" stored or a "0" applied and a "1" stored will add in the left leg and cancel in the right leg. Thus a match between applied and stored information is indicated by a current of magnitude I in the right leg and a mismatch by a current I in the left leg. Figure 3 shows how a match or mismatch is sensed by the match cryotron in the top branch of the comparison line. If a number of these bits are assembled into a word and a comparison is made on all of the bits in the word the top comparison branch will be superconducting only for an exact match.

Comparison on a Portion of the Word. It is usually desirable to match with only a few of the bits that make up the word, ignoring the other bits. This can be achieved by providing a comparison bypass branch around the match cryotron at each bit position. This bypass branch is disabled in any bits used in the comparison by applying "comparison bit selector" currents. Thus the comparison line will be superconducting only if all interrogated bits in a word match the corresponding bits of the applied word.

Readout of Stored Information. Figure 3, also shows a readout capability incorporated in the cell. In this configuration the comparison branches in each bit position of the selected word are also used for readout. This requires, if more than one match exists, the switching of current from the top comparison branches of all non-selected words as described below.

To read out stored information a "0" digit current of magnitude I is applied upwards. This current divides, sending I/2 upward into each leg. These currents combine with the stored circulating current of I/2. If a "0" has been stored the result will be a current I upwards in the right leg and no current in the left leg. If a "1" has been stored the result will be a current I upwards in the left leg and no current in the right leg. Thus a stored "0" will switch the mismatch cryotron resistive, forcing the comparison current through the comparison branch in that bit, leaving the sense cryotron superconducting. A stored "1" will switch the match cryotron resistive, forcing the comparison current through the comparison bypass branch, switching the sense cryotron resistive. Consequently, when a current is applied to the sense line a voltage will be read there for a "1" but not for a "0".

Selection and Control Logic

The function of the selection and control logic is to locate the first matching location that falls into the category being searched, so that read and write operations can be performed on that word only, and finally after that word has been processed, to have it bypassed during subsequent comparisons. These operations are performed by the sequencing ladder, the access inhibitor cells and the comparison bypass cells shown in Fig. 4.

The Sequencing Ladder. The sequencing ladder, shown separately in Fig. 5, selects the word of the class being searched that is stored nearest the "top" of the memory. The ladder is controlled by the comparison branches from each memory location. In the case of a "matching" word, current flows in

the top branch; in a mismatched location it flows in the bottom branch. In all words before the first match, the control rungs will be blocked by a current in the bottom comparison branch. For instance, Fig. 5 shows cryotron B1 driven resistive for a mismatch in word 1. Cryotron A1 remains superconducting, allowing current to flow from the top of the ladder down to word #2. Here, where the first match occurs, the rung will be superconducting (cryotron B2) and the left rail resistive. The current will deflect from the left rail to the right rail through the superconducting rung, identifying this word as the first matching word. Since this current is now in the right rail, it cannot reach matching words further down the ladder.

A match is indicated externally if no voltage appears across the rails of the ladder. If no match exists, all the rungs will be resistive and a voltage will be sensed across the ladder.

Access Inhibit Cells. These cells are shown in Fig. 4. Their function is to inhibit access to all words except the selected one, by storing a circulating current. This current is first stored in all of these loops simultaneously by applying sequential and overlapping current pulses to the access inhibit set line and the access inhibit line. In the selected word, this circulating current will be quenched due to current in the control rung of the ladder. This allows write and comparison bypass set currents to operate on the selected word. It also enables comparisons other than the selected one to be overridden. By applying a read-out strobe pulse, comparison currents in all unselected words will be diverted to the lower comparison branch. Only in the first selected word will the comparison current remain in the top comparison branch. It is this operation which enables the comparison line to be used for read-out.

The Comparison Bypass Cells. These are used to eliminate a matching word from further consideration after operations on it have been completed. This is accomplished by storing a circulating current in the associated comparison bypass cell thus making cryotron L in the top comparison branch resistive. A current is stored in the cell by applying overlapping and sequential pulses to the comparison bypass set lines and the comparison bypass lines. The access inhibit cells ensure that the comparison bypass set current is routed to the comparison bypass cell in the selected word only. Once a circulating current has been stored in this cell, blocking a match for the previously selected word, the system can be returned to the comparison mode to search for the next matching word. After all matching words have been processed the comparison bypass reset current is applied, quenching circulating currents in all comparison bypass cells simultaneously.

Modes of Operation

These are discussed in connection with Fig. 4.

1. Comparison - The contents of all words in the entire memory can be compared simultaneously. This comparison can be made on the basis of any or all bits that are contained in the word. Any bits that are not pertinent can be masked out of the comparison.

On those bits that are to be compared, a digit current corresponding to a "1" or "0" and a comparison bit select current are applied. On those bits that are to be masked from the comparison, only the digit current corresponding to a "0" is applied.

Next, the comparison set line is pulsed. In those words where a match exists, the comparison current (which is always kept on) is forced to the top comparison branch. This switches the sequencing ladder current (which is maintained throughout the comparison cycle) through the rung of the first matching word. It is blocked from the rungs of all non-matching words.

Overlapping and sequential access inhibit and access inhibit set pulses are next applied, storing a current in all access inhibit cells. This current will be quenched in the first matching word by the current in the rung of the sequencing ladder.

Access to all but the first matching word is inhibited by the currents now stored in the access inhibit cells. Therefore Read, Write and Comparison Bypass operations will be effective only on the first matching word.

It is of interest to note that sequential access to all the locations in the memory irrespective of their content can be obtained by comparing with all the input-output register bits masked. This procedure is required when the memory is to be filled with zero's before commencing operations.

2. Read - The Read Operation must always be preceded by a Comparison operation.

The top branch of the Comparison line is used for reading. In all matching words the comparison current has been diverted to this branch of the comparison line. By pulsing the read-out strobe the comparison current will be switched to the lower comparison branch in all except the first matching word. This happens because the first matching word is the only one that does not have current stored in its access inhibit cell. Thus the comparison current can be maintained through cryotron I, only in the first matching word.

All masked bits have "0" digit and sense currents applied. A stored "1" in the selected location will then be indicated by a voltage on the sense line of that bit. The unmasked bits need not be read out since they are already known.

3. Write - The Write Operation must also be preceded by a Comparison Operation. (To write in the first empty register a match is made with all 0's).

Digit currents are applied (corresponding to the word to be written) followed by a write pulse.

Digit currents are required even for masked bits. Therefore it will be necessary to read out the masked bits and re-write them.

4. Comparison Bypass - This operation is used to exclude the first matching word from further consideration after it has been dealt with. A current is stored in the comparison bypass cell of the first matching word by applying overlapping and sequential comparison bypass and comparison bypass set currents. When all matching words have been bypassed, an attempt to match will result in a voltage across the sequencing ladder. The comparison bypass currents are then erased by applying a comparison bypass reset.

Experimental Results

Several Data Addressed Memory plates have been tested in all modes of operation. An arbitrary pattern was stored and various comparisons were made. Read, Write and Comparison Bypass operations were also performed. These tests were performed by manually switching DC currents of 200 ma.

Time constant measurements were made on several loops using single 200 ma. pulses. These measurements were in agreement with the time constants calculated from the geometry. The method used[1] was to store a current in a loop. This current was quenched by switching a cryotron in the loop (identified as the 'switch' cryotron in table I), resistive for an interval less than the calculated time constant. The current remaining in the loop was monitored by measuring the critical gate current of a 'sense' cryotron controlled by this current. This method was used to measure the following:

1. The time constant for transfer of a current from the lower to the upper branch of the comparison loop when pulsing the Comparison Set.

2. The time constant for transfer of a current from the upper to the lower branch of the comparison loop when pulsing the read-out strobe in the presence of a persistent current in the access inhibit cell. This differs from the previous time constant because there is a low inductance resistive path in parallel with cryotron I.

3. The time constant for quenching a circulating current in the Access Inhibit cell.

4. The time required to invalidate a match in words 1 and 2 and to transfer the sequencing ladder current from word 1 to word 3. This was measured by observing the quenching of a circulating current in the Access Inhibit loop of word 3.

Typical results of these measurements are shown in table I. (The switch and sense cryotron identifications refer to Fig. 4.) Nominal CFC dimensions are shown in table II. The widths and lengths of the various interconnections are indicated in Fig. 6 which is a photograph of the actual layout.

Discussion

Future Prospects

Since in the present work the substrate area was larger than any published previously, and since the logic design had never been previously tested, the crossed film cryotrons used were deliberately made large, and an extremely low density layout was used. By reducing the cryotron dimensions (operating at a somewhat lower temperature to maintain gain), and by using a denser layout, it should be possible to accomodate sixteen 21-bit words on the presently used substrates. 1000 such substrates interconnected and contained in a 1 cubic foot cryostat, would constitute a memory of useful capacity. Approximately 160 input-output leads would be required for such a system.

Since substrate-to-substrate interconnections possess a much higher inductance than those printed on the substrates, it is desirable to design the system so that interconnections are not included in loops driven by cryotrons. Several of the ladder circuits such as the comparison bypass set circuit do not satisfy this criterion but can be modified so that they do. A simple ladder circuit is shown schematically in Fig. 7A. The interconnection inductance will limit the operating speed of such a circuit. This problem can be overcome by the modification shown in Fig. 7B, at the cost of an extra bypass line on each plate. By using this bypass line, current can be directed through any or none of the other lines on each plate. Since the interconnection inductance is not included in a cryotron loop, it no longer affects the circuit operating speed. This technique is also useful for minimizing the effect of sneak currents, and has been discussed previously in this connection.[9] It should be mentioned that in this type of circuit the comparison bypass set current will initially divide equally between all lines on one substrate. Thus, there must be enough lines on one plate to limit this current to a safe value or the current rise time must be limited to prevent partial switching in unselected words.

A more complex type of circuit, in which the top rung of the ladder on one plate controls the bottom rung of that on the next plate, is required for the sequencing ladder because this serves as the data transmission channel between plates. The time delay in this circuit can probably be reduced to the vicinity of 50 nanoseconds by using a special low inductance interconnection between the plates together with some extra cryotron amplifiers. This time delay will probably be the factor limiting overall system speed since in the comparison cycle a signal may have to be propagated through every substrate.

Conclusions

The experiment described above proves that fully operating crossed film cryotron circuits can be obtained on 6-inch by 3-inch substrates. In assessing this result it should be taken into account that because of the specific purposes of

this project, no component redundancy was used and all film deposition was performed under the simplest manual control. The good circuit performance under these rigorous conditions is probably due to the use of ample design tolerances and to the operation of the storage cells such that no controls ever have to distinguish between different non-zero current amplitudes.

For example, a worst case analysis of the cell of Fig. 3 leads to a comparison ratio of 6.5/1. This is the ratio of current in the right leg of the storage cell to the current in the left leg of the storage cell in a "matching" bit and includes consideration of a five percent change in inductance of one leg due to layout and fabrication variations, a 10 percent change in digit current due to driver drift, and a transfer of only 90 percent of the digit current from the left leg to the right due to incomplete switching. This allows a large margin for variation in control widths, cryotron parameters or drift in operating point. Another advantage inherent in the properties of the DAM is that if access to a faulty location can be inhibited, system operation will not be affected, (except that the number of available locations is reduced by one).

Estimates presented in the preceding section indicate the feasibility of a 300,000 bit memory with a 50 microsecond comparison cycle time. Such a system should be particularly valuable since it is quite compatible with existing room temperature data processors.

Acknowledgments

The authors would like to extend sincere thanks to P.H. Boucheron for his major contributions to the data addressed memory mentioned in the text; to J.W. Bremer with whom some of the earliest superconducting devices incorporated in the memory were evolved; to D.A. Donath for valuable criticism; and to H.E. Tanis who fabricated the cryotron circuits.

References

1. V.L. Newhouse, J.W. Bremer, and H.H. Edwards, "The Crossed Film Cryotron and Its Application to Digital Computers," Proc. E.J.C.C., pp.255-260; December 1959.

2. N.S. Prywes and H.J. Gray, Jr., "Multi-List Organized Associative Memory," Moore School of Electrical Engineering, University of Pennsylvania, January 1962.

3. P.H. Boucheron, General Electric Co.-HMED, Syracuse, N.Y., private communication.

4. R.R. Seeber, Jr., "Associative Self-Sorting Memory," Proc. E.J.C.C., pp. 179-188; December, 1960.

5. A.J. Learn, "Superconducting Computers," Electronics, vol.34, pp.50-51; November 24, 1961.

6. P.M. Davies, "A Simplified Superconductive Associative Memory," Proc. S.J.C.C.; May, 1962.

7. A.E. Slade and H.O. McMahon, "A Cryotron Catalog Memory System," Proc. E.J.C.C., pp. 115-120; December 1956.

8. A.E. Slade and C.R. Smallman, "Thin Film Cryotron Catalog Memory," Solid State Electronics, vol. 1, pp. 357-362; September, 1960.

9. V.L. Newhouse, "Superconducting Circuits for Computing Machines (Review)," Electro-Technology, vol. 67, pp. 78-89; April, 1961.

10. W.L. McDermid and H.E. Petersen, "A Magnetic Associative Memory System," IBM J. Res. and Dev., vol. 5, pp. 59-62; January, 1961.

11. J.R. Kiseda, H.E. Petersen, W.C. Seelbach and M. Teig, "A Magnetic Associative Memory," IBM J. Res. and Dev., vol. 5, pp.106-121; April, 1961.

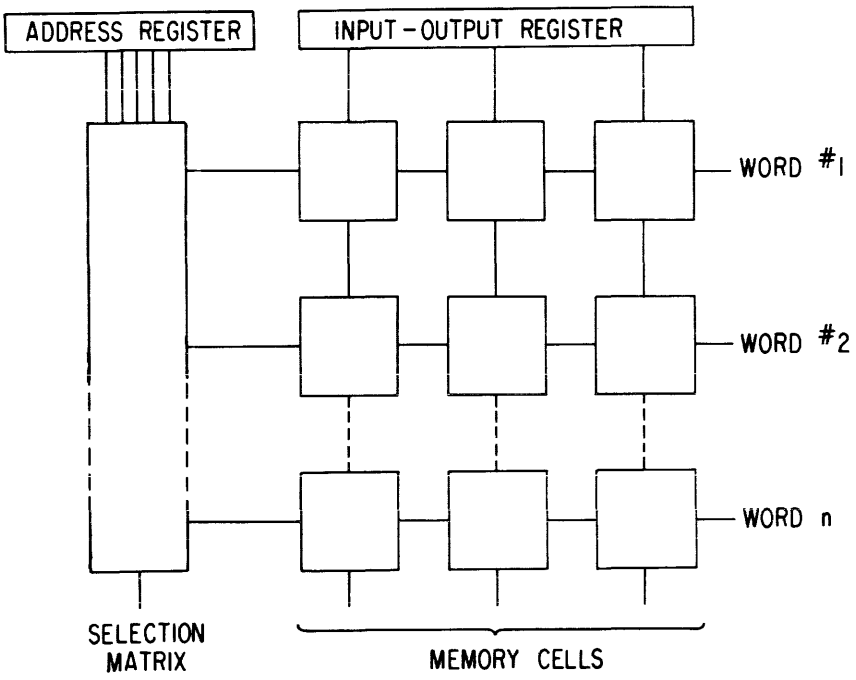


FIGURE 1 LOCATION ADDRESSED MEMORY SYSTEM.

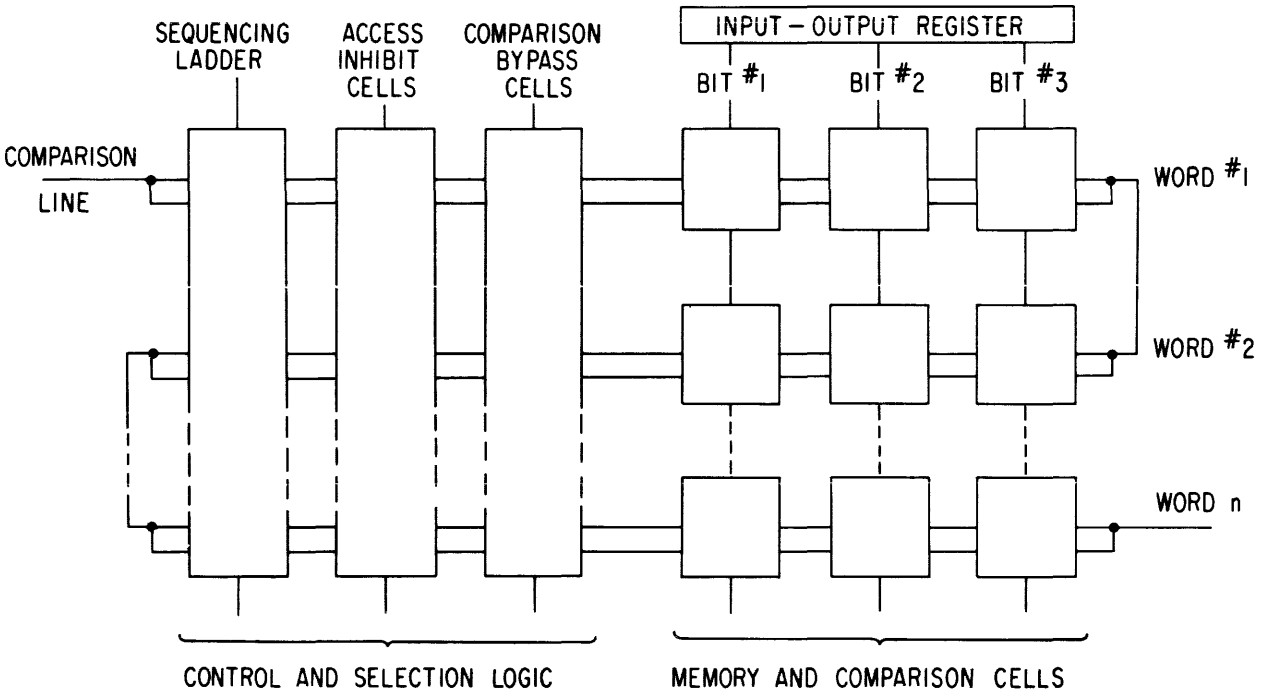


FIGURE 2 DATA ADDRESSED MEMORY SYSTEM.

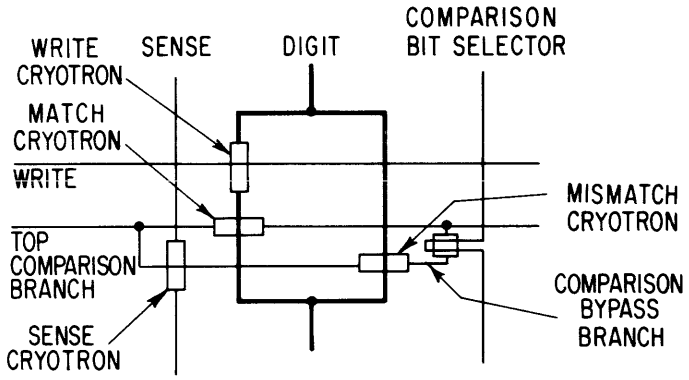


FIGURE 3 MEMORY CELL.

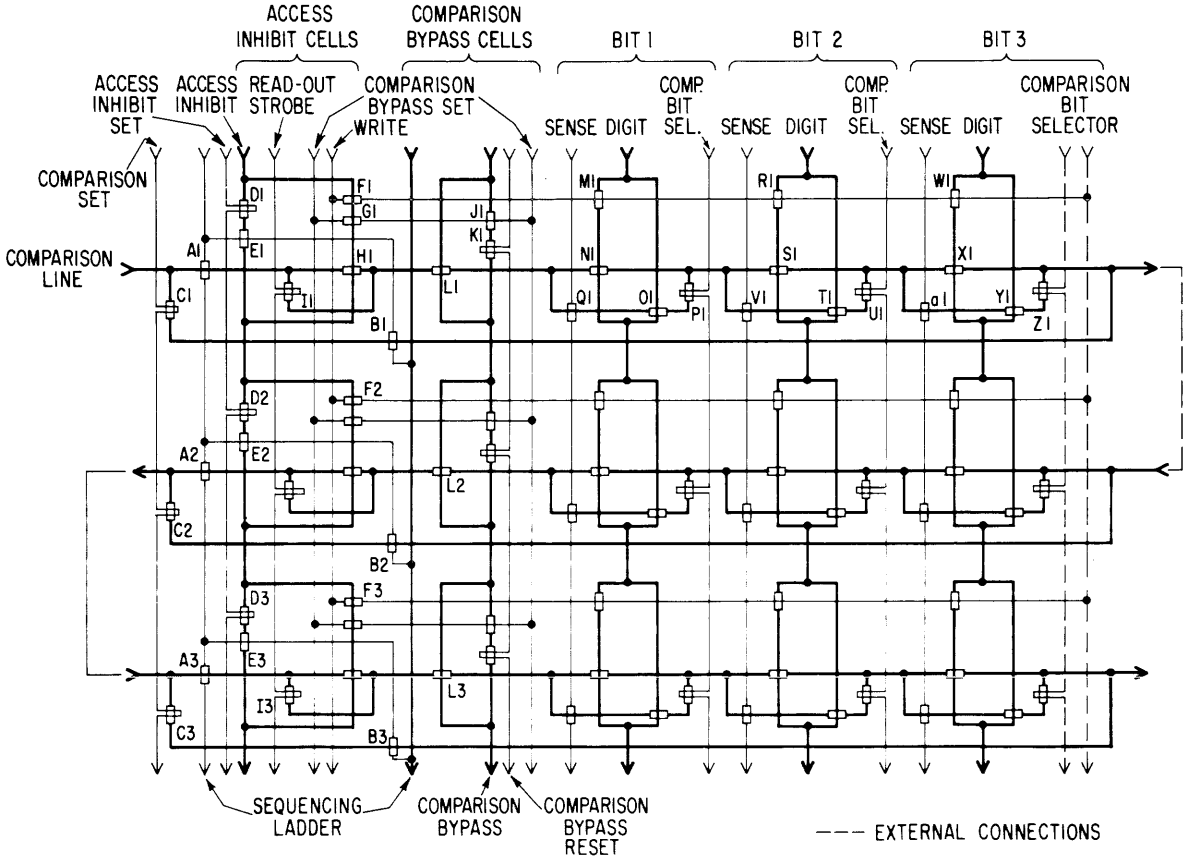


FIGURE 4 MEMORY SCHEMATIC

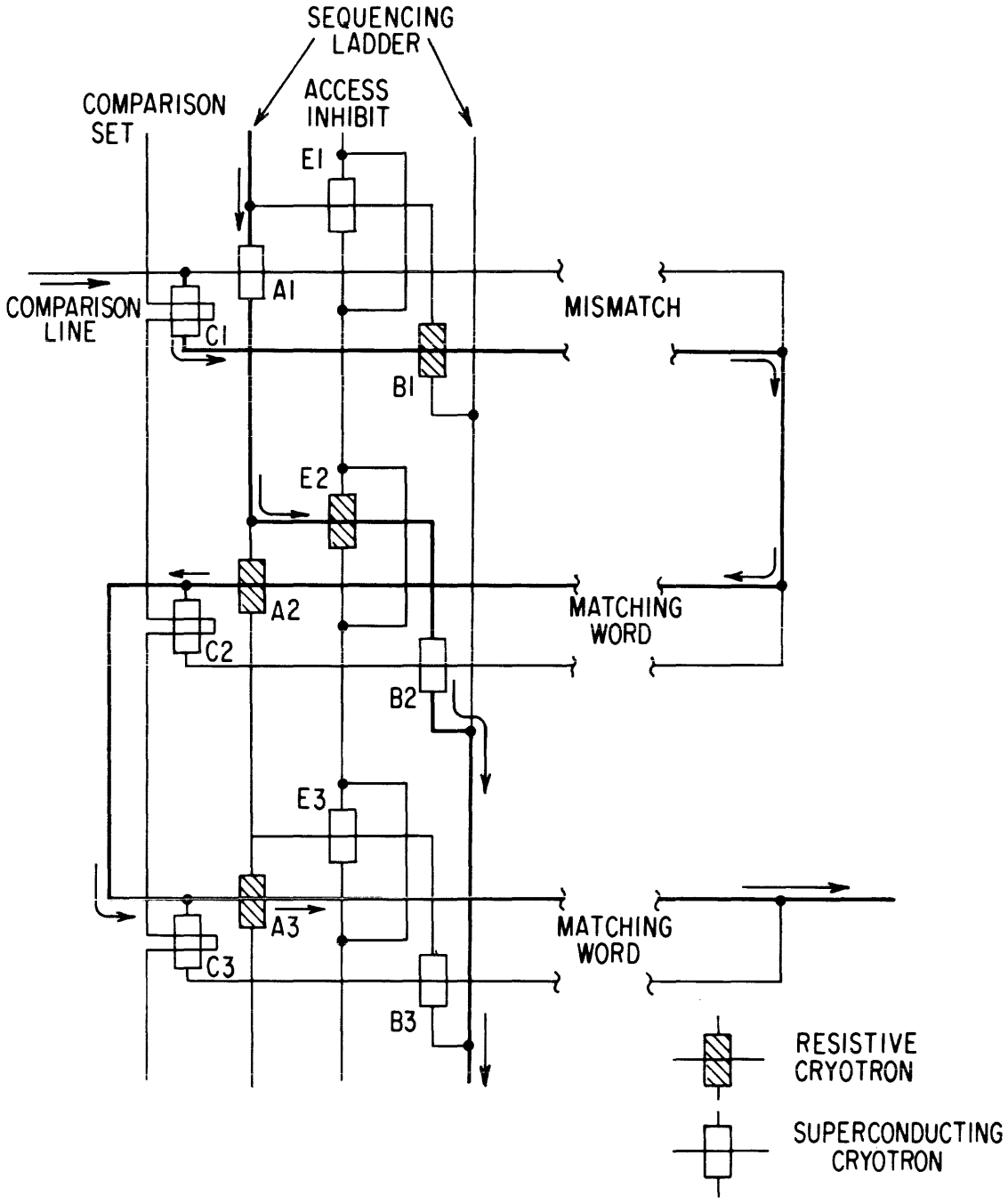


FIGURE 5 SEQUENCING LADDER.

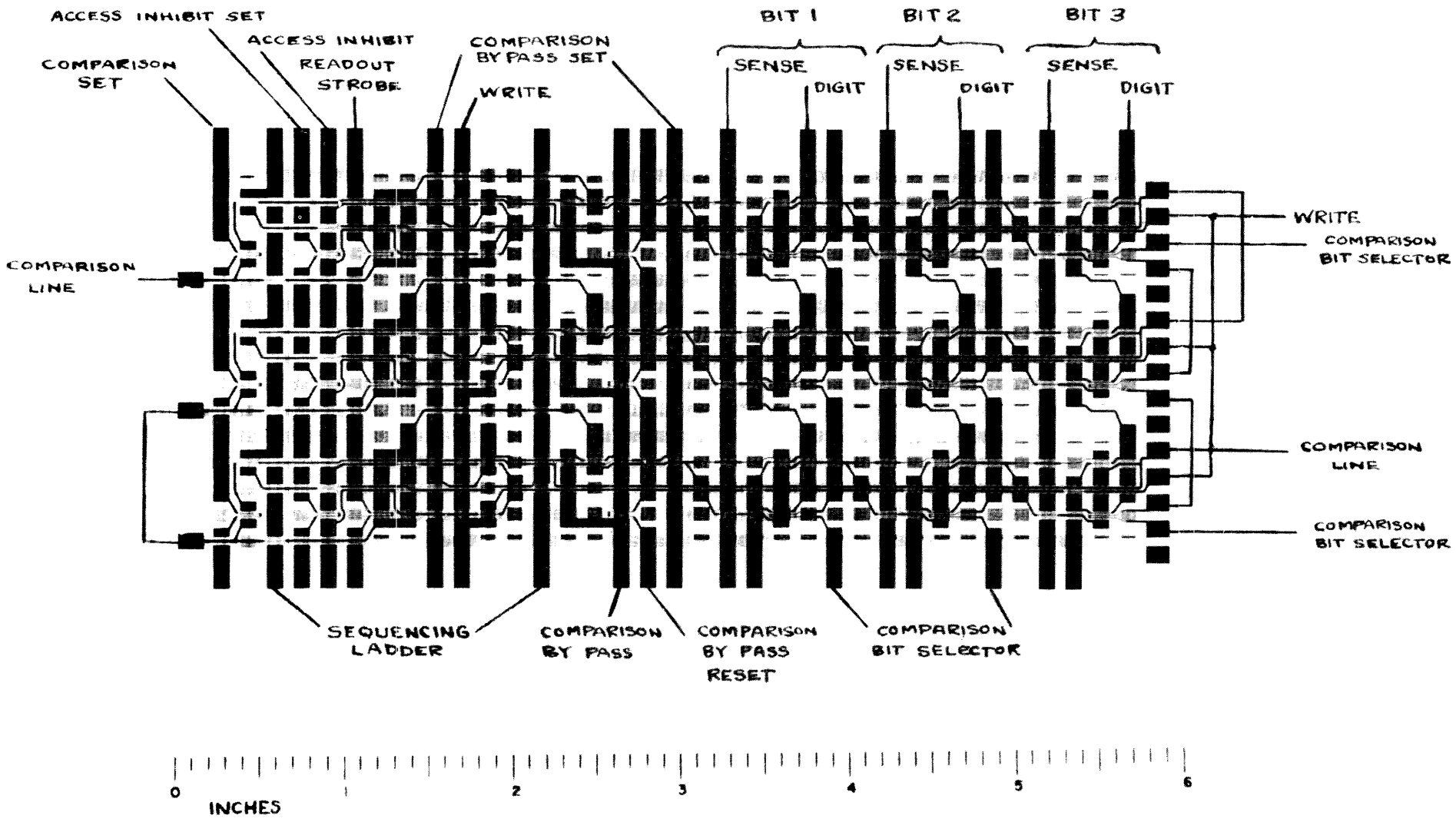


FIGURE 6 DATA ADDRESSED MEMORY LAYOUT.

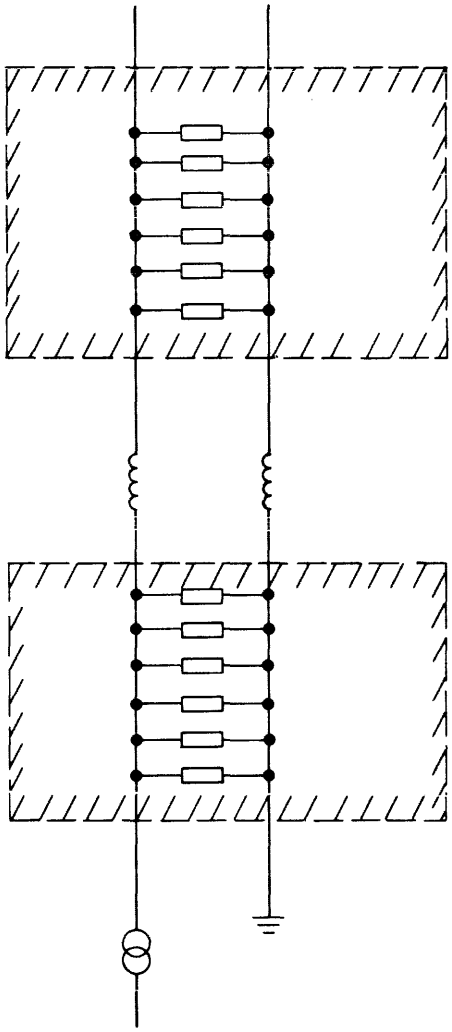


FIGURE 7A SIMPLE LADDER.

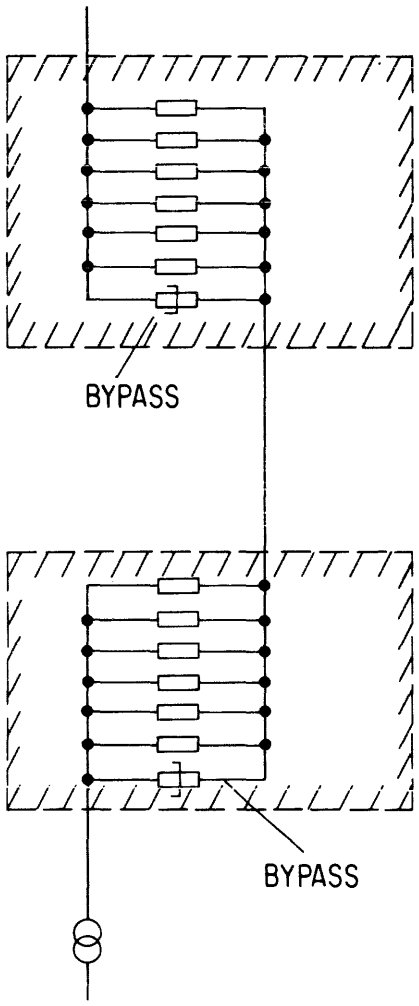


FIGURE 7B MODIFIED LADDER.

SUBSTRATE #	TEMPERATURE IN °K T_c	OPERATING T	ΔT	NOMINAL CFC GATE RESISTANCE	SWITCH CFC	SENSE CFC	TIME CONST. IN MICROSEC. MEAS.	CALC.	REMARKS
12/5/61	3.714	3.635	0.079	280					MANUAL SWITCHING OF COMPARISON FUNCTION
12/6/61	3.710	3.630	0.090	300					ALL MEMORY FUNCTIONS DEMONSTRATED
1/3/62	3.831	3.752	0.079	700	I 3	A3	5.0	5.4	H3 HELD RESISTIVE COMPARISON LOOP
					C 3	A3	2.0	1.8	TIME CONSTANTS
1/12/62	3.807	3.741	0.066	700					ALL MEMORY FUNCTIONS DEMONSTRATED
1/10/62	3.707	3.621	0.086	250	D3	F 3	1.0	1.5	ACCESS INHIBIT TIME CONSTANT
					L1,L2	F 3	5.0	—	REACTION TIME OF COMPARISON LOOP AND SEQUENCING LADDER

TABLE I TEST RESULTS.

LAYER		WIDTH	THICKNESS
		MICRONS	MICRONS
Pb SHIELD		—	1.0
SiO INSULATION		—	0.8
Sn GATES		1600	0.3
SiO INSULATION		—	0.8
Pb	ACTIVE CONTROLS	55	1.0
	INACTIVE CONTROLS	330	1.0

TABLE II CONSTRUCTION DETAILS.

CIRCUITS FOR THE FX-1 COMPUTER

Kenneth H. Konkle
Staff Member, Digital Computer Group, Lincoln Laboratory*
Massachusetts Institute of Technology
Lexington, Massachusetts

Summary

A set of computer logic circuits capable of 50-megapulse operation is described. Included are gated and mixing pulse amplifiers, a static flip-flop, a diode logic unit with current-steering amplifier, a passive delay line, and an active variable delay circuit, all of which are designed to operate with terminated 75-ohm transmission lines. Ten-nanosecond pulses and 20-nanosecond flip-flop transition times are achieved through use of very-high-speed MADT transistors. The circuits have been successfully employed in the FX-1, a small general purpose computer with a high-speed magnetic-film memory.

I. Introduction

The FX-1 Computer is a small, high-speed, general-purpose digital computer designed and built, at the Massachusetts Institute of Technology, Lincoln Laboratory, to be a realistic test vehicle for high-speed logic circuits, packaging techniques, and magnetic-film memory.[1]

It is a parallel, binary, stored-program machine with a 12-bit word length (plus parity) and addressing capability for 1024 words. The present memory, however, contains 256 words plus a 16-word toggle switch storage for read-in and check-out programs. The transistor logic circuits operate at a 50-megapulse rate; that is, the contents of flip-flop registers may be interchanged or modified at 20-nanosecond intervals. The magnetic-film memory has a read-write cycle time of 350 nanoseconds. Approximately one-million single-address instructions can be performed per second. The instruction code is fairly small but provides relative and deferred addressing. The total number of transistors in the machine exclusive of memory is 3500.

Transistors

The logic circuits have been designed around a germanium p-n-p Microalloy Drift transistor, the L-5447, which was developed and brought into production early in 1960 by the Lansdale Division of Philco Corporation with support from Lincoln Laboratory. A later smaller version of this transistor is the T-2217, and commercial modifications are available as the 2N769 and 2N976. The important characteristics of this transistor, as shown in Table I, are its high gain-bandwidth product and its low base resistance, collector capacitance, hole-storage, and saturation voltage. However, the gain-bandwidth product falls off above 10 ma. of collector current as shown in Fig. 1, thus limiting the class of usable circuits to those having relatively high load impedance.

In addition to the L-5447, a few 2N781 germanium epitaxial mesa and 2N708 silicon epitaxial mesa transistors were used where their higher current and dissipation capabilities were needed.

Design Constraints

At the desired speeds the transmission delay associated with just a few inches of wiring becomes comparable to the rise time of the signals. Terminated transmission lines are, therefore, indicated for most of the logic circuit interconnections in order to reduce crosstalk and reflections.

Since transmission line impedances are relatively low, the logic circuit is a transformer-coupled pulse amplifier so that the high load impedance needed for best performance from the transistor can be provided.[2,3]

II. Circuits

Gated Pulse Amplifier

The gated pulse amplifier, shown in Fig. 2, is somewhat unusual in that it has degeneration in the emitter circuit. Its operation can be demonstrated by considering the effect of a negative 1.7-volt step applied to the input (the base of Q1), assuming that the gating point (the collector of Q2) is grounded. The input voltage and the emitter resistor determine a maximum value for the collector current in Q1. This current is more than adequate to bring the transistor into saturation, thereby applying the supply voltage to the divider consisting of the collector RC network, the pulse transformer and its load, and the emitter RC network. However, the current in the magnetizing inductance of the transformer increases with time until eventually the available collector current is exceeded. The transistor then comes out of saturation and terminates the output pulse.

This circuit combines the advantages of saturated operation (the output is determined primarily by passive components) with the fast turn-off time of a nonsaturated circuit. If an output pulse from a similar circuit is applied to this circuit, the pulse is amplified and restandardized to a minus 1.7-volt peak amplitude and a 10-nanosecond width. Shorter than standard pulses are lengthened by hole storage while longer pulses are shortened by the shaping action described above.

The capacitor in the emitter circuit speeds-up the output rise time. The most important

*Operated with support from the U.S. Army, Navy, and Air Force.

function of the RC network in the collector is to reduce the collector supply voltage at the end of the pulse so that the peak collector voltage is reduced during the transformer overshoot. The collector RC network also causes some droop in the output pulse, although this has not been found objectionable. The overshoot voltage of the output pulse is determined by the magnetizing inductance of the transformer and the load resistance, which is always 37 ohms, hence no clamping is required to limit the back voltage on the transistor.

The design of the pulse transformer is crucial to the correct performance of this circuit. It is wound on a very small pair of Ferramic Q2 cup cores. The 14-turn primary and 3-turn bifilar secondary are wound over each other with the distributed capacitance and inductance carefully controlled in order to provide the proper characteristic impedance and resonant frequency. The secondary turns are lumped near the end of the primary which is ac grounded.

A special cable terminator is used at the input to the pulse amplifier to compensate for the capacitive, non-linear input impedance. A 100-ohm resistor can be used with some increase in rise time and delay.

Transistor Q2 of the gated pulse amplifier, Fig. 2, enables or disables the amplifier. If Q2 is saturated the circuit operates as previously described; however, if the gate transistor is cut off, collector current will not flow in Q1 and an output pulse will not be produced. Stray capacitance at the gating point is charged to minus 1.1 volts when Q2 is cut off, in order to prevent a spurious output pulse. Since the pulse amplifier draws current from the gate transistor only when it is amplifying a pulse, Q2 may be used to enable or disable several amplifiers, provided the amplifiers are not pulsed simultaneously. Several gate transistors may be paralleled to obtain more complex gating logic. The gate transistor input, which is either at 0 or minus 1.8 volts, comes from a flip-flop or other level amplifier via a terminated 75-ohm transmission line. Up to 5 gate transistor inputs can be driven from a 75-ohm line.

Each pulse amplifier can drive two 75-ohm lines; if only one line is used, a 75-ohm padding resistor must be added. The output of a line may drive two pulse amplifiers if they are located within several inches of each other. Thus a fan-out of four is obtained with certain geometrical limitations.

The output pulse from the gated pulse amplifier has a minus 1.7-volt peak amplitude, 10-nanosecond width, 2-nanosecond rise time, 3.5-nanosecond fall time, and 1.3-volt overshoot. The circuit exhibits unity voltage gain for input pulses larger than minus 1.3 volts and has essentially no output for input pulses smaller than minus 0.5 volts. This provides good discrimination against spurious pulses and noise, and adequate margins with normal pulses.

The pulse propagation delay in this circuit is 3 nanoseconds of which nearly half occurs in the transformer. A change in the gate transistor input level must precede the input pulse by at least 10 nanoseconds to produce a proper output pulse. This "gate set-up time" is a function of stray capacitance at the transistor collector.

The maximum repetition rate for the pulse amplifier is limited to 25 megapulses per second by transformer recovery time. However, this limitation only rarely prevents the FX-1 from operating at an effective rate of 50-million register transfers per second.

Mixing Pulse Amplifier

A further extension of the basic pulse amplifier is shown in Fig. 3. In this circuit, the mixing pulse amplifier, transistor Q3 is added between the collector of the pulse input transistor and the pulse transformer in order to isolate the input transistor (now called a pulse inverter) from the large voltage swing at the transformer primary. Transistor Q3 operates as a grounded-base amplifier with a small amount of base degeneration. It saturates during the pulse and unsaturates at the end in a manner similar to that of the original pulse amplifier. The input signal to the mixing transistor is approximately 1.5 volts with a pulse current of 15 ma.; hence, the input impedance is 100 ohms. Several parallel pulse inverters may feed this low-impedance point, producing an output pulse whenever an input pulse is applied to any one of them. As is shown in Fig. 3, pulse inverters may be gated by gate transistors as in the case of the gated pulse amplifier. Because of the third transistor, the pulse delay time is increased to 4.5 nanoseconds; however in all other respects its characteristics are equivalent to those of the gated pulse amplifier.

Fig. 4 shows an input and output pulse for a mixing pulse amplifier. The slight ringing is due to the distributed capacitance of the transformer windings.

Flip-flop

The circuit schematic of the flip-flop is shown in Fig. 5. Transistors Q1 and Q4 are mixing pulse amplifiers similar to those previously discussed. The pulse transformers have been modified to provide secondary center taps. Transistors Q6 and Q7 form a conventional saturated RC-coupled flip-flop. Triggering is accomplished by driving both the "off" and "on" transistors to their new states by the input pulse. As a result the output transitions, as seen in Fig. 6, are very symmetrical. In addition, this triggering method improves the rise and fall times of the output since the capacitors in the cross-coupling arms of the flip-flop can be made smaller. These capacitors need only remove the stored charge from the 1N903 silicon mesa trigger diodes. Transistors Q5 and Q8 (2N781's) are output

emitter-followers, each capable of driving a single terminated 75-ohm line. Resistors placed in series with the bases of the emitter-followers prevent oscillation and ringing on the outputs. Standard output levels are 0 and minus 1.8 volts with rise and fall times of 5 to 8 nanoseconds. Note that the clamp voltages for the flip-flop are locally generated by the silicon diode net in the upper right corner. This has been done in all FX-1 packages in order to simplify power distribution. Transistors Q2 and Q3 provide steering for the complement input. These transistors are similar to mixing pulse amplifiers; however, their bases are biased from the outputs of the flip-flop. Therefore, an inverted pulse applied to their common emitters is steered either to the clearing or setting transformer. As seen in Fig. 6, the delays within the flip-flop, due to the transformers, diodes, etc., are greater than 10 nanoseconds, which is adequate to allow pulse dodging. The flip-flop can be used in shift registers or for register interchanges without auxillary delay or storage. Each input terminal can be driven from up to 6 pulse inverters, with an input pulse arriving at one terminal or another at 20-nanosecond intervals. The pulse inverters may be conditioned by gate transistors as described in the pulse amplifier discussion. The transition times of the flip-flop are less than 20 nanoseconds.

Register Driver

The circuits previously described comprise the bulk of those in FX-1. Several others are required for the control and in-out sections of the machine. One of these is the register driver which is used to amplify command pulses for driving a flip-flop register. It consists of a mixing pulse amplifier followed by a grounded-base amplifier using an NPN silicon epitaxial mesa transistor and an output transformer. It will drive eight 75-ohm lines with an output pulse slightly wider and slower than that obtained from a standard pulse amplifier.

Timing Circuits

The control section of a computer requires circuits which can generate the chains of command pulses needed to perform instructions. In most computers, these command pulses are obtained from a clock, however, in FX-1 they are obtained from a large complex multipath delay-line loop. The exact path taken through the control is a function of the states of control flip-flops. Hence, a sequence of pulses in space and time is produced which is determined by the instruction to be performed. This type of control was used because it is more easily realized in a machine employing pulse logic circuits.

Delay Line

Two types of delay are used in generating the pulse sequences. One is a passive delay line, the other an active variable delay circuit. The passive delay line, shown in Fig. 7, is a folded strip-transmission line etched on one side of thin double-clad etched-wiring stock. By using a very thin dielectric, 0.01-inch thick Teflon, and narrow lines, a delay of 4 nanoseconds at 75-ohm impedance can be obtained in an area of approximately 2 square inches. Due to the fineness of the strip line there is a signal loss of 0.035 volts per nanosecond. Therefore, a special pulse amplifier was designed which produces a 60% larger pulse for driving the delay line. This pulse can traverse up to 40 nanoseconds of passive delay line before it is attenuated to the point that it needs amplification. Approximately 300 of these lines are used for spacing and trimming high-speed command pulses.

Variable-Delay Circuit

The active variable-delay circuit, shown in Fig. 8, is used for delays exceeding 45 nanoseconds, as for example, in memory-strobe timing, parity-check timing, and carry-propagation timing in the adder. Transistors Q1 and Q2 form a single-shot multi-vibrator which is triggered by turning on Q2. The lower 1N903 diode at the collector of Q2 applies a positive step to the 82-μμf. timing capacitor. When the capacitor has charged, Q1 starts to conduct again, cutting off Q2. This same diode then opens allowing the pulse transformer to overshoot. This overshoot is amplified and standardized by the output pulse amplifier to produce the delayed pulse. Delays can be produced in the range from 45 to 160 nanoseconds; larger values require adding an external capacitor.

Diode Logic Circuit

When a large number of terms are to be combined into a single gating function, pulse logic is not desirable because the pulse must be passed sequentially through a pulse amplifier for each term involved, resulting in an excessively long logic delay. Level logic reduces the delay by producing the complex gating function for a single pulse amplifier. It is also superior for memory parity computation, where it permits the use of a pyramidal rather than a serial arrangement of gates.

The parity function of 4 bits can be obtained with a 3-level logic circuit, that is, an "and/or/and," if the inputs are available in both their normal and inverted forms. Normal and inverted gating functions are also necessary in delay-line-loop control logic, since a pulse is usually routed down either one or another of several delay paths. These requirements were met by using a 2-level diode-logic net followed by a current-steering amplifier which provides the third level of logic and the complementary output levels. The outputs of the current-steering amplifier are shifted with Zener diodes, clamped to locally generated voltages, and then amplified by emitter-followers so that terminated output lines can be driven. Several additional inputs were added to the diode net so that the unit

would be more generally useful.

Fast silicon mesa diodes (1N903's) are used in the diode nets, L-5447 transistors in the current-steering amplifier, and 2N781 epitaxial mesa transistors as the emitter-followers. The delay time of this level logic circuit is 10 nanoseconds; the input and output levels are at 0 or minus 1.8 volts.

Level Amplifier

The level amplifier is an RC-coupled saturated L-5447 inverter driving a 2N781 emitter-follower, similar to half of the flip-flop of Fig. 5. It has a delay of 6 nanoseconds, and will drive one 75-ohm line with standard FX-1 levels having 5- to 8-nanosecond rise and fall times.

Schmidt Trigger

The Schmidt trigger circuit, Fig. 9, is used to amplify and sharpen slowly changing signals coming from switches, push buttons, or the lower speed logic circuits of the in-out equipment. Its output is a standard FX-1 level which can, if desired, be applied to a special pulse amplifier to produce a standard FX-1 pulse on the negative transition of the input signal.

III. Construction

A three-level method of construction is used in FX-1. Circuit components are wired into small plug-in-units which plug onto trays. These trays contain most of the logic interconnections and are in turn plugged into boxes which contain the intertray wiring. This 3-level arrangement reduces the size of the machine and the wiring delays, without limiting accessibility.

Plug-In-Units

The circuit components are assembled in a cordwood fashion between two double-sided etched-wiring cards with plated-thru holes to form the plug-in unit, Fig. 10. Each unit, which measures 1 3/4" x 2 1/4" x 1", contains approximately 60 components and 10 transistors, which corresponds to one flip-flop or four gated pulse amplifiers. Small commercially available components are used, including 1% molded deposited carbon resistors and miniature ceramic capacitors. The assembly of the cordwood units proved to be quite simple and, contrary to expectations, repair has also been easy. Approximately 360 plug-in-units of 13 different types were used in FX-1.

Trays

The tray, shown in Fig. 11, will mount and interconnect up to 20 plug-in-units. The entire high speed central processor section of the FX-1 was built on 26 of these trays, 12 of which are identical. Figure 11 shows the two interconnection methods used on FX-1 trays. The first method uses 75-ohm Teflon-insulated miniature coaxial cable, RG-187U. The second method uses a three-layer etched-wiring board with the center layer as a ground plane; the width of the etched lines on the faces, and the insulation thickness between them and the ground plane are controlled to produce 75-ohm strip transmission lines. Connections to or through the ground plane are made with plated-thru-holes.[4,5]

Figure 12 shows a cross-section of the three-layer etched-wiring board. It is constructed by first etching two tray cards; one has the vertical wiring, the other has the horizontal wiring and the ground plane. Clearance apertures are etched in the ground plane where connections are not desired to the plated-thru-holes. The two cards plus two very thin cover layers are then laminated together and drilled. When the holes are drilled, the copper edges of the wiring lands and ground plane are exposed. Copper is then chemically deposited and followed by heavy electroplating. This copper on the two cover-layer faces is finally etched away leaving plated-thru-holes with small lands around them.

FX-1 was initially run using trays with coaxial-cable wiring. Eight of these have since been replaced by their etched-board equivalents with excellent results, and the remainder will be replaced in the near future.

Complete Machine

Figure 13 is an overall view of the FX-1 in its temporary frame. The central processor is located in the upper left in three boxes, each containing up to 10 trays. The trays plug into 86-pin double-sided printed circuit connectors, from the rear of the computer. Transmission line wiring of the tray is carried through the connectors as a wire pair with one side grounded. Intertray connections on the front of the boxes uses RG-187U coaxial cable.

Cooling is obtained with low velocity air from the back of the computer.

The 256 word magnetic film memory with its drive circuits and sense amplifiers is located in the bottom left. The racks to the right contain the power supplies, control panel, toggle switch storage, in-out buffer and control, photoelectric tape reader, and XY-display scope.

IV. Results & Conclusions

The circuits described in this paper have been used to build a computer, the FX-1, which can perform instructions in less than one microsecond, and whose accumulator can add two 12-bit numbers in 160 nanoseconds. The 3500 transistor central processor has been in operation since July 1961. Debugging of the machine proceeded very rapidly. All signal waveforms were extremely clean and free from noise. Since initial debugging, the operation of the central processor has been reliable and entirely satisfactory.

Although the operating speeds were referred to as 50 megapulse per second; that is, 20 nanoseconds per register transfer, these speeds have not yet been obtained. At present, a 24-nanosecond register transfer rate is used because of the long gate-set up delay; however, modifications to the gate transistor circuit are now being studied which will allow operation at the stated speed.

The circuits and construction techniques employed are adequate for the construction of larger high-speed computers with improvement needed only in the flexibility of the level logic circuit and the component packing density.

V. Acknowledgement

The author wishes to thank Leopold Neumann and John Laynor who were deeply involved in the circuit design, Elis Guditz who was largely responsible for the construction techniques employed, and to Dr. Donald Eckl whose group supported the transistor development contract.

VI. References

1. Raffel, J. I., T. S. Crowther, A. H. Anderson, and T. O. Herndon, "Magnetic Film Memory Design," Proceedings of the IRE, vol. 49, no. 1, pp. 155 - 164, January 1961.

2. Carroll, W. N. and R. A. Coopper, "Ten Megapulse Transistorized Pulse Circuits for Computer Applications," Semiconductor Products, vol. 1, no. 4, pp. 26 - 30, July/Aug. 1958.

3. Neumann, L., "Transistorized Generator for Pulse Circuit Design," Electronics, vol. 32, no. 4, pp. 47 - 49, April 3, 1959.

4. Guditz, E. A., "Plated Component Connections for Micro-miniature Circuits," Proceedings of 1959 Electronic Components Conference, Philadelphia, Pa., pp. 166 - 171.

5. Guditz, E. A., "Connections Plated with Wiring," Electronics, vol. 32, no. 51, pp. 96 - 97, Dec. 18, 1959.

Maximum Ratings

Collector Voltage, V_{CB}	-12 volts
Collector Voltage, V_{CES}	-12 volts
Collector Voltage, V_{CEO}	-7 volts
Emitter Voltage, V_{EB}	-2 volts
Total Device Dissipation at 25°C	35 mw

Electrical Characteristics

Static Characteristics	min	typ	max
Collector Cutoff Current, I_{CBO} (V_{CB} = -5v)		0.3	5 μa
Emitter Cutoff Current, I_{EBO} (V_{EB} = -2v)		.05	1 ma
DC Current Gain, h_{FE} (I_C = -20 ma, V_{CE} = -0.5v)	25	60	
Saturation Voltage, V_{CE}(SAT) (I_C = -10 ma, I_B = -0.5 ma)		.16	.25 v
Base Input Voltage, V_{BE} (I_C = -10 ma, I_B = -0.5 ma)		340	430 mv
High Frequency Characteristics			
Output Capacity, C_{ob} (V_{CB} = -5v, I_E = 0, f = 4 mc)		1.5	3 pf
Input Capacity, C_{ib} (V_{EB} = -1v, I_C = 0, f = 4 mc)		4.5	8 pf
Gain Bandwidth, f_T (V_{CE} = -0.5v, I_E = 20 ma f = 80 mc)	100	250	mc
Gain Bandwidth, f_T (V_{CE} = -5v, I_E = 7 ma, f = 200 mc)	600	850	mc
Hole Storage Factor, K_s' (I_B = 2 ma)		20	25 nsec

Table I

Important Characteristics of L-5447 & T-2217

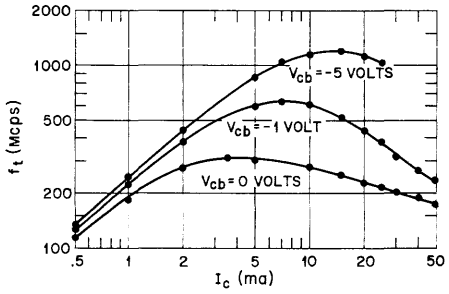


Fig. 1 Gain-Bandwidth vs Collector Current for the Philco L-5447 Transistor

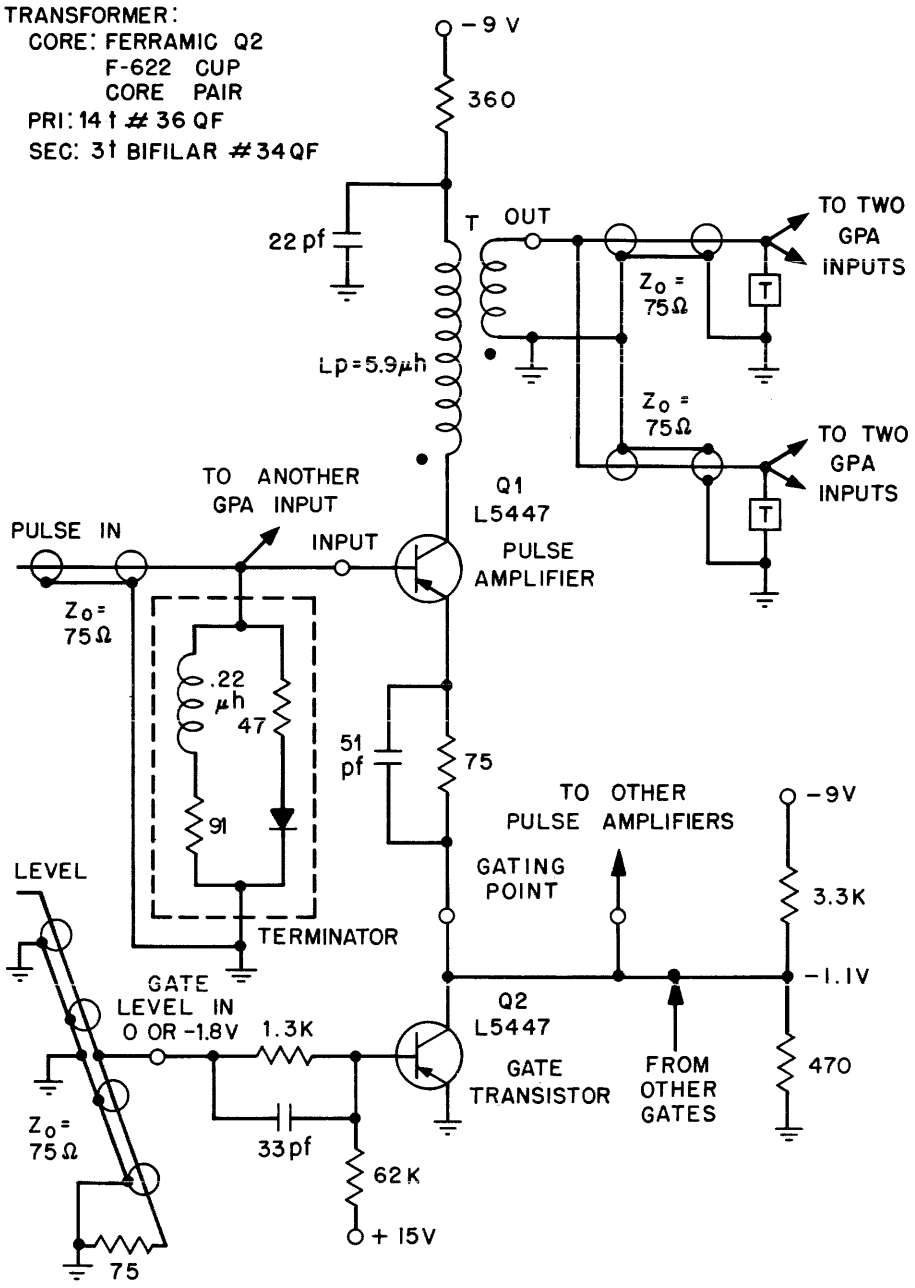


Fig. 2 Basic Gated Pulse Amplifier

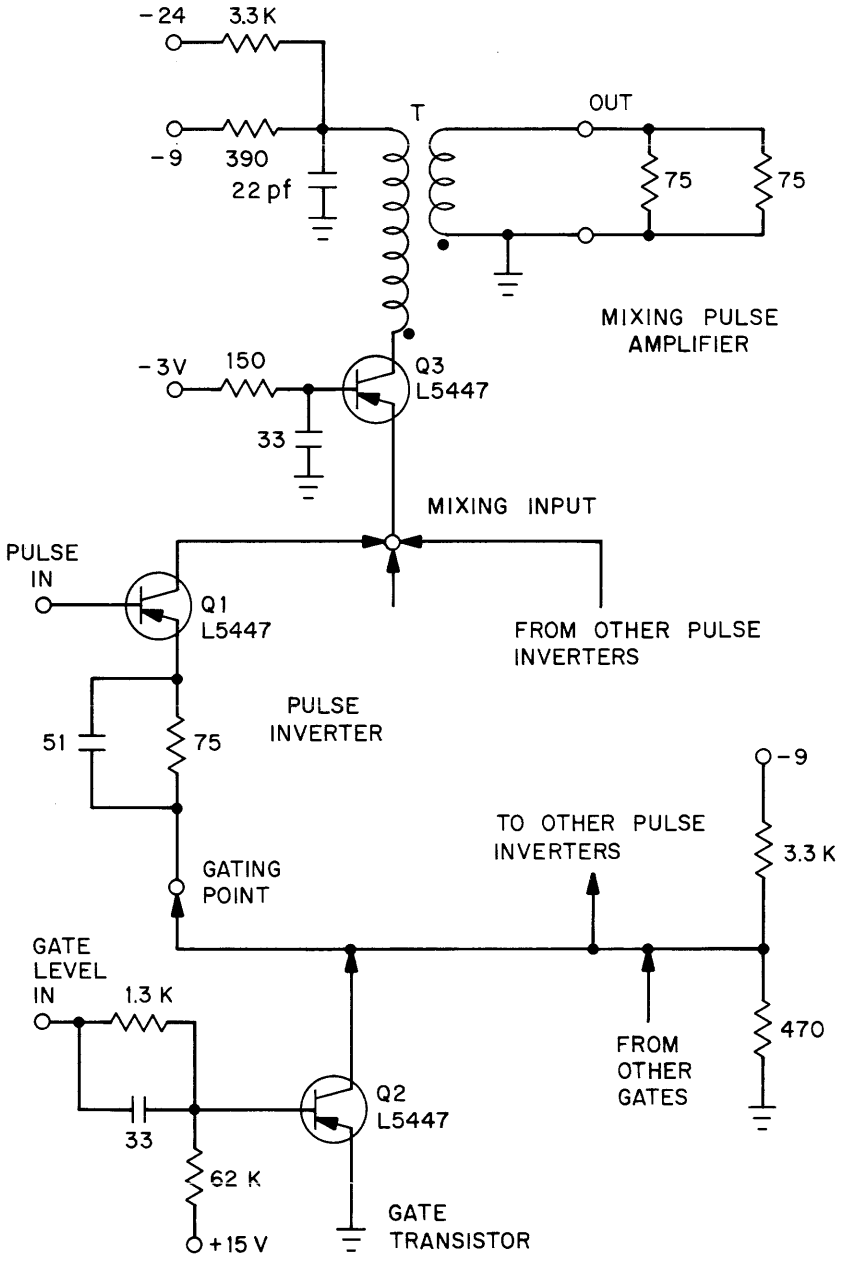


Fig. 3 Mixing Pulse Amplifier

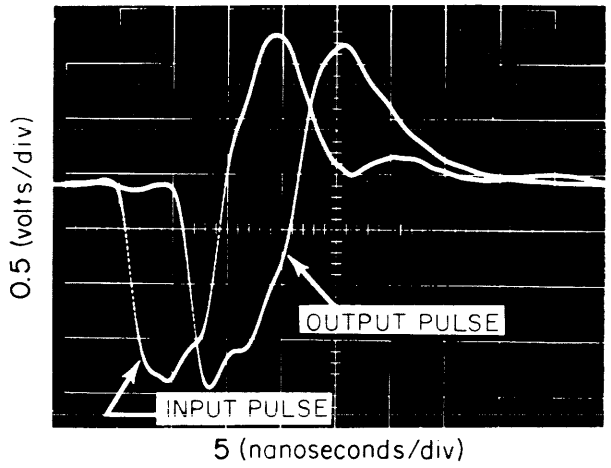


Fig. 4 Input and Output of Mixing Pulse Amplifier

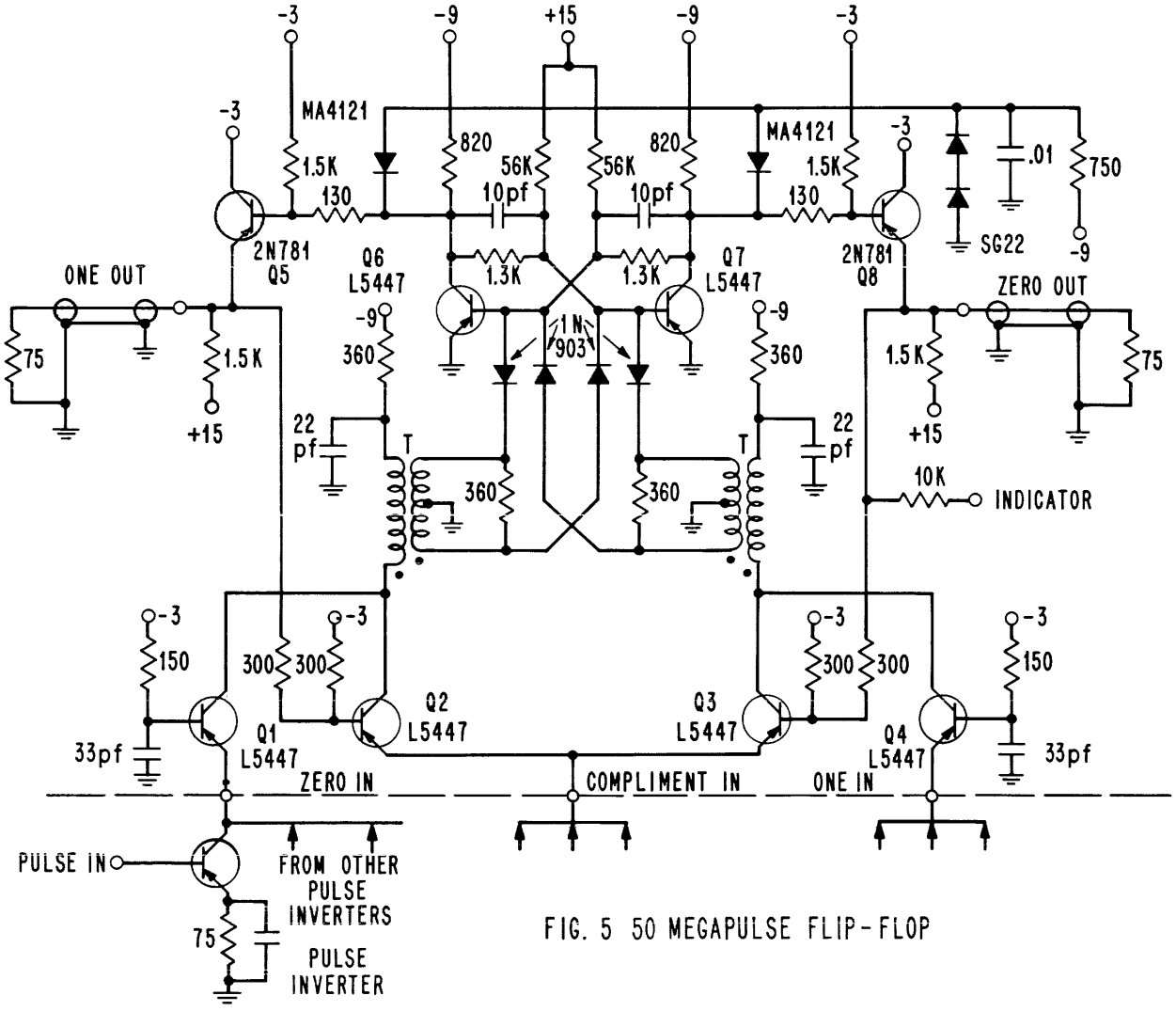


FIG. 5 50 MEGAPULSE FLIP-FLOP

Fig. 5 50 Megapulse Flip-Flop

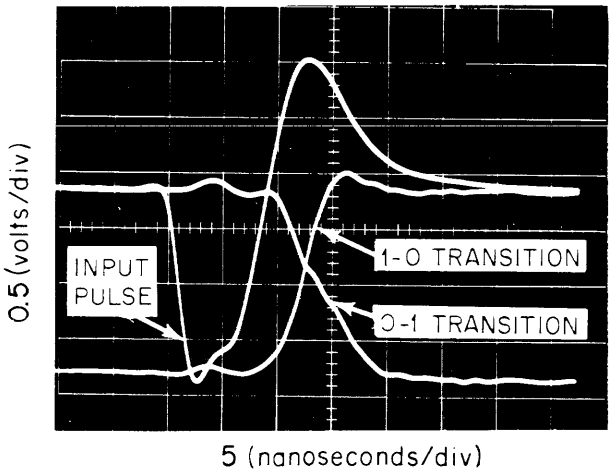


Fig. 6 Input Pulse and Complementary Level Outputs of Flip-Flop

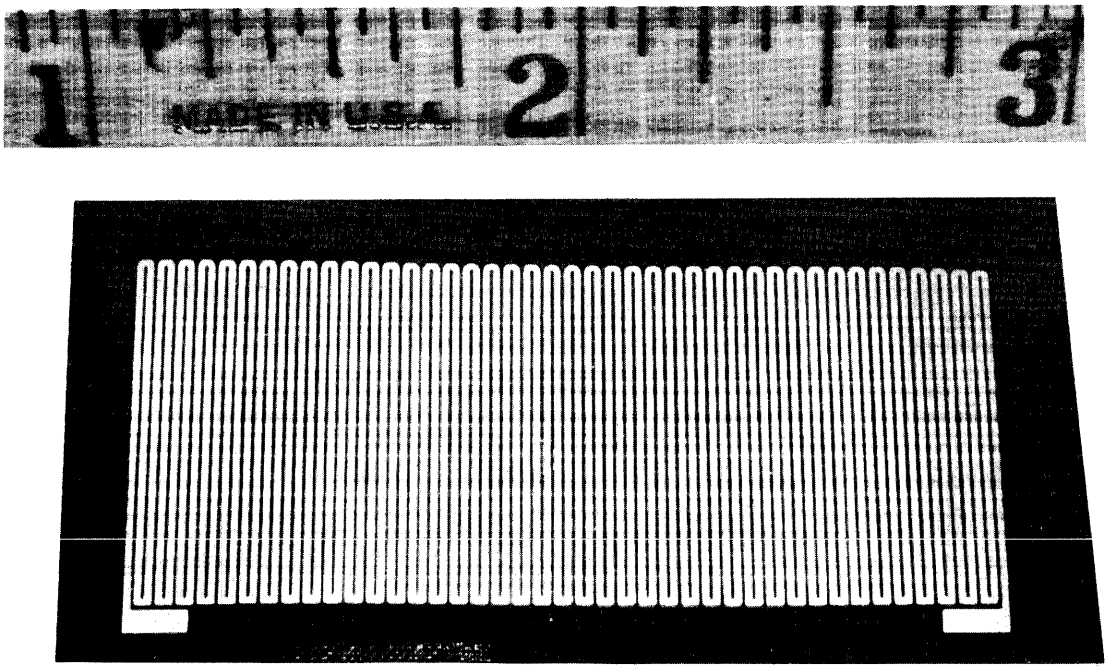


Fig. 7 Delay Line

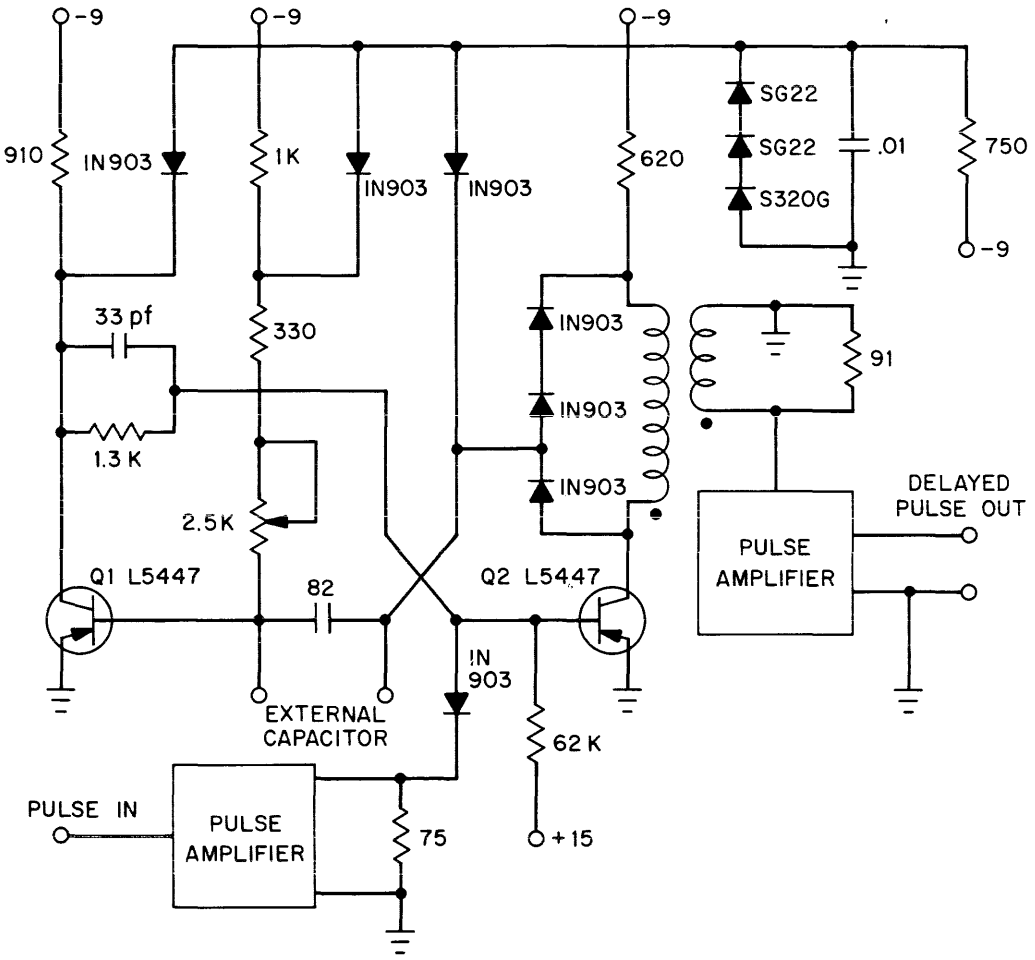


Fig. 8 Variable Delay Unit

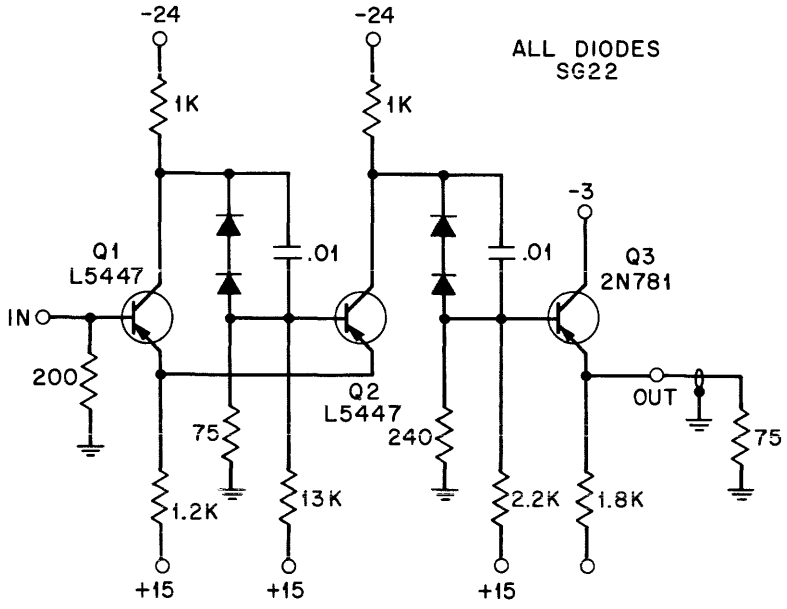


Fig. 9 Schmidt Trigger

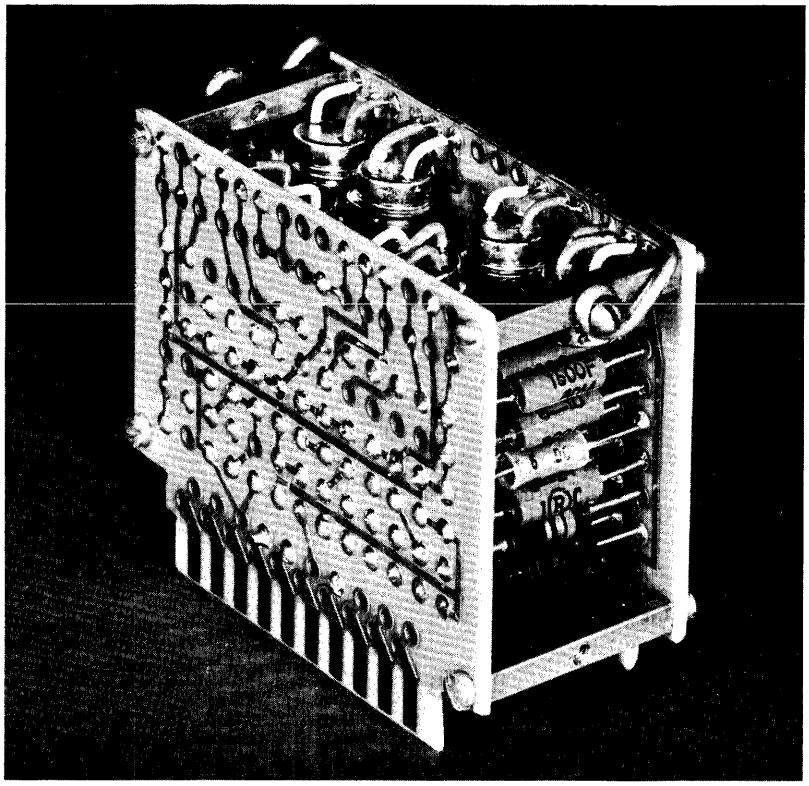


Fig. 10 FX-1 Plug-In-Unit

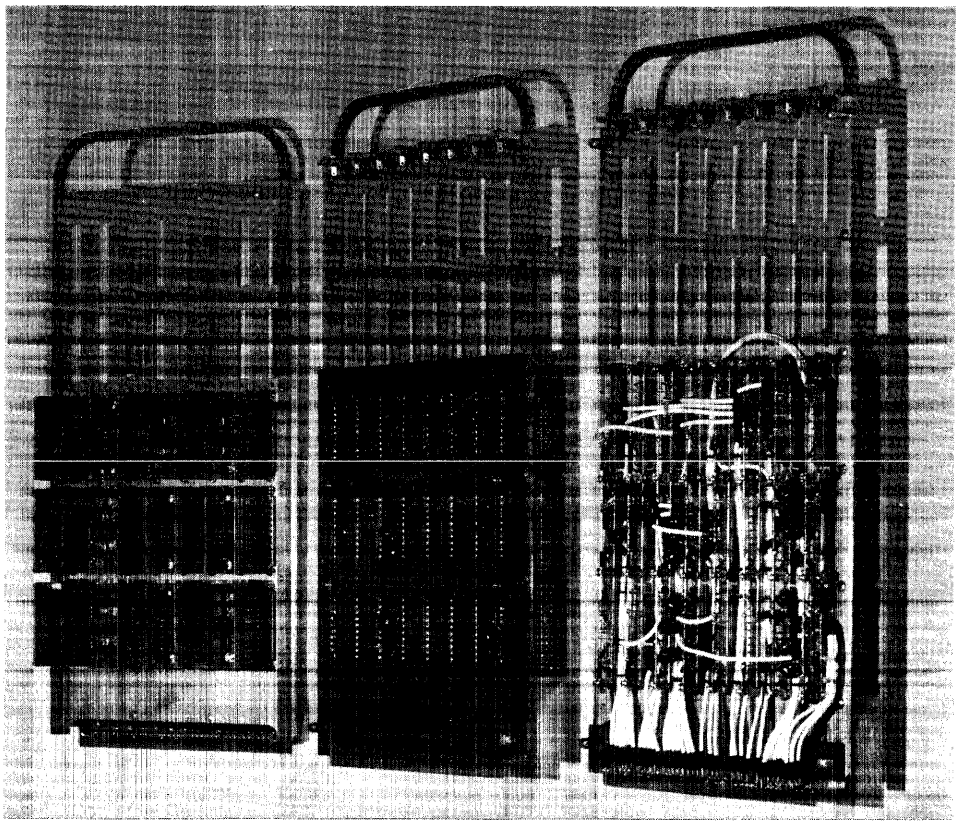


Fig. 11 FX-1 Trays Showing Coaxial & Multilayer Etched Wiring

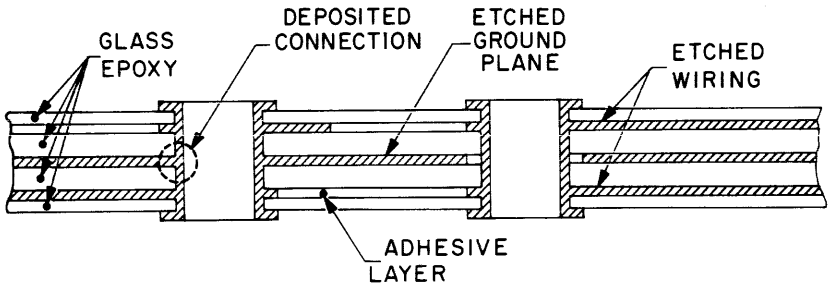


Fig. 12 Three Layer Etched-Wiring Cross-Section

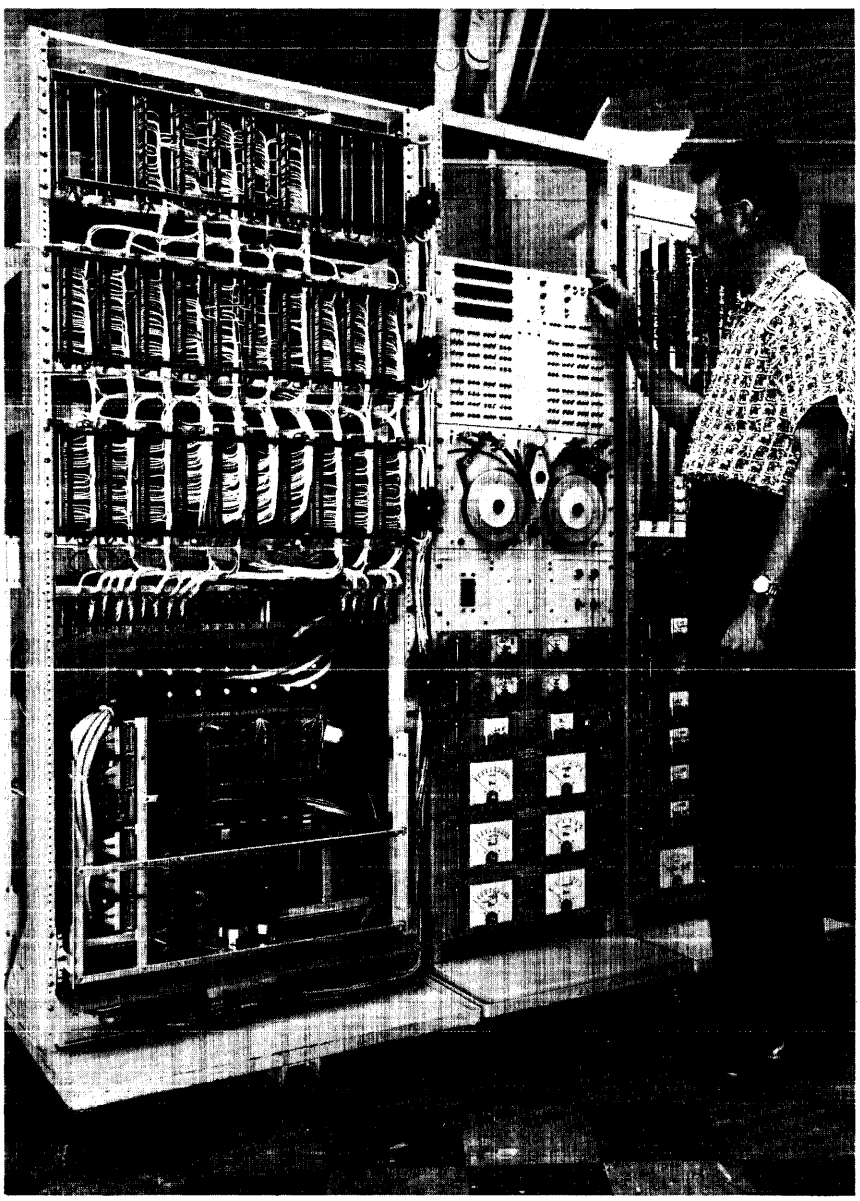


Fig. 13 Overall View of the FX-1 Computer

ON-LINE MAN-COMPUTER COMMUNICATION

J. C. R. Licklider and Welden E. Clark

Bolt Beranek and Newman, Inc.

Cambridge, Massachusetts and Los Angeles, California

Summary

On-line man-computer communication requires much development before men and computers can work together effectively in formulative thinking and intuitive problem solving. This paper examines some of the directions in which advances can be made and describes on-going programs that seek to improve man-machine interaction in teaching and learning, in planning and design, and in visualizing the internal processes of computers. The paper concludes with a brief discussion of basic problems involved in improving man-computer communication.

Introduction

On-line communication between men and computers has been greatly impeded, during the whole of the short active history of digital computing, by the economic factor. Large-scale computers have been so expensive that -- in business, industrial, and university applications -- there has been great pressure to take full advantage of their speed. Since men think slowly, that pressure has tended to preclude extensive on-line interaction between men and large-scale computers. Inexpensive computers, on the other hand, have been severely limited in input-output facilities. Consequently, the main channel of on-line man-computer interaction, in the world of commerce and in the universities, has been the electric typewriter.

In critical military systems such as SAGE, the economic factor has been less restrictive and the need for man-computer interaction greater or more evident. However, the SAGE System, the pioneer among computerized military systems, is "computer-centered" -- less so in operation than in initial design, but still clearly computer-centered -- and that fact has had a strong influence upon man-computer interaction in military contexts. The computers and their programs have tended to dominate and control the patterns of activity. The scope for human initiative has not been great. Men have been assigned tasks that proved difficult to automate more often than tasks at which they are particularly adept.

For the kind of on-line man-computer interaction required in computer-centered military systems, a console featuring a Charactron display tube, a "light gun," and arrays of display lights and push buttons proved effective. At one time, about four years ago, at least 13 different companies were manufacturing such consoles -- different in minor respects but all alike in basic concept. Until recently, therefore, on-line man-computer communication could be summed up in the phrase: electric typewriters and SAGE consoles.

Increasing Need for Man-Computer Symbiosis

During the last year or two, three trends that bear upon on-line man-computer interaction have become clear. First, the cost of computation is decreasing; it is no longer wholly uneconomic for a man to think in real time with a medium-scale computer. Second, time-sharing schemes are beginning to appear in hardware form; the economic obstacle fades as the cost of a computer is divided among several or many users. Third, more and more people are sensing the importance of the kinds of thinking and problem solving that a truly symbiotic man-computer partnership might accomplish:

1. Military officers are eager to regain the initiative and flexibility of command they feel they lost to the computers in computer-centered command and control systems, but they want to retain the storage and processing services of the computers.

2. A few mathematicians are finding computers very helpful in exploratory mathematical thinking. Working closely with powerful computers and graphic displays, they are able to see at once the consequences of experimental variations in basic assumptions and in the formulation of complex expressions.

3. Several persons responsible for the programming of computerized systems are beginning to believe that the only way to develop major programs

rapidly enough to meet hardware time scales is to substitute, for the large crews of programmers, coders, and clerks, small teams of men with sophisticated computer assistance -- small teams programming "at the console." With statement-by-statement compiling and testing and with computer-aided bookkeeping and program integration, a few very talented men may be able to handle in weeks programming tasks that ordinarily require many people and many months.

4. In war gaming and even to some extent in management gaming, there is a growing feeling that the value of exercises will increase greatly if the pace can be speeded. On-line interaction between the gamers and computers is required to speed the pace.

5. In the planning and design of systems of many kinds, digital simulation is recognized as a valuable technique, even though the preparation and execution of a simulation program may take weeks or months. There is now a growing interest in bringing the technique under direct and immediate control of planners and designers -- in achieving the availability and responsiveness of a desk calculator without losing the power and scope of the computer.

6. In the field of education, some of the far-reaching possibilities inherent in a meld of "programmed instruction" and digital computers have become evident to many.

7. The complex equipment used in exploratory research, now in scientific laboratories and perhaps shortly in space, requires overall guidance by scientists but, at the same time, detailed control by computers. Several groups are currently interested in "semi-automatic laboratories."

The foregoing considerations suggest that man-computer communication will be an active field during the next few years and that efforts to facilitate productive interaction between men and computers will receive wide appreciation.

Man-Computer Complementation

The fundamental aim in designing a man-computer symbiosis is to exploit the complementation that exists between human capabilities and present computer capabilities:

a. To select goals and criteria -- human;

b. To formulate questions and hypotheses -- human;

c. To select approaches -- human;

d. To detect relevance -- human;

e. To recognize patterns and objects -- human;

f. To handle unforseen and low-probability exigencies -- human;

g. To store large quantities of information -- human and computer; with high precision -- computer;

h. To retrieve information rapidly -- human and computer; with high precision -- computer;

i. To calculate rapidly and accurately -- computer;

j. To build up progressively a repertoire of procedures without suffering loss due to interference or lack of use -- computer.

It seems to us that the functions listed, a through j, are the essential ingredients of creative, intellectual work. In most such work, they are not strung together in simple temporal sequence, but intimately interrelated, often operating simultaneously with much reciprocal interaction. For that reason, the conventional computer-center mode of operation, patterned after that of the neighborhood dry cleaner ("in by ten, out by five"), is inadequate for creative man-computer thinking; a tight, on-line coupling between human brains and electronic computers is required. We must amalgamate the predominately human capabilities and the predominately computer capabilities to create an integrated system for goal-oriented, on-line-inventive information processing.

In associating capabilities a through f primarily with human beings and capabilities g through j primarily with computers, we are of course describing the present state of affairs, the technology in which we now must work, and not asserting any essential discontinuity between the domains of human and machine information processing. There is always the possibility that human competence in g through j can be significantly increased, and it is almost certain that machine competence in a through f will develop rapidly during the next decades. At present, however, we think that man and computer complement each other, and that

the intellectual power of an effective man-computer symbiosis will far exceed that of either component alone.

Steps Toward Man-Computer Symbiosis

To bring men and computers together in tight synergic interaction, we must make advances in several contributory fields. Among the most important appear to be: time sharing and other possible solutions to the economic problem; memory and processor organization for contingent retrieval of information and programming of procedures; programming and control languages; and on-line input-out equipment, including integrated displays and controls. The groups with which we are associated have been working in those areas. It is disappointing to find that the areas appear to grow more rapidly than we can explore them and to realize how trivial are our accomplishments relative to the requirements. However, we are beginning to have some tangible results, and it may be worthwhile to illustrate briefly the following three:

1. A system for computer-aided teaching and computer-facilitated study.

2. A man-computer system for use in the planning and design phases of architectural and constructional problems.

3. Two programs that display aspects of the internal processes of a computer during execution of programs.

Computer-Aided Teaching and Learning

Exploration of ways in which a computer can facilitate teaching and learning raises several problems in man-computer communication. Effective teacher-student relations involve nearly continuous interchange of information, and anything that interferes with the communication is likely to impair effectiveness.

The importance of rapid, convenient student-teacher communication has demonstrated itself quite clearly in experiments with a simple, automated, language-vocabulary-teaching system. One version of the system, Tutor 1, uses a computer typewriter as the communication link between the student and the machine. Let us examine first the procedure briefly and then the problem of typewriter communication between student and computer.

The typescript of the sample German-English lesson, shown in Fig. 1, illustrates the procedure. In a session with Tutor 1, the student initiates activity by typing "0." The computer then asks him whether or not the student wants detailed instructions. The student replies by typing "s" for "No, start the lesson." The computer selects a German word at random and presents it. The student then types an English word that he thinks is equivalent in meaning and terminates his response by hitting the "centered-dot" key. If the response is acceptable to the computer, the computer types "+" for "correct." (Brevity is crucial.) If it wants another English equivalent, the computer then types the German word again. If it does not want another English equivalent, it types the item score and the cumulative score to date and offers a comment on the student's performance. When the student misses a word, the computer types "-" for "incorrect" and "ta" for "Do you want to try again?" If the student replies "y" for "yes," the computer presents the missed German word again. If the student replies "n" for "no," the computer types an English equivalent and requires the student to copy it. And so forth, as illustrated.

The first thing we found out about Tutor 1 was that students (children and adults) who type well like to use it, whereas students who do not type well may be attracted at first but soon tire of the lesson. During the development of the program, several variations were tried out. Those that speeded the pace of presentation or streamlined the procedure of response were the most successful. A version that eliminated the requirement that the student type the response -- that allowed him to respond vocally or subvocally and then trusted him to score his answer -- was greatly preferred by students who typed only fairly well or poorly; good typists liked "type-the-answer" versions better. With one type-the-answer program, designed to avoid all possible interruptions, students who type well sat for two or three hours at a time, industriously adding new German, French, or Latin words to their vocabularies, occasionally checking their cumulative scores, but never asking for coffee breaks.

Twenty years from now, some form of keyboard operation will doubtless be taught in kindergarten, and forty years from now keyboards may be as universal as pencils, but at present good typists are few. Some other symbolic input channel than the typewriter is greatly needed.

We make some use of the light pen and "light buttons" associated with multiple-choice questions and answers displayed on the oscilloscope. When the alternative courses of action can be laid out in a tree-like branching structure, it is convenient to let the computer ask a multiple-choice question via the oscilloscope display and to arrange the program in such a way that touching the light pen to the button associated with particular response brings forth a subordinate question appropriate to that response. With four familiar alternatives, the operator can make a selection every second or two (i.e., select at a rate of 1 or 2 bits per second), which is adequate for some purposes, though not truly competitive with talking or expert typing (up to 20 and 40, respectively, bits per second in situations in which the pace is not limited by judgmental processes).

In computer-aided teaching, the restriction to a small ensemble of multiple-choice responses sometimes precludes truly convenient, natural communication, and it leads into controversy with those who think that the "constructed response" methods are inherently superior. In our work thus far, it appears that the difference in effectiveness between constructed-response and multiple-choice procedures is small compared with the difference between a convenient, fast response mode and an inconvenient, slow one. Convenience and speed influence markedly the student's enjoyment of his interaction with the computer and the lesson. The most important sub-goal, we believe, is to maximize the amount of enjoyment, satisfaction, and reinforcement the student derives from the interaction. And good student-teacher communication appears to be absolutely essential to that maximization.

Good man-computer communication is important, also, in systems in which the computer serves to facilitate learning without taking the initiative characteristic of most human teachers. We are working on a system, Graph Equation, the aim of which is to facilitate a student's exploration of the relations between the symbolic and graphical forms of mathematical equations.

The program displays, for example, the graph of a parabola (see Fig. 2), and below the graph it displays the equation,

$$y = a(x-b)^2 + c \quad . \qquad (1)$$

Associated with each of the parameters, a, b, and c, is a potentiometer that controls the value of the parameter. The student can vary the parameter values at will and see, directly and immediately, the correspondence between the configuration of those values and the shape of the parabola. We are in the process of substituting, for the potentiometers, "light scales" with pointers operated by the light pen and of displaying numerical coefficients instead of letter parameters on the oscilloscope. Even in the present crude form, however, the system is an effective aid. It presents the linkage between the symbolic and the graphical representation in a dynamic way. It lets the student explore many more configurations than he could explore if he had to plot graphs on paper. And it lets him see "answers" while he is still thinking about "questions" -- something we think may be very important in learning.

We plan, of course, to have the Graph Equation system operate dynamically in the other direction, also. The student will draw a rough parabola. The computer will fit an accurate parabola to the rough one and display the accurate one. At the same time, the computer will calculate and display the coefficients. The completed system, we hope, will provide the student with a flexible, responsive study tool. It will not have much practical value as long as it is restricted to parabolas, of course, but it should be possible, with a faster machine, to handle Fourier transforms, convolution integrals, and the like.*

Often the student must manipulate characters of text with reference to pictorial or graphical information. We have been able to handle some of these functions but still lack an integrated system for communication of interrelated symbolic and pictorial information between the student and the computer.

* The work on computer-aided teaching and learning is supported by the United States Air Force under Contract No. AF33(616)-8152 and is monitored by the Training Psychology Branch, Behavioral Sciences Division, Aerospace Medical Research Laboratory, Aeronautical Systems Division, Air Force Systems Command.

Computer-Aided Planning and Design

In starting to explore the field of computer-aided planning and design of systems, we have focused on hospitals. Hospitals pose very interesting and difficult -- and we believe to a large extent typical -- system problems because the relative importance of the various planning factors varies from one local context to another, because so many kinds of interest and experience are relevant and eager to make themselves felt, and because tangibles and intangibles are so intimately interrelated. One of the main aims in setting up a computer system to facilitate hospital planning is therefore to provide a means through which general guide lines and local constraints can interact. Another is to permit several persons with various backgrounds and interests to look at tentative plans from their own differing points of view and to manipulate and transform the plans during the course of their discussion. A third (since the intangible factors must ultimately be converted into tangible, physical form) is to give the planners a way of sketching out their suggestions and then relating them, quickly and conveniently, to all the other considerations that have been introduced.

Coplanner, a computer-oriented planning system with which we have been working, is essentially:

1. The PDP-1 computer with typewriter, oscilloscope, light gun, and magnetic-tape unit.

2. An ensemble of empirical data describing the commerce (communication of information, transportation of objects, and movement of personnel) that goes on in typical hospitals.

3. An ensemble of programs for accepting, storing, retrieving, processing and displaying information.

In our work thus far with Coplanner, we have experimented with hypothetical hospital situations, using two or three members of our own group and an outside expert or two as the planning team. In preparation for a team planning session, we load into core the programs most likely to be wanted first and make ready the tapes containing the rest of the programs, the ensemble of empirical data, and the material generated in previous planning sessions.

The members of the planning team then sit before the oscilloscope. They start to discuss, for example, a hospital that is expanding its plant and must relocate and enlarge its X-Ray Department. They come to the question: Where should the X-Ray Department be located, relative to the other departments and facilities, in order to minimize the cost of its interdepartmental commerce?

One of the members of the team retrieves, through the computer, a record of previous analyses that provides data on the major components of X-Ray commerce:

a. transport of patients,

b. trips by doctors and internes to supervise x-ray examinations, to study x-ray films, and to consult with personnel of the X-Ray Department,

c. communication not involving movement of personnel, and

d. routine personnel activities such as entering or leaving duty stations and taking meals and breaks.

In response to typewriter commands, the computer then prepares and displays several graphs to summarize the quantitative commerce data. The graphs are mainly distribution graphs and histograms. Since they refer to hospitals of the same type and size, but not to precisely the one being planned, intuitive judgment suggests modifications to take into account various features of the local context. Members of the team make the adjustments in the process of discussion. All they have to do to increase the height of a bar in a histogram is to touch the top of the bar with the light pen and lift it to the desired level. Usually there will be discussion of the change and several successive adjustments of the graph. If the graph is a frequency histogram, raising one bar automatically lowers the others. Efforts have been made to create a favorable context for exercise of the planners' intuitive judgment. Provision is made for labeling, filing, and later processing alternative quantitative summaries if the planners do not agree fully on a single summary.

Figure 3 shows two graphs of the type developed in this phase of the planning discussion.

The planners of course have several different ideas concerning the new layout of the hospital. To make these ideas

concrete, they display prepared floor plans -- a separate plan for each floor of each version -- or sketch them directly on the screen of the oscilloscope, using the light pen as a stylus. Sketching is facilitated by the computer, which posts a background outline plan having the proper dimensions and showing existing structures that cannot readily be altered. In its "straight-line" mode, the computer plots straight lines even if the sketcher's lines are wavy. In its "preferentially-parallel-to-axes" mode, the computer plots lines precisely parallel to the x axis if the sketcher makes them approximately so, etc. On their sketches, which they can readily file away and recall for revision, the planners label the various departments and the stairs, elevators, dumbwaiters, etc. Each label, typed on the typewriter, appears at the top of the oscilloscope screen, and then is adjusted to desired size, trapped by the light pen, moved to its proper location on the plan, and dropped there. Each label serves as a storage and retrieval tag for the sketch to which it is attached. The plan can therefore be made up in small parts and displayed as a whole. Within a few months, the program will be capable of filing and retrieving assemblies by name.

Having tentatively worked out their ideas about X-Ray commerce and sketched several physical arrangements, the planners now turn to the problem of evaluation. First, they select one of the commerce-distribution hypotheses and one of the physical layouts for examination and designate them as input data to a fast-time simulation program that converts the commerce pattern from a set of statistical distributions to a sequence of individual trips and calls. Then they apply a program that finds the best routes for the trips and calls and computes expected durations and costs. In calculating cost, the amounts of time spent by various categories of personnel are weighted appropriately. The weighting function can, of course, be discussed and varied by the planning team. The calculated cost provides an evaluative measure for the selected layout under the selected commerce hypothesis. Actually, several different evaluative formulas are ordinarily used. The corresponding cost figures are saved for later use.

The evaluative procedure is then applied to other combinations of layout and commerce hypothesis. When all the combinations have been treated, the planners recall the cost figures and compare them. On the basis of this comparison, they usually discard all but the best two or three schemes. They modify the best ones, introduce new considerations developed as a result of the study, and make further simulation and evaluation runs.

Figure 5 shows an output-display prepared by the evaluation program.

If the planners are inclined to go into detail in certain areas, Coplanner is prepared to assist them. An elevator-simulation routine, for example, provides a dynamic display of elevator operation under the loads specified by a selected commerce-distribution hypothesis and a determination of best routes. Direct dynamic simulation has important roles to play in work of this kind because it appeals to non-mathematical planners more directly than does queuing-theory analysis performed with the aid only of symbolic assumptions and equations. Sometimes dynamic simulation is a substitute for the abstract theory; sometimes it is an introduction to the abstract theory; sometimes it is a check upon the abstract theory.

In the preceding discussion, one small facet of the hospital planning problem was used to illustrate the approach we are advocating. We have developed a fairly powerful system to facilitate planning in the area discussed and in related areas. In other areas, the system is only starting to develop. The computer parts of the system are not intended, we should emphasize, to calculate optimal plans or designs; they are intended to provide memory, manipulative, computing, and display functions in such a way that they can be integrated with the more intuitive functions supplied by the human parts of the system.*

Visualizing the Operation of Computer Programs

The covertness of the operation of the programs of electronic computers makes it difficult for us to develop of them the same direct, perceptual kind of comprehension that most of us have of familiar mechanisms, the moving parts of which we can see and touch. The great speed with which the programs run

* Coplanner was developed under USPHS Project W-59, Collaborative Research in Hospital Planning. J.J. Souter, A.I.A., and M.B. Brown, M.D., past and present Project Directors, and J.I. Elkind and W.E. Fletcher participated in the formulation of the system.

adds to the difficulty, of course, but we are in the habit of solving the speed problem -- for example, through "slow motion." Unless a window or a plastic model will provide solution, however, we are in the habit of letting the problem of covertness go unsolved. We tend to be satisfied with extremely indirect procedures for interrogation and for drawing inferences. In the case of the human brain, for example, a neurophysiologist may try to construct a model of an internal process on the basis of waveforms recorded from 10 or 100 of the million or billion neurons involved, plus microscopic inspection of several slices of the tissue prepared in such a way as to render visible one or another feature of its architecture. Our approach to computers is comparable: When trouble arises and the results do not turn out as we expect them to, we may try to figure out what is going on by examining with the aid of a typewriter control program the contents of supposedly critical registers, one register at a time, even though we cannot hope to look at more than a hundred of the thousands or tens of thousands of registers involved. Alternatively, we may ask for a printout of the contents of many registers at some particular point in the running of the program, hoping to reconstruct the dynamic pattern of events from the static view provided by the printout.

Considering the problem posed by covertness leads one to think about the procedure, introspection, used as the basic experimental tool in such early psychological laboratories as Wundt's and Titchener's, and still widely employed in the development, if not in the formal testing, of psychological hypotheses. Human introspection is a useful procedure despite its severe shortcomings. How much more useful it would be if those shortcomings were overcome -- if all the processes of the brain were accessible to the reporting mechanism; if the reporting mechanism could describe all the aspects of those processes; if the reports were detailed and accurate; if introspecting did not interfere with the process under examination.

That thought leads immediately to the idea of a computer analogy to, or improvement upon, human introspection. Clearly, computer introspection can be freed of all the shortcomings mentioned, except the last, and the last one can be turned to advantage. Displaying its own internal processes will of course interfere with the computer's execution of its substantive programs, but only by appropriating memory space and time. Often, there is memory space to spare, and programs normally run too fast for the operator to follow them perceptually. The conclusion, therefore, is that it might be interesting to experiment with programs that display various aspects of the internal operation of the running computer.

Two such programs, written for the PDP-1 computer, are Program Graph and Memory Course. Program Graph was written with the hope that it would facilitate the introduction to computer programming and provide displays through which certain individual or "personality" characteristics of programming style may be seen. Memory Course was intended mainly for use in "debugging" computer programs. Both programs make use of a trace routine that executes the instructions of the object program in normal, running sequence and, after each execution, (a) records in core registers the contents of the accumulator, input-output register, and program counter, (b) does some incidental bookkeeping, and (c) turns control over to the display routines. The display routines develop graphs of types to be illustrated.

The graphs displayed by Program Graph are illustrated in Fig. 6. In Fig. 6A, as each instruction of the object program is executed, its location is plotted as ordinate, and the cumulative number of executions is plotted as abscissa. (Roughly speaking, therefore, the graph represents active memory location versus time.) Both the ordinate and the abscissa scales run from 0 to 1777 (octal). The interpretation of the graph is quite direct: straight-line parts of the graph represent straight-line parts of the program; jumps represent jumps or subroutine calls; serrations represent loops. The subroutine structure is revealed clearly. If the operator knows the general course the program should follow, he can detect and locate gross faults readily.

Figures 6B-6D show, for the same object program, the contents of the accumulator as a function of time. The abscissa scale again runs from 0 to 1777. In Fig. 6B, the ordinate scale covers the range from -2^{17} to 2^{17}; in Fig. 6C, it runs from -2^{13} to 2^{13}; and in Fig. 6D, it runs from -2^{7} to 2^{7}. Evidently, the accumulator is heavily engaged in computations involving small numbers.

Figures 6E-6G show, for the input-output register, what Figs. 6B-6D showed for the accumulator.

Figure 6H displays the instruction codes. Each instruction code is a two-digit octal number. The ordinate scale extends from 02 (and) to 76 (operate, which is an augmented instruction, the augmentation not shown). The most heavily used instructions are 20 (load accumulator with contents of) and 24 (deposit contents of accumulator into).

Figure 6I displays the memory references and the augmentations. Both are shown here; either class may be suppressed.

In Fig. 6J, all the graphs of Figs. 6A-6I are displayed simultaneously. Because the points are shape-coded, it is possible, though difficult, to reconstruct in detail the sequential pattern of a program from graphs of this type. They might therefore find application in historical documentation of very critical computations, such as those concerned with rocket launching and air defense. In any event, the composite representation conveys an impression of the great capability computer's have to introspect upon their internal processes and report about them in detail.

As we leave this topic, we should perhaps mention the phenomenon that appears when Program Graph is equipped for recursive operation and set to display its own operation. The result, of course, is only a recursion of beginnings, terminated by overflowing of the pushdown list. This effect is not entirely foreign to human introspection.

The routine, Memory Course, plots a grid-like map of memory and displays, against the background of the grid, the course through memory taken by the object program. The dots of the grid represent memory registers, and the dot that represents the register containing the instruction presently being executed is encircled. As control passes from one instruction to another of the object program, a line is drawn connecting the corresponding registers. The effect is hard to illustrate in a still photograph because its effectiveness depends largely upon the kinetic character of the display. However, Fig. 7 may convey an approximate impression. Because the photograph integrates over time, it shows a longer segment of the program's course through memory than one sees when he views the oscilloscope directly.

Program Graph and Memory Course are but two of many possible schemes for displaying the internal processes of the computer. We are working on others that combine graphical presentation with symbolic presentation. Symbolic presentation is widely used, of course, in "debugging" routines. If many symbols are displayed, however, it is not possible to proceed through the program rapidly enough to find errors in reasonable time. By combining graphical with symbolic presentation, and putting the mode of combination under the operator's control via light pen, we hope to achieve both good speed and good discrimination of detailed information.*

Problems to be Solved in Man-Computer Communication

Among the problems toward which man-computer symbiosis is aimed -- problems that men and computers should attack in partnership -- are some of great intellectual depth and intrinsic difficulty. The main problems that must be solved to bring man-computer symbiosis into being, however, appear not to be of that kind. They are not easy, but their difficulty seems due more to limitations of technology than to limitations of intelligence.

What we would like to achieve, at least as a sub-goal, is a mechanism that will couple man to computer as closely as man is now coupled to man in good multidisciplinary scientific or engineering teams.

For a psychologist to telephone a mathematician and ask him, "How can I integrate $\int (dx/(1-x^2))$?" required, in one empirical test, 105 seconds, including 65 seconds devoted to dialing and formalities with the mathematician's secretary plus 32 seconds of preamble with the mathematician. To ask the mathematician that particular question is, of course, wantonly to waste his time -- 170 seconds of it, in this case, since all he needed to say was: "Look it up in any table of integrals," and all he did say was that sentence embedded in a context of encouragement and courtesy. (To find a table of integrals[2] and then to locate the entry took the psychologist, who missed the relevant formula on his first pass and started over at the beginning after scanning 569 entries, 7 minutes and 25 seconds.)

---

* Preliminary study of these displays of internal computer processes was supported through a contract with the Council on Library Resources.

What we would like the computer to do for us, in the context of the foregoing example, does not require such a deep solution as an algorithm for formal differentiation; it requires merely good communication and retrieval. We would like to have an arrangement that would let the psychologist write on his desk input-output surface:

$$\int \frac{dx}{1-x^2} = \text{what?} \quad , \qquad (2)$$

and then let the computer replace the "what?" -- in perhaps 2 or even 20 seconds -- by the expression:

$$\frac{1}{2} \log \frac{|1+x|}{|1-x|} \quad . \qquad (3)$$

In the example, our aspiration would not stop, of course, with the display of expression (3) in symbolic form. The psychologist would surely want elucidation. His next request might be "Please plot a graph," or, if the novelty were worn off, simply "Graph." We would then like to have the computer display on the input-output surface a figure[3] such as Fig. 140 in Dwight's Tables. The figure would, of course, be plotted from computed points, not retrieved from storage. It would be no trouble for the computer to calculate and present it in a few seconds. (For the psychologist to plot a rough graph of the integral took 12 minutes. For another person to locate a published figure (Dwight's) took 17 minutes: a little more than 16 to get to the document room and thumb through books that did not contain the figure, and then a little less than 1 to pick up Dwight's book and scan as far as page 29, where the figure is.)

Five Immediate Problems

Consideration of many such examples as the foregoing and of what would have to be done to put the computer's clerical power conveniently and responsively under the control of human initiative suggests that the main essential steps to man-computer symbiosis are the following:

1. For the economic reason mentioned in the Introduction, develop systems for sharing the time of digital computers among many users on a split-millisecond basis. With J. McCarthy and S. Boilen, one of us is working on a small-scale prototype of such a system with five user stations.*

2. Devise an electronic input-output surface on which both the operator and the computer can display, and through which they can communicate, correlated symbolic and pictorial information. The surface should have selective persistence plus selective erasability; the computer should not have to spend a large part of its time maintaining the displays. The entire device should be inexpensive enough for incorporation into a remote console. An interesting approach to the man-to-machine part of this problem is being taken by Teagher.[4] We are employing an oscilloscope and light pen to fulfill the function, but they do not meet the cost and selective-persistence requirements.

3. Develop a programming system that will facilitate real-time contingent selection and shaping of information-processing procedures. The system must permit trial-and-error operation based upon "tentative computation": it will often be necessary to go back to the beginning or to an intermediate point and to try a different attack. We are experimenting with interpretive systems for on-line assembly of procedures from sub-procedures,** and we are planning work on console compiling, intermeshed with testing and contingent application of procedures as they are required by the human components of the man-computer partnership.

4. Develop systems for storage and retrieval of the vast quantities of information required to support, simultaneously at several user stations, creative thinking in various areas of investigation. For economic reasons, such systems must almost certainly be hierarchical, moving information from large-capacity, fast-access storage as (or shortly before) the information is needed. To achieve the desired effectiveness, it will probably be necessary to make advances in the direction of parallel-access, associative memory with preliminary activation based upon apperceptive relevance. In this area, we believe, much fundamental study of

* The work on time-sharing is supported by Grant R68568 from the National Institute of Health.

** One of the systems is based on a typewriter control program, Process Control, written by D. Park.

information indexing and of memory organization will be necessary before truly satisfactory hardware can be designed, but it appears that quite a bit can be accomplished directly through development of memories -- probably read-only memories -- with very large capacity and moderately fast access and through the application of existing keyword or descriptor techniques.

5. Solve the problem of human cooperation in the development of large program systems. It appears that the development of effective human cooperation and the development of man-computer symbiosis are "chicken-and-egg" problems. It will take unusual human teamwork to set up a truly workable man-computer partnership, and it will take man-computer partnerships to engender and facilitate the human cooperation. For that reason, the main tasks of the first time-sharing computer system with many remote stations may well be in the areas of language and procedure development.

In the five problem areas just mentioned, "to begin is everything," even if it is necessary at first to build research systems along lines that would be uneconomic for widespread application. If we neglect the arguments of economics and elegance we can think at once of ways of solving, or at least starting to solve, the problems. These ways will probably be adequate to test the premise that man-computer symbiosis will be able to achieve intellectual results beyond the range of men alone or of computers programmed and operated in conventional ways.

Four Long-Term Problems

In four other areas, the problems to be solved appear -- if they are not simplified beyond recognition in the effort to make them tractable -- to be deep and intrinsically difficult. The first of these areas is computer appreciation of natural written languages, in their semantic and pragmatic as well as in their syntactic aspects. The second is computer recognition of words spoken in context by various and unselected talkers. The third is the theory of algorithms, particularly their discovery and simplification. The fourth is heuristic programming. We believe that these four areas will in the long term be extremely important to man-computer symbiosis, but that man-computer partnerships of considerable effectiveness and value can be achieved without them. We suspect that solutions in these areas will be found with the aid of early man-computer symbioses, rather than conversely.

An Intermediate Problem

A system combining an elementary form of computer speech recognition, computer recognition of carefully hand-printed characters, and simple light-pen editing techniques, would provide, we think, a very convenient and effective communication link between man and computer. The problems involved in creating such a system seem to us to be intermediate between the five and the four. They may be solved in time to permit the use of correlated voice-hand input in the earliest man-computer partnerships, but, if the required solutions are not ready, it would not be good to wait for them.

References

1. James J. Souder, Madison Brown, Weldon Clark and Jerome Elkind, Collaborative Research in Hospital Planning, United States Public Health Service, Project W-59, to be published in the spring of 1962.

2. B. O. Peirce, A Short Table of Integrals, 3rd edition, Boston: Ginn and Company, 1929.

3. H. B. Dwight, Table of Integrals and other Mathematical Data, 3rd edition, New York: The Macmillan Company, 1957.

4. H. M. Teager, Semi-Annual Progress Reports dated January 1961, June 1961 and January 1962, M.I.T. Computation Center. Also, Quarterly Progress Reports 2, 3, 4 and 5 of the Real-Time Time-Sharing Project, M.I.T. Computation Center.

```
0

Good afternoon.  This will be
your German-English Lesson No. 4.
If you are ready to start at
once, please type "s."  If you
would like to review the
procedure, please type "p."

s

reichen          to hand• +
reichen          to pass• +
     64               64   good
öffnen           to offer• - ta   n
to open          to open•
   -120              -56   poor
arbeiten         to arbitrate• - ta   y
arbeiten         to look• - ta   n
to work          to work•
   -184              -240   Dumbkopf!
kochen           to cook• - ta   y
kochen           to boil• +
      0            -240   okay
öffnen           to open• +
     64            -176   hot dog
rauchen          to smoke• +
     64            -112   admirable
arbeiten         to work• +
     64             -48   good
kochen           to boil• +
     64              16   very good
machen           to make• +
     64              80   Keep it up.
     80

That's it.  You did well.  I'll
be looking forward to the next
lesson.
```

Fig. 1 -- Typescript of a short illustrative lesson in which a computer plays the role of instructor in language-vocabulary drill. The student typed "0" to start the session, "s" to start the lesson, the English words (and terminating dots) in right-hand column, and the abbreviations of "yes" and "no" in response to the computer's "ta" ("Do you want to try again?"). The computer typed the remainder, including scores and comments. The procedure is explained in the text.

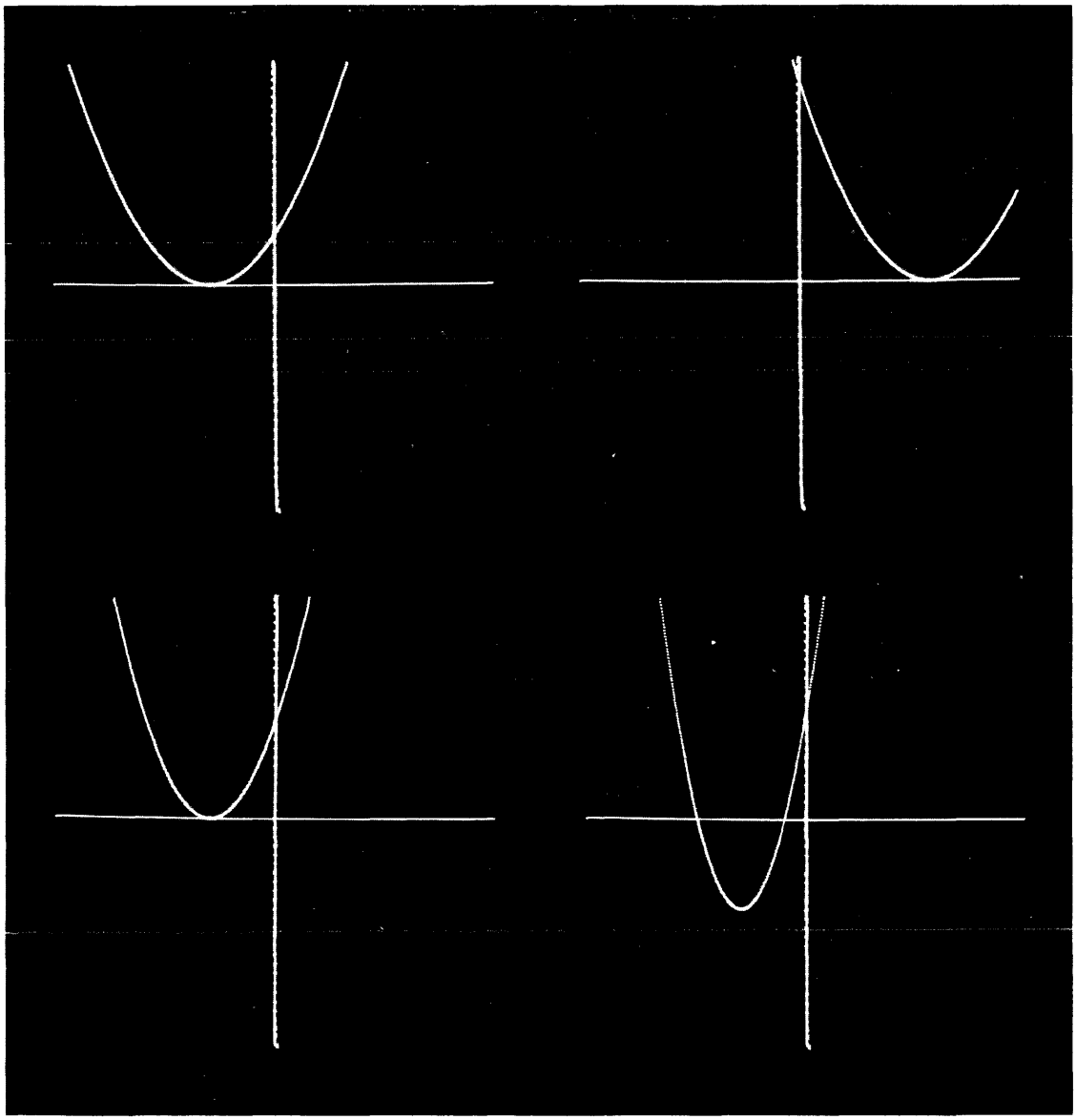


Fig. 2 -- Parabolas displayed by computer to facilitate student's exploration of relations between graphical and symbolic representations of mathematical expressions.

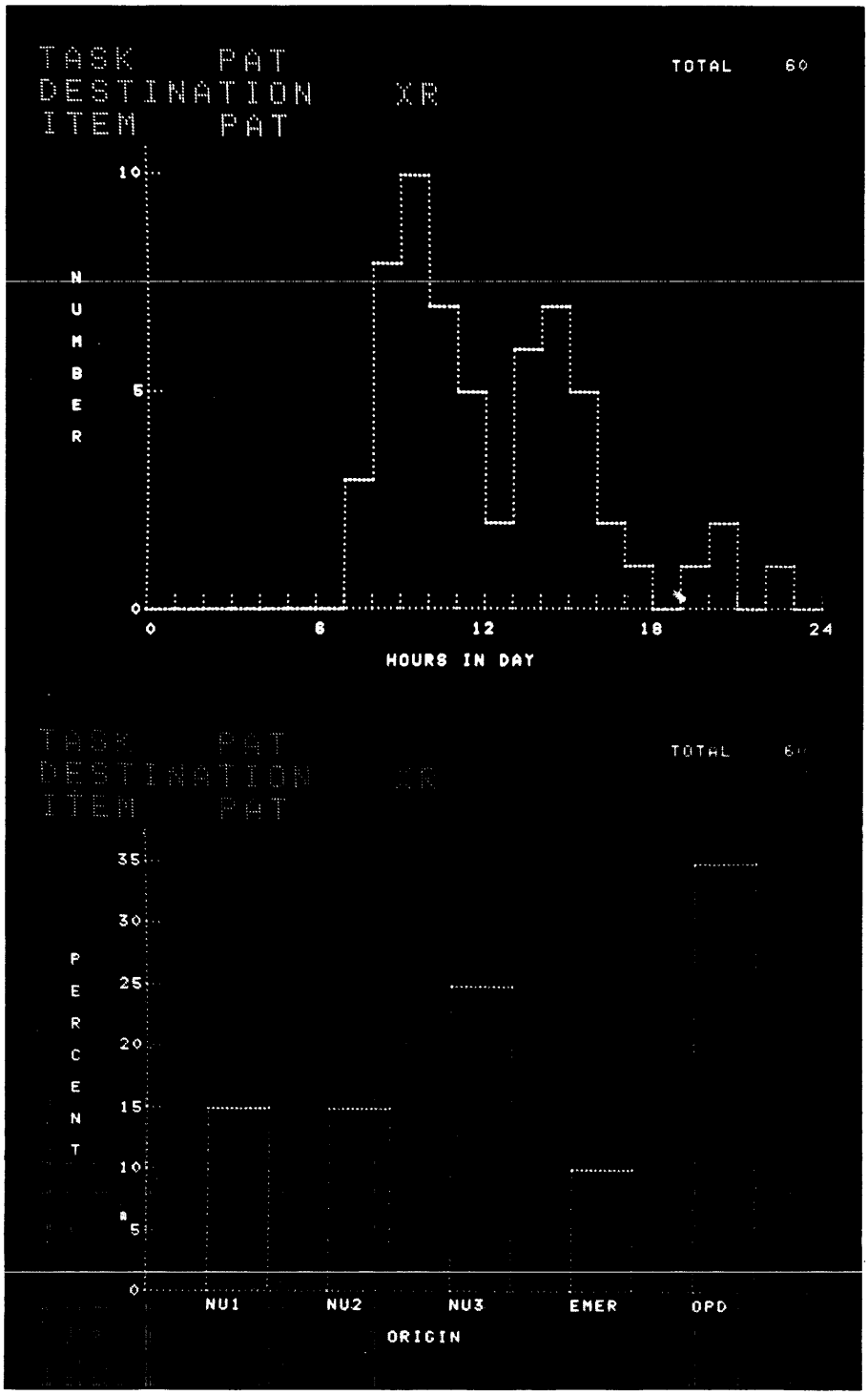


Fig. 3 -- Oscilloscope displays of several aspects of a projected inter-departmental "commerce" pattern in a hypothetical hospital. The upper graph shows the anticipated time distribution of patient transport trips to the X-Ray Department. The lower graph shows the conditional distribution of those trips among departments of origin.

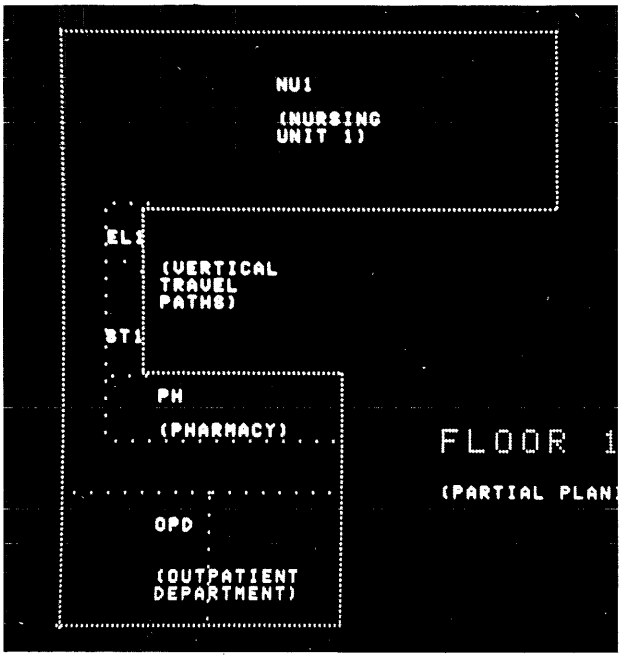


Fig. 4 -- Oscilloscope display of an outline planning sketch of one floor in a hypothetical hospital.

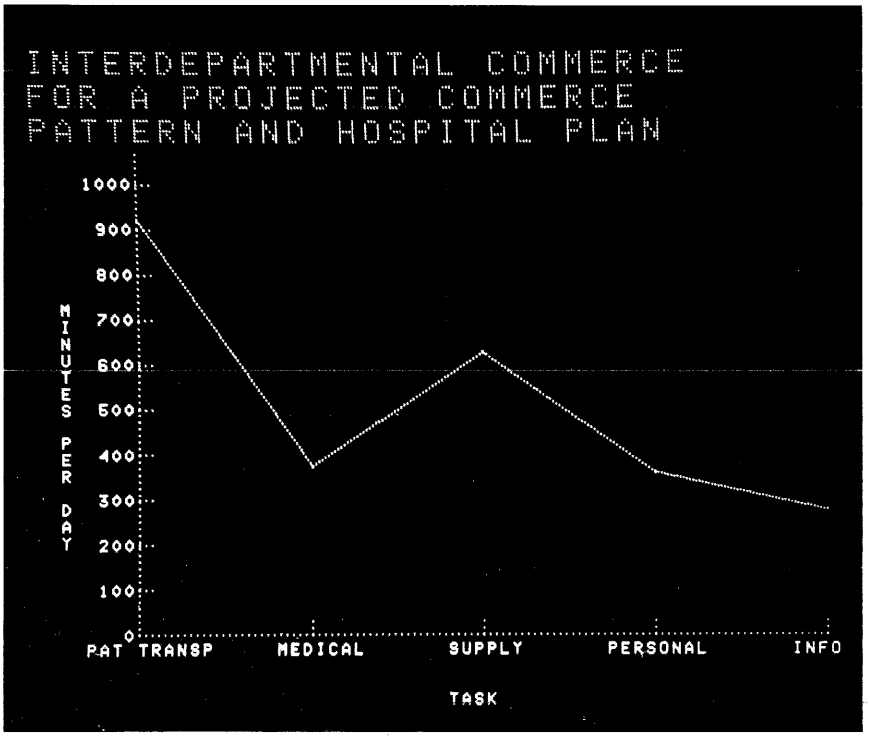


Fig. 5 -- Oscilloscope display of the performance, in respect of "commerce," of a proposed hospital plan. Scale time is defined as man-minutes spent in transit. The contributions of individuals to this quantity are weighted by coefficients associated with their personnel categories.

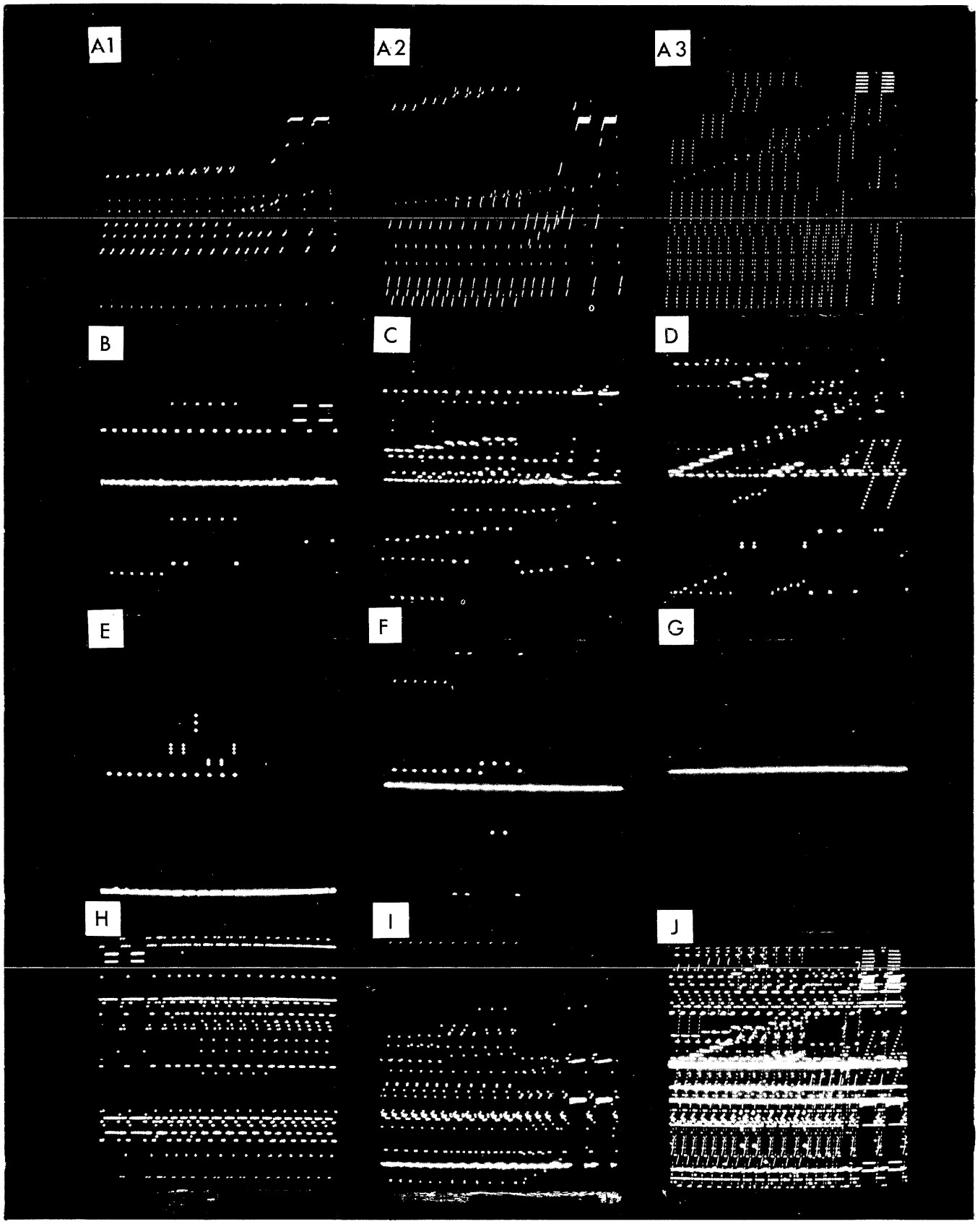


Fig. 6 -- Photographs of oscilloscopic displays made by Program Graph. See text for interpretation.

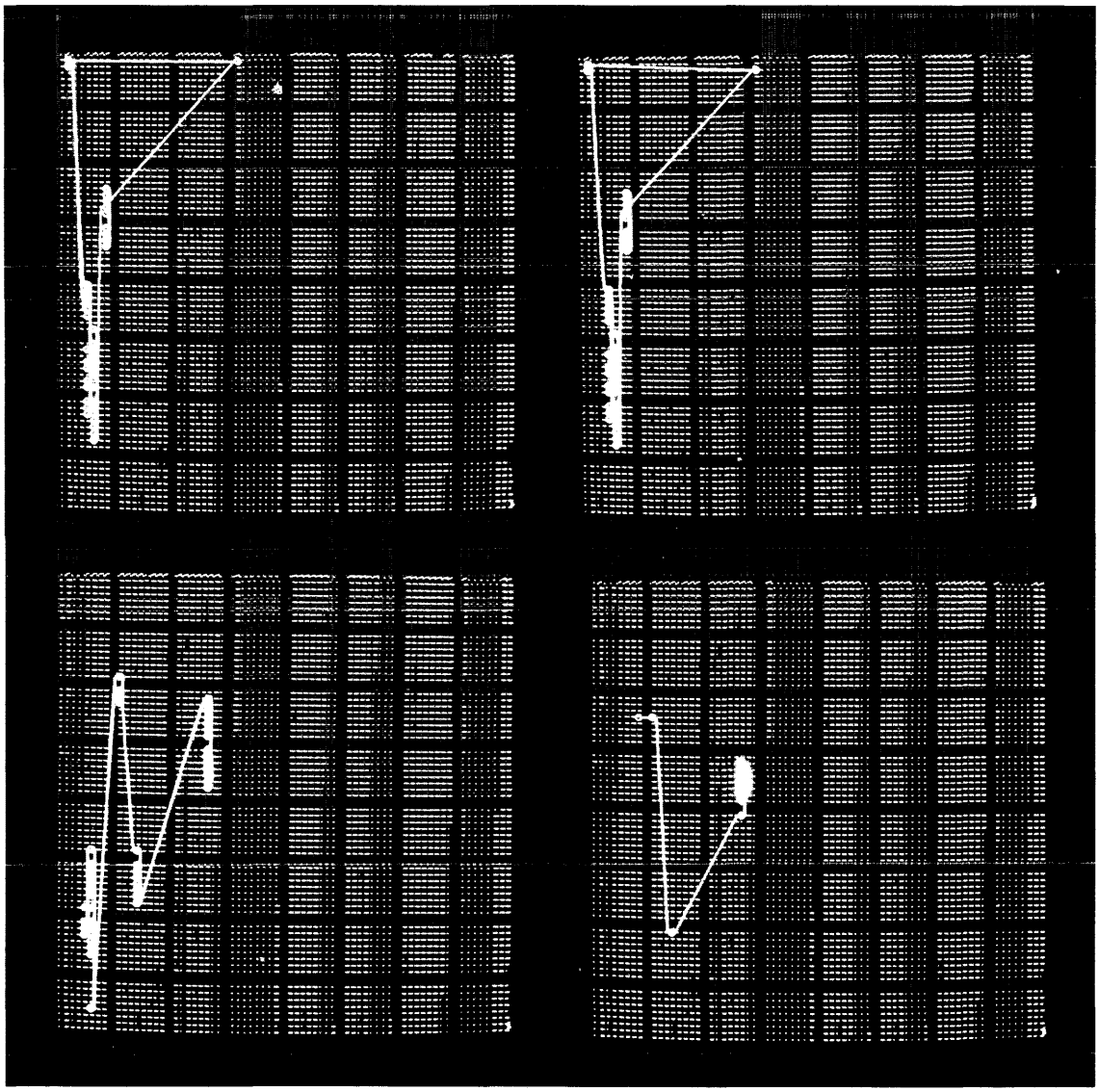


Fig. 7 -- Photograph of oscilloscopic display made by Memory Course. See text for interpretation.

SOLUTION OF NONLINEAR INTEGRAL EQUATIONS USING ON-LINE COMPUTER CONTROL

Glen J. Culler* and Robert W. Huff**

Ramo-Wooldridge, a Division
of Thompson-Ramo-Wooldridge, Inc.
Canoga Park, California

Summary

Experiments have been carried out using on-line control of a digital computer to obtain solutions for certain non-linear integral equations. The computer program did not anticipate any particular method of solution; each user constructed his own method during the solution process. Easy cases, those for which the straightforward iteration converges, were solved with half as many iterations. Difficult cases that were both globally and locally divergent were solved through careful control by the problem solver. Photographs displaying the solution process for two sample cases are included.

Introduction

The basic philosophy of on-line computation is principally that the originator of the problem shall, in a fairly detailed fashion, actually direct its solution by the computer, thereby bringing to bear his own insights and knowledge based on the problem's structure or physical significance. To make possible an operation of this sort requires a fairly sophisticated computer with suitable interrupt capability and a form of input-output equipment which provides for rapid, sophisticated displays of graphical or numerical information from the computer, and offers convenient means for controlling the computer's activities by easily and accurately feeding back information to the computer in graphical or numerical form.

A system of this type can be realized in many ways. At the present time a satisfactory version of the requisite input-output equipment exists in the form of a Display and Analysis Console (DAC) developed at Ramo-Wooldridge.*** For output, this console has two 17" scopes (with resolutions of 10^3 lines in each direction) which permit the display of line segments as well as points. Input of information to the computer is accomplished by means of a crosshair, whose position changes in response to a control lever, and whose coordinates are, on push button command, transmitted to the computer, leaving a small cross displayed on the scope; a light gun (photocell) which can be used to select, with complete accuracy, any desired luminous point of a display; a series of push buttons, each of which calls up a sub-routine from the computer (a subset of these have an overlay feature which allows for conveniently changing from one problem to another); and a set of numerical keys used for feeding in numerical information. Using this equipment we have begun to investigate and explore the potentialities of on-line solution of scientific problems, and in particular, to develop the techniques necessary for literally carrying out mathematical analysis at the DAC keyboard. We report here the results of one of our first experimental attempts in this mode of operation.

To explore the potentialities and characteristics of on-line scientific computing, it is clear that one must attempt to solve a variety of suitably chosen, difficult problems. In order that the results have a content and value beyond a mere exercise in the use of displays, we imposed the following criteria:

a. the problem must be one which presents real difficulties for conventional computer techniques and mathematical analysis;

b. the problem must be one as yet unsolved (for any soluble problem can usually be solved in many ways, and the ability to do so using a new technique may not indicate the latter's real value); and

c. the problem should be one of significance for some area of current scientific research, that is, a problem whose solution is of value in its own right, apart from any interest in the methods used to obtain it.

---

*On leave from University of California, Santa Barbara, California.

**University of California Radiation Laboratory, Berkeley, California.

***This is a console component of AN/FSQ-27 equipment developed under the auspices of Rome Air Development Center, USAF.

Once a problem was selected, we adopted the additional ground rule that the computer program to be used should involve no mathematical or computational methods beyond those needed to construct the basic operations of analysis, thus requiring the user to compose his techniques at the DAC.

The first problem selected was taken from the Bardeen-Cooper-Schrieffer (BCS) theory of superconductivity. It was suggested and formulated by Dr. J. Robert Schrieffer, of the University of Illinois, while a consultant to this Laboratory, in July 1961. We will concentrate here upon the mathematical formulation of the problem and the techniques used to solve it. A discussion of the physical significance and interpretations of results will be published elsewhere.

The BCS Integral Equation

In a normal (non-superconducting) metal, an elementary excitation or "quasi-particle", associated with momentum p, has energy $\epsilon(p)$, whereas according to the BCS theory, the elementary excitations in a superconductor have energy

$$E(p) = \sqrt{\epsilon^2(p) + \Delta^2(p)}$$

The "energy-gap parameter", Δ, which may be considered as a function of p, E or ϵ, satisfies a non-linear integral equation

$$\Delta(E) = k \int_0^{\omega} d\epsilon' \frac{K(E, E')\Delta(E')}{E'} \qquad (1)$$

where

$$E' = \sqrt{(\epsilon')^2 + \Delta^2(E')}$$

and

$$K(E,E') = \left\{ f(E + E') + f(E'-E) - C \right\} \frac{\tanh E'/2T}{}$$

with

$$f(E) = \frac{1}{2} - E + E^2 \ln \left| \frac{1 + E}{E} \right|$$

The constant k measures the strength of the attractive force between electrons which arises from their interaction with the ionic lattice vibrations (i.e., from the emission and absorption of phonons); C gives the strength of the Coulomb repulsion between electrons (relative to the phonon induced attraction); and T is the temperature. The equation (1) always has the trivial solution $\Delta \equiv 0$, but only for a superconductor does there exist, in addition, a non-zero solution. This is the case for C = T = 0, i.e., in absence of the Coulomb repulsion between electrons every material would (in this theory) be a superconductor at absolute zero. The physically interesting questions are:

a. For given C and k, what is the largest temperature for which (1) has a non-trivial solution? This will be the critical temperature.

b. For T and k given, what is the largest value of C which permits $\Delta \not\equiv 0$? On the basis of this, one can, in principle, predict which materials should be superconductors and which should not.

c. For given k, T and C, what is the shape of the solution $\Delta(E)$? From this one can find the density of states and thereby explain, quantitatively, the phenomenon of electron tunnelling through an insulating film separating two superconductors.

On-Line Approach

Our method of solution, in conformity with the ground rule concerning the simplicity of the programming, requires only that the computer, given a trial $\Delta(E')$, evaluate the definite integral in equation (1) for a number of different values of x. The operator then merely controls the input in such a manner as to produce agreement between the (normalized) input and output curves. To aid himself in this, the operator can, upon pushbutton command, call up "Present Input", the function $\Delta(E')$ used in the integration on the righthand side of (1); "Present Output", the function $\Delta(E)$ on the lefthand side of (1) which results from the integration (normalized to agree with "Present Input" at E = 0); and "Present In-Out Ratio", the ratio of these two functions. Lack of agreement between input and output is readily apparent from the DAC display, when one is far from a solution. On the other hand, when one is near a solution the deviations of the in-out ratio from unity provide a very sensitive test.

Additional features available to the operator are the following:

a. In addition to "Present Input", "Present Output", and "Present In-Out Ratio", he can call up "Past Input", "Past Output", and "Past In-Out Ratio", these being the corresponding quantities resulting from the previous integration.

b. On pushbutton control, it is possible to expand or to contract the displayed curves in the x or in the y direction, by as many powers of two as desired.

c. Of the two DAC 17" scopes, one is used to display the function $\Delta(E)$ while the second one displays $\ln \Delta$ as a function of $\ln E$, thereby facilitating examination of the asymptotic behavior of the solutions.

In addition to the sophisticated display capability represented by the above features, an essential requirement of such on-line operation is the ability to give directions and input data easily and rapidly to the computer. The former is accomplished by means of the DAC's push-buttons. For the latter, we have available the numerical keys of the DAC, the crosshairs and the light gun. The numerical keys are used to feed in the values of the parameters C, T and $\Delta(0)$ at the start of each new case. The crosshairs provide one means of controlling the selection of the input curve, $\Delta(E')$. At each stage in the solution of the problem, one has the option of using the output from the previous integration as input for the next. However, for many values of the parameters ($C \neq 0$) this process fails to converge. The alterations in the output curve which must be carried out at each stage in order to achieve convergence to a solution are conveniently made by means of the crosshairs. With the insertion of only a few crosshair points and the use of relative interpolation, it is possible to produce the desired changes in the output curve with a minimum of labor and without gross alterations of its mathematical character. (The technique of relative interpolation uses the curve to be modified as a basis for the interpolation rather than using some preselected class of functions.) In addition to this capability of relative interpolation, the program provides the possibility of using, as input for the next integration, any desired linear combination of input and output curves, over all or a portion of the domain of the functions. Also, when starting a new case it is convenient to have available some previous solution as a basis for relative interpolation. As one learns how changes of parameters effect the solutions, relative interpolation can be used to alter the previous solution in anticipation of the form of the coming solution. Finally, the asymptotic form of the solution can be obtained when it has a power-law behavior. The light gun, when pointed at two points on the (asymptotically) straight $\ln \Delta$ vs. $\ln E$ curve, was used to instruct the computer to display both the power-law based on these two points and the portion of the solution in that asymptotic region.

Armed with this collection of very simple techniques, the operator at the console can compose remarkably powerful methods of solution. For example, we have observed, in solving the BCS integral equation, that on successive iterations certain portions of the curve tend to oscillate while other parts grow or decay in a monotonic fashion. Given the facilities described above, it is quite easy to control such behaviors by employing different strategies for different portions of the same curve, and thereby achieve rapid convergence to a solution. To accomplish this using conventional computer techniques would not only require a detailed prior knowledge of the mathematical structure of the problem and of the properties of the solution, but would also entail programming difficulties of no small magnitude. This provides a nice illustration of one of the principal advantages of the on-line method, namely the operator's ability to use information obtained in the course of the solution to alter and improve the very method of attack.

Two Examples of the Solution Process

As an illustration of the use of on-line computation for the solution of physical problems, we present here the solution of the non-linear integral equation (1) described above for two representative sets of parameter values.

In view of the fact that for given values of C, T and k, equation (1) may have no solution other than the trivial one $\Delta \equiv 0$, we specify for each case C, T and the value of Δ at $E = 0$, i.e., at the Fermi surface. We are then guaranteed that a solution exists and, for given values of these parameters, k comes out as a kind of eigenvalue. (One can subsequently represent the results by giving $\Delta(0)$ as a function of k, C and T so that nothing is lost thereby.)

As explained above, instead of a program in the conventional sense, the computer has simply a collection of sub-routines. The principal one of these is simply evaluation of the integral specified in equation (1), i.e., given an "input" function, the computer carries out this definite integral for ninety-six values of E (in the dimensionless units used here the upper limit in equation (1) is $\omega = 12$), rescales the resultant function of E to agree with the specified value, $\Delta(0)$, at $E = 0$, and thereby obtains an "output" function. (The factor needed for the rescaling also provides an approximate value of k.) If, in any way, one obtains an input function which agrees in shape with the resultant output function, one has a solution of the integral equation (1).

For a first illustrative case, we choose the parameter values $\Delta(0) = .01$, $C = T = 0$. As initial input, we use a constant function, i.e., $\Delta(E) \equiv (0)$. When the Iterate button is pressed, the computer takes this input and carries out the integration required for equation (1). (The range of integration is divided into 1536 points and requires two minutes of computation time.) When the computer signals, via a panel light, that the computation is finished, we press the Present Input button and obtain a picture of the input function, i.e., the straight line shown in Figure 1. On pressing the Present Output button,

we obtain the result for the integration, also shown in Figure 1.

At this point we elect to do a straight iteration. We press the Iterate button and the computer takes the result of the previous integration, i.e., the previous output, and uses it as an input for the next integration. When this is completed, we can call up, as shown in Figure 2 Past Input (shown dotted), Present Input (the same as the present output of Figure 1), and Present Output. Since the latter two show a general agreement, continued iteration is indicated. One more pass brings us to the situation of Figure 3 where again Past Input, Present Input and Present Output are shown. Since the negative peaks in the latter are off scale, we push Contract Y which rescales the ordinate by a factor of two, resulting in the display shown in Figure 4. The Present Input and Present Output curves now agree so closely as to be nearly indistinguishable, and we therefore push the Present In/Out key which calls up a display of the ratio of input to output, also shown in Figure 4. One more iteration brings us to the situation shown in Figure 5, where Present Output (now quite indistinguishable from Present Input), Present In/Out Ratio and Past In/Out Ratio (dotted) are displayed. (In general, dotted curves are used to display results of the previous integration.) A last iteration brings us to the excellent solution shown in Figure 6 where the Present In/Out ratio is almost identically equal to one. It should be emphasized, of course, that this ratio provides a very sensitive test for the agreement of input and output functions.

At this point, we push the Print button causing the computer to print a permanent record of the solution obtained. In addition, we might wish to have the solution at a finer scale, in which case pushing the Expand x button will produce the desired result. Figure 7 shows the solution at the original scaling, expanded by one factor of two and expanded by two factors of two. This expansion in x can be carried out at any stage during the solution process and is frequently useful in that it allows us to work with increased accuracy upon that portion of the curve where there is significant structure, ignoring the asymptotic region.

We now consider a second example, this time with the parameter choice $\Delta(0) = .1$, $T = .1$, $C = .5$. To begin with, we try the same approach which proved so effective for the previous case, i.e., we use a constant as an initial input. The resultant Present Input and Present Output are shown in Figure 8. We push the Contract Y button in order to get the output curve on scale, and then carry out another iteration resulting in the display shown in Figure 9, where Present Input and Present Output are given. In contrast to the first case discussed, the input and output curves show no qualitative similarity. If, undaunted, we continue the straight iteration, i.e., use the present output curve of Figure 9 as new input, we obtain the result shown in Figure 10 where Past Input (dotted), Present Input and Present Output are displayed. An additional iteration brings us to the even more disastrous situation of Figure 11 (Past Input, Present Input and Present Output). Present Input and Output at a contracted scale are shown in Figure 12. Clearly we are not on the right track. But, wait, look back at Figure 10. The last output is very nearly equal to the initial input, $\Delta(E) = \Delta(0)$. These differ only by the presence of two shallow minima. Now if we had a linear problem and could neglect these minima, we would have the average of the three curves shown in Figure 10 as our solution. With this in mind we push the Restart with New Parameter button which takes us back to the beginning of the problem, and this time use the crosshair capability to sketch in a guess for the input function. The crosshaired points are shown in Figure 13, while Figure 14 shows the polygonal curve interpolated by the computer to go through these points (Present Input) and the resulting Present Output function. Contraction in y gives the display of Figure 15. Doing one more iteration brings us to Figure 16, where Past Input, Present Input (just below it) and Present Output (the top curve) are shown. An additional iteration brings us to Figure 17, where again Past Input (dotted), Present Input (the highest curve) and Present Output (the bottom curve) are shown. Note that the iteration process is definitely divergent, since over almost the entire range the Present Output curve lies outside the region bounded by the other two.

To render the process convergent, we instruct the computer to use as new input not the previous output, but rather a linear combination

$$\lambda \Delta_{in} + (1 - \lambda)\Delta_{out}$$

In the present case, we choose a value of $\lambda = 0.4$, since this will make the new input lie approximately midway between the top two curves of Figure 17. The result is shown in Figure 18 where we display Past Input and Output (dotted) and Present Input and Output. Note that Present Input, the lower of the solid curves, indeed lies four-tenths of the way between Past Input and Past Output. Since the Present Input and Present Output curve lies much closer together than Past Input and Past Output, we are clearly going in the right direction. The result of one more iteration, with the same value of λ, gives us the result of Figure 19 where the input and output curves are now sufficiently close as to make examination of the Present In/Out ratio, also shown there, sensible. Since the principal deviations of the in/out ratio from unity are associated with the zero crossings of the solution, we now revert to a value of $\lambda = 0$ in order to correct that portion of the solution as rapidly as possible. One iteration with $\lambda = 0$ brings us to the situation of Figure 20 where Present Output and Present In/Out ratio are shown. (The crosshairs are also displayed, since

in their standard position the horizontal one provides a unit reference.) Note that the in/out ratio has greatly improved. In Figure 21 we see the result of one more iteration with $\lambda = 0$. The in/out ratio continues to improve in general, save in the asymptotic region where we notice that it is again divergent, the Past In/Out ratio being closer to unity than the Present In/Out ratio. Since unity lies six-tenths of the way from the latter to the former, we now choose a value of $\lambda = 0.6$ and iterate once again. The result, as shown in Figure 22, is an in/out ratio which deviates from unity at only two of the three zero crossings of the solution. Experience has shown that the most rapid convergence is obtained, not by leaving λ fixed, but by altering it in a suitable fashion from one iteration to the next. In the present case, since we are interested in improving the solution near the zero crossings, we again remove the linear combination feature, i.e., take $\lambda = 0$, thereby obtaining the situation shown in Figure 23. An additional iteration with $\lambda = 0$ leads to Figure 24 and we see that because of the divergence the solution is getting worse in the asymptotic region, and hence we are not improving the zero crossing situation. Observing that one now lies seven-tenths of the way from the Present In/Out ratio to the Past In/Out ratio, we iterate once more, this time with a value of $\lambda = 0.7$, obtaining thereby the excellent final solution shown in Figure 25. As before, we can expand the scale if desired, as shown in Figure 26.

The two examples above do not illustrate the use of relative interpolation as a means for perturbing a large amount of data with very little input information. Figure 27 shows the above solution with four crosshair points inserted and Figure 28 shows the resulting relative interpolation (solid curve) as well as the above solution (dotted). Actually, this was done to construct an input for a new case (namely $\Delta(0) = .1$, $T = .3$, $C = .5$) which proved to be a good first guess.

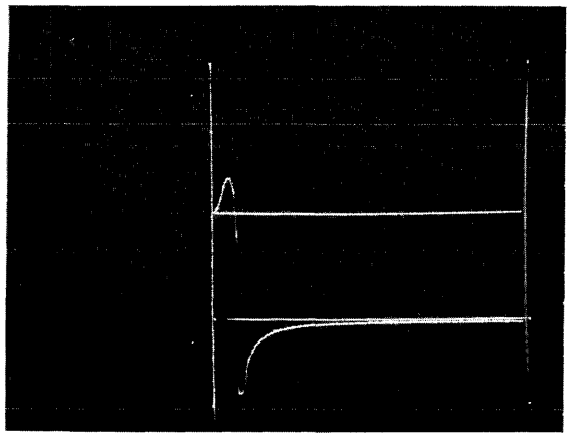
Comments

From our experience thus far there is no question concerning the value of this on-line approach for so-called scientific computation problems. In computing the various curves of physical interest in this first problem (such as $\Delta(0)$ vs. T for constant k and C) the ability to see what was happening and then decide what to do next proved to be both fascinating and efficient. We were able to run through all the cases needed for such a curve and never vary significantly from a solution. On the other hand, when we attempted to find our first solution in a new region of the parameter space, it was often surprisingly difficult.

It is possible to attack a broad variety of problems not of the form attempted here and to extend our work in this direction. However, we recognize that this form of computing is no panacea. When problems really can be automated, they usually should be, notwithstanding occasional gains in efficiency. Also, within the context of a computing center, this on-line operation only makes sense if the main facility is otherwise occupied while the problem solver is scratching his head to decide what to do. But, finally, we are gratified to see that certain forms of cut and try mathematics will soon be reasonably available on a push button basis for the practitioner of the art of classical analysis.

Acknowledgements

We wish to express our gratitude to Dr. J. Robert Schrieffer for his insight in formulating a problem so satisfactory to our needs; to Dr. Burton D. Fried for his good physical intuition and enthusiasm, which were of decisive value in solving the more difficult cases; to Roy Sather and Robert Bolman for their patient cooperation and help in making this computational experiment a success.



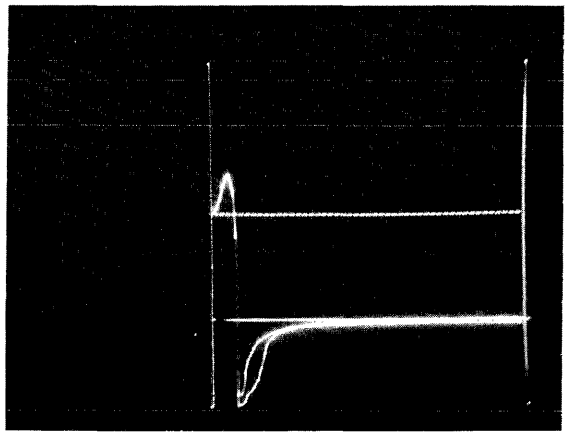
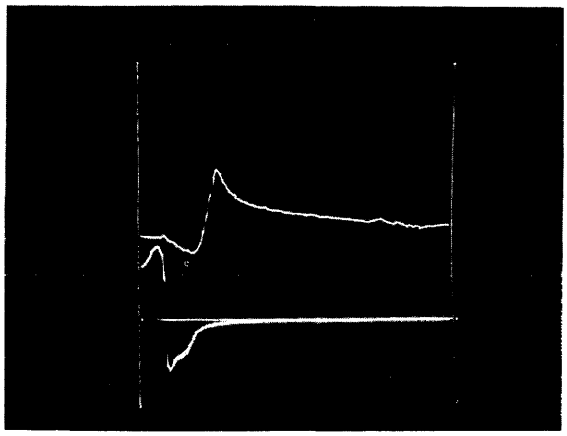


Figure 1: Present Input and Present Output $\Delta(0) = .01$, $C = 0$, $T = 0$.

Figure 2: Past Input (···), Present Input and Present Output.



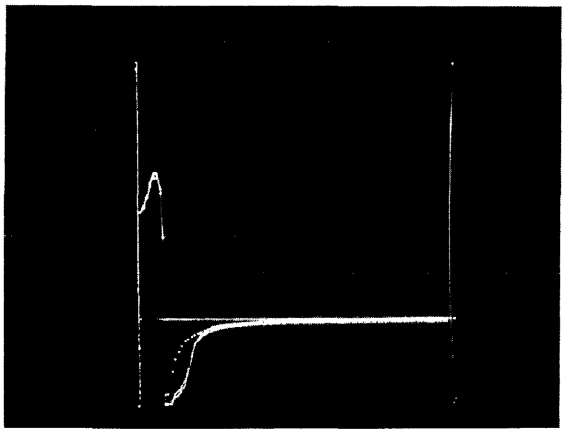
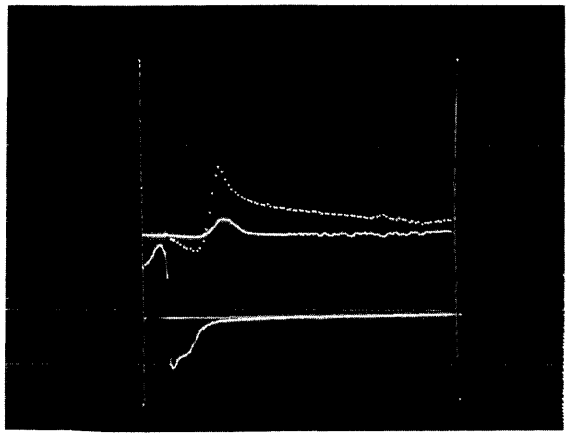


Figure 3: Past Input (···), Present Input and Present Output.

Figure 4: Present Input, Present Output, and Present In/Out.



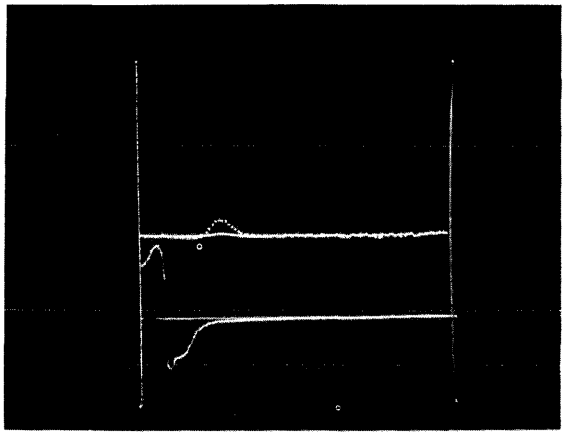
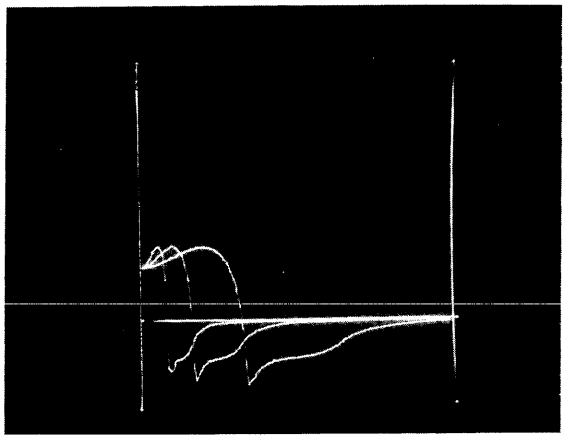


Figure 5: Present Input, Present Output, Past In/Out and Present In/Out.

Figure 6: Present Input, Present Output, Past In/Out and Present In/Out.



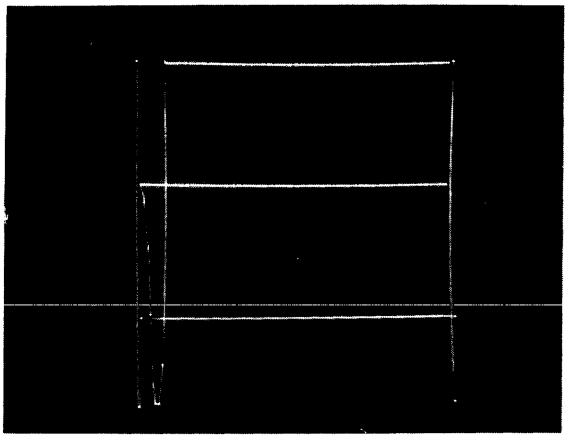
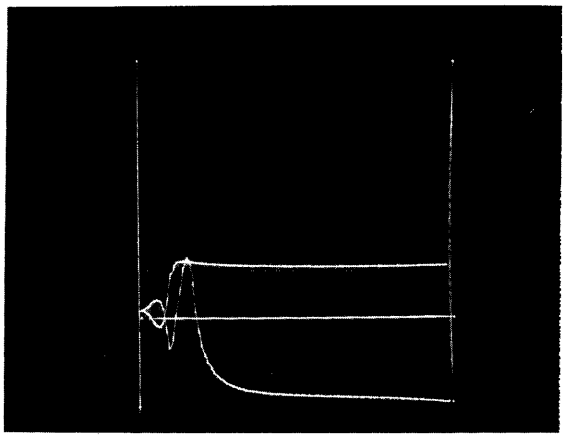


Figure 7: Present Output, Present Output with Independent Variable Expanded by 2 and by 4.

Figure 8: Present Input and Present Output $\Delta(0) = .1$, $T = .1$, $C = .5$.



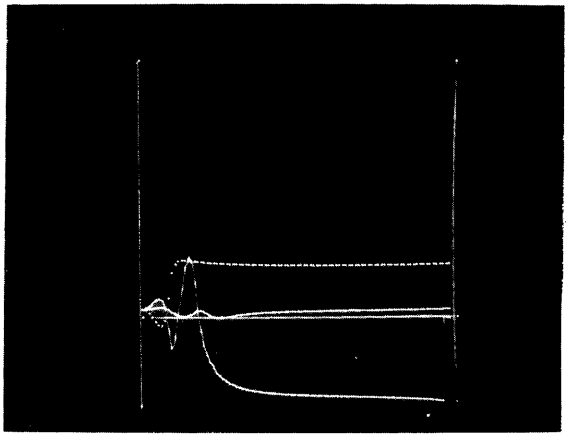
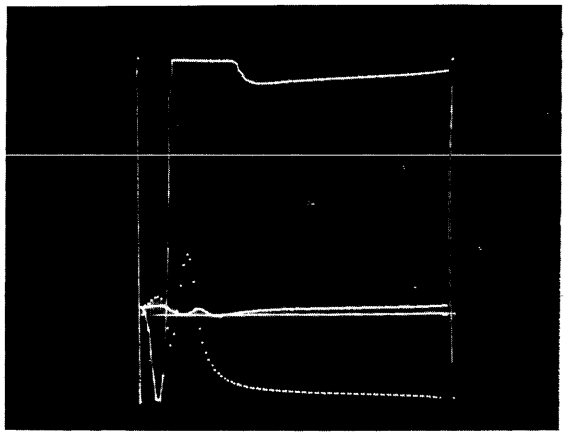


Figure 9: Present Input and Present Output.

Figure 10: Past Input, Present Input and Present Output.



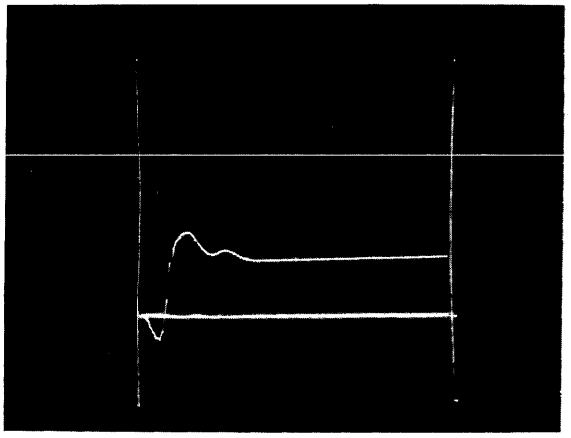
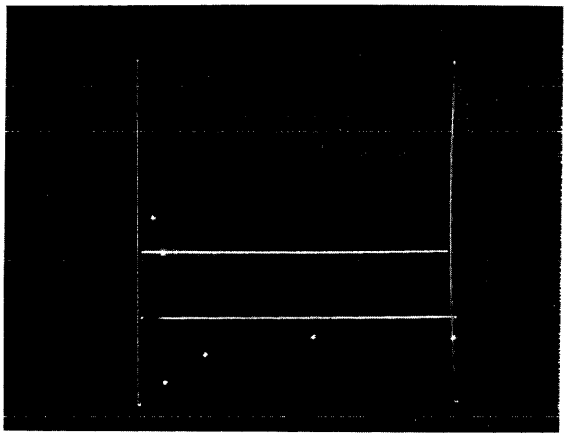


Figure 11: Past Input, Present Input and Present Output.

Figure 12: Present Input and Present Output Contracted by 4 in Y-Direction.



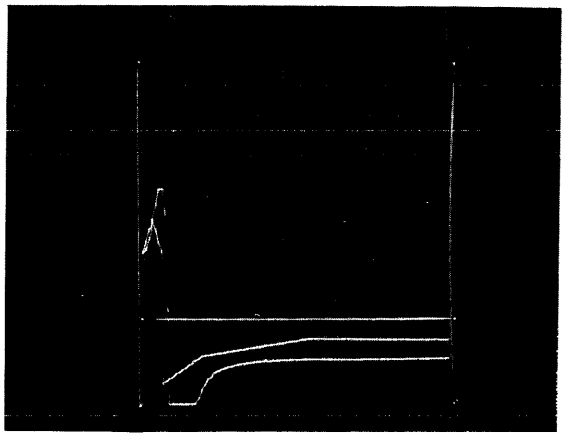
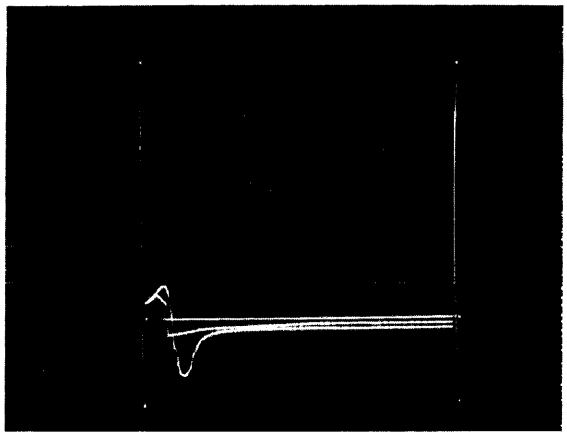


Figure 13: The Constant Function Y = Δ(0) and Five Crosshair Points.

Figure 14: Present Input (polygonal), Present Output (off scale).



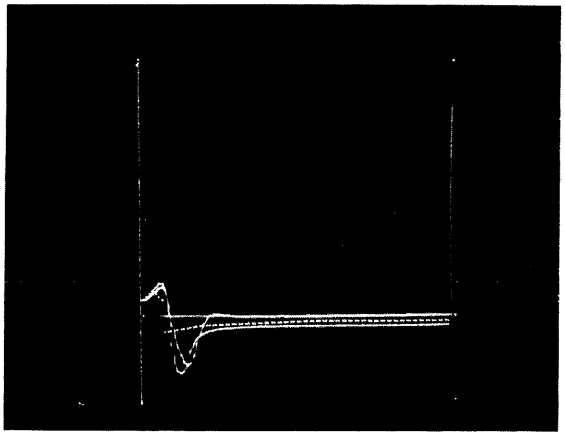
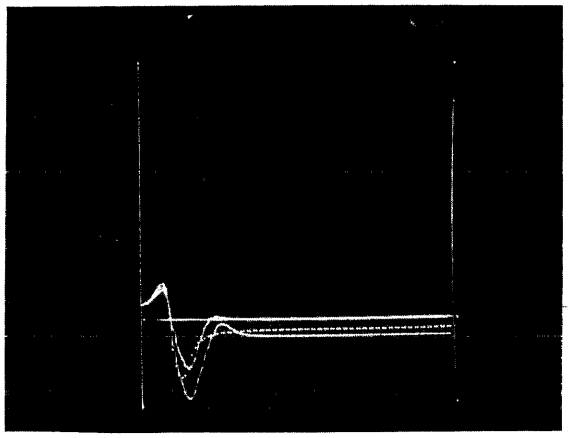


Figure 15: Present Input, Present Output Rescaled.

Figure 16: Past Input, Present Input and Present Output.



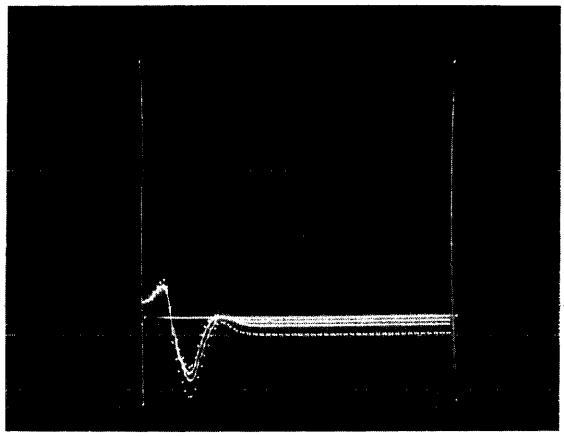
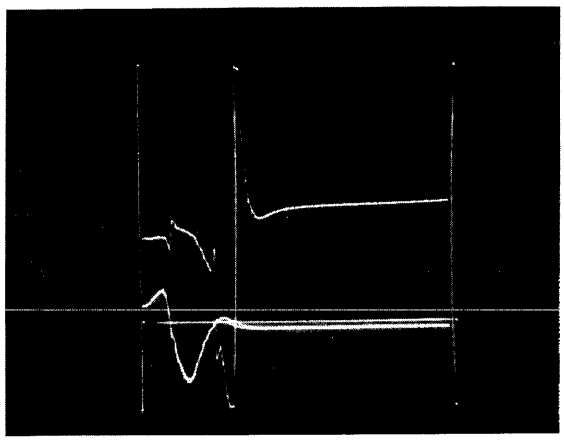


Figure 17: Past Input, Present Input and Present Output (note divergence).

Figure 18: Past Input, Past Output, Present Input (λ = .4), and Present Output.



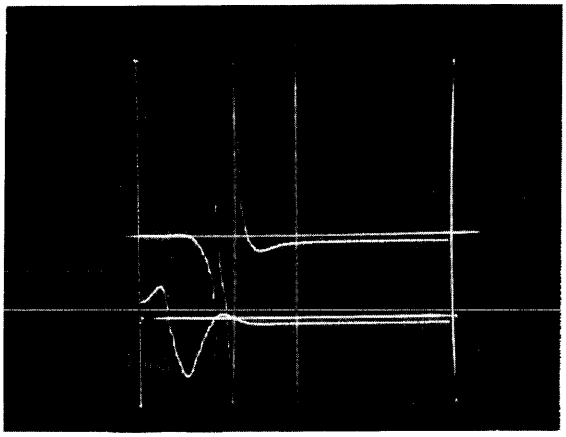
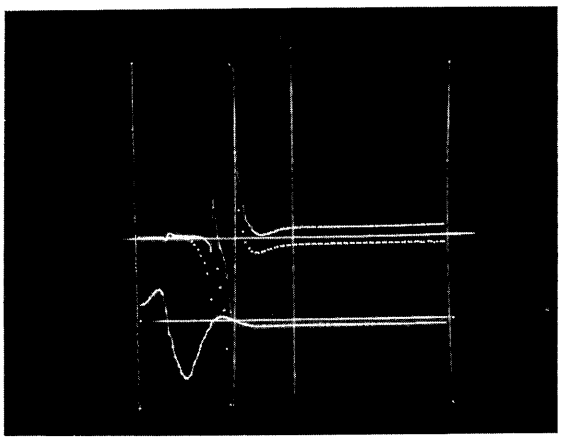


Figure 19: Present Input ($\lambda = .4$), Present Output and Present In/Out.

Figure 20: Present Output ($\lambda = 0$), Present In/Out.



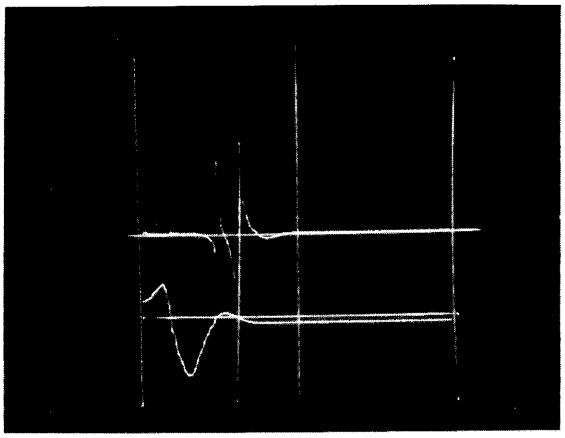
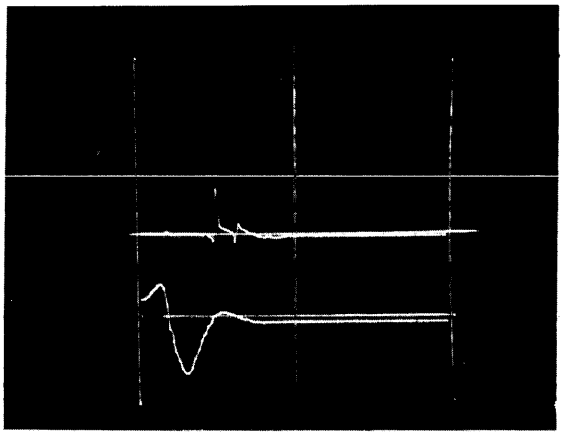


Figure 21: Present Output ($\lambda = 0$), Past In/Out and Present In/Out (observe tail oscillation).

Figure 22: Present Output ($\lambda = .6$), Present In/Out.



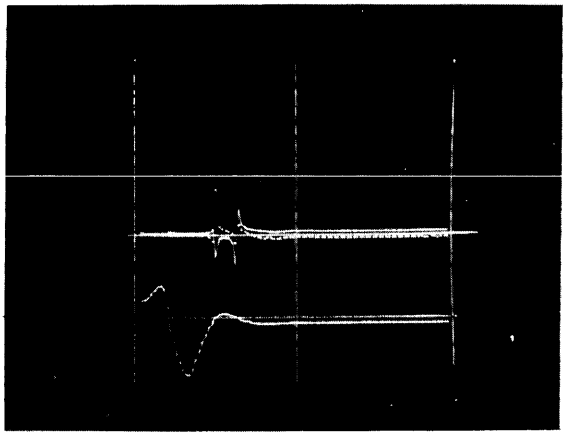
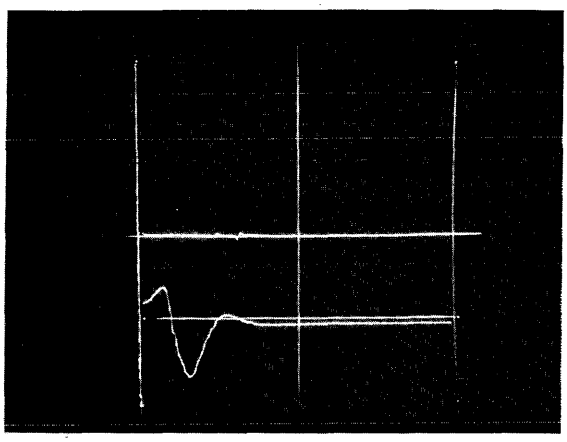


Figure 23: Present Output ($\lambda = 0$), Present In/Out.

Figure 24: Present Output ($\lambda = 0$), Past In/Out and Present In/Out (note over-all oscillation).



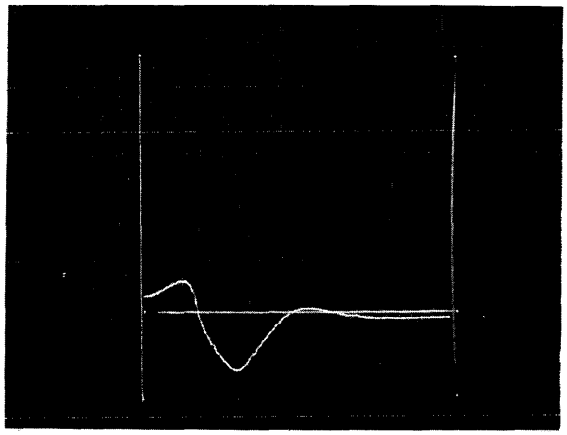
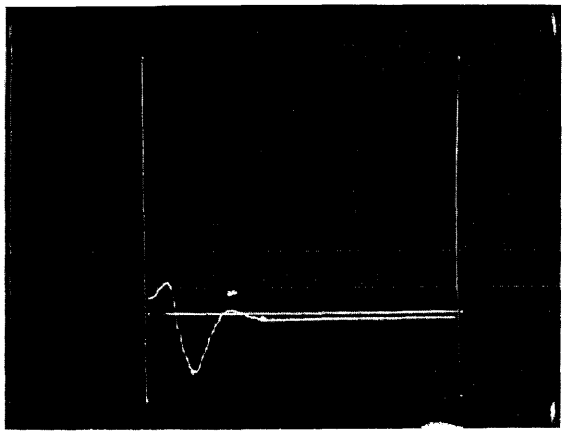


Figure 27: Present Output Contracted in x by the Factor 2 and Four Crosshair Points.

Figure 28: Past Output and Present Input Showing Results of Relative Interpolation.



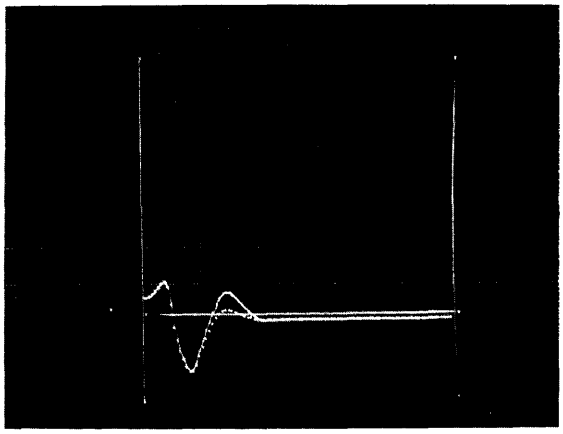


Figure 25: Present Output (λ = .7) and Present In/Out.

Figure 26: Present Output Expanded in x by the Factor 2.

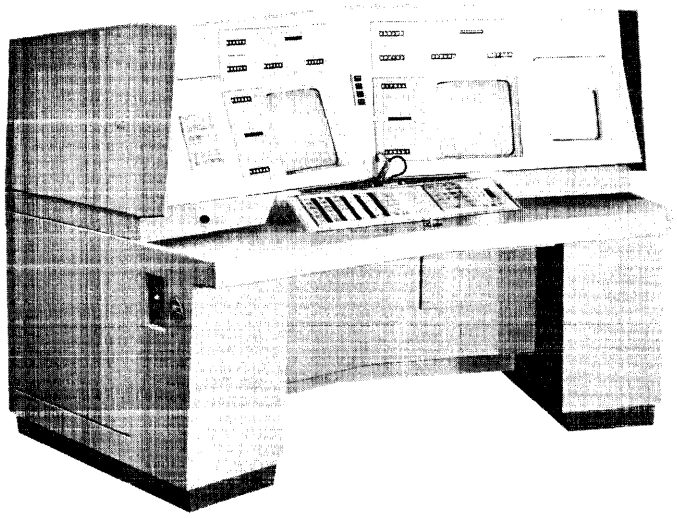


Figure 29: Front View of Display and Analysis Console (DAC)

ARE THE MAN AND THE MACHINE RELATIONS?

Burton R. Wolin

System Development Corporation

Santa Monica, California

Summary

Humans are considered as general purpose computers. They ask two questions of the environment, "What is most likely to be the situation next?", and, "What do I do now?" A research program is described which seeks to determine how, and how well, humans can answer the former question or predict the environment.

Introduction

As environments requiring control have become more complex, and the speeds of events in those environments have increased, there has been a trend to supplementing or replacing men, or the functions they have traditionally performed, with computers.

We can first ask the most general question about complex systems: What are they primarily intended to do?

The answer is that they are designed and produced to give some measure of prediction and control over some aspect or aspects of the real-world environment. To accomplish this general task they must have sensors, so that they can obtain data from the environment. They must also have some power to discriminate between events of interest in the environment and others. They must be able to analyze, synthesize, organize, and classify the data events of interest. They must, of course, have effectors and some control over them.

What have we described? We have described a government, a business enterprise, a military system, an air-traffic-control system, and a biological organism. We have described in general what has recently come to be called an "open system," that is, any system which maintains or increases its organization by feeding off the environment.

Chapman, Kennedy, Newell, and Biel[1] pointed out that a complex of men and hardware performing a complex task could be likened to an organism; that such system grows organization and learns from and adapts to its environment.

Chapman et al. studied the manual air-defense system, which was a system where men were the central components. They performed all of the significant operations in the system. They discriminated, classified, and calculated. They figured out what the situation was and what to do about it.

This manual air-defense system was also, interestingly, one of the first systems which became inadequate to cope with the increasingly complex, fast changing environment. The system was modified, in two stages; first, to employ a special-purpose computer to aid the control function, and second, to employ a general-purpose computer to aid in the performance of the system functions generally.

The supplementing or replacing of men by computers has been forced upon us so fast that occasionally some unease is felt about the whole process. In many ways we were neither ready, nor prepared, for this development.

There are many statements made, based on traditional beliefs, about what men can and cannot do, or about how they should and should not be used. These statements lack a basis of evidence and are quite imprecise.

It is well known that given enough time, and a problem which is not too complex or too imprecise, men can do a respectable job of solving such a problem. Exactly what the relations are between enough time, amount of complexity, and degree of precision are not known. All we have really said is that men have limitations. We also know amazingly little about how men do solve problems, make decisions, and perform in general high-level intellectual functions. We do have a set of terms for the sort of activities men engage in in some fashion when they are engaged in high-level intellectual pursuits. Some of these terms have already been used earlier, like analyze, synthesize, organize, and classify. Close inspection of such terms and their definitions is unsettling. More questions are raised than answered.

Just one example of a statement based on traditional beliefs about men, and how careful inspection of the statement led to unanswered questions and a program of research, might be helpful.

The statement: Men learn from experience. The consequence: The important decisions in systems must be made by men. The analysis:

1. How is the importance of a decision measured?

2. What do men learn from experience?

3. How does the type of experience a man has influence or determine how

much and what is learned?

4. What are the measurable characteristics of men which influence how much and what will be learned from (particular) experiences?

One cannot go to a book and get the answers to these questions. Later, a research program will be described which is intended to help answer some of these questions.

Some system designers have decided that the best way to deal with man's limitations or what man cannot do is to eliminate him insofar as possible from a system. They attempt to design completely automatic functions to be performed wholly by the computer. To the extent that any such attempt fails, which sometimes happens, man is used as backup: The function becomes "semiautomatic." It is frequently found, however, that the man can't help. To the extent that the automatic function fails, the semiautomatic function also fails. This failure of man to remedy the problem reinforces the attitude which such designers started with - that men are no darned good. This is the wrong conclusion. Where other system designers have started out to design semiautomatic functions, and have treated men with the same care they would any component, such semiautomatic functions have proven highly successful.

If one remembers that we are here discussing man-computer systems which operate in real-time with high data loads upon complex problems, one other comment is in order. The designers of a function frequently find that they have a choice as to the set of operations to use to fulfill a function. At least they have an apparent choice. However, the particular characteristics of the computer with which they must work almost inevitably reduces the number of choices actually available. A general-purpose computer is general purpose in much the same way a man is general purpose: Given enough time, it can perform a function more or less well.

Having seen men working alone fail and succeed, and men and computers working together both fail and succeed, it seemed reasonable to assume that not enough was known about either men or computers or their interaction with one another. One way of finding out more is to do research.

The research program I shall describe has two objectives. The first objective is to study the behavior of men in complex environments to find out what they can and cannot do well, and what factors limit or increase their effectiveness. The second objective is to study men as analogues of general-purpose computers, to determine how they perform complex functions, so that we can learn how to program (and maybe even design) computers to perform such functions when the real-time requirements will make use of the man impossible.

Such complex enviroments as those against which we wish to apply man are available in many real cases. Actual systems operating in actual environments could provide a laboratory for such research. However, as anyone knows who has ever tried, research to solve the problems of a large-complex man-computer system using that system as a laboratory vehicle presents difficulties. First, research is very slow, because changing anything in a system is a major enterprise. Second, research is imprecise, because changing anything in the system may have far ranging effects which were unanticipated, unsought, and very confusing. Third, the research is somewhat wasteful, because even if a particular problem is solved, the investigator is highly uncertain if the solution will ever work again. The generality of what he has found is unknown.

It seems a reasonable proposition that if the problems one encountered in working on systems could be extracted or abstracted and generalized, the appeal to research could be simplified. One always investigates some aspect of a system at a time anyway, rather than all aspects at once.

What important problems in systems can be extracted, abstracted, and generalized? Human operators in systems face two major situations over and over.

First, some have to make decisions about what to do: They have to answer the question "what do I do now?" To answer that question they have to answer a prior question, "what is the situation?" or "what is most likely to be the situation next?"

Second, for some operators there are rules like "if A then do B?" These operators have the problem of determining when A has occurred, or anticipating when A will occur. Of course, translating to a question, these operators have a job very similar to that of the former ones: "What is the situation?; etc."

It is possible to decompose general questions into more detailed questions, and this we can do for "what is the situation?" To answer this, we may need the answers to many more specific questions, like

What is it?
Where is it?
How many are there?
Where is it going?
What is it doing?
When is it going to happen again?

The operators may have to process data which contain not only the answer to the question they seek but irrelevant or incompatible data as well. The data which contain the pertinent information may be at a detailed level, so that the operator must summarize or aggregate

them, or conversely he may have to perform operations which break out details from aggregated events.

Our research program has included experiments which are concerned with processing of complex data events and those which are concerned with processing of strings or sequences of simple data events.

The Model

We assume that our human general-purpose computer has the task of obtaining the answer to a question. He receives data events which have been generated by some sources(s) and perhaps operated upon already by some other device(s). The data events may contain information relevant to the question to be answered. The relevant information in the data events may or may not be adequate to reduce the operator's uncertainty completely and may or may not contain redundant elements.

These same data events may contain noise, or irrelevant or incompatible data.

The data events may be presented all at once to the operator or may be presented over some interval of time as a sequence.

In any case, the operator must perform certain operations to obtain or produce the answer to the question. Loosely described, he must separate relevant from irrelevant data and perform coding operations on the relevant data to organize it and extract the answer to his question. This means that the operator, like an artificial automaton, can be thought of as asking a sequence of questions, performing necessary operations to obtain the answer to each as they are raised, and going on to the next until he has the answer to the final question. Also, as with the computer, neither the questions themselves nor the operations to obtain the answers to them need be in a fixed, predetermined sequence.

If the foregoing analogy holds, then the transfer function through our human general-purpose computer can be flow diagrammed, and the logical operations which must be performed can be specified and perhaps measured. It would then be possible to do two things; We could estimate how long it would take a human operator to extract the answer to a question from any data event, and we could determine how to program the human operator and arrange the data event so that we obtain optimal performance. Optimal performance, of course, may turn out to be inadequate, so we can expect to find limitations to the use of humans for many data-processing tasks.

Recapitulating, complex data events are generated by a source or sources. These data events may contain relevant information with varying amounts of redundancy. They may also contain noise or irrelevant or incompatible data. The human data processor, like a computer, performs operations upon the data event. The desired output or product is an answer to some question: Data processing is the extraction of information from data events. Figure 1 illustrated the notion of an information-to-data ratio. It can be seen that the more redundancy and noise there are in the data, the harder it is to find the information. Figure 2 shows the general model described previously. This model will be the basis for the discussion to follow.

The Research Program

Rather than accept traditional beliefs about what men can or cannot, should or should not, do, our approach is to subject these beliefs to empirical study.

What of the belief, which we have already cited, that important decisions are the responsibility of man?

Ward Edwards, at the September 1961 meeting of the Human Factors Society, gave a paper (published in the April issue of Human Factors) contending that men are not good decision makers but are good situation analyzers. He was referring specifically to the case where data are fuzzy, incomplete, or probabilistic in nature.

Edith Neimark[4] and Edwards[2] have both conducted experiments where their subjects had to make a decision. They could purchase relevant information as they wished. There are large and consistent individual differences between subjects. The subjects did not maximize to expected values, but tended somewhat in that direction. Interestingly, the subjects tended to buy too much information. That is, they paid for and obtained more information than they needed but did not use it in the best possible way in making a decision. It should be noted that all the information was relevant. Where it was not, one would expect as much or more purchase, but even less effective decision.

What have we seen? We have seen that men can be considered in some sense like general-purpose computers; they solve arithmetic and logical problems if given enough time or if the problem isn't too complex. However, when the problems are complex and time is limited, men have trouble problem solving: They do not behave like the ideal rational man. We have also seen that certain traditional beliefs about man's capabilities and limitations have tended to influence design of man-computer systems, but that these beliefs have lately been subject to inspection.

It will be remembered that a system, of the sort with which we are concerned, obtains data from and about the environment of interest. What the system can learn about the environment, or what questions it can conceivably answer about the environment, depends on the nature of

the environment itself. Additionally, a limiting factor is the nature of the data about the environment available to the system. If the data events come to the system in strings or as a sequence, predicting or controlling the environment requires that the system find the order or pattern of the environmental events.

An environment can have two different kinds of order or pattern. One, for all practical purposes, can be thought of as determined or completely predictable. This simply means that the probability of any deviation from some rule is very low. Many events are of this kind. For instance, an overwhelming percentage of the time one can predict that a depression of a light switch will extinguish an electric light, or that when a letter 'Q' is seen in the English language that it will be followed by a letter 'U'. The general description of this kind of event is a very high conditional relation between some two events such that if A, then, effectively, always B.

The second kind of order or pattern in an environment is stochastic order. This kind of order is common in environments and can be troublesome. The old coin flipping problem is a good illustration of a simple case of this kind of order. If the coin is unbiased, a sequence of such events will tend to contain an equal number of head and tail events. If the coin is biased, a sequence of such events will tend to contain the numbers of heads and tails representing the amount of the bias. If the coin tossing mechanism is biased, or if coins with different amounts of bias are systematically rotated in the order of tosses, one can expect to get probabilistic contingencies between successive events. The differences between these cases are important.

If two events are each equally probable for all reasons, any prediction one can make will lead to chance success.

If two events are not equally probable, but the only source of order is in the event frequencies, then the best strategy is to predict always the most frequent event.

However, if there is some source of order in addition to the event frequencies, a much more complicated situation obtains: It is necessary to identify the contingent relations and adopt a more complex optimal strategy. Let us illustrate the difference in the latter two cases.

Figure 3 shows two generators, both resulting in sequences having .5 A events as the over-all event frequency. The generator shown in figure 3b has an additional property: Redundancy of no mean amounts between odd and even draws.

If an analyzer were presented with a sequence from one of these two generators, it would have two questions to answer: First, what characteristics does the generator have, and second, what strategy should be employed to optimize predictions?

If the analyzer were sensitive only to event frequencies, the first question for both generators would be answered that the generator is producing an equal number of A and B events.

If, on the other hand, the analyzer were able to find additional redundancy, it would report quite a different answer to the first question with respect to the sequence generated from b.

We haven't yet said anything about the answer to the second question, about strategies to optimize predictions. Remember that our analyzer is a human computer. It can be shown that the best strategy is to always pick A on odd draws and B on even draws. Does he optimize?

Quite simply, the answer is no. The human computer tends to find the pattern in sequences generated from sources like 3b, but to match the characteristics of the source. That is, he comes to predict A on odd draws about .7 times, and on even draws about .3 times. Now, it can be shown that by chance .7 of the A events he predicts on odd draws will be A's, and .7 of the B events he predicts on odd draws will be A's, while .3 of the A events he picks on odd draws will be B's, and .3 of the B events he picks on odd draws will be B's. Contrariwise, the same for even draws. Thus matching the generator 3b will result in, on the average, accurate prediction of .7 A events on odd draws and .7 B events on even draws. This is substantially better than an analyzer which determines over-all event frequencies only, where the prediction success would be .5 for A and B events on both odd and even draws. However, it can also be shown that the variance is not at a minimum when a matching strategy is employed for prediction. That is,for a given sample sequence from generator 3b, predictive success can be much worse by chance than .7. Variance is reduced to a minmum by always predicting A on the odd draw, and never predicting A on the even draw for sequences from generator 3b. If this be so, why doesn't our human computer employ the better strategy? The answer seems to be--and this statement does not come easily from a hard-headed behaviorist--that our human computer is a teleo-logician. What does this mean? It means, first, that humans have goals. In this case the goal is to approach perfection insosfar as possible. It means, second, that humans seek order or pattern in the world: they sometimes try to find the rules which govern the generation of events.

You will note that it is only after we have been able to describe completely the nature of the generator that it is possible to state the optimal strategy, and that in our example, we knew the characteristics of the generator from the outset. The human computer does not know the nature of the generator in an overwhelming number of cases with which he must concern himself. The important point here is that adopting an optimal strategy and discovering the characteristics of a generator

concurrently are incompatible and logically impossible operations. Once one decides to optimize, any additional unfound source of order won't matter. If one seeks more order, and expects to find it, it is ridiculous to consider any strategy optimal prior to finding that order.

Two uncertainties tend to keep human computers from settling on an optimal strategy unless the order seems completely predictable. The first uncertainty is whether the generator is stationary. That is, the rules by which the generator operates may change. The second uncertainty is whether there are additional sources of redundancy which the human computer has not yet been able to find. Generator 3b has only digram redundancy, or event pairs relationship, but a generator may have higher order, or N-gram relationships which would improve prediction, or even make the sequence completely predictable. An example of a very difficult case would be a random sequence of two kinds of events ten events long which then repeated itself. The problem in some sense is similar to learning to spell a long word, or even worse, to recognize variants as being the same long word when one's students are the generators.

A generator, then, can have many sources of order, or redundancy, such as a language has: The patterning may be very complex and the relations subtle. The order may be either stochastic in nature, or completely predictable, or some of each (such as U following Q but A, L, R, . . . following B). What appears to be stochastic order at one level of organization can become determine order at a higher level of organization (houses in a tract all being constructed in different orders and sequences all end up looking identical).

It should be clear from the above discussion that what at a very detailed level may be considered as a simple binary choice--that is, an event will or won't occur, or one or another of two events will occur--may turn out to be anything but a simple binary choice when strings of such events have been appropriately grouped into subsets. It is less obvious, but still true, that the entities which are produced by organizing lower order elements are recognizable as entities, or specific ordered sets, and not simply as collections of elements. These higher-order entities are manipulated, not the elements of which they are composed.

In addition to inventing generators which produce strings of events having various kinds of stochastic and determined order, which can be thought of as artificial languages, we have used English language prose to get at some of the same phenomena.

The technique we have used was one developed by Shannon.[5] Our human computer is asked to guess his way, letter by letter, through a passage of English prose. He is told after each guess whether the letter he guessed is right or wrong. He has a twenty-seven letter alphabet, A to Z plus space. Shannon used this technique to measure average redundancy in English. His analysis was made at the letter level. That is, he measured the average number of guesses for the nth letter, given n minus one prior letters. Our interest, and therefore our use of the technique, is different. As the previous discussion indicates, we are interested in organization through levels. Specifically, we wish to know what characteristics of the organization of data events influence redundancy or the ability of our human computers to predict and how our human computers go about the job of reorganizing and predicting as they accumulate data.

Table I shows three sequential arrangements of eleven words taken from the five hundred most frequently used words in the English language (actually it is possible to devise about two dozen sentences using these eleven words, and fifteen different sentence variants were given to five subjects each). As we go from top to bottom in our table, from variant to variant of the word sequences, it can be seen that the sentence is decomposed; that is, less and less integrated, or organized.

Our human computers guess their way through the highly organized variant 1 with an average of 121 guesses. An average of 173 guesses were required to get through variant 2. An average of 233 guesses were needed to get through variant 3. (Actually variant 3 was presented as a list of words, not a sentence). There was no overlap in scores between these two groups. Table II shows the data for five subjects each on these three variants.

The sentences (or list for variant 3) have 53 character positions. Actually, 10 of the character positions were completely redundant in all cases for all subjects. For example, once RUNNI was guessed, no more than one guess was needed to complete RUNNING (space).

Now, it may look like this was a very clever experiment with a clean hypothesis. Actually, the hypothesis that word arrangement would influence uncertainty was a good guess, but how it would do so turned out to be complicated.

The identification of exactly what variables contribute to "organization" is a difficult problem. However, the data show one distinct feature (which does nothing to answer the question). Whatever contributes to organization, a sequence which is well organized results in rapid early decrease in uncertainty, whereas a poorly organized arrangement shows either linear or uneven decrease in uncertainty.

Our fastest subject guessed his way through the 43 character positions in 94 guesses, or an average of just under 2.2 guesses per letter. Subject 5, guessing his way through the list of poorly arranged words in variant 3 of Table II, took 258 guesses, or an average of exactly 6

guesses per letter. It should be remembered that in both cases 10 character positions required only one guess each.

These data have permitted many other analyses not discussed here, but I believe the point is clear. Data processing by a human general-purpose computer is clearly a function of the organization characteristics of the data to be processed. This seems to be true because the human general-purpose computer is searching for organization in the data and uses data-processing techniques which yield the best results when the data are highly organized.

Can a computer help a man who is attempting to predict events where the organization or pattern is not known to the man? The question should perhaps be modified to how rather than whether the computer can help. To use the computer simply as a data transducer may not help and may impair the man's performance. If the computer can be used to organize the data, there is little doubt that computers can contribute mightily to the intellectual pursuits of man.

A Few Words about Automation of Research

The data we have obtained to date on both artificial and natural languages were obtained and processed by manual methods, or blood, sweat, and tears.

To obtain the data from 65 subjects for the sixteen word arrangements described above required an experimenter to spend 12 to sixty five minutes on each subject individually. Data reduction and analysis took weeks.

Since preparation of this paper began, our Systems Simulation Research Laboratory has become operational for human data processing research We currently are collecting data from six subjects simultaneously. Shortly the capacity will be twenty four at once. This means that in three hours on the computer, we will increase our data-collection capacity over that of a man 120 times. Weeks of data reduction and analysis is reduced to minutes. In the intellectual pursuit called research there is no question that a computer can help significantly.

The key term for describing our laboratory is "general purpose." The computer, a Philco Transac, is a general-purpose computer. We program the computer with a general-purpose procedure-oriented language called JOVIAL[6] which is a first cousin to Algol. The programs we write are highly parameterized and modularized so they can be used for a variety of purposes. The console equipment for communication between experimental subjects and the computer is modular, so that we can assemble a console from standard parts. Further, this equipment has been designed so that given modules are truly general purpose: the content, or meaning of displays and button actions are independent from the equipment. Like the telephone system, our system doesn't care what is talked about.

We have programs which generate, make up and distribute displays, read and interpret button insertions of our subjects, record specified data, and organize, reduce and analyze that data. The programs can be used for quite different research purposes. The attempt has been to design the laboratory on the order of a quick change artist, so that it can be made to resemble that which the investigation requires. Only one question remains: given the physical facility, do we have the intellectual facility to exploit our opportunity?

References

1. Chapman, Robert L., Kennedy, John L., Newell, Allen, and Biel, William C. The Systems Research Laboratory's Air Defense Experiments, Management Sci., 1959, 5(3), 250-269.

2. Edwards, W. Probability Learning in 1000 Trials, J. Exp. Psychol., 1961, 62, 381-390.

3. Edwards, Ward. Dynamic Decision Theory and Probabilistic Information Processing, Human Factors, 1962, 4(2), in press.

4. Neimark, Edith. Information-Gathering in Diagnostic Problem Solving: A Preliminary Report, Psychol. Record, 1961, 11, 243-248.

5. Shannon, C. E. Prediction and Entropy of Printed English, Bell Syst. Tech. J., 1951, 30, 50-64.

6. Shaw, C. J. A Programmer's Look at JOVIAL in an ALGOL Perspective, Datamation., 1961, 7(10), 46-50.

TABLE I Gradual Decomposition of a Sentence

1. The boy just got in time to the school running fast.
2. The boy just got to running fast the school in time.
3. Time the just to running got boy school fast in.

TABLE II Total Guesses For Sentences from Table I

	CONDITION		
Subj.	IX	XV	XVI
1	154	117	215
2	164	117	215
3	177	121	230
4	183	121	246
5	186	130	258
X	173	121	233

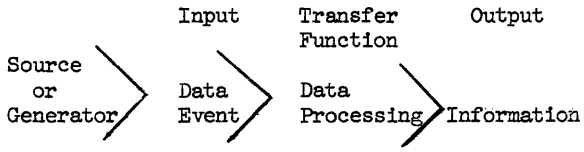
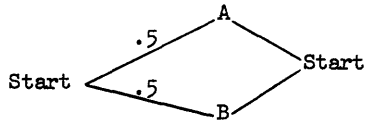
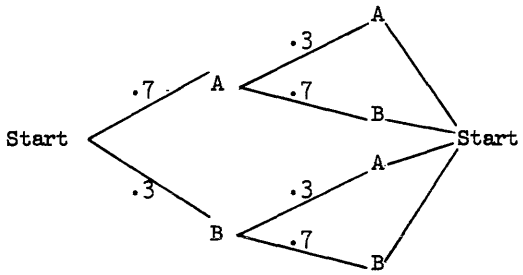


Figure 2. Model for Data Processing



a



b

Fig. 3. Two generators with common event frequencies, different redundancy.

INFORMATION

Data
Redundancy
Noise
irrelevant data
incompatible data

Figure 1. Information-to-Data Ratio

PROBLEMS IN THE STUDY OF THE NERVOUS SYSTEM

Belmont G. Farley
Staff Member, Digital Computer Group
Lincoln Laboratory*, Massachusetts Institute of Technology
Lexington, Massachusetts

This paper is a survey of the main experimental and theoretical difficulties encountered in the study of the nervous system. These difficulties are illustrated by examples of the uncertainties still existing in knowledge of the behavior of neurons, both individually and in groups, and in the interpretation of experimental observations. Concepts of the reduction of data from electrophysiological experiments are discussed and compared with those in physical experiments.

Introduction

The nervous systems of human beings and lower animals provide the organization which enables them to survive, propagate, and to produce the astonishing variety of behavior which we know. To understand this organization is perhaps the most interesting and important problem faced by science today. Little progress, however, has been made toward such understanding, and the main reason for this state of ignorance is the unique difficulty of the problem itself. It is the purpose of this paper to analyze briefly the nature of the difficulties, and to show how automatic computers, both analog and digital, are among the indispensable tools for research on nervous function.

For the purposes of this discussion, we will divide the study into three areas, namely experiment, data processing, and model construction and calculation. Since no one, least of all the writer, could hope to cover these areas in any completeness, the discussion will be limited to comparatively few points most likely to serve the purposes of the present session.

Experiment

The experimental problem comes first in the discussion of any scientific investigation. In study of the nervous system, this is a very broad field, covering the range of animals from very simple to very complex, and the range of experimental scope from the cell (or even smaller) to total behavior.

A few words concerning the neuron itself are in order here, principally to point out that some rather generally held assumptions about it may not coincide with the facts. This example will also serve as a reminder that there are still considerable uncertainties about functional behavior even at the neuron and "inter-neuron" level. It has been rather widely assumed, particularly in engineering circles, that neuron action is all-or-none, and that its function is represented satisfactorily by a simple logical element. Indeed, reference to the engineering literature discloses numerous examples of simple logical devices which have been given the appellation "neuron", without qualification or explanation. This is not well supported by experiment, but wishful thinking may carry the unwary even further. "Since the neuron is a simple logical element, the nervous system must be a logic system somewhat on the order of a digital computer with the neuron as the fundamental logical decision unit" represents a not untypical chain of reasoning, explicit or implicit. Now, as a matter of fact, no one can disprove that the neuron is a fundamental logical unit by incontrovertible experimental evidence. However, this castle is built on very flimsy foundations, because there is no experimental evidence which supports it either, and there is much which is hard to explain on such a basis. It is clear that the neuron is much more complicated than a simple gate. The all-or-none action is approximated quite well by the cell body and main axonal processes (although time is not quantized), and in this case both experiment and theory are reasonably well developed. However, neurons also may possess many fine axonal endings and many processes emanating from the cell body called dendrites. Unfortunately, it is not known whether these parts of the cell support all-or-none activity or not. In the case of the dendrites there is not even agreement on whether they conduct toward or away from the cell body in normal function in the central nervous system, but they apparently have time constants of 10 - 100 times the cell body. Furthermore it is quite possible that the dendrites can influence one another directly without the intermediary of action potentials of cell body and axon. Thus, the precise nature of the action and the influence of one neuron upon another in the central nervous system is still a matter of doubt. The main experimental difficulties here are due to the neuron's small size, high packing density, and high susceptibility to damage.

To assume that the complex non-linear biochemical system we call the neuron is a simple logical element is thus to go considerably beyond the evidence. To go further, and assume it plays the role of a fundamental decision element in a logical system of the type ordinarily studied in symbolic or computer logic is even more daring. It would not be at all surprising to find analog or quasi-analog decision systems in the brain. After all, many of the variables it must handle are analog, and digital realizations of analog functions are often clumsy and inefficient.

*Operated with support from the U.S. Army, Navy, and Air Force.

Even if computer or symbolic logic is a useful analogy to brain function, the digital computer itself furnishes an analogy which illustrates the extrapolative nature of the assumption. The logical structure of a computer is, of course, independent of its hardware realization, and the "logical" designer may know little or nothing of the circuits and components used to implement his equations or logical schematic diagrams. He deals mainly with basic logical building blocks such as "gates." The assumption about the neuron's role which we have been discussing amounts to considering the neuron as one of these basic logical building blocks. However, the neuron could instead correspond to some part of a particular "hardware" realization, perhaps only a small part. Such a discrepancy between assumption and reality can lead to serious confusion in interpretation of experimental data, and later in theory construction. At this early stage in the study of the nervous system, it is clearly important to keep the tentative nature of the ideas in mind, and to remember that absence of contrary evidence is not equivalent to supporting evidence.

Most of the work on the central nervous system has been done on animals under some degree of anesthesia. Only in the past few years have techniques been developed for normal animals, but we will proceed immediately to consideration of the study of the normally behaving animal, since it is here that the experimental problems must be faced in their full strength. Here we will content ourselves with emphasizing the number of skills and disciplines in which a successful experimenter must be expert. As a specific example, we might consider an experiment in which known stimuli are given to an animal behaving as naturally as possible. Electrophysiological signals and behavioral measures are to be recorded in an attempt to relate the stimuli, responses, behavior, and ultimately, brain function to one another. In order that the animal be as unrestrained and as comfortable as possible, it is desirable that the brain signals be transmitted without wires to a recorder by a transmitter small enough so that the animal is essentially unencumbered. The range of techniques required for such an experiment is obviously tremendous, and each technique must be pushed to the limit. First, it is hardly necessary to mention here the purely engineering problems. Stimuli must be very carefully controlled, so that stimulus generators and transducers must be precisely calibrated. Electrophysiological signals are very small, ranging as low as a few micro-volts. Amplifying and transmitting such signals by means of packaging small and light enough not to inconvenience a cat, for example, presents obvious engineering challenges.

Needless to say, preparation of the animal further requires knowledge and techniques of animal husbandry, anesthesiology, sterile brain surgery, physiology, anatomy, mechanical engineering of probes and holders, bio-chemistry, animal psychology, and a host of others almost too numerous to mention. Total time of such an experiment will never fall below many hours, and may often extend for days or weeks. A high percentage of complete or partial failures must also be expected, particularly in the earlier stages. Since it is necessary to repeat experiments a number of times before a given result can be considered reliable, it is not unusual for a single experiment to require thirty or even fifty preparations. It should be clear that a highly trained and experienced experimenter or group of experimenters is required.

In advanced experiments it will be necessary to allow the electrophysiological and behavioral state of the animal, and the response to the stimulus, to determine collectively a new stimulus. As a very simple example, when the animal becomes habituated to a given stimulus, it may be desired to change the stimulus. As experiments of this type progress, they will become more and more complex. Thus, a complex automatic control system will be necessary, and it is at this point that we encounter the first necessity for automatic computers, both analog and digital. Very little work of this sort has been carried out as yet, although preparations are going ahead at more than one research center, and it can be expected to grow rapidly. Here there exist many problems of design and use of computers which have hardly been touched.

Data Processing

If the experimenter should be successful in his attempt to get reproducible data, he is likely to be faced immediately by the second problem area - what to do with the data, of which there is often so much, and of such variability and complexity as to baffle the observer.

In the case of our cat, again, it is found that the cortical response to a simple stimulus such as an auditory click has very different forms at different times. The behavioral and electrophysiological state of the animal at these times may be impressionistically described in a rough way as "alert", "drowsy", "light sleep", "deep sleep", etc., and most of such work has in the past had to depend on such estimates by the experienced observer. What this observer actually does is to attempt to specify "states" by the appearance of the animal, his electroencephalogram, muscle potentials, and any other information at hand. That is, he classifies the data in some "meaningful" way. (The "classification" may of course sometimes result in its limiting form - a continuous function.) It has been necessary in the past for the observer to use his own brain to effect this classification, for lack of any other technique. Of course, the human brain is the best instrument for such classification and is likely to remain so for a long time.

Note that this classificatory brain function is also used in the simpler sciences, such as physics. However, its use has been so commonplace that it is scarcely conscious, both because the interesting properties of the experimental data are usually obvious, and because it is generally so easy to control the condition of the

experiment. It is of course the brain of the physical experimenter which tells him that his apparatus is in proper adjustment, that conditions are constant, and in short that his experiment is producing meaningful data. He has learned this through long experience in the laboratory, (usually expressed by saying "he has good technique") but just how he knows these things is never precisely written out, and indeed, it would be very difficult to do so. Unfortunately it is formidable difficulties of just this nature that must be overcome in neurophysiology (and in behavioral science in general).

Although the brain is so good at classification, it also has serious limitations and disadvantages. One observer cannot tell another precisely "what" he sees (or thinks he sees). Even worse, he cannot be sure he sees the same "thing" today that he saw last month, or even yesterday. There is also the well-known tendency to see what one hopes or expects to see. These disadvantages are, of course, usually described by the term "non-objective," as applied to human perception. In physics, these disadvantages of the brain have been minimized by choice of problems which are so clear-cut that the very existance of the difficulty can be ignored except by philosophers. (It should be mentioned that it is beginning to be necessary to deal with such problems even in physics. The precise specification of a "significant" cloud chamber track configuration is an example. It is safe to predict that physics will be forced into this area gradually as the clear-cut problems are all used up.) However, the scientist who hopes to connect brain function and behavior must face the problem of objective classification at the start, because the nature of his study demands it.

Now it is clear that there is no question of suddenly replacing the human brain in all classification by a precisely specified process or processes. To do so would presuppose a solution to the very problem under investigation. Although a considerable amount of work has been done in the past few years on processes to carry out some form of learned perception, it must be admitted that existing devices, interesting though they are, show a negligible ratio of performance when compared to that of the human brain. Nevertheless, work to date at least shows that fruitful attack on the problem is possible, and we need not despair in the face of the difficulty and conclude that an objective science of brain and behavior is not possible, as so many have done in the past. It is not impossible; it is just excruciatingly difficult.

At this point it should be obvious that the automatic computer must again be called on to play a part. This tool, used properly, makes possible the development and use of the complex, precisely defined processes necessary to classify the tremendous quantities of data, as well as to perform more conventional statistical processing.

The writer and several colleagues developed a rudimentary classification process for brain waves some time ago, using the Lincoln TX-0 computer. This work convinced us that the type of program outlined above was feasible, but it is clear that there is no "royal road", and that many improvements are required in techniques of data gathering, improvement of computer capacity and use, as well as in classification procedures, before computer processing can make substantial inroads on the problem. A considerable change in instrumentation and viewpoint will also be necessary. In the meantime, we will have to use our heads and computers both.

Model Construction and Calculation

We are now ready to confront the third major difficulty. So far we have discussed the problems involved in getting data, and in processing it in meaningful ways. Suppose that we now have a well-developed experimental technique, and have been fortunate enough to have discovered repeatabilities in the data which correlate with controllable or distinguishable states of the animals. That is to say we have solved the first two problems. We would now like to construct a theory (or mathematical model) to describe the experiment, which satisfies all the known "boundary conditions" of physiology, reproduces our final experimental results to a reasonable quantitative precision, and predicts the results of new feasible experiments. In other words our theory should satisfy the requirements of any good scientific theory. This theoretical area is in the most unsatisfactory state at present. Good theories are scarce except perhaps in the case of axon behavior. It might not be too much to say that in the main area of interest - non-trivial functional organization of the nervous system - there is not a single theory which meets the criteria mentioned above. There are a number of reasons for this. In part it is because of the uncertain nature of the data itself which we have emphasized. It is important to realize that the study of the more involved functions of the nervous system is still in an early stage, and since the functions are so complex, it takes some time, often years, to establish the existence and nature of particular effects. In other words, in many cases, the experimental and processing problems are not yet in good enough shape to proceed with theory. Even in cases where those problems are under reasonably good control, the complexity of the boundary conditions and data is so great that it has not been possible to construct sufficiently descriptive models. Finally, even when a possible model can be conceived, it is almost invariably too complex (and too non-linear) for predictive claculations by the tools of mathematical analysis. Here, again, it is natural to turn to computing aids, analog or digital, to study the behavior of theoretical models.

We might first inquire if these tools could aid in finding or constructing theoretical models in the first place. It is conceivable that they might, since the problem is somewhat akin to constructing a proof of a theorem. However, workers in "artificial intelligence" have not interested

themselves in this area. Although it is of such importance to science, there appears to be little or no work along these lines. In any case, this problem appears to be so much more difficult of solution than the classification problem that no useful results in our complex area are to be expected for a long time. The most we can hope for in the near future is computational, processing, and display aids to the human model maker.

However, if reasonable precisely stated models can be obtained by any means, computers can be used to calculate the outcome from appropriate assumptions, no matter how complex or non-linear the model. The only questions that arise refer to size and speed of computer. Here there is such a variety of possible models that they cover the full capacity range of present-day computers, the more complex models taxing even the largest facilities. The "neuron-like" network model being investigated by the writer requires essentially the full capabilities of the Lincoln TX-2, and the model is a very small system compared to the brain. There are many other models of interest, however, which require much smaller computing facilities.

Discussion

We have seen how involved are the problems faced by the investigator of the nervous system, and how distant is the goal of understanding the functional organization of the brain. It might be worth while to emphasize again that the attack on the problem requires skills commensurate with the difficulties. It is true that in this field no one can hope to acquire the full breadth of knowledge and techniques in complete detail any more than one can have a full command of physics or engineering; but contribution to understanding of brain function requires a knowledge of the basic background of the science, commencing with an appreciation of actual experiments in neurophysiology. This implies a commitment for a period of years with no more room for dilletantism than in any other science. For those interested, a short reading list is appended; a glance through the three volumes of the Neurophysiology Handbook alone should convey the magnitude of the required background.

Automatic computers can be expected to play an increasingly important role as tools in all three major areas of the study. They must not be considered as golden keys which unlock any door; they are no substitute for hard work, but are indispensible because of the nature of the problem.

Most existing computers and computer installations are less than appropriate to our purpose. A "family" of machines containing analog as well as digital features are needed, with their designs specifically oriented toward the problems we have discussed, and integrated organizationally into the brain research laboratory. Only in this way will these problems be approached with maximum efficiency; and the difficulties are so great that reduced efficiency can mean years added to the periods between major advances. It is apparent that large problems of design, organization, support, and training must be faced, but that is inherent in the nature of the science, and the game is definitely worth the candle.

Suggested Reading

1. Adrian, E. D., F. Bremer, and H. H. Jasper, Eds., Brain Mechanisms and Consciousness, Charles C. Thomas, Springfield, 1954.

2. Brazier, M. A. B., The Electrical Activity of the Nervous System, MacMillan, New York, 1953.

3. Bullock, T. H., "Neuron Doctrine and Electrophysiology," Science, 129, 997-1002, 1959.

4. Clark, W. A. Jr., "Average Response Computer (ARC-1)," Quarterly Progress Report, Research Laboratory of Electronics, MIT, 114-117, April 15, 1958.

5. Communications Biophysics Group, Processing Neuroelectric Data, MIT, Research Laboratory of Electronics, Technical Report 351, 1959.

6. Delisle Burns, B., The Mammalian Cerebral Cortex, Williams & Wilkins Co., Baltimore, 1958.

7. Eccles, J. C., The Physiology of Nerve Cells, The Johns Hopkins Press, Baltimore, 1957.

8. Farley, B. G., "Self-Organizing Models for Learned Perception," Self-Organizing Systems, Pergamon Press.

9. Farley, B. G., "Some Results of Computer Simulation of Neuron-Like Nets," Federation Proceedings, Symposium on Computer Techniques in Physiology, 21, 92-96, 1962.

10. Farley, B. G., and W. A. Clark, "Activity in Networks of Neuron-Like Elements," Information Theory, Fourth London Symposium, Butterworths, London.

11. Farley, B. G., L. S. Frishkopf, W. A. Clark, Jr., and J. T. Gilmore, "Computer Techniques for the Study of Patterns in the Electroencephalogram," IRE Transactions on Bio-Medical Electronics, BME-9, 1962, No. 1.

12. Field, J., Handbook of Physiology, Section 1: Neurophysiology, 1, 2, 3, American Physiological Society, Washington, D. C., 1959.

13. Granit, R., Receptors and Sensory Perception, Yale University Press, New Haven, 1956.

14. Harlow, H. F., and C. N. Woolsey, Eds., Biological and Biochemical Bases of Behavior, The University of Wisconsin Press, Madison, 1958.

15. Hebb, D. O., The Organization of Behavior, John Wiley, New York, 1949.

16. Magoun, H. W., The Waking Brain, Charles C. Thomas, Springfield, 1958.

17. Rosenblith, W. A., Ed., Sensory Communication, The MIT Press and John Wiley & Sons, Inc., New York, 1961.

18. Sholl, D. A., The Organization of the Cerebral Cortex, Methuen, London, 1956.

19. Stevens, S. S., Ed., Handbook of Experimental Psychology, John Wiley & Sons, New York, 1951.

20. A Symposium on Dendrites, Electroencephalography and Clinical Neurophysiology, Journal Supplement No. 10.

21. Tinbergen, N., Study of Instinct, Oxford University Press, 1951.

22. Young, J. Z., Doubt and Certainty in Science, Oxford University Press, 1957.

23. Brazier, M. A. B., Ed., "Computer Techniques in EEG Analysis," The EEG Journal, Supplement No. 20.

NEURAL ANALOGS

Leon D. Harmon

Bell Telephone Laboratories, Inc.

Murray Hill, New Jersey

Introduction

Information processing in the nervous system is receiving increasing attention from researchers in the communications sciences. Stimulating and effective liaison between neurophysiologists and engineers is apparent on several fronts and is expanding rapidly.

It is presently popular to call such interactions between biology and engineering 'Bionics', although it is not clear that much information is added by the term. Bionics is a formless sack into which a wide variety of strange bedfellows are thrown. Like Cybernetics, it represents the drawing together of many disparate fields, but actually constitutes no discipline in itself.

Basically there are two quite distinct areas of activity. In one the background and orientation of the engineering sciences are used to promote understanding of physiological systems. Here the concepts of information processing, computer operations, pulse coding, and digital and analog logic can be usefully brought to bear. A stimulating and catalyzing effect of the one discipline on the other has already occurred and is rapidly increasing. Sophisticated comprehension of nervous-system operation must ultimately depend on assistance from the communications sciences.

The other area of activity contains attempts to extract useful or at least suggestive information from biology (notably neurophysiology) to develop new approaches to communications and automata. The idea is that nature can provide cues, either from single elements or from entire systems, which can be useful to communication engineering.

Neural Analogs

One of the most prolific of these activities (certainly among the most vigorous) straddles both of the areas mentioned above. This is neural modeling. The art is not new; a simple electrical model was described over half a century ago.[1]* Simulation with chemical models[2-4] followed, and more recently, flexible electrical analogs[5-8] have appeared. Mathematical modeling, while necessarily constrained to oversimplification, has been extensive.[9-17] Some of the mathematical models have been formulated by neurophysiologists to clarify and consolidate their findings.[18-21] The advent of the high-speed digital computer has made it possible to explore small network behavior,[22-25] although once again it has been found necessary to simplify greatly.

Two quite different kinds of neural modeling have resulted. It is the purpose of this paper to emphasize the differences between these two approaches, to review briefly some of the main streams of activity in neural modeling, and to show, by way of example, the results of one particular line of investigation - the work dealing with real-time electronic neural analogs.

In one category of neural modeling the intent is to investigate functions of the nervous system by simulating the parameters of the biological original closely. The analogs which follow from this philosophy may supplement physiological research in a unique way. They can provide a means of examining the possible operations of nervous structures which may not be easily explored in vivo. They can be used to test theories of neural organization, to discover functions of nerve cells not previously seen, and to investigate the information-processing properties of nervous tissue. They are valid only insofar as they contain enough modeling parameters of sufficient accuracy, and insofar as their actual and predicted behavior is in agreement with that of the living system. Very often such consonance is difficult to establish because of our incomplete knowledge of the nervous system. This is especially true of the brain itself. Perhaps our best hope at the moment is to try to understand what seem to be the simpler peripheral structures.

In the second kind of neural "modeling" the intent generally is to explore single-element logical behavior or the mass actions and self-organizing properties of ensembles of relatively simple, quasi-neural elements. The resemblance to biological neurons is superficial, the abstraction deliberately having been made gross and incomplete. Such systems may have intrinsic interest and application but should not necessarily be expected to yield physiological understanding.

Within this group, two subdivisions can be conveniently made. There are, for example, the mathematical models and computer simulations to investigate adaptive and self-organizing systems,[26-32] and the analyses and the hardware analogs to explore logics of quasi-neural elements.[33-37]

---

*The references cited throughout this paper are intended to be representative rather than exhaustive.

The principal impetus for this line of investigation derives from the reputed great flexibility and computing power of nervous systems. It has been tempting to abstract some of the properties of neurons and to investigate the network behavior of elements having similar properties. It is certainly true that networks of nonlinear elements having modifiable interconnections can produce interesting behavior. Such properties as threshold firing, temporal summation, inhibition, and refractoriness lead to complex and intriguing results. It is also probably true that in the future such systems can be made to adapt to "experience" in ways that are useful and non-trivial. However, relevance to biological computers (brains and nervous systems) is by no means established, certain oracular pronouncements notwithstanding.

Several lines of investigation should be mentioned which are related to neural modeling but which cannot be categorized as such. Observations of the complex functions of nervous systems have led to the speculation that hardware analogs may be useful to computer technology. Consequently there has been some effort directed toward the development of new devices which incorporate some of the black-box functions of neurons.[34,36,38]

It is frequently presumed that analogs of neural mechanisms provide useful novelty in the engineering sense. This has by no means been established. In the first place nature has not yet shown us new engineering tricks; threshold devices, pulse logic, inhibition, and analog memory have been around for a long time. While it is true that some device development and some systems research have been spurred by contemplation of biological facts, no really new engineering secrets have been wrested from nature; the advances have resulted from the designers' ingenuity.

Another related area has to do with the testing and modeling of information processing in biological organisms; here the frame of reference is more nearly psychophysical than physiological. Some aspect of the behavior of an organism is analyzed, and a conceptual system is constructed on the basis of this analysis. The elements of the system are formal logical operators, not neural analogs. The model is then used as a basis for prediction and understanding of additional behavior. Novel and useful applications to hardware systems sometimes follow.[39,40]

Some Considerations of Neural Functions

There is often a strong temptation for non-physiologists to take an oversimplified view of the nervous system. Frequently such views arise from limitations imposed by our analytical tools, and the restrictions are candidly admitted. More frequently, however, oversimplified abstractions of nervous structures arise from an incomplete understanding of the immensity of the problem.

Let us briefly examine some of the information that neurophysiology and neuroanatomy have provided. Consider, for example, the human visual system. Reasonable estimates place the number of photoreceptors in each eye at 1.3×10^8, the number of interconnecting elements in the retina at about 10^9, and the number of transmission lines to the brain (optic nerve) at 10^6. The entire brain contains something like 10^{10} elements. If these estimates are in serious error, they very likely are too small. For example, recent electron-microscopic examination of the frog's optic nerve raised the previously accepted estimate of fiber count from 3×10^4 to 5×10^5 as a lower limit.[41] Thus it is seen that the nervous system has many orders of magnitude more elements than our largest computers. But what about the complexity of each of these elements, the neurons?

Until about thirty years ago the neuron was thought to be a simple element. Its apparent "all-or-none" behavior led to the picture of an uncomplicated relay which fired off an electrical pulse when its input threshold was exceeded. This simplicity vanished with more knowledge. Discrete neural signals are now thought to be confined to the output (axon) portion of the cell. Furthermore, there is evidence that in many situations all-or-none responses are less significant than continuous (amplitude-variable) phenomena. Consequently, all-or-none phenomena have been relegated to a special case and, in the view of some neurophysiologists, even to a minor role.[42,43]

Both pulse and continuous activity are probably involved in the transmission of events from one neuron to another. When a cell fires, an action potential (all-or-none pulse or "spike") propagates along the axon. This pulse may be an isolated event, or it may be one of a train where the intervals are time-variable. To a first-order approximation these pulses are alike (100 mv, 0.5 msec). Propagation is an active process; each pulse is continuously regenerated as it traverses the axon.

Axons terminate at synapses, which are structurally discrete junctions between cells. With a few exceptions there appears to be no trans-synaptic pulse conduction. It seems that in most synapses a presynaptic pulse triggers a unique chemical response at the junction. Thus, an all-or-none pulse traveling along an axon dies as it reaches a synapse. Subsequently a post-synaptic potential is generated in the recipient neuron. This may occur either at a dendrite (input fiber) or at the cell body itself, depending on the site of the synaptic connection.

This new electrical event has a continuous range of intensity. It is a "graded" potential which flows slowly and with attenuation (unlike a spike), and it may last for many tens of milliseconds. Such a signal may either excite or inhibit the cell. Also, a given presynaptic fiber may either excite or inhibit at different

times, depending on the state of the receiving neuron. Firing of a neuron may depend on thousands of convergent inputs.

These and other complexities handicap comprehensive mathematical analysis of networks of neurons. For instance, there exists no analytical method for deducing the function of an anatomical structure from its geometrical or topological configuration. Also, there is some, but not much, direct evidence for synaptic change as a function of use; such evidence as does exist is incomplete and controversial. The entire subject of synaptic mechanisms is difficult and complex.[44-48] Furthermore, practically nothing is known about the mechanisms of memory and learning. With regard to network connectivity, there is little evidence for randomness. Some striking evidence indicates that the visual system, for example, is anatomically highly specified,[49,50] even at birth.[51] Finally, it is important to realize that owing to difficulties of measurement, many of the input-output relations of neurons have not been directly observed - they are inferred.

This highly condensed account of neural action has been presented to show that neural events are far from simple, and that our knowledge is by no means complete. Several recent reviews[42,43,52] give excellent and convincing accounts of the situation.

Now, what about our analogs? We must note that modeling implies abstraction which in turn often implies simplification. The descriptions of neural events given above include many subtleties, yet they omit many physiological facts. For example, despite the term "all-or-none", spike amplitude often is a variable; spike waveform and overshoot have many forms; baseline ("resting") potentials are functionally related to recent firing activity. Even the most accurate neural models ignore these and many other properties. It is generally assumed that such omissions are reasonably "safe" and that the ignored parameters lead to unimportant effects. If functionally accurate modeling is intended, such assumptions can be tolerated only so long as the behavior of the analog and the original are not divergent. In the case of quasi-neural modeling (as in the adaptive systems approach) where functional accuracy is apparently of little concern, the simplifications are extreme. Rarely, for example, are refractory recovery or the complicated synaptic transmission functions included.

Electronic Models of Neurons

Electronic neural analogs can provide a flexible means for exploring the operation of the nervous system. It is relatively easy to incorporate a large number of parameters representative of the biological cell. The model may then be used as an analog computer to examine functions that may be exceedingly difficult if not impossible to predict or to analyze fully. The studies described below take for their premise that such simulation can have validity and that continued exploration with such models can lead to a better comprehension of neural events. In most of these analogs a strong attempt was made to produce a reasonably accurate physiological representation.

One of the earliest electrical models was described by Walter[53] in 1953. An array of gas-discharge lamps was made to behave in an axon-like manner. The system was devised to demonstrate all-or-none impulse propagation, facilitation, accommodation, and inhibition.

Burns[5] introduced a thyratron model in 1955 to elucidate his electrophysiological investigations of repetitive firing in nervous tissue. He demonstrated the properties of after-discharges (sustained firing following stimulus removal) owing to interacting recovery mechanisms following conditioning stimuli. Once again the system was principally an analog of axonal mechanisms.

Meanwhile the quiet neurophysiological revolution described by Bullock[43] was taking place. The relative importance assigned to axonal transmission and simple discrete neural action was dwindling. About this same time an evolution was also taking place. Neural modeling began to come into vogue.

In 1959, Harmon[6] described a transistorized model of a neuron. In a subsequent analysis of the circuit (Harmon and Wolfe[54]), it was shown how the parameters of threshold, all-or-none output, refractory recovery, inhibition, and repetitive firing could be made to approximate those of the biological neuron.

The following year an improved solid state neural model was reported by van Bergeijk and Harmon.[55] Two experiments were described in which the "neuromimes" (as they are now called) had been used to examine modes of information processing in the peripheral nervous system. In one, the "bug-detector" of Lettvin's frog's eye[56] was modeled; in the other, dynamic range extension in the auditory system was explored.

In the same year Babcock[7] examined in considerable detail the single-unit and small-network properties of a transistorized neural analog. Accounting for a large number of the biological parameters, it was made to demonstrate a variety of interesting behavior. Included were summation, division, facilitation, counting, and frequency discrimination. In addition, a time-variable synaptic function was employed to obtain some primitive adaptive pattern recognition. This work underlies a current and quite interesting series of explorations of neural property-filtering (Babcock, et al[57]).

Küpfmüller and Jenik[8] in 1961 described a transistorized analog in which the input-output pulse relationships were analyzed in some detail. An interesting contribution of this model was

its demonstration of the use of multiple inputs to obtain addition and multiplication.

An Example of One Approach

If one desires to simulate the biological neuron closely in the hope of producing physiologically meaningful results, the black-box equivalent parameters must be carefully chosen. Having produced a model, it is then incumbent upon the investigator to establish the validity of the analog; only then is it reasonable to expect physiological significance.

In the neuromime mentioned earlier,[55] the operating parameters were made as consistent as possible with the corresponding biological values. Multiple excitatory inputs provide variable input integration characteristics to simulate gross aspects of dendritic behavior. Threshold is a time-varying function depending on the recent firing history of the unit. During firing, the model is absolutely refractory; after the output pulse, a decaying exponential return to resting threshold occurs. Inhibition can be introduced to negate excitatory influences. The model has a latency or transmission delay which varies with excitation level and firing frequency. Provision for accommodation and adaptation is readily made by simple external feedback loops. The basic circuit is relatively simple, containing five transistors, two diodes, two capacitors, and nineteen resistors.

The accuracy of the model was tested (Harmon[58]) by replicating a number of classical neurophysiological experiments. As a result of these tests, properties such as time-intensity trade, burst firing, repetitive firing, accommodation, and transient change of excitability were shown to be consistent with those of the biological neuron. In addition, several properties were described which were built in by implication and discovered later, but for which no physiologically equivalent results have yet been found. For example, there is a pulse-deletion phenomenon which is due to asynchronous firing of one unit by another; this produces unexpected non-integral ratios of input to output firing frequencies. Future testing of such predicted behavior constitutes an interesting and challenging aspect of neural modeling.

A study has been made describing in detail the use of this analog to examine the dynamic range extension of the spiral innervation of the ear (van Bergeijk[59]). In this work an attempt was made to deduce the function of a known anatomical structure; the problem concerns the ability of the auditory system to handle a dynamic range of 110 db, while a single neuron's responses span only 20-25 db. A number of physiological experiments were suggested; results agree with predictions quite well.

The model has been used to explore a possible neurological origin of the flicker-fusion phenomenon (Levinson and Harmon[60]). It was shown that a relatively simple configuration (apparently doing little violence to known anatomy and physiology) can with fair accuracy account for both physiological and psychophysical data. A prediction of a new psychophysical effect made on the basis of the model was reasonably well confirmed; this effect deals with human response to a flickering light of complex waveform.

Although our neuromimes are only models, and as such will never be more than rough approximations to the living cell, the accuracy of approximation is good enough to enable us to study several problems in neurophysiology. These investigations have been successful, not in leading to an end point, but rather in suggesting a number of new experiments, both in physiology and in psychophysics.

Conclusion

Modeling of the nervous system is expanding and is diversifying. Present efforts can be roughly divided into two schools. In one, the intent is to produce models which represent biological function as closely as possible and which we hope can be used to extend neurophysiological investigations. In the other, the idea is to explore the behavior of quasi-neural elements, to describe and analyze the behavior of devices having some of the gross functions of neurons.

Electronic analogs designed for accurate representation provide a flexible means for investigating properties of the "biological computer". It is not claimed that such models are complete or even exceedingly precise. The present state of neurophysiological knowledge itself precludes this. Rather, sufficient approximations exist to yield interesting and possibly useful functional equivalence. It would appear that neural modeling, if done realistically and checked closely by physiology and psychology, has considerable power in aiding our comprehension of nervous structure and function.

It can be shown that a considerable number of properties inherent in the living cell are present in some of these models without having been explicitly built in. One might speculate that a valid theory of neural operations could be based on a relatively simple set of explicit parameters. If so, the hope is raised that fundamental understanding of nervous system operation may be less complicated than the present mass of experimental data seems to suggest.

References

1. Herman, L., "Zur Theorie der Erregungsleitung und der electrischen Erregung", Pflug. Arch., 75, 574-590 (1899).

2. Lillie, R. S., "The Passive Iron Wire Model of Protoplasmic and Nervous Transmission and its Physiological Analogues", Biol. Rev. 11, 181-209 (1936).

3. Yamagiwa, K., "Iterative Excitability in Lillie's Nerve Model", Jap. J. Physiol., 1, 269-276 (1950).

4. Tasaki, I., and A. F. Bak, "Voltage Clamp Behavior of Iron-Nitric Acid System as Compared with that of Nerve Membrane", J. Gen. Physiol., 42, 5, 899-915 (1959).

5. Burns, B. D., "The Mechanism of After-Bursts in Cerebral Cortex", J. Physiol., 127, 168-188 (1955).

6. Harmon, L. D., "Artificial Neuron", Science, 129, 3354, 962-963 (1959).

7. Babcock, M. L., "Reorganization by Adaptive Automation", Univ. of Ill., Electrical Eng. Res. Lab., TR-1, Nonr.-1834(21), Project NR049-123, 141 pp. (1960).

8. Kupfmuller, K., and F. Jenik, "Uber die Nachrichtenverarbeitung in der Nervenzelle", Kybernetik, 1, 1, 1-6 (1961).

9. McCulloch, W. S., and W. Pitts, "A Logical Calculus Immanent in Nervous Activity", Bull. Math. Biophys., 5, 115-133 (1943).

10. Culbertson, J. T., "Hypothetical Robots and the Problem of Neuroeconomy", Rand Corp. Report P-296, 58 pp. (1952).

11. Minsky, M. L., "Theory of Neural-Analog Reinforcement Systems and its Application to the Brain-Model Problem", Ph.D. Dissertation, Princeton University, University Microfilms, Ann Arbor (1954).

12. Uttley, A. M., "The Probability of Neural Connexions", Proc. Roy. Soc., B-144, 229-240 (1955).

13. Beurle, R. L., "Properties of a Mass of Cells Capable of Regenerating Pulses", Phil. Trans. Roy. Soc. London, B-240, 55-94 (1956).

14. Kleene, S. C., "Representation of Events in Nerve Nets and Finite Automata", in Automata Studies, C. E. Shannon and J. McCarthy, Eds., 3-41, Princeton Univ. Press (1956).

15. von Neumann, J., "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components", in Automata Studies, C. E. Shannon and J. McCarthy, Eds., 43-98, Princeton Univ. Press (1956).

16. Rashevsky, N., "Mathematical Biophysics", 3rd ed., 2 vols., Dover, N. Y. (1960).

17. Jones, R. W., C. C. Li, A. U. Meyer, and R. B. Pinter, "Pulse Modulation in Physiological Systems, Phenomenological Aspects", IRE Trans. Bio-Medical Electronics, BME-8, 1, 59-67 (1961).

18. Hodgkin, A. L., and A. F. Huxley, "A Quantitative Description of Membrane Current and its Application to Conduction and Excitation in Nerve", J. Physiol., 117, 500-544 (1952).

19. Fitzhugh, R., "A Statistical Analyzer for Optic Nerve Messages", J. Gen. Physiol., 41, 4, 675-692 (1958).

20. Freygang, W. H., Jr., "Some Functions of Nerve Cells in Terms of an Equivalent Network", Proc. IRE, 47, 11, 1862-1869 (1959).

21. Rall, W., "Mathematical Model of Dendritic Neuron Electrophysiology", Biophysical Journal, (in press).

22. Rochester, N., J. H. Holland, L. H. Haibt, and W. L. Duda, "Tests on a Cell Assembly Theory of the Action of the Brain, Using a Large Digital Computer", IRE Trans. Inf. Theory, IT-2, #3, 80-93 (1956).

23. Davidson, C. H., and D. R. Smith, "Oscillation Characteristics of Large Neural Networks", paper presented to the 15th ACM National Conference, Milwaukee (1960).

24. Reiss, R. F., "The Digital Simulation of Neuro-Muscular Organisms", Behavioral Science, 5, 4, 343-358 (1960).

25. Farley, B. G., and W. A. Clark, "Activity in Networks of Neuron-Like Elements", in Proc. 4th Lond. Symp. Inf. Theory, C. Cherry, Ed., 242-251, Butterworths, London (1961).

26. Clark, W. A., and B. G. Farley, "Generalization of Pattern Recognition in a Self-Organizing System", IRE - Proc. WJCC, 86-91 (1955).

27. Perotto, P. G., "Un Nuovo Componente Elementare per Reti Logiche Donate di Alte Facolta", Elettronica, 4, 1-11 (1958).

28. Kamentsky, L. A., "Pattern and Character Recognition Systems - Picture Processing by Nets of Neuron-Like Elements", Proc. WJCC, 304-309 (1959).

29. Crichton, J. W., and J. H. Holland, "A New Method of Simulating the Central Nervous System Using an Automatic Digital Computer", Univ. of Michigan, Willow Run Labs. Rep. No. 2144-1195-M (1959).

30. Willis, D. G., "Plastic Neurons as Memory Elements", in Proc. Int. Conf. Inf. Processing, 290-298, UNESCO, Paris (1959).

31. Caianiello, E. R., "Outline of a Theory of Thought-Processes and Thinking Machines", J. Theoret. Biol., 2, 204-235 (1961).

32. Rosenblatt, F., "Principles of Neurodynamics" Cornell Aeronautical Labs., Inc., Buffalo Report #1196-G-8, 616 pp. (1961).

33. Coates, C. L., and E. A. Fisch, "Design of a Solid State Neuron Circuit", G. E. Electronics Lab. Report, R60ELS-39 (1960).

34. Crane, H. D., "Neuristor Studies", Stanford Electronics Labs. Report 1506-2, Nonr. 225(24), NR373-360, 191 pp. (1960).

35. Loebner, E. E., "Image Processing and Functional Retina Synthesis", Bionics Symposium, WADD Tech. Rep. 60-600, 309-337 (1960).

36. Widrow, B., "An Adaptive 'Adaline' Neuron Using Chemical 'Memistors'", Stanford University Electronics Lab. Report TR-1553-2 23 pp. (1960).

37. Cote, A. J., Jr., "Simulating Nerve Networks with Four-Layer Diodes", Electronics, 34, 41, 51-53 (1961).

38. Klaczko, S., unpublished studies on the feasibility of using neuron-like elements as digital computing elements in high-speed computers, Telefunken GmbH, Konstanz, Germany.

39. Reichardt, W., "Analysis of a Non-Linear Biological Filter", Nuovo Cimento, Suppl. 2, Vol. 13, series 10, 608-616 (1959).

40. Reichardt, W., "Uber das optische Auflösungsvermögen der Facettenaugen von Limulus", Kybernetik, 1, 2, 57-69 (1961).

41. Maturana, H. R., "Number of Fibres in the Optic Nerve and the Number of Ganglion Cells in the Retina of Anurans", Nature, 183, 4672, 1406-1407 (1959).

42. Bishop, G. H., "Natural History of the Nerve Impulse", Physiol. Rev., 36, 3, 376-399 (1956).

43. Bullock, T. H., "Neuron Doctrine and Electrophysiology", Science, 129, 3353, 997-1002 (1959).

44. Eccles, J. C., and A. K. McIntyre, "The Effects of Disuse and of Activity on Mammalian Spinal Reflexes", J. Physiol., 121, 492-516 (1953).

45. Fatt, P., "Biophysics of Junctional Transmission", Physiol. Rev., 34, 674-710 (1954).

46. Bullock, T. H.,"Neuronal Integrative Mechanisms", in Recent Advances in Invertebrate Physiology, B. T. Scheer, Ed., 1-20, University of Oregon Pubs., Eugene, Oregon (1957).

47. De Robertis, E., "Submicroscopic Morphology of the Synapse", Int. Rev. Cytology, 8, 61-96 (1959).

48. Grundfest, H., "Synaptic and Ephaptic Transmission", in Handbook of Physiology, Sec. I: Neurophysiology, Vol. I., J. Field, Ed., pp. 147-197, American Physiological Soc., Washington, D.C. (1959).

49. Sperry, R. W., "Mechanisms of Neural Maturation", in Handbook of Experimental Psychology, S. S. Stevens, Ed., 236-280, John Wiley and Sons, Inc., New York (1951).

50. Maturana, H. R., J. Y. Lettvin, W. S. McCulloch, and W. H. Pitts, "Anatomy and Physiology of Vision in the Frog", J. Gen. Physiol., 43, 6, Suppl. (2), 129-175 (1960).

51. Walk, R. D., E. J. Gibson, and T. J. Tighe, "Behavior of Light-and-Dark-Reared Rats on a Visual Cliff", Science, 126, 3263, 80-81 (1957).

52. Rapoport, A., and W. J. Horvath, "Information Processing in Neurones and Small Nets", WADD Report TR-60-652, AF33(616) - 6272 (1960).

53. Walter, W. G., "The Living Brain", p. 280, W. W. Norton and Co., Inc., New York (1953).

54. Harmon, L. D., and R. M. Wolfe, "An Electronic Model of a Nerve Cell", Semiconductor Products, 2, 36-40 (1959).

55. van Bergeijk, W. A., and L. D. Harmon, "What Good are Artificial Neurons?", Bionics Symposium, WADD Tech. Rep. 60-600, 395-406 (1960).

56. Lettvin, J. Y., H. R. Maturana, W. S. McCulloch, and W. H. Pitts, "What the Frog's Eye Tells the Frog's Brain", Proc. IRE, 47, 11, 1940-1951 (1959).

57. Babcock, M. L., A. Inselberg, L. Lofgren, H. von Foerster, P. Weston, and G. W. Zopf, Jr., "Some Principles of Preorganization in Self-Organizing Systems", Univ. of Illinois, Electrical Eng. Res. Lab., TR #2, Nonr, 1834(21), NR049-123, 110 pp. (1960).

58. Harmon, L. D., "Studies with Artificial Neurons, I: Properties and Functions of an Artificial Neuron", Kybernetik, 1, 3, 89-101 (1961).

59. van Bergeijk, W. A., "Studies with Artificial Neurons, II: Analog of the External Spiral Innervation of the Cochlea", Kybernetik, 1, 3, 102-107 (1961).

60. Levinson, J., and L. D. Harmon, "Studies with Artificial Neurons, III: Mechanisms of Flicker-Fusion", Kybernetik, 1, 3, 107-117 (1961).

THE CAUDAL PHOTORECEPTOR OF THE CRAYFISH: QUANTITATIVE STUDY OF RESPONSES TO INTENSITY, TEMPORAL AND WAVELENGTH VARIABLES

William R. Uttal and Hedwig Kasprzak

Thomas J. Watson Research Center

International Business Machines Corporation

Yorktown Heights, New York

Abstract

Quantitative measurements of the multicellular activity recorded with gross electrodes from the ventral nerve cord of the crayfish have been made to determine the adaptation function, the spectral sensitivity and the relation between stimulus intensity and response amplitude of the simple photoreceptor located in the sixth caudal ganglion. A digital computer analysis technique was used to discriminate between those fibers associated with the photic stimulation and other spontaneously active neurons and to perform the bookkeeping operations of peak detection, and tabular and graphic display of amplitude and interval distributions. The neural output was found to adapt considerably over a period of one minute with high activity rates (produced by high level stimuli) showing a greater percentage decrease in activity than lower rates. A constant energy spectrum was produced by means of a grating monochrometer and the luminosity function plotted by measuring the total amount of activity in a two second sample. The photosensitivity function was found to closely approximate the luminosity curve of the dark adapted human eye. The relation between stimulus intensity and response amplitude was found to be well fitted by a power function with an exponent of .53.

Introduction

Since Prosser's discovery of the photosensitivity of the sixth ventral ganglion of the crayfish, only a few investigators (Kennedy, 1958;[6] Welsh, 1934;[18] Stark and Hermann, 1961[11]) have pursued studies of this interesting sense organ. Many significant questions have not been answered as yet, and a number of important functional relationships remain undefined. For example, the specific photosensitive tissue has not been identified, nor until now has its luminosity curve been determined. Furthermore, knowledge of the neural organization remains limited and we do not know how many neurons are involved in this photoreceptor process and how they interact with other sensory systems.

One of the most perplexing problems obstructing further investigation has been the enormous amount of data which must be handled to arrive at quantitative measures of these neural phenomena. High speed digital computers provide the solution to these problems through their ability to automatically process the critical measures of thousands of sequential single cell action potentials in a few seconds.

Thus the combination of an appropriate neurological preparation of an almost ideal level of complexity and a sufficiently powerful analysis tool has led to the present work. As a result, we have been able to make quantitative measurements of some of the relationships required for an understanding of the organization of this simple photoreceptor system. The specific goals of this work were to determine the differential responsiveness of those fibers in the ventral nerve cord which are activated by light while the wavelength, intensity and temporal dimensions of the stimulus were varied. These differences were then used as a starting point to attack some of the above problems.

Unexpectedly, these measurements became extremely significant to another field of sensory investigation - human vision - because of certain similarities between our results and the earlier work of Wald and Brown (1958)[15] in photochemistry and Stevens (1961)[12] in psychophysics.

An attempt has been made to integrate the results of our experiments with those photochemical and psychophysical efforts.

General Procedure

Material and Method

Crayfish (tentatively identified as Cambarus astacus) were obtained from a commercial supplier in Wisconsin. Initially an extremely high mortality rate in storage was encountered but this problem was almost completely eliminated by keeping the animals in a refrigerator maintained at approximately 50° F. During the experiments the crayfish were kept in a hollow refrigerated tray through which water was circulated at a constant temperature of 35° F. The entire caudal nerve cord was exposed by surgical removal of the chitonous epidermis and the light source then focused on the sixth ventral ganglion--the structure containing the as yet unidentified photoreceptor tissue.

Stimulation

The caudal photoreceptor was stimulated by incadescent light sources powered by 6v A.C. voltage packs. For the temporal and intensity studies, an Ealing projection lamp housing a six volt coiled filament bulb was used. In the spectral sensitivity study, however, a Bausch and Lomb 250mm. grating monochromator was used which provided a continous spectrum originating from a ribbon filament tungsten incandescent bulb. Exit and entry slit widths were kept constant at 1 mm. to control the purity of the stimulating light. An equal energy spectrum was generated (using a calibrated solar cell to equate the energy levels) by regulating the voltage applied to the ribbon filament lamp. Because of the extremely long time constants of the preparation, the light source was arranged to be switched on and off manually.

Recording

Simple and conventional response detection techniques were used throughout the experiment. The recording electrode (of 0.005 inch platinum wire) was formed into a hook and was used at the connective between the 4th and 5th abdominal ganglion to simply lift the entire nerve cord above the remaining tissue. An alligator clip was clamped to abdominal muscle tissue to serve as an indifferent reference electrode. The signals recorded were conveyed to a Tektronix 122 preamplifier and thence to an oscilloscope, which performed the dual function of further amplifying the signal while also displaying the recorded signals for visual monitoring. The vertical output of the oscilloscope was fed into a remote power amplifier which elevated the signal voltage to a peak to peak output of 10 volts and clipped the noise below an adjustable threshold level. The output of this amplifier was fed into the data analysis unit to be described below. The entire system is shown in block diagram form in Fig. 1.

Data recording was controlled by a clockwork timer which allowed the amplified neural signal to be recorded for a specified predetermined length of time. This unit has been described in detail in another publication (Uttal and Roland, 1961[13]) and we shall only briefly describe its operation here. The amplified neural signal was digitized into a pure binary code at the command of the control logic unit, which provided the necessary sequence of control voltages. Data were digitized 12,000 times a second and written at this rate on a modified IBM 727 Magnetic Tape unit.

Data Analysis

The data recorded on the magnetic tape were then processed on an IBM 7090 computer under control of several special computer programs specifically designed for the analysis of our recordings.

The typical program performed the following three-stage operation upon the raw data:

1) The raw data were read into the computer and inspected by a peak detection program. The raw data were essentially in the form of a table showing the value of the analog voltage 12,000 times a second. The peak detection program first reduced this enormous table to a smaller one which tabulated the significant measurements of each individual spike potential. Thus, a new table was formed in which each sequential spike was numbered, its amplitude tabulated, and the time at which it occurred also indicated.

2) Two tables indicating the distribution of amplitudes and the distribution of intervals between sequential spikes were then tabulated.

3) Two histograms were automatically plotted by the computer presenting the tabular information in stage 2 in a pictorial form which was ideally suited for visual or simple statistical evaluation. Figure 2 is a sample printout of the computer plotted amplitude histogram for a highly sensitive preparation stimulated by white light. It can be seen that a relatively narrow band of activity has been generated by the stimulus which is easily distinguishable from other unrelated activity.

Our usual measure of response was to simply sum the activity within the bandwidth associated with light sensitive fibers. Because pooled interval data from many fibers rapidly approaches a random distribution as the numbers of fibers increase (as described by Cox and Smith, 1954[1]) the amplitude data are more applicable to present experiments and have been used exclusively for the determination of our results.

This pulse amplitude discrimination technique is similar to that suggested by Hichar (1959)[4] and Stark (1959).[10] The primary advantage that accrues to the researcher from the amplitude discrimination technique is the ability to distinguish between the neural activity associated with one set of neurons from those of another independent system. Because all amplitude bands are analyzed simultaneously by our computer method at high speed, the data are available without tedious rerunning of prerecorded analog signals. The application of a general purpose digital computer method to amplitude analysis of a multifiber preparation is similar to the interval analysis done on single fibers by Gerstein and Kiang (1960).[2]

Experiments and Results

The Intensity Function

In general, the responses to light stimulation recorded by our system appeared to be the responses of a small number of cells. This is in accord with the earlier observations reported by Prosser (1934)[9] and Kennedy (1957)[5] although (see discussion), we differ from Kennedy on the actual count. The coding of the intensity-response function has also been studied by Kennedy (1958).[6] His published data do not include the dynamics of the dimly illuminated region of the curve--a gap which this current experiment has attempted to fill, nor were his data presented in a mode suitable for our analysis.

Unfiltered, the illuminance of the light falling on the caudal photoreceptor was approximately 1100 ft. candles (measured by a Macbeth illuminometer). The incident illuminance was then varied by means of neutral density filters which were inserted by hand in the light pathway. After a fifteen-minute dark adaptation period, the stimuli were presented in a "constrained random order" (presentation without replacement until all items had been presented once) to avoid any timing or sequence effects from biasing our data. Sample records of 2 seconds duration were digitized and then the neural activity was allowed to return to its resting level before the next sample was taken.

Figure 3 is a graph of the pooled results from fifteen runs on thirteen different animals. The response curve exhibits a monotonically increasing function for increases in light intensity over its full range. More significantly, this function is well fitted by a straight line on a logarithmic plot but deviates considerably from a straight line on a semi-logarithmic plot. This datum will be of special significance in the discussion of the spectral effectiveness function below.

Temporal Characteristics

In the earliest reports of investigations of the caudal photoreceptor, it had been determined that the temporal characteristics of this response were of extremely long time constants. Both Prosser (1934)[9] and Welsh (1934)[18] described in qualitative terms the extreme length of the latency periods and the slow adaptation. Stark (1959)[10] further described this phenomenon by showing the low frequency characteristics of the bode plot, a graphic means of displaying the servoanalytical properties of a dynamic system.

Our computer analysis technique was used to further define some of these temporal relations. Our experiment measured the effect of stimulus intensity on the adaptation and latency of the sequence of neural signals to a continuous photic stimulus of four different intensities. Each point on Fig. 4 represents the average of 13 measurements of neural activity at that particular time. To obtain these data, one-minute records were taken starting at the onset of illumination with continuous digitization of the signal throughout that period. A special computer program was prepared which broke this one-minute record into 12 sequential five-second sections and performed on each in turn the analysis described above. Figure 4 thus shows the trend of the activity over this one-minute period. This figure clarifies some of the less quantitative comments of the authors mentioned above. Clearly the neural responses do show some adaptation ranging from a 40 per cent decrease for the highest intensity of light to what appears to be the absence of adaptation reported for lower light intensities. There is also clear evidence of the increase in latency with decreased stimulus intensities.

Spectral Effectiveness

The determination of the stimulating effectiveness of the various wave lengths of light should be based upon an equal response criterion because of the nonlinearity of biological detectors . Human luminosity curves, for example, are usually based upon the energy required for a threshold detection. However, it is also possible to determine the luminosity function of a photoreceptor in a simpler experimental paradigm using a constant-energy criterion if one knows the correction factor to be applied from the stimulus-response relationship. Since the latter relationship was available to us, we were able to implement the simpler constant-energy experiment, and we will perform this correction later in the discussion.

A Bausch and Lomb 250 mm. grating monochromator was used as the light source. Particular wavelengths were selected by hand and energy levels equated by manually controlling the voltage applied to the ribbon filament lamp. Equal energies had been calibrated for every ten millimicra between 460 and 560 millimicra

and every twenty millimicra between 400 and 460 and 560 and 640. The image of the exit slit of the monochrometer was brought to a focus on the sixth caudal ganglion. After dark adaptation, selection of the wavelength and adjustment of the energy level, the light was turned on. As soon as the latent period was completed and the response stabilized, 2 second samples were recorded on our digital data logging unit for subsequent computer analysis.

The records were then analyzed in the usual way and a plot made of the relative stimulating efficiency of the various wavelengths as shown in Fig. 5. For procedural reasons, the two sections of the curve divided by the dotted line were gathered in separate experimental sessions, but each point on the curve represents the average of 10 different measurements with each measurement carried out on a different animal. The stimulating efficiency curve can be seen to peak at approximately 500 millimicra, the same wavelength Kennedy (1957)[5] reports as the peak of the absorption curve for a microspectrophotometric measurement of a curious red colored pigment he observed in the sixth ganglion. Since the complete details of his measurement have not been published as yet, and we have not been able to observe the pigment; a more specific comparison may not be made at this time.

It can also be seen that the sensitivity of the photoreceptor is minimal above 600 millimicra. At the shorter wavelengths some sensitivity is indicated below 400 millimicra, the lowest wavelength measured in our experiment.

Discussion

As with any gross electrode recording technique there remains a question of sampling -- i.e., what is the proportion of responses which are being recorded in relation to the total number of responsive cells? It appears from our records that the responses of many different sized cells are indeed being detected. The response from photically activated fibers is a medium sized potential usually half the microvoltage of those sensory fibers activated by mechanical manipulation of the telson. These large, mechanically activated cells (apparently the giant fibers) have such high speed adaptation characteristics that their responses are present only at the onset and offset of the stimulus.

It was clear that many different sized and functioned cells were contributing, from time to time, to our recordings. Because the range of spike amplitude associated with photic stimulation is relatively narrow, we may assume that the responses of fibers slightly more distant are not being significantly attenuated, but rather that distant cells contribute in full amplitude to the pooled records. Further evidence that a small number of cells truly represent the totality of photoresponsive neurons is forthcoming from records from a split cord preparation. The connective between two caudal ganglia was split along the transverse midline and only one half was hooked with the recording electrode. As Fig. 6 shows, the preparation led to a highly periodic pulse train characteristic of a single fiber. These data have led us to conclude that there are but two active fibers conveying information about the incident light intensity on the sixth ganglion to more anterior levels of the nervous system - one on either side of the almost bilaterally symmetrical nerve cord. This conclusion suggests that Kennedy's (1957)[5] estimate of 4 or 5 photosensitive fibers may have been too high.

The question of the specific identification of these two cells has not yet been answered. However, as an initial attack upon this issue we have prepared an atlas of microphotographs of cross sections of the crayfish ventral nerve cord. A sample section is shown in Fig. 7. The wide variety of cell sizes is immediately obvious but our optical photomicrograph does not show the complete details. A recent electronmicrograph, by McAlear and his collegues (1961)[8] of the circumesophageal connective, has the advantage of greater resolution. For our experiments, however, only the fibers resolved in Fig. 7are probably significant.

The problem of identification of the photosensitive material in the receptor can also be approached by determining the spectral response curve of the receptor and comparing these data with the absorption spectrum of known extracted photochemicals. As mentioned above, the determination of the spectral effectiveness curve with an equal energy spectrum, however, does not lead to the preferred luminosity function directly because of the nonlinearity of the intensity-response relation.

The data in Fig. 3 indicating the relationship between the stimulus amplitude and response is found to be a power function of the following form:

$$\text{Response} = k\ \text{Stimulus}^{.53}$$

(For values above threshold and below saturation) where k is an arbitrary constant dependant only upon the units used. A power function of the same general form is used by S. S. Stevens (1961)[12] and his collegues to describe magnitude functions of many sense modalities in man. More specifically, the exponent of the power function for the crayfish caudal photoreceptor is very close to their measures for visual magnitude estimates of small stimuli in the dark adapated human eye. For larger stimulus patterns, the exponent is lower and we suggest that this lowering is an effect of the

reciprocal inhibitory interactions described by Kuffler (1952)[7] for the vertabrate and Hartline and Ratliff (1957)[3] for the invertabrate visual systems. Since the reciprocal interactions are minimized in the measurements we made (only a single cell was found on each side of the nerve cord) a point source experiment and our diffuse stimulus experiment are equivalent in the crayfish and we feel our results, therefore, reflect predominantly photochemical phenomena.

This analysis lends further support to Stevens' suggestion that it is the transducer which is responsible for the "compression" function rather than some central learning phenomena as proposed by Warren and his collegues (1958).[17] Our conclusions also emphasize the significant role that peripheral processing can play on perception by accounting for a complex psychophysical function exclusively in terms of the peripheral transform.

Now, since we are in possession of both the stimulus response function and the constant energy spectral sensitivity function, it is possible to correct the obtained spectral effectiveness curve and develop the desired luminosity curve by defining a new value of the response R_o called R_c, the corrected response, according to the following rule:

$$R_c = kR_o^{\frac{1}{.53}}$$

where k is an arbitrary constant. The curve showing the relationship between response and stimulus wavelengths is then in a form comparable to the absorption spectrum and psychophysical sensitivity data so elegantly used by Wald and Brown (1958)[15] to demonstrate the photochemical basis of human vision.

These corrected data were then normalized by dividing their values by the value at 500mμ, the wavelength of maximum response, and plotted superimposed (in Fig. 8) upon Wald & Brown's (1958)[15] comparison of the human rhodopsin absorption spectrum, the human luminosity curve theoretically corrected for ocular transmission and finally, the human luminosity curve experimentally determined for aphakic subjects. It can be seen that the relationship is extremely close for all four curves. A slight discrepancy exists near 460 millimicra which is similar to that found by Wald (1951)[14] in a comparison of frog rhodopsin and the human aphakic luminosity curve.

Wald and Hubbard (1957)[16] recently investigated the absorption spectrum of rhodopsin extracted from the compound eye of the lobster and, interestingly enough, their measurements differ considerably from our data even though the lobster and the crayfish are closely related decapod crustaceans.

The similarity between the crayfish spectral response and that of the human visual system, coupled with the similarity between the stimulus response power functions, suggest that there is a very similar photochemical mechanism operating in each case. However, the difference between the temporal properties of the two must not be overlooked. The difference is so great, varying from tens of milliseconds in the vertebrate eye to six or seven seconds in the caudal photoreceptor, that a significantly different mechanism is suggested for the spike potential generation process. This difference may be related to the site of action of the generator potential or some other phylogenetic difference in neural structure rather than a photochemical effect and does not, therefore, vitiate the photochemical similarities described above. Future research will be required to unravel the interacting physiological mechanisms of photochemical transduction, the generator potential and the resulting pulse encoding process.

It is clear, however, that the similarities which do exist offer the experimenter an almost novel opportunity to explore some properties of human vision, both physiological and psychological, in a highly reduced model preparation.

Summary

The dynamics of the caudal photoreceptor of the crayfish are extremely well suited to experimental inquiry because of the simplicity of the system. This study has attempted to explore certain of the functions mediated by this sensor and to integrate these results into the framework from adjacent disciplines. Progress has been made in the following areas:

1. The stimulus-intensity response-amplitude relationship was shown to fit a power function of the following form:

$$\text{Response} = K\,\text{stimulus}^{.53}$$

This is strikingly close to that recorded by other investigators for the analogous human psychophysical study.

2. The spectral sensitivity curve was determined for an equal energy stimulus spectrum and corrected by means of the stimulus response relation which we have found to be a power function. The data could then be superimposed upon equivalent values for human rhodopsin absorption and visual sensitivity with only a slight discrepancy in the shorter wavelengths.

3. A considerable difference was noted between the long time constants of adaptation and latency observed in the crayfish caudal photoreceptor and the analogous more rapid measurements in other invertebrate and vertebrate photoreceptors.

4. The nature of the responses to photic

stimulation suggested that there are only two neurons conducting coded information about light intensity from the sixth ganglion.

5. Because of the similarity in the power function relating stimulus intensity with response level and the similarity of the spectral sensitivity curve, it is suggested that there is a similar or even common photochemical mechanism underlying the response of both the human eye and the crayfish caudal photoreceptor. Because of the dissimilarity between the temporal characteristics, a different mechanism of spike action potential generation is suggested. Further research is necessary to completely analyze the interrelations between transduction, code generation and transmission.

References

1) Cox, D. R. & Smith, W. L. On the superposition of renewal processes. Biometrika, 1954, 41, 91-99.
2) Gerstein, G. L. & Kiang, N. Y-S. An approach to the quantitative analysis of electrophysiological data from single neurons. Biophysical Journal, 1960, 1, (1), 15-28.
3) Hartline, H. K. & Ratliff, F. Inhibitory interaction of receptor units in the eye of limulus. J. General Physiol., 1957, 40, 357-376.
4) Hichar, J. K. Pulse height analysis of spontaneous activity in the crayfish nervous system. Medical Electronics, Proceedings of the Second International Conference on Medical Electronics, 1959.
5) Kennedy, D. Responses of a crustacean photoreceptor. (Abstract) Federation Proc., 1957, 16, 71.
6) Kennedy, D. Responses from the crayfish caudal photoreceptor. Am. J. Opthal., 1958, 46, 19-26.
7) Kuffler, S. W. Neurons in the retina: organization, inhibition and excitation problems. Cold Spr. Harb. Symp. Quant. Biol., 1952, 17, 281-292.
8) McAlear, J. H., Camougis, G. & Thibodeau, L. F. Mapping of large areas with the electron microscope. Biophys. Biochem. Cytol., 1961, 10, 133-135.
9) Prosser, C. L. Action potentials in the nervous system of the crayfish. II. Responses to illumination of the eye and caudal ganglion. J. Cell. Comp. Physiol., 1934, 4, (3), 363-377.
10) Stark, L. Transfer function of a biological photoreceptor, Wright Air Development Center Report 59-311, 1959.
11) Stark, L. & Hermann H. T. Transfer function of a biological photoreceptor. Nature, Lond., 1961, 191, (4794), 1173-1174.
12) Stevens, S. S. The psychophysics of sensory function, in Sensory Communication (Walter A. Rosenblith, ed) 1-33, The M.I.T. Press & John Wiley & Sons, Inc., New York, 1961.
13) Uttal, W. R. & Roland P. A. A terminal device for entry of neuroelectric data into an electronic data processing machine. EEG & Clin. Neurophysiol., 1961, 13, 637-640.
14) Wald, G. The photochemical basis of rod vision. J. Opt. Soc. Amer., 1951, 41, 949-956.
15) Wald, G. & Brown, P. K. Human rhodopsin. Science, 1958, 127, 222-226.
16) Wald, G. & Hubbard, R. Visual pigment of a decapod crustacean; the lobster. Nature, Lond., 1957, 180, 278-280.
17) Warren, R. M., Sersen, E. A., & Pores, E. B. A basis for judgments of sensory intensity. Amer. J. Psychol., 1958, 71, 700-709.
18) Welsh, J. H. The caudal photoreceptor and responses of the crayfish to light. J. Cell. Comp. Physiol., 1934, 4, 379-388.

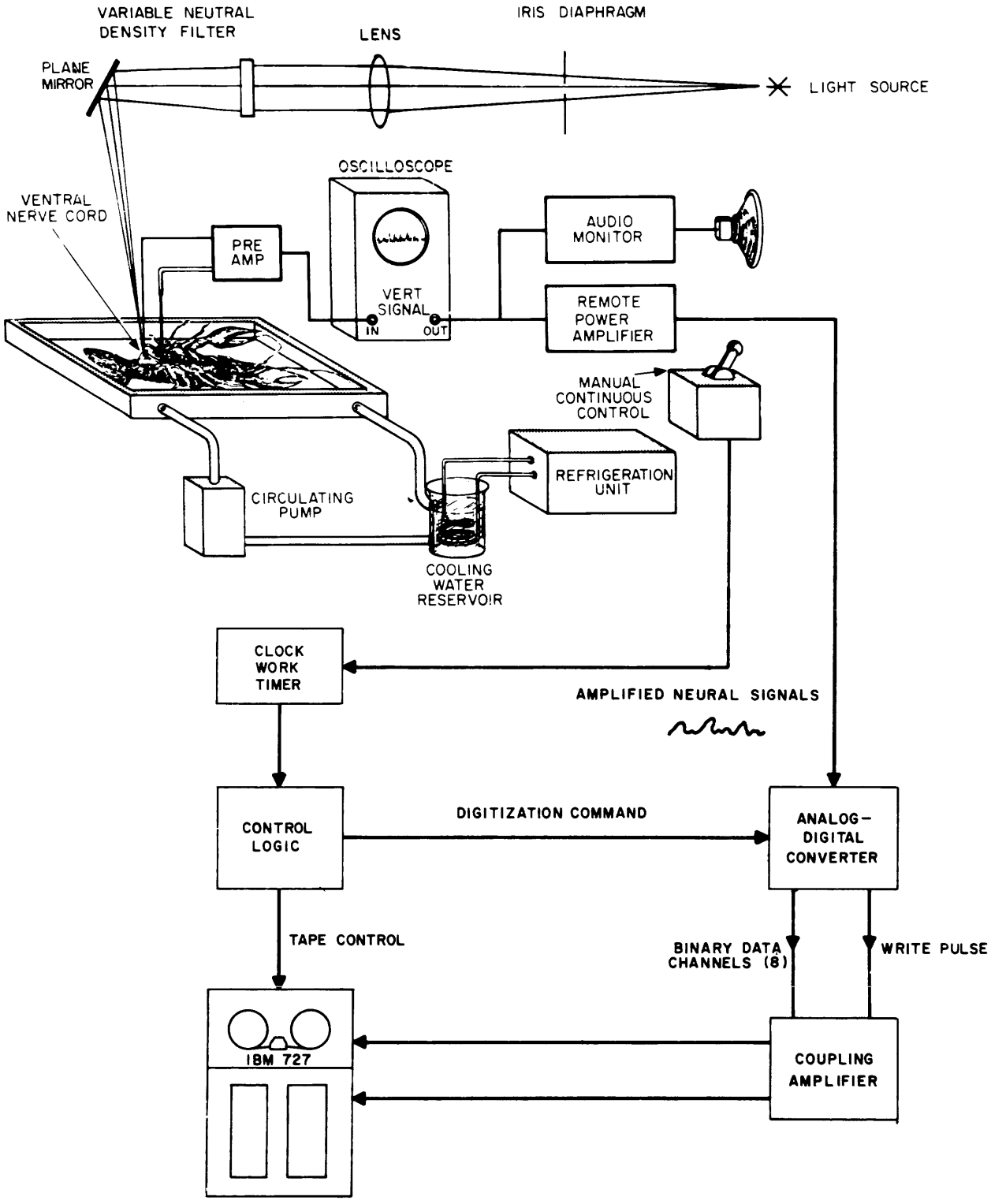


Fig. 1 A Block Diagram of the stimulating, recording and data analysis units.

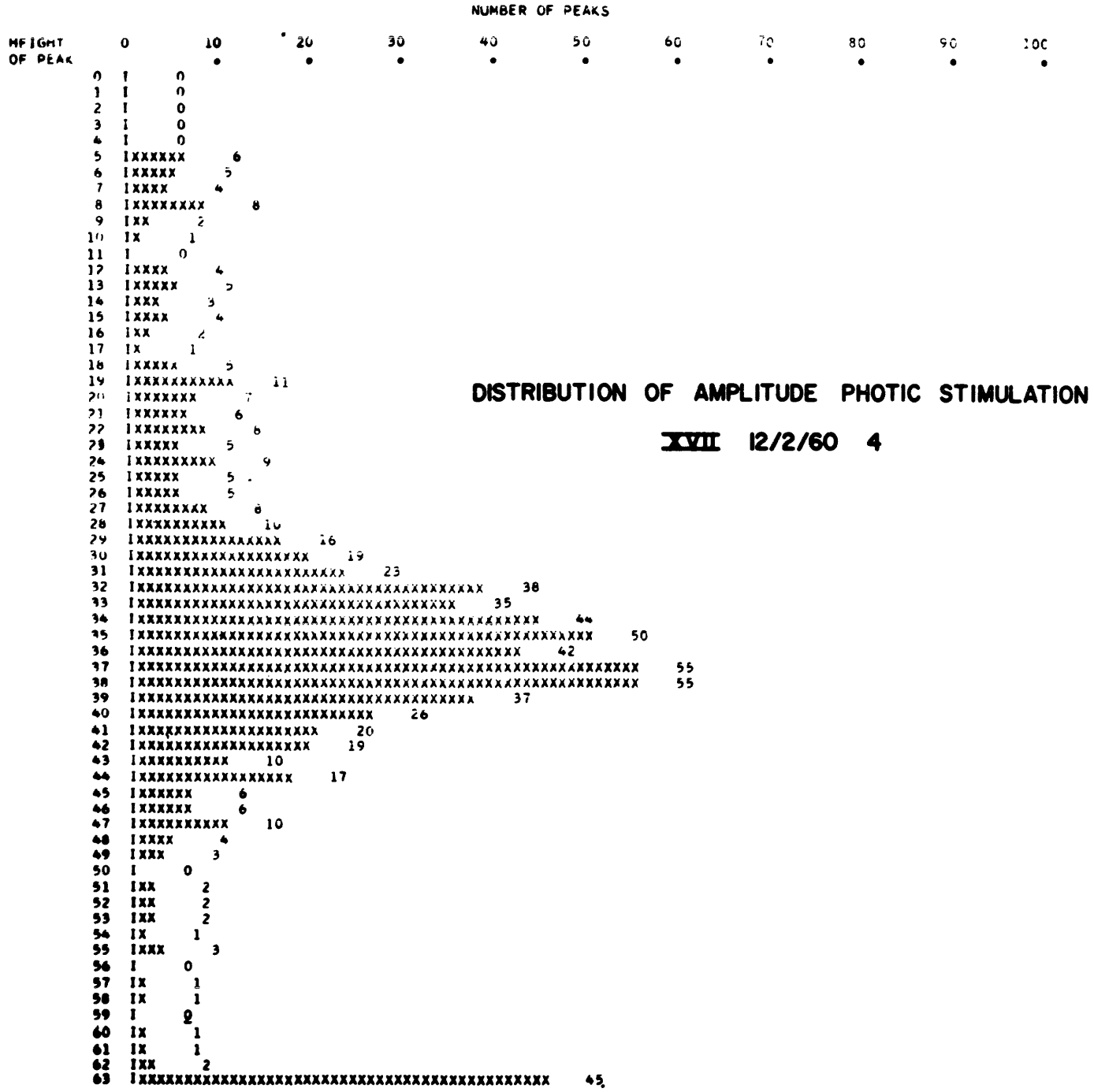


Fig. 2 A sample computer printed histogram for responses elicited by white light.

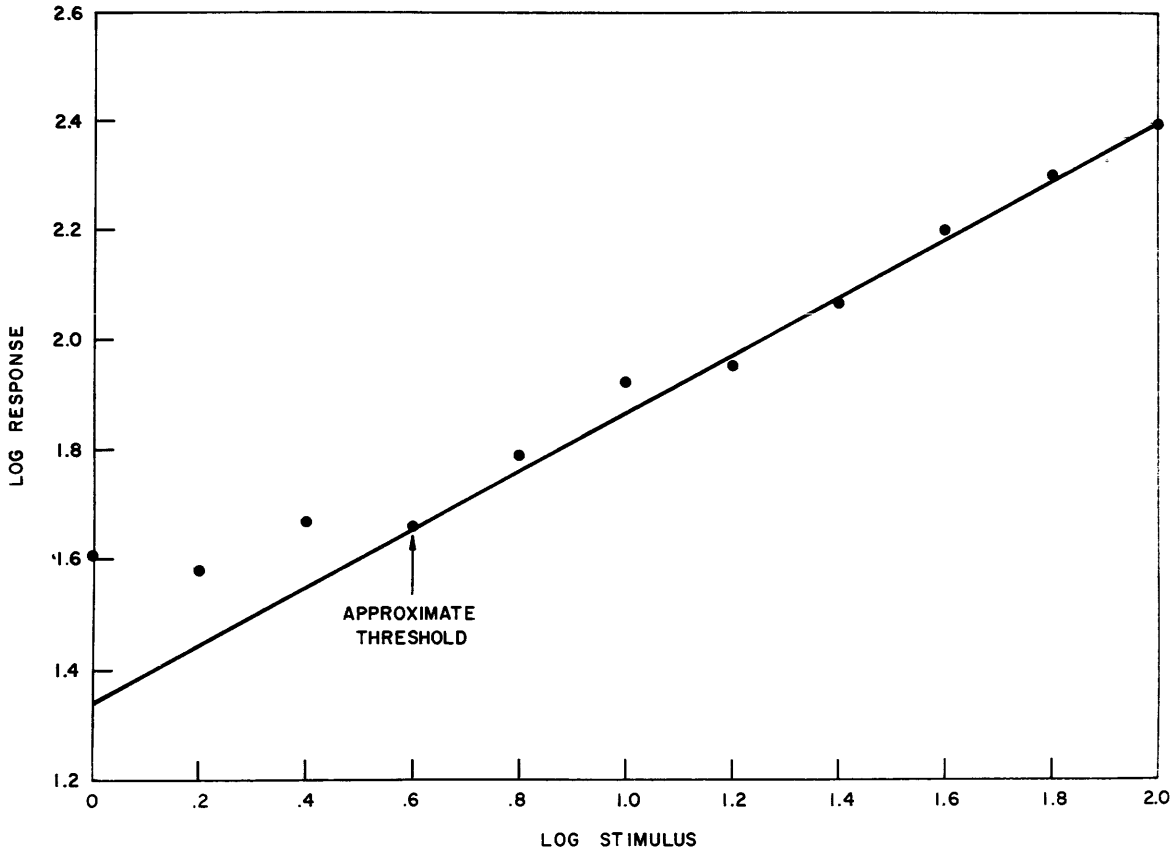


Fig. 3 A plot on logarithmic scales of the results of the stimulus response experiment showing the fit to the straight line representing:

$$\text{Response} = K\ \text{Stimulus}^{.53}.$$

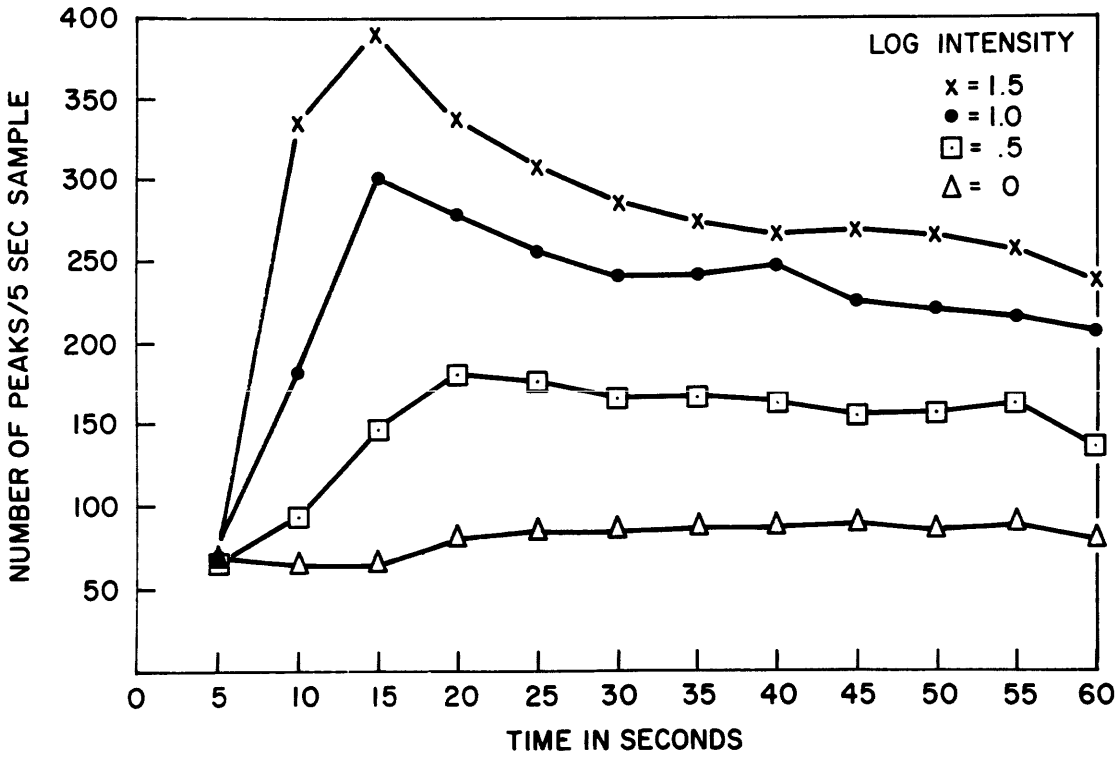


Fig. 4 Results of the adaptation experiment showing the shift in latency and

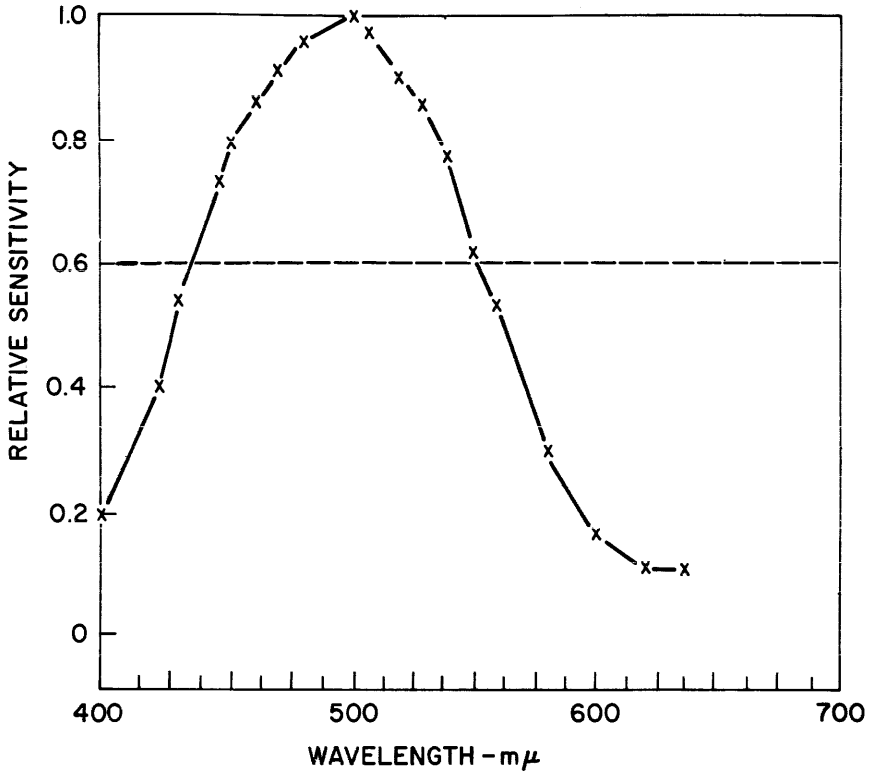


Fig. 5 The results of the spectral effectiveness experiment. Data have been normalized, but not corrected as in Fig. 8.

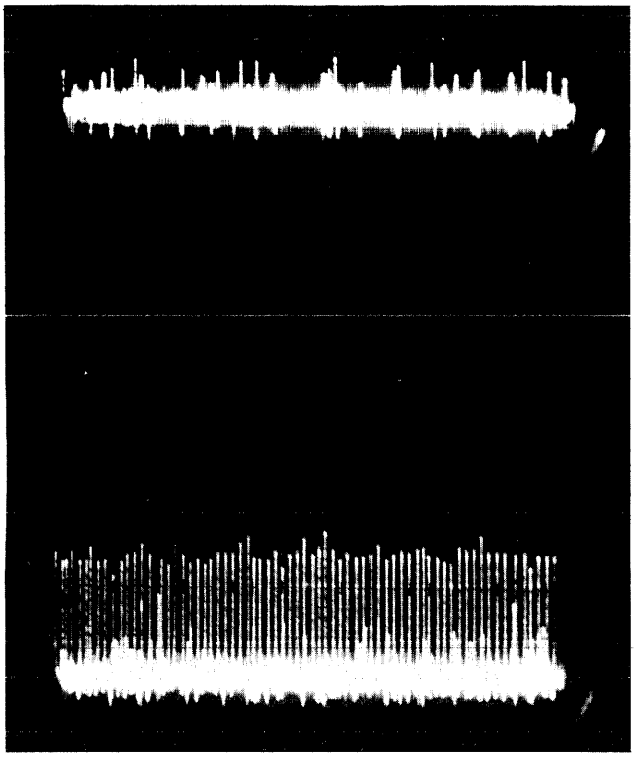


Fig. 6 An oscillograph of the responses from a split cord preparation. The upper plate shows spontaneous activity in the unstimulated preparation while the lower shows the highly regular activity produced by photic stimulation.

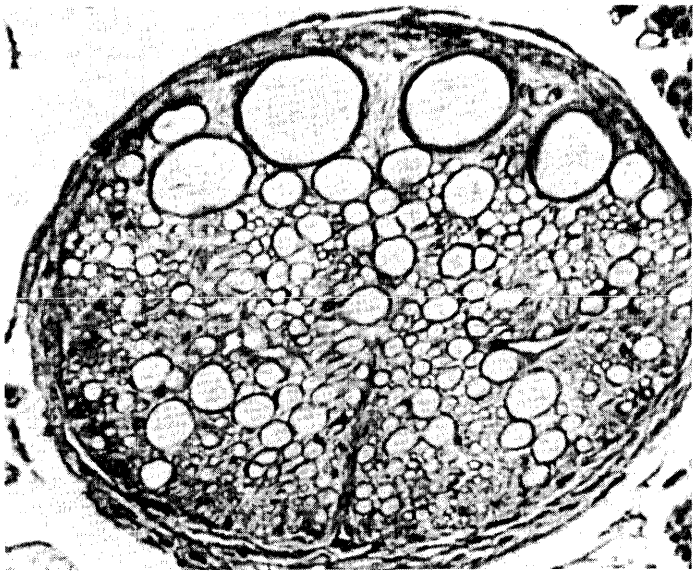


Fig. 7 A sample cross section of the caudal nerve cord stained with osmic acid, (6μ section).

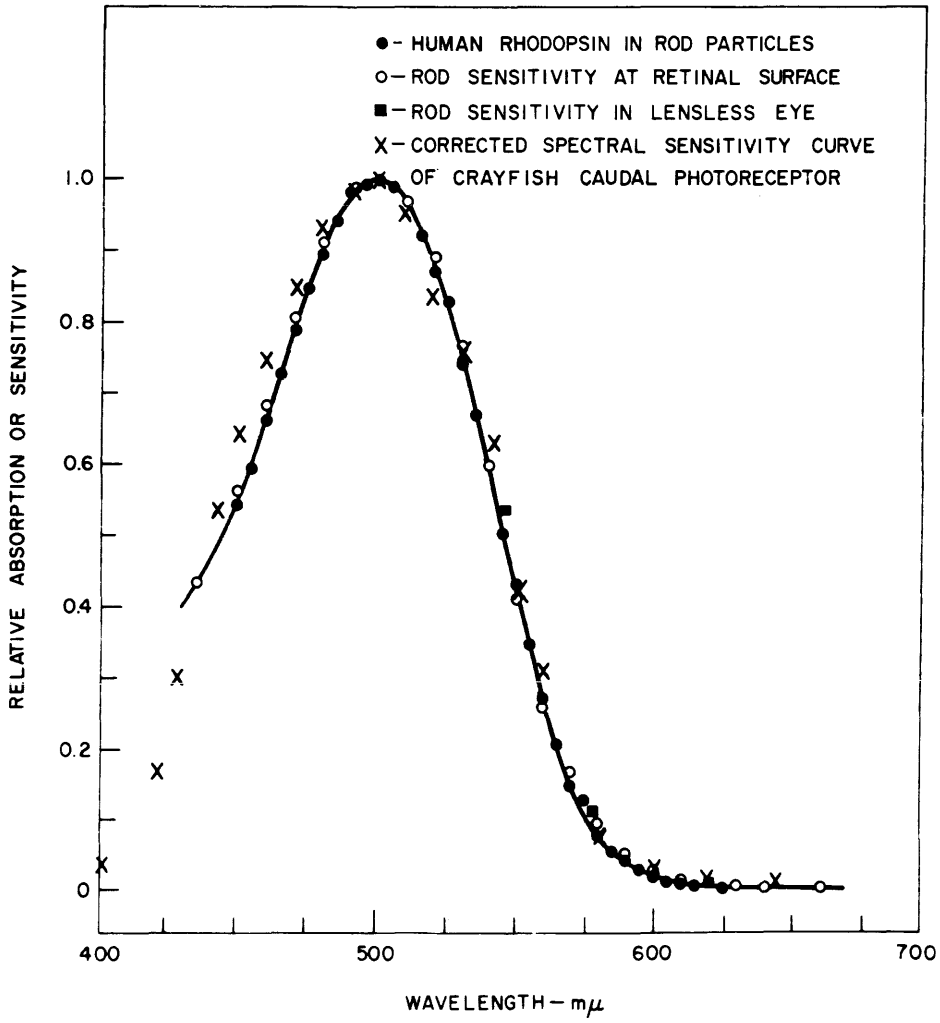


Fig. 8 A plot of the corrected results of the spectral effectiveness experiment superimposed upon Wald and Brown's (1958) three way comparison of photochemical, theoretical, and psychophysical results.

A THEORY AND SIMULATION OF RHYTHMIC BEHAVIOR DUE TO RECIPROCAL INHIBITION IN SMALL NERVE NETS

Richard F. Reiss

Librascope Division

General Precision, Inc.

Glendale, California

Summary

On the basis of a specific conceptual model of signal processing in neurons, together with some fragmentary arguments and evidence in physiological literature, an elementary theory of a "multivibrator effect" producible by reciprocally inhibiting neurons is developed. The results of exploratory simulation experiments are described, and speculations on the possible role of the multivibrator effect in semiautomatic muscle control systems are presented.

Introduction

The investigations reported here constitute one phase of a continuing study of small nerve nets. The most highly developed nervous systems contain, and are built upon, more primitive subsystems, each of which is responsible for basic units of behavior. These subsystems are to some extent autonomous but must also act in cooperation with each other within a hierarchical framework. Although an understanding of the hierarchy as a whole may be the ultimate objective, such understanding can scarcely be more than superficial unless and until the capabilities, the demands and dependencies, of the parts are known in some detail.

Thus the repertoire of behavior that can be produced by small groups of neurons is an important problem both for the biologists and psychologists who seek to understand complex nervous systems for their own sake, and for those of us who are motivated by a desire to eventually synthesize inferior, equivalent, or perhaps even superior machines. As empirical data accumulates, particularly in recent years, the picture of an individual neuron becomes ever more complicated. In the face of this physiological evidence, it is increasingly difficult to justify the characterization of a single neuron's signal-processing capabilities by such concepts as simple logical elements. And consequently our views of the potential behavioral range of small nerve nets must be correspondingly expanded.

The theory and simulation experiments discussed in this paper all revolve around the rhythmic behavior that might be produced by just two neurons which inhibit each other, a case that does not look very promising at first glance. The conceptual and simulation models of neural signal processing used are also rather simple. Nevertheless, the expenditure of considerable time and effort investigating the basic neuron couplet has produced what could only be called exploratory results. An exhaustive, definitive study has yet to be carried out. The possible functions of such couplets when connected with a few other neurons, sensors, and effectors are only touched upon here. It is clear that we have done no more than scratch the surface of that problem. And finally, a methodical correlation of our results with existing physiological data and attempts to predict future empirical discoveries have only just begun.

A secondary but important objective in this research is the comparative evaluation of analog and digital simulation techniques for the study of small nerve nets. We believe that in the long run the development of powerful simulation tools will prove crucial to the advance of nerve net theory. However, that subject is beyond the scope of this paper and receives only casual mention. The section describing briefly the analog and digital models is included primarily to provide other investigators a basis for comparing their results with ours. The reader may well wish to skip from the description of our conceptual model to the section on reciprocal inhibition theory.

A Conceptual Model of the Neuron

The digital and analog simulation models used in these studies are based on a simple conceptual model of signal processing in neurons. This conceptual model has few if any novel aspects when compared to those underlying previous simulation models. The emphasis on "fatigue" is somewhat unusual, but the functional effects of fatigue are similar to those of "adaptation" as employed, for example, by Harmon[8,9] and Taylor[17] in their analog models.

It will be assumed that the state of a neuron at any particular moment is completely defined by the values of three state variables S, θ, and E. The value of S represents a quantity of "stimulant" produced by pulses from other neurons, i.e. by all previous inputs. For a given set of parameters, S has a maximum possible value S_{max} and a minimum possible value S_{min}; S_{max} is a positive number, and S_{min} is a negative number. An "excitatory" input pulse raises S toward S_{max}, whereas an "inhibitory" input pulse decreases S toward S_{min}. As a consequence of these limits on excursions of S, we obtain a process resembling "defacilitation" in real neurons. As S approaches S_{max}, excitatory inputs lose their

effectiveness regardless of pulse magnitude. The same applies to inhibitory inputs if S is driven negatively toward S_{min}. Initially S=0, and in the absence of incoming pulses S, whether positive or negative, decreases exponentially to zero with the passage of time.

The state variable S corresponds roughly to the degree of polarization in a synaptic membrane. When S=0, it represents the normal resting potential across the membrane. Positive excursions of S represent depolarizations, and negative excursions represent hyperpolarizations. However, a neuron receiving pulses from two or more axon terminals may have two or more distinct synapses. In this situation S represents the combined effects of all synaptic membrane states at that point in the neuron where new pulses are generated, i.e., where the neuron "fires" and initiates a pulse which propagates out along its axons to other neurons. Hence the spatial and temporal summation of all inputs to the neuron are represented by the lumped variable S.

It is assumed that each input pulse has associated with it a number which characterizes its nature (excitatory or inhibitory), the axon terminal at which it appears, and the spatial and temporal degradation of the resulting synaptic response which occurs within the neuron between the synapse and the locus of firing. An excitatory input pulse is represented by a positive number; an inhibitory pulse, by a negative number. These numbers will be called "terminal coefficients" because they will be assumed to be the same for all pulses arriving at a particular terminal. Thus each axon terminal is assigned its coefficient, and this number will then characterize all pulses passing through that terminal and the corresponding synapse.

This conceptual model focusses attention on that small area in a neuron where new pulses are generated, i.e., the "spike initiation locus." The three state variables S, θ, and E are actually assumed to describe just the state of the spike initiation locus; but since we are not concerned here with the complexities of intraneural signal interactions, we shall continue to use these variables to represent the state of the neuron as a whole. As a consequence of this drastic simplification, two other assumptions are introduced. First, all pulses are of very short duration and resemble axon spikes in form. The "ballistic effect" of a synaptic membrane, which smears an incoming axon spike over a considerable time interval, is ignored. An incoming pulse causes an almost instantaneous change in S (although the addition may not be algebraic due to the defacilitation imposed by S_{max} and S_{min}). Second, all time delays involved in the generation and propagation of a pulse are lumped into a set of "transit times" for a particular neuron. In effect these simplifying assumptions push axon terminals up against the spike initiation locus with the result that a neuron is represented by nothing more than a spike initiation locus and a branching axon having various transit times associated with its various terminals.

In the simulation experiments described below, the generation of a pulse - i.e., "firing" - has no effect on the value of S. Only incoming pulses and the passage of time influence the behavior of S; there is no "resetting" of S subsequent to firing. This may constitute a very serious distortion of events in some real neurons. However, the investigations of Eccles, Fatt, and Koketsu[5] on Renshaw cells in spinal ganglia and the empirical and simulation studies of cortical neurons by Burns[1,2] indicate that for at least some classes of neurons the act of firing does not wipe out the stimulating potentials and currents which caused firing. To our knowledge, there have been no broad comparative studies of the effects of varying degrees of stimulant destruction due to pulse generation.

The second state variable θ is a "firing threshold." The most important parameter associated with θ is its "resting" value θ_r. Initially $\theta=\theta_r$. If firing occurs (a new pulse generated), θ is assumed to rise to a very large value and then decay, with the passage of time, back to its resting value. The assumptions regarding the type of decay are different in the digital and analog models and will be described at the appropriate points below.

We shall define two types of hypothetical neuron in terms of the criterion for firing. A relaxation neuron fires whenever $S \geq \theta$. A single strong excitatory pulse can raise S well above the resting threshold θ_r, causing the neuron to fire. The threshold goes effectively to infinity immediately after firing and then decays toward θ_r. If the decay of S toward zero is sufficiently slow and the decay of θ toward θ_r is sufficently fast, then at some moment the threshold will fall below the value of S, causing the neuron to fire again. A single excitatory pulse can cause such a neuron to generate a series of pulses with a constantly decreasing frequency. The electronic analogs constructed by Burns[1] can exhibit this type of "after discharge," and the electronic analogs used in our studies can also be adjusted to operate in this "relaxation mode."

For our purposes, the most important feature of a relaxation neuron is that it can generate pulses which are not synchronous with excitatory input pulses. Its firing frequency can differ greatly from the frequency of excitatory input impulses. Clearly, a relaxation neuron may have an output frequency many times greater than the input frequency and thereby act as a "frequency amplifier." This type of behavior appears to have far-reaching consequences for nerve-net theory in general, but the subject will not be pursued here.

The digital simulation of a relaxation neuron presents a messy, costly problem, particularly if a small computer is used. Each time an excitatory pulse arrives or the neuron fires it becomes

necessary to calculate whether and when θ will next fall below S. If neither S nor θ decay in a linear fashion, the simultaneous solution of the functions governing S and θ decay will generally require time-consuming iterative computations. To avoid such complications in our digital simulation model, a somewhat more crude conceptual model of neural firing was postulated. A coincidence neuron fires at time t if and only if: (1) a pulse arrived at time t, and (2) $S \geq \theta$ after this pulse has acted on S. Thus a coincidence neuron can fire only at those times when input pulses (whether excitatory or inhibitory) arrive. This model requires the simulation program to test for firing conditions ($S \geq \theta$) only when input pulses are present, and the prediction problem is eliminated.

Obviously, a coincidence neuron must fire at frequencies which are the same as or subharmonics of the input frequencies. This model corresponds to a real neuron which responds to both the absolute magnitude and rate of change of stimulating potentials or currents. We shall not speculate on the existence of such real neurons here because comparative simulation experiments (discussed below) indicate that the basic reciprocal inhibition networks constructed from coincidence neurons behave in essentially the same way as reciprocal inhibition networks constructed from relaxation neurons. It is likely that other types of network would produce radically different types of behavior depending upon which type of neural model is used.

The behavior S and θ without fatigue is illustrated in Figure 1. The top graph shows the amplitude and times of arrival of input pulses (vertical marks) from several axon terminals. Marks above the base line denote excitatory pulses and marks below denote inhibitory pulses. The second graph shows the response of S (solid line) and θ (dashed line). Initially S=0 and $\theta=\theta_r$. Typical values of the limits S_{max} and S_{min} are indicated. The first two excitatory inputs at times a and b are of the same amplitude, but the second is less effective due to defacilitation. A third excitatory input at time c raises S above θ and firing occurs. The generation of an output pulse is indicated in the third graph. At the time of firing, θ rises very sharply to a very high value and then decays back toward θ_r. The interval during which $\theta > S_{max}$ is the absolute refractory period, and this is followed by the relative refractory period during which $S_{max} > \theta > \theta_r$. At times d and e inhibitory inputs drive S negative. Excitatory inputs at f and g raise S above θ, which has returned to θ_r. The input at time h fails to cause firing. However, this is a relaxation neuron and the decay rates of θ and S are such that θ falls below S at time i, thereby causing a third output pulse to be generated. A final inhibitory input at j hastens the decay of S to zero.

If this were a coincidence neuron, it would behave in the same way except at time i; the threshold would fall below S for a short time, but no firing would occur. Thus the third output pulse would be absent.

The third state variable, E, is introduced into the conceptual model as a means for producing "fatigue." It is intended to represent a quantity of potential energy stored in the neuron. When the hypothetical neuron is fully "rested," $E=E_{max}$, a positive number which E never exceeds. The generation of an output pulse upon firing reduces E by some amount, and then energy flows into the neuron from its environment raising E back to E_{max} in time. It is assumed that E can never reach zero and, therefore, always has a positive value. The rate at which E returns to E_{max} will be called the "recuperation rate," and the amount by which E is reduced when the neuron fires will be called the "fatigue decrement."

The value of E influences the generation of pulses by altering the decay rate of the threshold θ. As E decreases the decay rate of θ decreases, thereby producing greater absolute and relative refractory periods. As E approaches zero the total refractory period approaches infinity. The relation between changes in E and θ are illustrated by Figure 2. Here $\theta=\theta_r$ and $E=E_{max}$ initially, and it is assumed that the neuron is of the relaxation type. In order to simplify the discussion, S is suddenly raised above θ_r at time a and maintained somehow at a constant value until shortly after time g (as indicated by the dashed line). The upper graph shows the behavior of θ; and the lower graph, the corresponding behavior of E. At time a, firing causes E to be reduced by the fatigue decrement. Between a and b the recuperation process raises E back toward E_{max}, but the decay of θ to S is so rapid that firing occurs before E is fully recuperated. Each time the relaxation neuron fires the value of E is further reduced and the decay rate of θ is thereby reduced. By time f an equilibrium has been reached; the period between firings, as determined by the decay of θ and the value of S, is such that the fatigue decrements are just balanced by the recuperation process between firings. This firing frequency will be called the "fatigue equilibrium frequency." Of course, the situation illustrated by Figure 2 is artificial because normally S will be fluctuating rather than constant; the interactions of S, θ and E become quite complex, and an equilibrium may never appear.

Our use of the term "fatigue" in these conceptual and simulation models should not be taken too seriously, but it seems more appropriate than "adaptation" because E is affected by the generation of pulses (with the presumed expenditure of considerable energy) rather than the arrival of input pulses. Adaptation is usually defined in terms of the response of sensory neurons to relatively steady "generator potentials" and is attributed to repolarizing processes in synaptic membranes. In our conceptual model this would be represented by some process acting on S, either self-induced by the value of S and/or caused by

firing. The process might take the form of a gradual increase in the decay rate of S as a function of the mean value of S over some time interval. It may be noted that Taylor[17] introduces an "adaptation network" into his neural analog only when the model represents a sensory neuron subjected to steady generator potentials, and that the network is inserted in the excitatory input section where it is unaffected by firing and does not influence the threshold.

Brazier[3] summarizes the major behavioral differences between adaptation and fatigue in neurons as follows: (1) adaptation develops and disappears more rapidly than fatigue; (2) anoxia hastens fatigue but has little effect on adaptation; and (3) an adapted neuron will respond quickly to changes in stimulus intensity whereas a fatigued neuron responds sluggishly or not at all. In regard to item 1, our conceptual model can approximate the rapidity of onset and disappearance of adaptation if large fatigue decrements and recuperation rates are postulated. Item 3 describes a behavioral difference that can be spanned in our conceptual model by adjustment of parameters, particularly input pulse magnitudes, S_{max}, decay rate of S, and the algorithm governing θ decay as a function of E.

In brief, the use of the term "fatigue" in our conceptual and simulation models is to some degree arbitrary since "adaptation" can be approximated by suitable assignment of parameters.

Digital and Analog Simulation Models

The foregoing conceptual model of neural behavior provided the basis for designing a digital simulation model and an electronic analog model. Although the simulation models differ in several respects, they both provide a satisfactory physical realization of the conceptual model. The digital and analog models of neurons will be called "elements" in order to clearly distinguish between the abstract, conceptual picture of a neuron and the corresponding physical models.

The Digital Simulation Model

The programmatic techniques used in the digital simulation model are based on those used in previous studies.[11,13] The program has been coded and operated by Miss Alice Opolak. A small computer, the RPC 4000, has been used throughout the digital simulation studies reported in this paper. The program admits up to 32 elements and as many output terminals per element, and four simulated "pulse generators" which provide controlled stimuli to networks of elements. A simulation "run" is initiated by reading in a description of elements desired, their connections and parameter, and an agenda governing the frequencies of pulse generators during the entire run. The pulse generators are independent of each other and their frequencies can be incremented or decremented automatically at any prescribed points during the run. Once the simulation program is activated, it continues until a specified amount of time in the nerve-net has been simulated. Two types of output are available and are printed out on-line: a list of every impulse generated by the elements (ordered by time of firing) and, at selected points during the run, a spectrum of periods between firings for each active element.

As noted previously, the elements simulated by the digital program represent coincidence neurons, a simplification dictated by economic considerations. The state variable S is associated with the parameters S_{max}, S_{min}, and a decay rate R. Each output terminal has its terminal coefficient h, which represents the amplitude of all pulses emitted by that terminal. If the terminal is excitatory, then h is positive; if the terminal is inhibitory, then h is negative. If an excitatory terminal emits a pulse, S in the post-synaptic element is instantaneously raised to a new value S' according to the rule:

$$S' = S + h\left(\frac{S_{max} - S}{S_{max}}\right)$$

Since the bracketed fraction approaches zero as S approaches S_{max}, defacilitation occurs with increasing values of S. For all excitatory terminals, $h < S_{max}$.

In computing the response of S to an inhibitory pulse ($h<0$), the algorithm above is used with S_{min} substituted for S_{max}. For all inhibitory terminals, $|h|<|S_{min}|$. It follows that inhibitory pulses also suffer defacilitation as S decreases toward S_{min}.

The decay of S is exponential. If S_o is the value of S at time t = 0, then in the absence of further input pulses, the value of S at any later time t is given by $S = S_o\mathcal{E}^{-Rt}$.

The resting value of the threshold is set at 1.0 in all elements. Other parameter values are assigned, for a desired effect, relative to this fixed point of reference. The unit of time is assumed to be a millisecond. The decay of threshold following firing in the digital element is not exponential. If an element fires at time t = 0, then in the absence of further firings the value of θ at any later time t is given by:

$$\theta = \frac{E_o^{-1} + t^2}{t + t^2}$$

where E_o is the value of E immediately after firing, i.e., after E has been decreased by the fatigue decrement. This algorithm was chosen because it produces a "supernormal" period ($\theta < \theta_r$) following the relative refractory period, similar in form to that observed in axon membranes.

The fatigue process is governed by the parameters E_{max}, P, and Q. Whenever the element fires, E is instantaneously reduced to a new value E' according to the rule:

$$E' = E - Q\left(\frac{E}{E_{max}}\right)$$

Since it is always the case that $Q < E_{max}$, the fatique decrement approaches zero as E approaches zero. Immediately after firing, E begins to rise exponentially toward E_{max}. This is the "recuperation" process. If E_o is the value of E immediately after firing (time t = 0), then in the absence of further firing the value of E at any later time t is given by:

$$E = E_o + (E_{max} - E_o)\varepsilon^{-\frac{1}{Pt}}$$

The relative values of P and Q determine the recuperation and fatigue rates for a given frequency of firing.

Each output terminal has an associated "transit time" Δ which represents the sum of all delays between firing and the arrival of a pulse at the spike initiation locus of a following element. Thus Δ simulates the lumped sum of synaptic and axon propagation delays. The duration of a pulse is zero.

The digital simulation program utilizes "event-controlled" sequencing, i.e., the program jumps in simulated time from one event to the next. The only events examined are the arrivals of pulses at output terminals, at which time they alter S and sometimes cause a new pulse to be generated through firing. Thus the simulation rate is largely determined by the number of pulses existing in the network at the moment. For all practical purposes, there are no constraints on the times at which firing can occur or, therefore, on the frequencies at which an element can fire.

A few more minor features of the program will be discussed below in connection with relevant simulation experiments.

The Analog Simulation Model

The conceptual model of a neuron, as described, has also been used as a basis for "analog elements" that can be hooked together in networks. These elements have been developed by E.D. Lewis for our studies of reciprocal inhibition. Figure 3 shows the five types of subunits which compose an analog element.

The excitatory and inhibitory integrators provide temporal summation of incoming pulses. The outputs of these integrators are of opposite polarity and are summed in the third type of unit. The output of the summing unit represents the state variable S. The fourth unit is a monostable flip-flop similar to those used by Harmon and Associates [8,9] in their analog systems. It has a resting threshold, and if the input S exceeds this threshold the unit fires, generating a sharply rising output pulse of controlled duration. Immediately after firing, the threshold rises to a very large value and then decays through an absolute and relative refractory period to the resting value. However, unlike the digital element, this unit will fire again if the threshold falls below the input potential S. Thus the analog element can simulate a relaxation neuron.

The fatigue integrator provides temporal summation of output pulses; the resulting potential is roughly equivalent to the inverse of the state variable E in the digital element. The output of the fatigue integrator determines the decay rate of the flip-flop threshold after firing. As the firing frequency increases, the absolute and relative refractory periods increase. Hence the fatigue processes in the digital and analog elements are similar.

Both the excitatory and inhibitory integrators are saturated by input impulses of sufficiently high frequency and/or amplitude. The limits imposed by saturation correspond roughly with S_{max} and S_{min} in the digital elements and cause defacilitation at high stimulus levels.

Two analog elements were used in the simulation experiments discussed here. The circuit components are such that the elements can operate in the range of 1 to 1000 pulses per second, i.e., in the normal range of real neurons. In most cases the pulse duration was around 1 millisecond. Delay lines were not used so that the transit time for a pulse traveling from one element to the other was essentially zero. The behavior of elements can easily be observed by oscilloscope and, because the operating frequencies are so low, recordings were made with a conventional ink polygraph.

We note in passing that for the parameter constellations used in the simulation experiments, the analog elements generally respond logarithmically to excitatory inputs. If a constant frequency train of pulses is input to a single element, its firing frequency rises to some peak value f_p and then, as a result of fatigue, declines to an equilibrium value f_e. Over the normal range of the input frequency f_i (about 20 to 500 pps), the peak and equilibrium frequencies approximately satisfy equations of the form $f_p = a + b \log f_i$ and $f_e = c + d \log f_i$, with $a > c$ and $b > d$. All four parameters, a, b, c, and d, are inversely proportional to the fatigue rate.

An Elementary Theory of Rhythmic Behavior Due to Reciprocal Inhibition

How might a steady stream of pulses cause and control rhythmic bursts of activity in neurons and/or muscles? This basic question motivated the simulation studies reported in this paper. Reflexes provide some of the best examples of rhythmic behavior, and the problem was succinctly summarized in 1932 by Creed et al as follows[4]:

> "Alternating reflexes have the feature that the the direction of movement holding for a phase in one sense is followed by a phase of opposite direction and this in turn by reversion to the earlier phase, and so on, the alternation proceeding under continuance of the same unaltered stimulus. Some of the alternating reflexes are of irregular alternation, e.g. the release reflex of the foot, the pinna reflex. A number of them are, however, regularly rhythmic, e.g. stepping, the scratch reflex, the shake reflex of the body, the shake reflex of the head, the to-and-fro snout reflex, biting reflex of lower jaw. In these, the movement proves to be not merely the alternate contraction and relaxation of a single prime mover or set of such; it is operated by alternate contraction of two opposed sets of prime movers, contraction in the one alternating with contraction in the other. The problem is how, under continuance of unaltered stimulation, does the initial contraction, e.g. of the flexor, instead of persisting during the maintained stimulus, die out despite that stimulus, and why, as it does so, does contraction of the extensor occur in its stead, again then to give way to flexion, and so on?"

Creed et al present a list of empirical data which show in some detail various features of alternating reflexes. Although they were working at a rather gross level, several of these items are of importance to our theoretical development and deserve summarizing: (1) An alternating reflex may continue to operate in the absence of afferent feedback from the limbs involved; (2) During the relaxation phase, a muscle (or synergetic group) may receive no excitation whatever, i.e.. the motor neurons are cut off entirely; (3) "The turning point is not abrupt, ... each phase presents a regular waxing and waning of activity"; (4) "It is noteworthy that the scratch and stepping (reflexes) are facilitated by asphyxial conditions"; (5) "The self-generated proprioceptive stimuli of the muscles which take part in (the alternating reflex) can regulate the act but are not essential to its rhythm."

Item 1 suggests that the rhythmic excitation does not stem from some feedback loop through afferents and therefore probably originates in the central ganglion. Items 2 and 3 indicate that while the switchover from one muscle to its antagonist is not instantaneous it does go to completion; i.e., excitation to one muscle is mostly, if not entirely, cut off. Item 4 is particularly important because, as noted previously, lack of oxygen hastens fatigue while it has little or no effect on adaptation. Hence if we are confronted by a choice between fatigue and adaptation in explaining the alternating mechanism, fatigue is to be preferred. Finally, item 5 indicates a useful role of afferent feedbacks in regulating the alternating mechanism.

In his classic work of 1906, Sherrington[15] discusses what he calls "reciprocal inhibition" by which he means "....the inhibition at one part always appears as the negative aspect of positive excitation at another." This is a very general definition; apparently Sherrington's "part" may be anything from a single neuron up to a whole segment of the nervous system together with its associated muscle groups. He generally uses "reciprocal inhibition" to describe, rather than explain, the behavior of various reflex systems. However, in discussing alternating reflexes he quotes at length a theory, proposed in 1903 by McDougall[12], wherein reciprocally inhibiting chains of neurons are used to explain the rhythmic alternations of contraction and relaxation. The network in Figure 4 is equivalent, using our diagrammatic conventions, to the network analyzed by McDougall and Sherrington. The topology is somewhat different due to their use of a now-defunct theory of inhibition at synapses. Arrowheads stand for excitatory axon terminals and solid circles for inhibitory terminals. McDougall argued, in effect, that if a_1 is more active than b_1, then the net stimulus to a_2 will be excitatory whereas the net stimulus to b_2 is inhibitory. Hence b_2 will be cut off while a_3, a motor neuron, will be active and cause the flexor to contract. After a time, "fatigue" will raise the synaptic resistance of a_2, thereby lowering its activity and indirectly removing some of the inhibition at b_2 so that it becomes more active and, in turn, suppresses a_2. This would provide a switch of excitation from a_3 to b_3 with a subsequent relaxation of the flexor and contraction of the extensor.

However, McDougall failed to recognize the possibility of alternately achieving excitation and inhibition of a_3 and b_3 when the inputs to a_1 and b_1 are of equal strength. He assumed that a_1 and b_1 must be stimulated at different intensities. Furthermore, he was primarily interested in the possible role of reciprocal inhibition in the visual apparatus of humans; and he carried out a variety of experiments on edge effects and related phenomena which supported his thesis to some extent. It is interesting that such contemporary theorists as Fessard[6] and Taylor [16] assume, as did McDougall, that the major function of reciprocal inhibition in vision is to amplify and sharpen the edges of figures projected on the retinal matrix.

The key ideas here are those of "reciprocal inhibition"--neurons inhibiting each other--and "fatigue." Combined, these ideas suggest an explanation of certain types of switching action. To our knowledge. McDougall's is the earliest clear theory of this type. Although the theoretical and simulation studies reported here stemmed by analogical reasoning from the behavior of free-running electronic multivibrators (a device unknown to McDougall and Sherrington), the ideas are essentially the same and McDougall clearly set the precedent.

In order to develop the theory of reciprocal inhibition further, we shall first concentrate on

the simplest conceivable network, as shown in Figure 5, and ignore its possible role in a larger system. The parameters of neurons a_2 and a_3 are approximately symmetric in value. The firing frequencies of a_1, a_2, and a_3 will be denoted by f_1, f_2, and f_3, respectively. We start with the naive assumption that the effective stimulation provided by an axon terminal of any neuron is the product of that neuron's firing frequency and the amplitude of the pulses characteristically produced by the terminal. If the terminal is inhibitory, it generates a negative stimulation. Let the excitatory terminals of neuron a_1 produce pulses of the same amplitude, namely, 1.0. The inhibitory terminals of a_2 and a_3 produce pulses of amplitude h_2 and h_3, respectively. Finally, let us assume that the firing frequency of an element is the simple sum of all input stimulation (and let the dimensions take care of themselves).

Thus for any given frequency f_1 of a_1, the frequencies of a_2 and a_3 are given by:

$$f_2 = f_1 - h_3 f_3 \qquad (1)$$

$$f_3 = f_1 - h_2 f_2 \qquad (2)$$

Since a negative frequency is impossible, if the right side of either equation has a negative value, the corresponding frequency is zero. This is the same as saying that if the net input to a neuron is inhibitory, that neuron will not fire at all.

For a given constant frequency of a_1, what will the network do? The most obvious answer is given by a simultaneous solution of equations (1) and (2), which yields:

$$f_2 = f_1\left(\frac{1-h_3}{1-h_2h_3}\right) \qquad (3)$$

$$f_3 = f_1\left(\frac{1-h_2}{1-h_2h_3}\right) \qquad (4)$$

If, for example, the inhibitory terminals produce rather strong pulses of $h_2 = h_3 = 2.0$ (strong compared with the excitory terminals of a_1), then $f_2 = f_3 = f_1/3$.

But there are two other simultaneous solutions to equations (3) and (4); namely, $f_2 = f_1$ and $f_3 = 0$, or $f_2 = 0$ and $f_3 = f_1$; if h_2 and h_3 are greater than 1.0. These are legitimate solutions because (1) and (2) are not ordinary equations. If the left side is zero, any negative value on the right satisfies the equality because a neuron whose net input is negative (inhibitory) will presumably have a frequency of zero.

Thus this simple model predicts that the network of Figure 5 can have three stable states: (1) both a_2 and a_3 firing at some fraction of the frequency of a_1; (2) $f_2 = f_1$ with a_3 totally suppressed; and (3) $f_3 = f_1$ with a_2 totally suppressed. We shall say that in the second case a_2 is dominant and in the third case a_3 is dominant. Dominance implies here that the other element is totally suppressed.

Suppose now that if a_2 is initially dominant, it gradually loses its sensitivity to excitatory pulses from a_1. In effect, the amplitude of pulses from a_1 decreases; (1) can be rewritten as $f_2 = h_1 f_1 - h_3 f_3$ where $h_1 < 1.0$ and is decreasing. If f_2 decays far enough, the right side of (2) becomes positive and a_3 begins firing. Each inhibitory pulse from a_3 hastens the decrease in the frequency of a_2, which in turn removes more inhibition from a_3, and so on. It is conceivable that this self-facilitating process, which we shall call "switchover", will terminate with a_2 suppressed and a_3 dominant. The critical point which initiates switchover occurs when f_2 decays to $f_2 = f_1/h_2$, which is the "switchover threshold." Similarly, there is a switchover threshold for a_3; namely, $f_3 = f_1/h_3$.

Therefore, on the basis of this model of neural behavior, a_2 and a_3 can become alternately dominant if each neuron loses its sensitivity as a result of repeated firing but regains that sensitivity during the time when its opponent is dominant.

The Multivibrator Effect

We shall call the alternating dominance of reciprocally inhibiting neurons "the multivibrator effect." In the aforementioned case, the sequence of dominance would be .... a_2, a_3, a_2, a_3 .... etc. The time interval over which a_2 can maintain dominance is the "dominance interval" of a_2, denoted by D_2. Likewise, the dominance interval of a_3 is D_3. Given symmetric parameters in a_2 and a_3, one should find that $D_2 = D_3$. Furthermore, the more rapidly the dominant neuron's frequency decays, the shorter its dominance interval. On the other hand, since the switchover threshold is inversely proportional to the impulse amplitude at the dominant neuron's inhibitory terminal, the stronger its inhibitory terminal, the longer its dominance interval. Consequently, asymmetric values of frequency-decay rates may, for example, be compensated by asymmetric strengths of inhibitory terminals; and it is possible that $D_2 = D_3$ even though parameters are <u>not</u> symmetric. But, in general, asymmetric parameters would likely make $D_2 \neq D_3$.

These are a few characteristics of the multivibrator effect in a network with reciprocal inhibition, as predicted by the algebraic approximations to real neural behavior. The simulation studies were undertaken as a result of such an analysis. Somewhat to our surprise, the predictions of this exceedingly crude algebraic model turned out to be quite good in respect to modes of behavior and the general effects of varying the several parameters. Exact quantitative comparisons were not possible because the digital and analog models are much more complex than the neurons assumed by the algebraic model.

The frequency decay in the dominant neuron, which must be present if switchovers are to occur, could, of course, be produced by either adaptation or fatigue. Although we know of no decisive empirical evidence on which to base a choice (in fact both could be operating), fragmentary physiological data, such as discussed above, suggests that fatigue is more likely to be the cause of frequency decays in dominant neurons, if indeed the multivibrator effect ever actually occurs in real nerve nets. To our knowledge the multivibrator effect has never been directly observed in living nervous systems.

Simulation Experiments on Simple Reciprocal Inhibition

The first series of simulation experiments were carried out on the digital model. Since the elements in this model represent "coincidence" neurons, there was some reason to fear that the harmonic character of their behavior would interfere with, if not prevent, the appearance of a multivibrator effect. A network of the type shown in Figure 5 was the subject of these experiments. Element a_1 was represented by one of the simulated pulse generators.

Initially the parameters of a_2 and a_3 were perfectly symmetric in values. Numerous tests over a wide range of input frequencies with various parameter constellations failed to produce anything more than rather synchronous firing in a_2 and a_3 at frequencies considerably lower than the input frequencies. This corresponded to the mode of operation predicted by equations (3) and (4) above. Neither element was able to gain dominance over the other and so the multivibrator effect could not appear.

The next step was to introduce asymmetry into the parameters. After only a few experiments, the multivibrator effect did appear. The pattern of firing shown in Figure 6 is typical of the behavior obtained in these early, and most subsequent, experiments. Here a_2 becomes dominant first, followed by a switchover to a_3, followed by a switchover to a_2, and so on. Inspection reveals two types of switchover in this sample record. The switchover from a_2 to a_3 does not involve any synchronous firing. The last impulse of a_2 is followed by an appreciable interval before the first pulse of a_3 appears. Furthermore, a_3 begins firing at the highest frequency attained during its dominance interval, i.e., the period between the first two pulses is as short as any in the dominance interval. The second type of switchover is illustrated when a_3 loses dominance. The last pulse of a_3 is synchronous with the first pulse of a_2, but there is a long period between a_2's first and second spike. Of course all firings here were synchronous with pulses from the generator a_1. However, both types of switchover have been seen in analog elements simulating "relaxation" neurons. Hence these do not appear to be artifacts introduced by the harmonic behavior of coincidence neurons. The second type of switchover evidently occurs when the state variables and input times are such that the dominant element manages to get off one final pulse just as the other begins firing. This last pulse, after the delay produced by the element's transit time, inhibits the onset of firing in the newly dominant element - thus the long period between first and second firings in the other element.

The dominance intervals of a particular element tend to be consistently terminated by the same type of switchover for a wide range of input frequencies. Typically, the element having the lesser "sensitivity" produces the first kind of switchover, which will henceforth be called a "gap" switchover, while the more sensitive element produces the second kind, the "synchronous" switchover. A third kind of switchover, the "overlap, is discussed below.

The decrease in frequency during the dominance interval, due to fatigue, is quite marked in Figure 6. The input frequency is constant, but the dominant element is able to maintain its maximum frequency (which may be equal to the input frequency, but is more commonly a subharmonic) for only two or three firings. The multivibrator effect cannot appear unless the fatigue equilibrium frequency is less than the switchover threshold frequency for the dominant element. The network of Figure 6 involves 18 parameter values in the digital model, and we have not been able to develop any algorithms for precise calculations of either the fatigue equilibrium or switchover threshold frequencies. It is not clear that the development of such algorithms would be worth the experimental effort required for induction and verification. The model is too arbitrary.

As noted, the multivibrator effect was obtained only after asymmetric parameter values were introduced. Many variations of asymmetry were tested (as many as five networks were simulated simultaneously with all but one parameter held constant), and a peculiar similarity of successful parameter constellations appeared. Although asymmetry was necessary to the occurrence of the multivibrator effect, the asymmetry apparently must be of the "self-cancelling" variety. For example, if the excitatory terminal contacting a_2 emits pulses larger than those emitted by the excitatory terminal contacting a_3 (which gives a_2 an advantage), then the inhibitory terminal contacting a_2 should also emit larger pulses than that contacting a_3 (which tends to cancel the excitatory advantage of a_2). We have not yet found a satisfactory explanation for this phenomenon.

Three Modes of Behavior

The two-phase multivibrator effect appears only over a rather well-defined range of input frequencies. The upper and lower limits of this range are a function of all parameters. Outside these limits, two other modes of behavior are commonly observed. By setting the input frequency initially at zero and raising it in stepwise fashion by small

increments, all three modes appear in sequence.

At very low input frequencies, one element dominates continuously. If there is mixed or synchronous firing of both a_2 and a_3 at low frequencies, it is unlikely that the multivibrator effect will appear at any frequency. As a rule, it is the more sensitive element that holds perpetual dominance at these low frequencies. We shall not attempt to develop a formal definition of "sensitivity" here. Because of the self-cancelling asymmetry, it is not always possible to predict which element is the more sensitive even though, in the digital model, all parameters are precisely known. But, generally, the element receiving the largest excitatory pulses, if it also has the largest S_{max} (least defacilitation), will be the most "sensitive" in our usage of the term.

As the input frequency is raised step by step, a transition from one element dominance to the two-phase multivibrator effect appears. This transition is not necessarily sharp. There may be a small range of input frequencies over which the multivibrator effect appears only sporadically, in which case there is no precise lower limit of the multivibrator range. As the input frequency is raised further, the dominance intervals become shorter and shorter until another transition range is reached. Here there is a transition from the multivibrator effect to mixed firing in both elements. In effect, the dominance intervals are reduced to one or two pulses and the concept of "switchover" becomes rather meaningless.

Small changes in parameters have little effect on the input frequency range of the multivibrator mode. A great many more experiments must be made before a clear picture of the effects of various parameters on this range can be put together. The fatigue rate appears to be the most important factor in determining the lower and upper limits of the multivibrator mode. For example, experiments on the digital model with a certain set of parameters which produced a low fatigue rate resulted in a frequency range of approximately 300 to 500 pulses per second for the multivibrator mode. Holding all other parameters constant, the fatigue decrement was doubled with the result that the frequency range was shifted down to approximately 200 to 400 pps. Thus, for this constellation of parameters, the magnitude of the frequency range remained essentially constant while the lower and upper limits were shifted downward by an increase in the fatigue rate. Experiments on both the digital and analog models have generally indicated that the higher the fatigue rate (within limits), the lower the input frequency at which the multivibrator effect first appears. The upper limit of the multivibrator mode behaves in a less predictable way.

Dominance Intervals as a Function of Input Frequency

The curves in Figure 7 are plots of the dominance intervals versus input frequencies for the two elements in a network, such as shown in Figure 5, using the digital model. The upper curve shows the dominance intervals in the more sensitive element. The multivibrator effect appeared at about 320 pps in this experiment, and the dominance interval of the more sensitive element was approximately 1-1/2 times greater than that of the other element. This difference is due to the asymmetric parameters. As the upper limit of the multivibrator mode (about 450 pps) is approached, the dominance intervals become nearly equal. Each point on this graph is the average of several dominance intervals. As might be expected in coincidence neurons, harmonic effects can cause the dominance intervals to fluctuate about an average value for any given input frequency. The analog elements, when simulating relaxation neurons, produce very small fluctuations, generally less than one or two percent.

Both the shape and frequency range of the curves in Figure 7 are typical of the experimental results produced by the digital model. We have had little success in obtaining the multivibrator effect over a wide range of input frequencies. The analog elements, on the other hand, can easily produce the multivibrator mode over much wider ranges (measured as the ratio of the upper to the lower limits). The curves in Figure 8 illustrate this clearly. Here the multivibrator effect appears at about 15 pps and has an upper limit above 100 pps. The upper-to-lower limit ratio is more than 7:1, whereas the digital model rarely produces ratios of more than 2:1. The absolute values of input frequencies for digital and analog models cannot be directly compared because of the differences in detailed behavior of the two models.

The curves in Figure 8 illustrate two extremes in the variation of dominance intervals with increasing input frequencies. In this analog experiment, there were considerable differences between the input and fatigue integrators of the two elements. One element (lower curve) produced dominance intervals that were almost independent of input frequency. The other element (upper curve) was able to produce long dominance intervals at the lower frequencies.

Effects of Transit Times

In the digital model, the delay between firing and the appearance of impulses at the element's output terminals is an independent parameter and therefore easily controlled. In the reciprocal inhibition network of Figure 5, the transit times of the inhibitory terminals could obviously play an important role in the multivibrator effect. Although relatively few experiments have been carried out to test the effects of variations in transit times, the results point to a general conclusion that the transit times should be small (relative to the periods between excitatory input pulses). If the transit times are several times larger than the period between input pulses, the two elements have a tendency to fire in synchronous bursts. Of course this is an interesting phenomenon in itself

and may have physiological applications. But, in respect to the multivibrator effect, it appears to be important as a contributing factor to the breakdown of the multivibrator mode at high input frequencies.

The transit times of the analog elements were essentially zero in the experiments to date. This may partially account for the relatively high upper limit of the multivibrator range observed in the analog model.

Dominance Ratios

In Figure 8, one element has a dominance interval more than twice that of the other element for low input frequencies. We shall say, then, that the "dominance ratio" is more than 2:1 in this case. Asymmetric parameter values are, of course, responsible when both of the reciprocally inhibiting elements are excited at the same frequency and synchronously. Dominance ratios between 10:1 and 15:1 have been achieved with ease simply by setting the fatigue rate much higher in one element than in the other. For a given input frequency, there is in principle no limit to the dominance ratio that could be obtained by appropriate assignments of fatigue rates. But, once the parameter values have been set, the variation in dominance ratio with input frequency is fixed. The dominance ratio may be relatively constant, as in Figure 7, or may vary considerably, as in Figure 8.

The introduction of independent excitatory inputs, however, opens up a new domain of behavior. The network in Figure 9, which is essentially the same as that proposed by McDougall and Sherrington (see above), makes it possible to excite a_3 and a_4 at different frequencies. Thus the dominance ratio may be influenced by outside signal sources. We have only begun to explore this small, but significant, modification of the basic reciprocal inhibition network. The most interesting simulation results achieved to date (using analog elements) suggest the following relationship. Let f_1 and f_2 denote the firing frequencies of a_1 and a_2, in Figure 9, and let D_3 and D_4 denote the dominance intervals of a_3 and a_4. If for a given pair of input frequencies the parameters are adjusted so that $D_3 \cong D_4$ when $f_1 = f_2$, then the dominance ratio approximately satisfies the equation

$$\frac{D_3}{D_4} = \left(\frac{f_1}{f_2}\right)^2 \qquad (5)$$

for an appreciable range of input frequency ratios. The fit is rough but definitely observable. If we consider the case where a_3 and a_4 are driving efferent neurons, then the number of pulses during a dominance interval is most important in determining the power developed by muscles under the control of these efferents. As a rude approximation, we can say that the number N_3 of pulses produced by a_3 during its dominance interval is proportional to the product $f_1 D_3$ and similarly, for the number N_4 produced by a_4. Utilizing equation (5), we arrive at the following algorithm:

$$\frac{N_3}{N_4} \cong \left(\frac{f_1}{f_2}\right)^3$$

This line of reasoning suggests that if a reciprocal inhibition net is driving a pair of muscles, the ratio of power developed in the muscles over a period of time is proportional to the cube of the input frequency ratio. Thus small differences between the two input frequencies could produce large differences in the power developed by the two corresponding muscles.

Effects of Noise

The digital simulation program has facilities for introducing pseudorandom fluctuations in the frequency of a pulse generator. The operator may specify the amount of "wobble" he desires as a percentage of the normal period between pulses. This may be considered a form of "noise," and its effects on the multivibrator mode in the single input network (Figure 5) have been tested. Wobbles of up to 10% have been introduced without interfering with the multivibrator mode. Near the upper frequency limit of the multivibrator mode dominance intervals fluctuate (by a smaller percentage) because there are only three or four firings during the interval. The wobble effect is generally insignificant because random changes in successive periods tend to cancel out over the much longer dominance interval.

Effects of Asynchrony

If the single input element in Figure 5 is replaced by two elements, as in Figure 9, which are firing at nearly identical frequencies, then excitatory pulses will arrive at essentially the same frequency but with constantly changing phase relations. This has been tested on the digital model with networks whose responses to a single input element were known. No appreciable effects were observed. The multivibrator mode appears to be quite stable in the presence of phase shifts due to asynchronous inputs.

Overlapping Switchovers

Two types of switchover, the "gap" and the "synchronous," were previously noted. A third type, which we shall call the "overlap," has appeared in some experiments and is illustrated by Figure 10. Here the dominance intervals overlap, the fatigued element firing shortly after the newly dominant element has become active. Although the factors producing overlap have not been isolated with certainty, it appears that relatively weak inhibitor terminals are mainly responsible. The phenomenon usually occurs at high input frequencies near the upper limit of the multivibrator mode.

Rapid Changes of Input Frequency

If the input frequency is suddenly raised, it generally induces a switchover. This effect is

most noticable in the analog model when simulating relaxation neurons. It is most likely due to the fact that the threshold of the nondominant element has returned to, or near, its resting value; and the element has had an opportunity to recuperate. Sudden decreases in the input frequency do not induce switchovers.

Summary and Interpretation of Reciprocal Inhibition Study to Date

The study reported in this paper constitutes only the first phase of investigations into the possible modes of rhythmic behavior and related switching processes in small nerve nets. It will have become apparent to the reader that many aspects of the simplest two-element reciprocal inhibition network remain to be methodically explored. Simulation experiments on extensions and mutations of the basic reciprocal inhibition network (not reported here) have already yielded distinctive, new behavior patterns. It now seems clear that a large variety of functions may be performed by small nerve nets, composed of a dozen or so neurons in a few basic configurations, when coupled with sensors and effectors. The major purpose of this concluding section is to suggest how reciprocal inhibition in its simplest form could support several types of semiautomatic control systems. We have not, as yet, made a careful search of biological literature for relevent empirical data; physiological evidence referred to here was come upon more or less by chance. The problem is to know what to look for in the literature, and this knowledge will be advanced by further theoretical and simulation studies however naive.

The most obvious application of reciprocal inhibition is the control of alternating reflexes, as noted by Sherrington and McDougall at the turn of the century. The rhythmic, alternate contractions and relaxations of antagonistic muscle groups provide the fundamental means of locomotion for a wide variety of swimming, crawling, walking, and flying animals. Locomotion is essential to their self-preservation. A simple neural means for producing such rhythmic behavior would have been very desirable during the course of animal evolution, particularly if the mechanism could be easily regulated by higher centers of a nervous system in the more complex animals.

Let us examine the basic reciprocal inhibition network from the standpoint of its possible use in controlling antagonistic muscles. The network in Figure 11 is composed of seven neurons and two antagonistic muscles, m_1 and m_2. Neurons c_1 and c_2 are reciprocally inhibiting and stand between the input neurons a, b_1, b_2 and the motor neurons d_1 and d_2. Assuming for the moment that b_1 and b_2 are quiet, it is clear that when the multivibrator mode is present due to the excitation of a, the muscles will alternately contract and relax in phase with the dominance intervals of c_1 and c_2. The mean frequency at which c_1 fires during its dominance intervals is proportional to the input frequency of a. This can be easily demonstrated by simulation for a variety of parameter constellations. If the inhibitory terminals produce strong impulses, the switchover threshold is only slightly below the input frequency with the result that dominance intervals are short; and the frequency of the dominant neuron decays only slightly during the dominance intervals.

Since the strength of muscular contraction is roughly proportional to the excitatory frequency, the following picture emerges: as the frequency of a increases, dominance intervals shorten, producing a more rapid alternation of contractions; and since the mean frequency during dominance increases, the strength of the contractions (per unit time) increase. For any period of time considerably greater than a dominance interval, each of the muscles is active for half of the period (on the average) regardless of the alternation rate. Thus the total power expended increases as the frequency of a goes up; and as a by-product of the characteristic decrease of dominance intervals at higher input frequencies, the alternation rate of muscular contraction also rises.

If the muscles of this crude system are responsible, for example, for locomotion, then a single stream of signals from neuron a could easily control the rate of speed produced by the relatively more complex muscle-skeleton apparatus. It must be presumed that if a's frequency drops so low that the multivibrator effect disappears, the frequencies of c_1 and c_2 are below those required for overt contraction of the muscles. This is not unreasonable, especially in light of Hoyle's discoveries described below.

The utility of neurons b_1 and b_2 may be simply described if we suppose that the muscles are responsible for the rearward thrusts of opposing legs in a crawling animal. So long as the dominance intervals of c_1 and c_2 are approximately equal, the path of the animal will waver about a straight line. Its speed is proportional to a's frequency. If now the frequency of b_1 rises above zero, the dominance interval of c_1 will become greater than that of c2 (with appropriate parameter values), with the result that the average force exerted by the left muscle will be greater than that of the right; and the animal will slew to the right. The greater the difference in frequencies of b_1 and b_2, the greater the corresponding dominance ratio, and hence the greater the turning rate. If we suppose that b_1 and b_2 are sensors, a simple phobic response to external stimuli is imposed on the more complex crawling mechanism. Not only would the animal turn away from the stimulus, but the stronger the stimulus, the greater the turning rate. If both b_1 and b_2 are equally stimulated, the to-and-fro wiggling motion might provide sufficient differences to initiate a turn.

If the frequency of neuron a or, rather, the combined frequencies of a, b_1, and b_2 are so high that the multivibrator effect is replaced by the more or less synchronous firing in c1 and c2, as

observed in simulation experiments, then contractions of the muscles would cease to be coordinated and would tend to be spasmodic. The experiments on transit times suggest that to obtain the multivibrator effect at high input frequencies, the transit times should be very short and, therefore, c_1 and c_2 should be close together.

The foregoing flight of fancy suggests how rhythmic muscular action could be modified by external stimuli. We now turn our attention to the regulation of the multivibrator mode by proprioceptive feedbacks. Consider the network in Figure 12. Stretch receptors r_1 and r_2 have been attached to the tendons of muscles m_1 and m_2. Suppose that the dominance intervals of c_1 and c_2 are normally equal but the irregularities in the environment momentarily impose greater loads on muscle m_1 than on m_2. The increased load is detected by r_1, whose frequency consequently increases. Since it has an excitatory termianl on c_1, the dominance interval of c_1 will be greater than that of c_2; and so will its mean frequency. Hence m_1 will be stimulated to exert greater force than m_2, and a greater fraction of the total power will be allocated to m_1. If the load increases equally on both muscles, the dominance intervals will remain equal, thus providing an equal distribution of force; but the mean frequencies during dominance will increase, thereby increasing the force of contraction in both muscles.

The network of Figure 12 suggests how proprioceptors, by regulating the dominance ratio and mean frequencies, permit the basic reciprocal inhibition mechanism to automatically adjust (within limits) to varying load conditions while maintaining coordinated contractions in antagonistic muscles. Unless the loads exceed the adaptation capabilities of the network, higher nerve centers need not be informed of the detailed fluctuations of loads.

It may be noted in passing that the same network might equally satisfy the control requirements if m_1 and m_2 constitute a synergetic group that must time-share a common load.

Recent discoveries of Graham Hoyle[10] are of particular interest within the framework of our reciprocal inhibition studies. He has been recording the motor neuron activity in the legs of intact locusts by means of implanted electrodes. His recordings show that even when the locust is at rest flexor and extensor muscles continue to receive the alternating excitation typical of walking movements. The frequencies are too low to produce overt movement, but the pattern remains. This suggests to us that if reciprocal inhibition is involved in the control of the locust leg muscles, with a network somewhat like that in Figure 11, then the lower limit of the input frequency range for producing the multivibrator effect is well below the input frequency required for actual movement. This would ensure that if the input is sufficient to cause movement, it is sufficiently high to produce the multivibrator effect and, therefore, the coordinated alternate contractions of flexor and extensor. Such a safety factor is consistent with the hypothesis that a reciprocal inhibition network could normally operate with a minimum of monitoring from higher nervous centers.

Another discovery of Hoyle's was, to us, very startling. He noticed that when a locust was engaged in its stereotyped "marching," the excitation to the extensor (line e) and flexor (line f) have the patterns illustrated in Figure 13. Hoyle hypothesizes that the movements required in marching are too rapid for the complete relaxation of both muscles each cycle; and that, therefore, one antagonist (the extensor in Figure 13) is continuously stimulated at a low frequency, so that it provides a reasonably strong elastic band against which the other muscle works. Thus more rapid alternations are possible.

This surprised us because on several occasions, while fiddling with the parameters of the analog model in attempts to obtain the multivibrator effect, just this pattern of firing appeared. Of course this was an annoying mode of behavior, being neither the multivibrator mode nor mixed firing; and so it received no attention. We have now returned to the simulation model in search of the conditions which produce Hoyle's pattern consistently.

A further complication observed by Hoyle was the periodic reversal of the pattern of excitation; i.e., the flexor is continuously stimulated while the extensor receives the bursts. If reciprocal inhibition is involved, then this clearly suggests that the fundamental pattern is caused by signal inputs rather than inherent asymmetries in the neural parameters. The regular and distinct switchovers from one pattern to the complement suggests the presence of a second reciprocal inhibition net with long dominance intervals and terminals on the fundamental net. This possibility remains to be investigated.

The multivibrator effect could, in theory, serve important functions in sensory perception and internuncial communication. However, consideration of such functions necessitates the introduction of three or more reciprocally inhibiting elements and is beyond the scope of this paper.

As mentioned earlier, such theorists as Fessard[6] and Taylor [16] argue that reciprocal inhibition probably serves to amplify differences in frequencies of adjacent neurons. Burns[2], on the other hand, suggests that it eliminates crosstalk, the spurious lateral spread of signals in densely packed layers of neurons. Yet again, Granit[7] supposes that reciprocal inhibition of motor neurons, whether direct or indirect, may provide negative feedback to limit their frequencies to match the low values ideal for exciting muscles. To our knowledge, no theorists since McDougall have seriously examined the possibilities of multivibrator action.

The best direct evidence for the existence of reciprocal inhibition networks in muscle control systems appears to be the special neurons described by, and named after, Renshaw[14]. The investigation of Eccles, Fatt, and Koketsu[5] expanded the evidence that Renshaw cells are responsible for rather widespread inhibition. They also suggest that Dale's principle to the effect that all axon terminals of a given neuron must be of the same kind (all inhibitory or all excitatory in our model) and, therefore, that motor neurons, having excitatory terminals (again we simplify the statement to suit our model), could not have inhibitory terminals on each other. The Renshaw cells may therefore be interposed to provide inhibitory terminals. The neurons b_1 and b_2 would serve essentially the same function as Renshaw cells.(Fig.14).

However, Eccles et al make no mention of the demonstrated or possible occurrence of a multivibrator effect. Unfortunately they, like Renshaw and others, excite these spinal neurons (in cats) only by brief antidromic volleys in the motor neurons, a procedure unlikely to bring forth multivibrator action.

Thus the empirical evidence is sketchy and only suggestive.

Conclusion

Reciprocally inhibiting neurons must be able to transform a relatively constant frequency stream of impulses into rhythmic bursts of impulses if our conceptual and derivative simulation models are reasonably valid. The multivibrator effect does not appear to be critically dependent on narrow ranges of parameter values. Minor differences in the parameters of two hypothetical elements facilitate rather than hinder the effect, which is important in light of the probable fact that no two real neurons are identical in behavior.

While the multivibrator-effect reliability appears over a substantial range of parametric conditions, it is not a rigid mode of behavior. The dominance ratio, mean frequencies during dominance, and lower and upper limits of the mode are all subject to regulation by signals. On the other hand, random fluctuations of the input frequencies have little effect. Thus reciprocal inhibition provides a viable mechanism for producing rhythmic behavior with varying degrees of automatic adjustment to prevailing environments.

On the basis of fragmentary empirical evidence, fatigue was chosen to produce the necessary frequency decay in a dominant element. However, the multivibrator effect might as well be produced by adaptation at synaptic junctions. If the effect occurs in real nerve nets, both adaptation and fatigue probably play important roles.

The material presented here constitutes an exploratory study of reciprocal inhibition between two elements only. Even this simple system requires and deserves more thorough investigation. We are not certain that all of the major, much less secondary, modes have been observed. The analog and digital simulation models have proved to be invaluable tools. The main advantages of the digital program have been the precise control and recording of events, the ability to automatically carry out sequential and simultaneous groups of experiments, and the ease with which model changes can be made. Its main disadvantages have been the necessity of transcribing printed output into firing plots suitable for rapid visual scanning and the economic necessity of restricting digital elements to coincidence firing. The analog's advantages were the ease with which parameters could be altered while observing the consequences directly, output in a form that can be rapidly scanned, relaxation firing, and small size. Its disadvantages were the difficulty of precisely controlling and observing the parameters and artifacts and the inability to execute sequences of experiments automatically.

Acknowledgments

We are pleased to acknowledge the support of the U. S. Air Force Office of Scientific Research, Contract AF 49(638)-1021, in the investigations reported here. Special thanks are due Miss Alice Opolak for her coding and operation of the digital program and her patient testing of numerous parameter variations. Similarly, we are indebted to Dr. E. D. Lewis for the design, construction, and operation of the analog simulation models.

References

1. Burns, B.D.: "The Mechanism of After-bursts in Cerebral Cortex" J. Physiol (1955), Vol. 127, 168-188.

2. Burns, B.D.: "The Mammalian Cerebral Cortex" London, 1958.

3. Brazier, M.A.B.: "The Electrical Activity of the Nervous System" 2nd Ed. New York, 1960.

4. Creed, R.; Denny-Brown, D; Eccles, J.C; Liddell, E.G.; Sherrington, C.S.: "Reflex Activity of the Spinal Cord" Oxford, 1932.

5. Eccles, J.C.; Fatt, P.; Koketsu, K.: "Cholinergic and Inhibitory Synapses in a Pathway from Motor-Axon Collaterals to Motoneurones" J. Physiol. (1954), Vol. 126, 524-562.

6. Fessard, A.: "The Role of Neuronal Networks in Sensory Communications within the Brain" Sensory Communication (W.A. Rosenblith, Ed.) New York, 1961.

7. Granit, R: "Receptors and Sensory Perception" Yale, 1955

8. Harmon, L.D.: "Artificial Neuron" Science (1959), Vol. 129, 962-963.

9. Harmon, L.D.; Wolfe, R.M.: "An Electronic Model of a Nerve Cell" Semiconductor Products, Aug. 1959, 36-40.

10. Hoyle, G.: Lecture: "Experimental Analysis in the Intact Insect" California Institute of Technology; Jan. 12, 1962.

11. Josephson, R.K.; Reiss, R.F.; Worthy, R.M.: "A Simulation Study of a Diffuse Conducting System based on Coelenterate Nerve Nets" J. Theoret. Biol. (1961), Vol. 1, 460-487.

12. McDougall, W.: "The Nature of Inhibitory Processes within the Nervous System" Brain (1903), Vol. 26, Part 2, 153-191.

13. Reiss, R.F.: "The Digital Simulation of Neuro-Muscular Organisms" Behavioral Science (1960), Vol. 5, 343-358.

14. Renshaw, B.: "Influence of Discharge of Motoneurons upon Excitation of Neighboring Motoneurons" J. Neurophysio. (1941), Vol. 4, 167-183

15. Sherrington, C.S.: "The Integrative Action of the Nervous System" Yale, 1906.

16. Taylor, W.K.: "Electrical Simulation of some Nervous System Functional Activities" Third London Symposium on Info. Theory (C. Cherry Ed.) London 1956.

17. Taylor, W.K.: "Computers and the Nervous System" Models and Analogues in Biology: Symp. No. 14, Soc. Exp. Biol. New York, 1960.

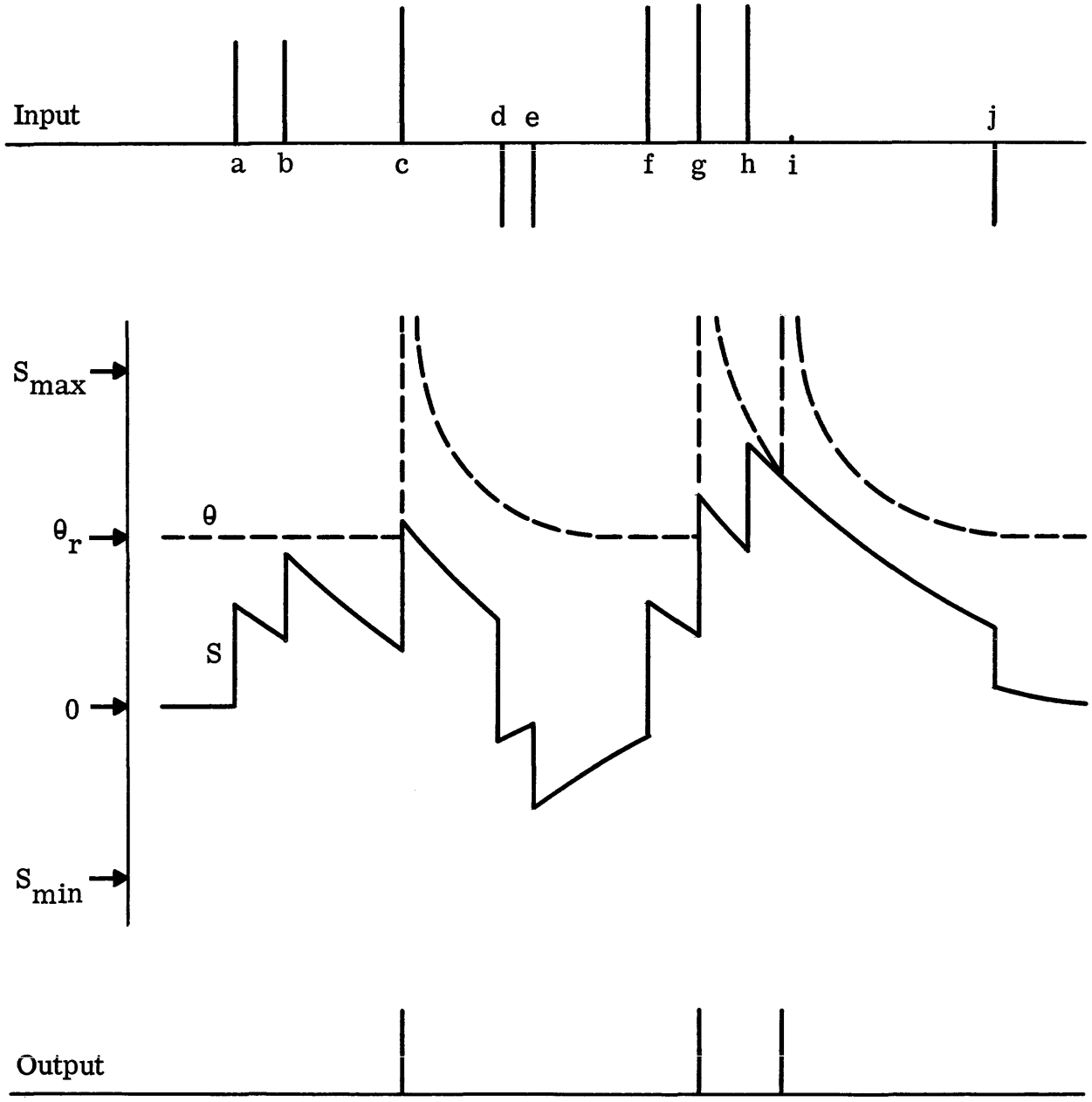


Figure 1. Response of S and θ to Excitatory and Inhibitory Impulses.

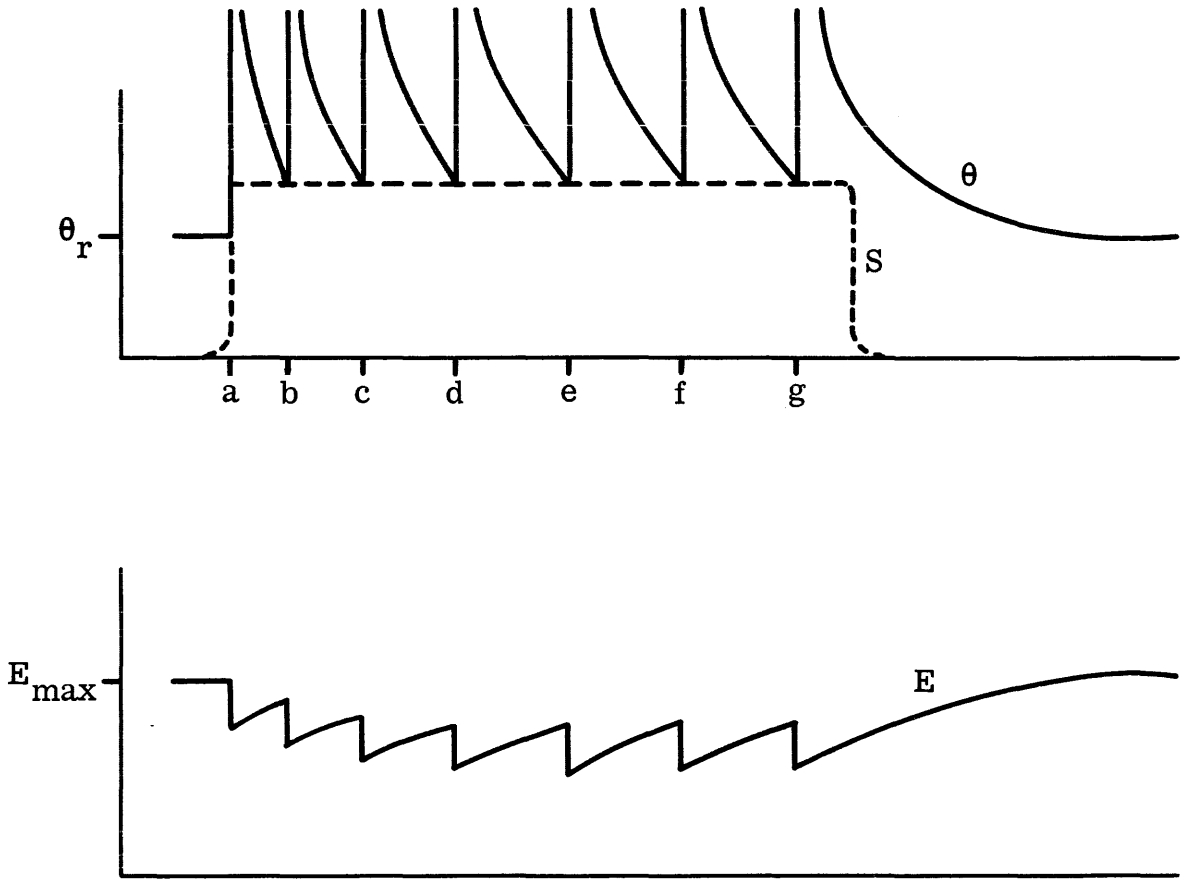


Figure 2. Fatigue Effects of Firing; Stimulant S Held Constant.

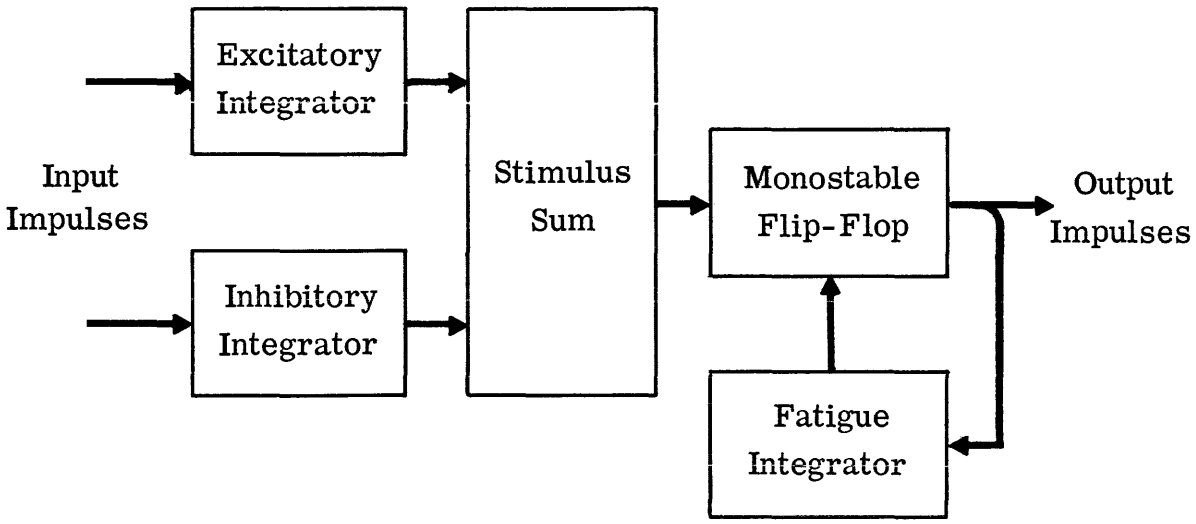


Figure 3. Major Components of Electronic Analog Neuron.

(Arrowheads ≡ excitatory axon terminals
Solid circles ≡ inhibitory axon terminals)

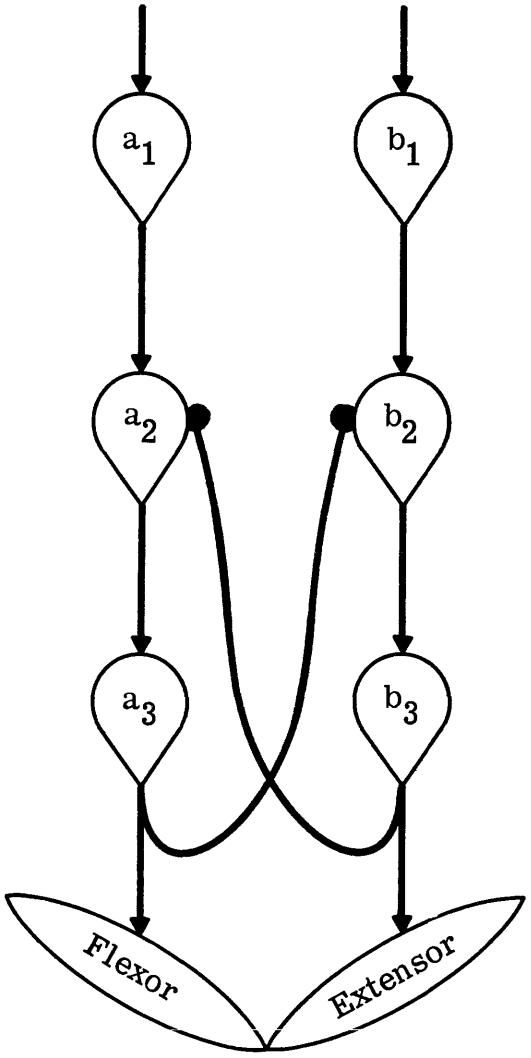


Figure 4. Reciprocal inhibition network equivalent to that proposed in 1903 by McDougall[12] to explain alternating reflexes.

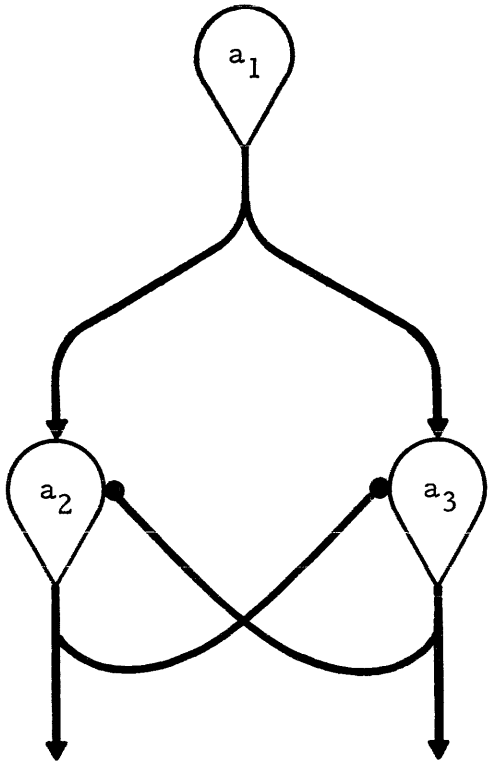


Figure 5. Basic reciprocal inhibition network with one input element (a_1).

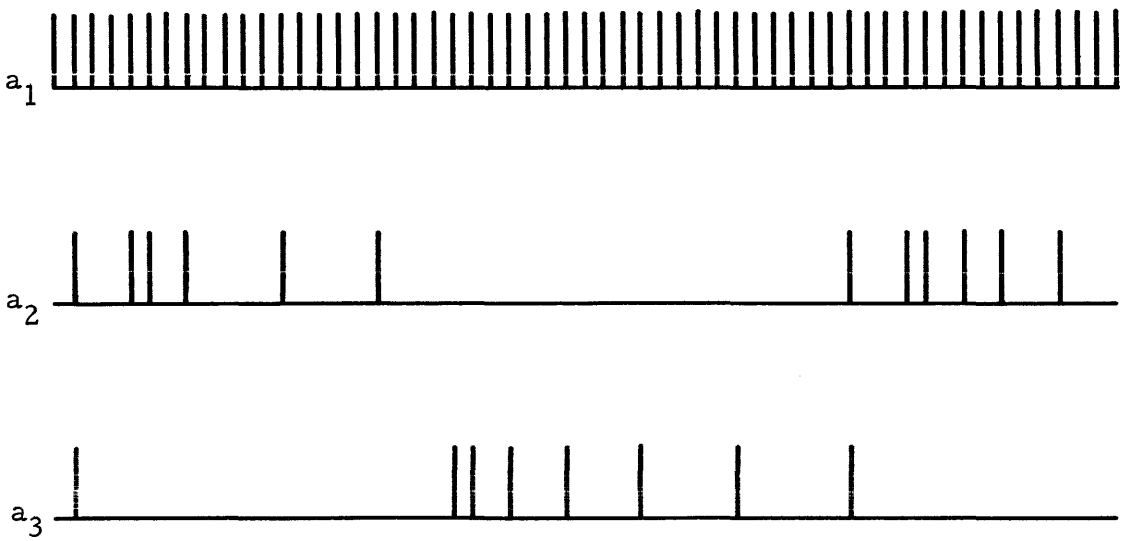


Figure 6. Firing patterns of elements in network of Figure 5, showing multivibrator effect with two types of switchover.

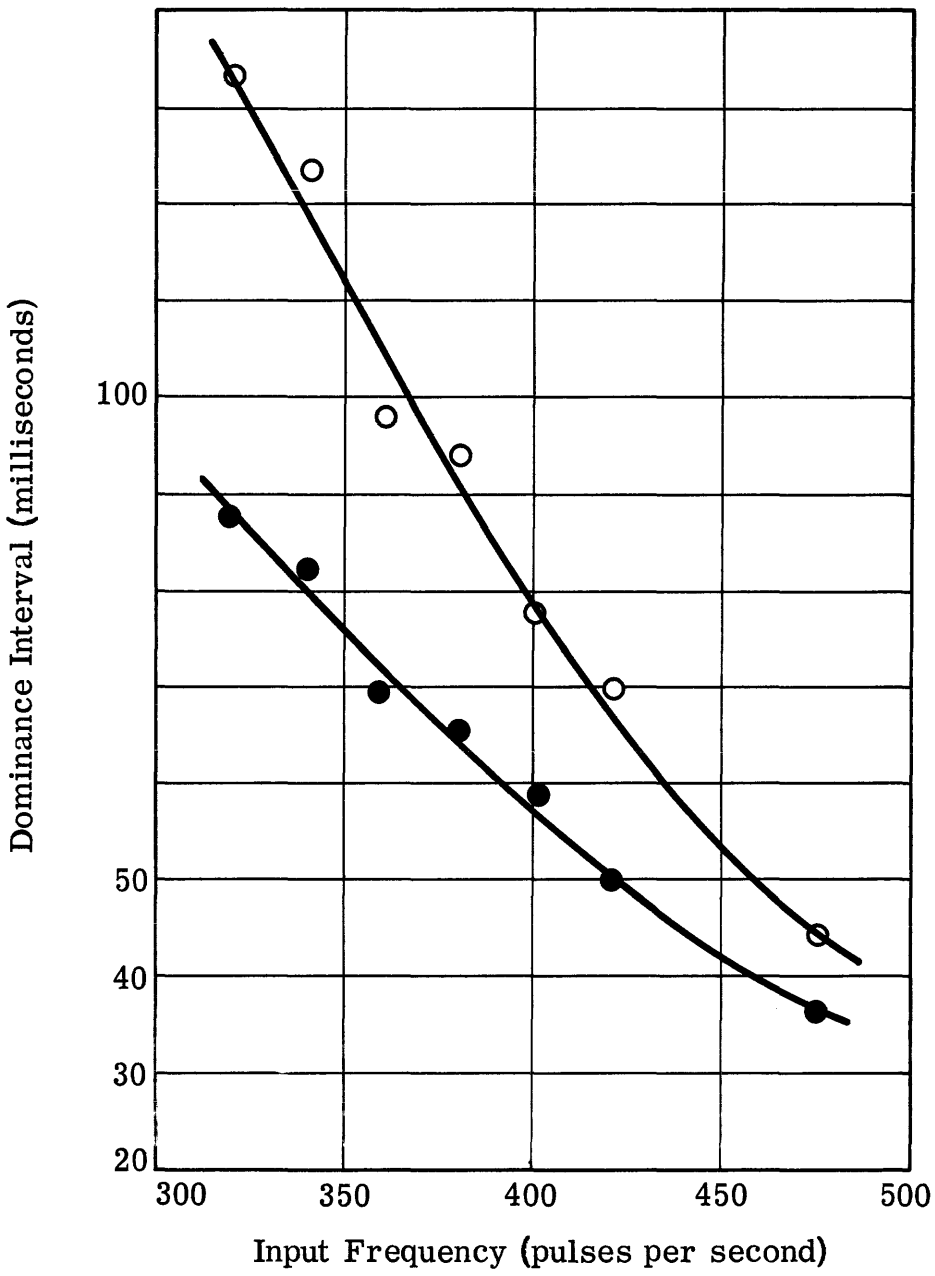


Figure 7. Typical variation of dominance intervals in two reciprocally inhibiting elements with asymmetric parameters. (Digital Simulation).

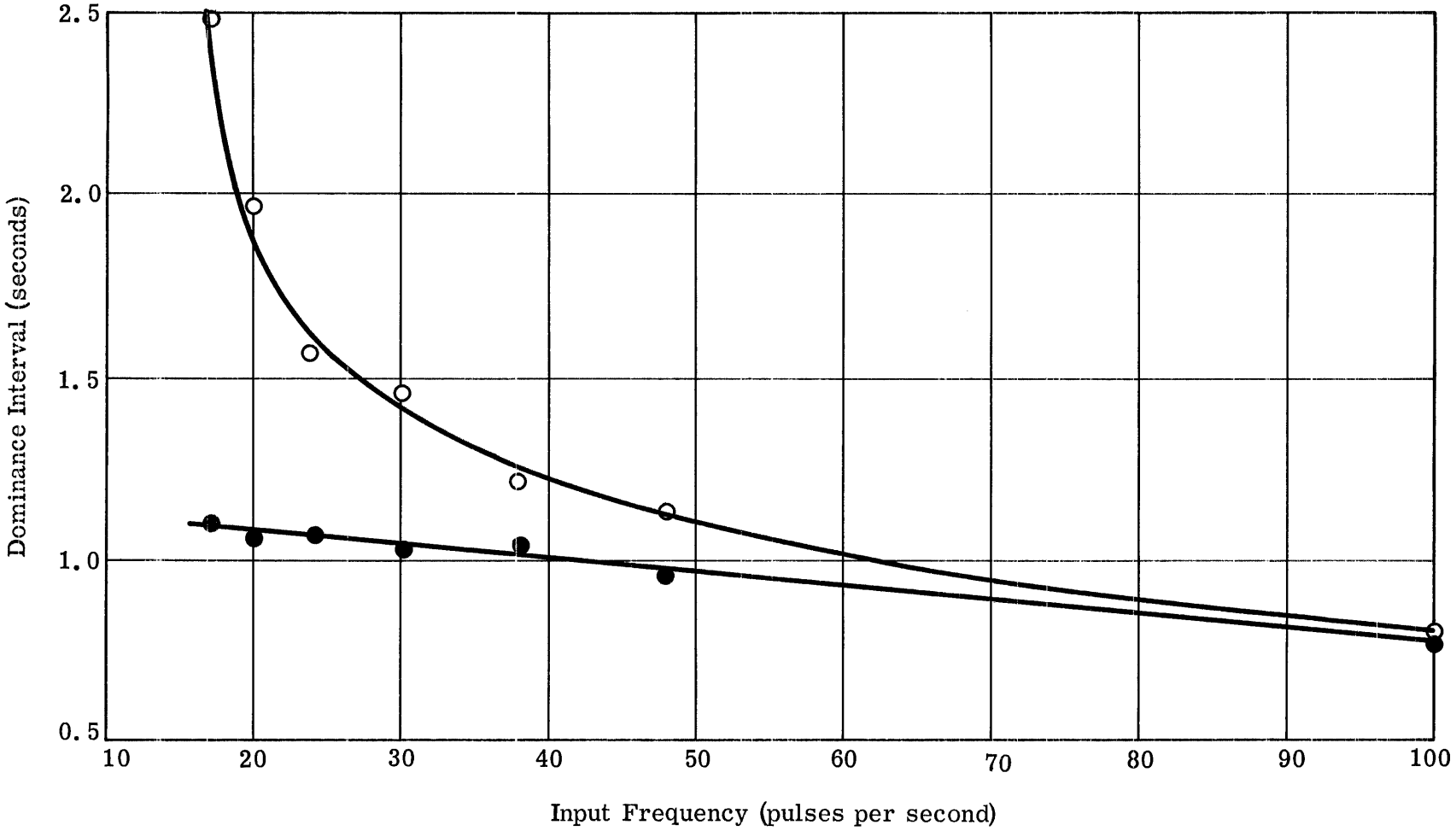


Figure 8. Variation of dominance intervals in two reciprocally inhibiting elements with highly asymmetric parameters. Illustrates linear and nonlinear extremes of dominance interval behavior. (Analog Simulation).

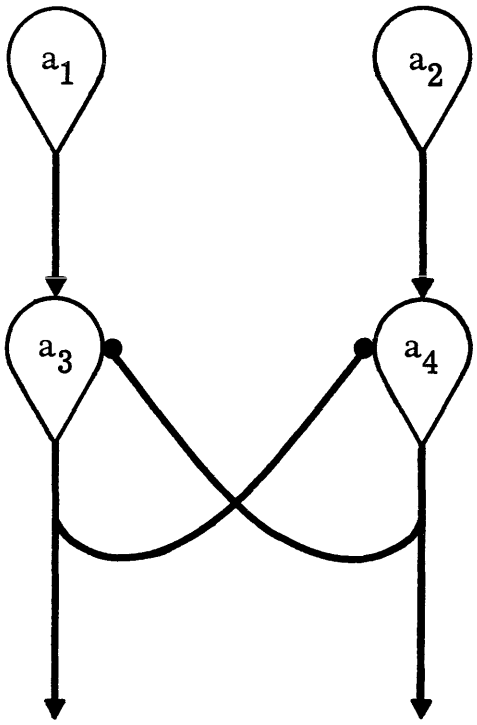


Figure 9. Basic reciprocal inhibition network with two, independent input elements.

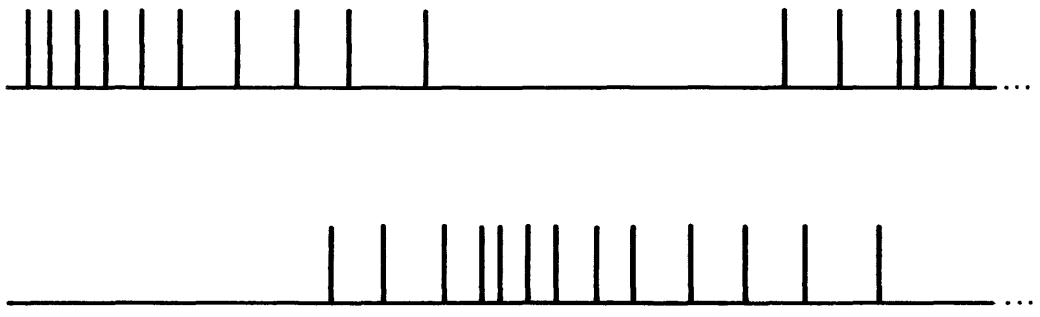


Figure 10. Multivibrator effect with overlapping switchovers.

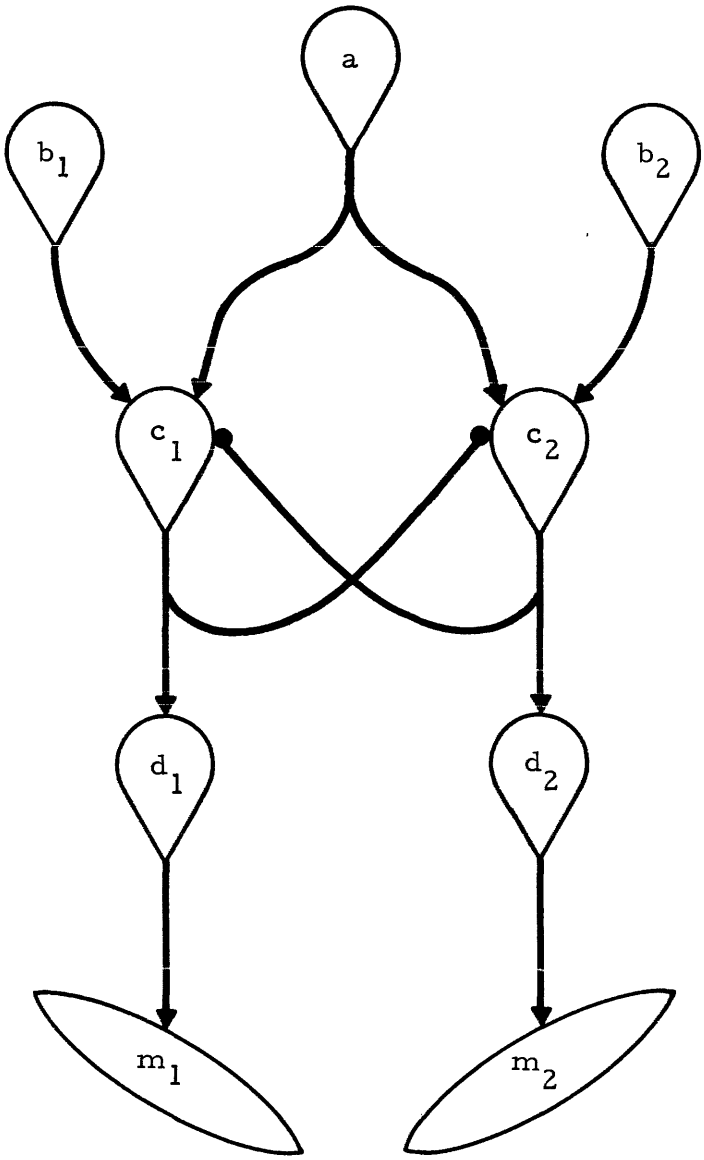


Figure 11. Reciprocally inhibiting elements driving efferents d_1 and d_2, with lateral inputs b_1 and b_2 for control of dominance ratio. Elements m_1 and m_2 represent antagonistic muscles.

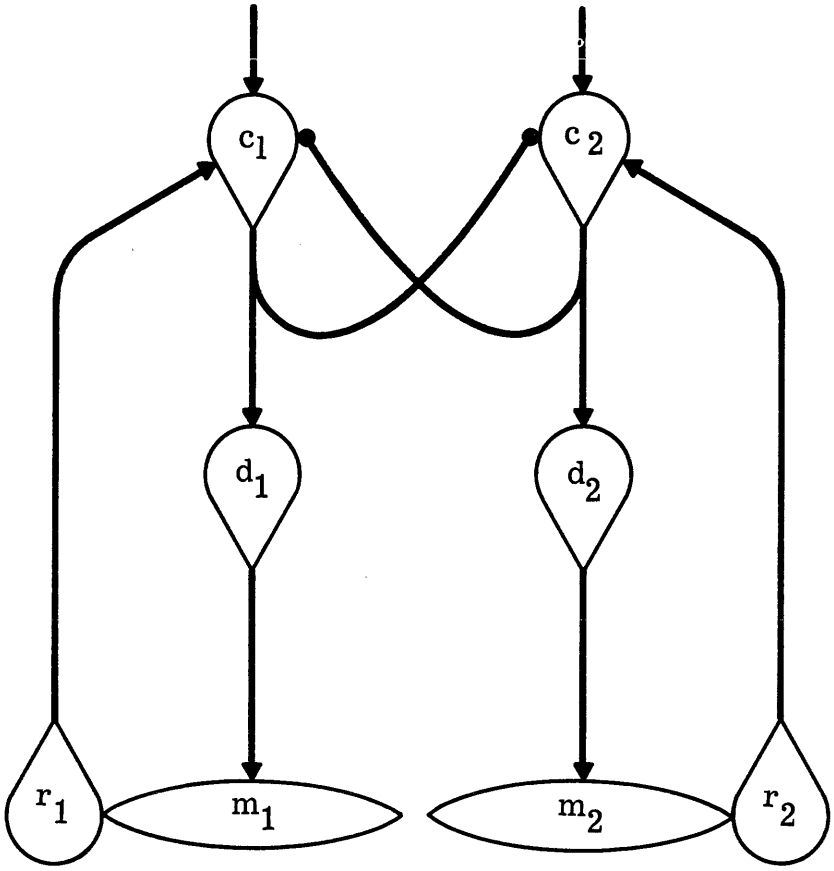


Figure 12. Reciprocal inhibition network with stretch receptors regulating dominance ratio to match unequal loads on m_1 and m_2.

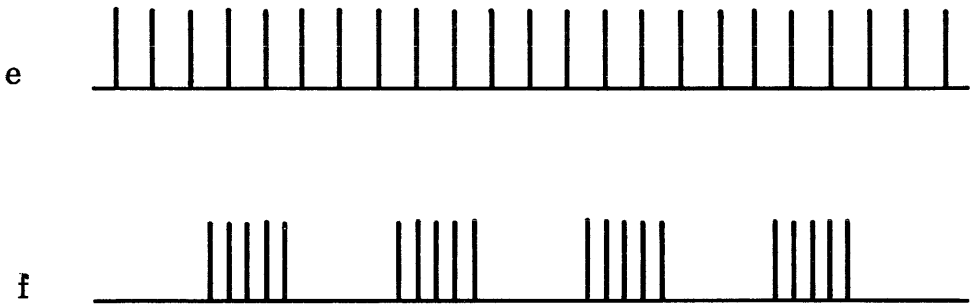


Figure 13. Semi-multivibrator mode due to asymmetric parameters.

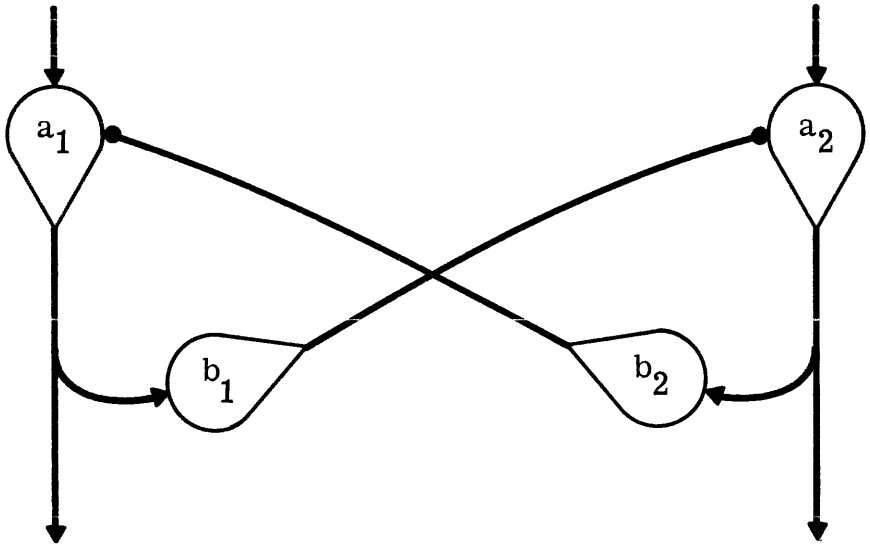


Figure 14. Reciprocal inhibition via intermediate elements b_1 and b_2 as suggested by the study of Renshaw cells (Eccles, Fatt and Koketsu[5]), and satisfying Dale's Principle.

THE MANIAC III ARITHMETIC SYSTEM

Robert L. Ashenhurst

Institute for Computer Research

University of Chicago

Chicago, Illinois

Summary

Unlike most computers, for which there is a formal distinction between "fixed-point" and "floating-point" numbers, the University of Chicago Maniac III computer handles all numbers in a single format (exponent and coefficient, with the coefficient in general not normalized). This permits several types of arithmetic to be defined, which differ in that results are adjusted (coefficient scaled) according to different rules. These rules are classified in terms of "significant-digit," "normalized," "specified point" or "basic" characteristics. Since the operand format in all cases is the same, numbers can be processed by the various arithmetics without intermediate conversion, thus adding a dimension of flexibility to the computing process.

This paper discusses the Maniac III arithmetic rules in some detail, showing how they embody the cited characteristics, and how consistent conventions for rounding, adjustment of zero and formation of low order parts are established. The trapping system used for the detection of anomalous results is also described.

1. Number Representation

Maniac III has a 48-bit word, which is partitioned for purposes of numerical manipulation into an 8-bit exponent part and a 40-bit coefficient (fraction) part. The exponent bits are numbered from right to left, and the coefficient bits from left to right, starting from 0 in either case. The format is diagrammed in Figure 1.

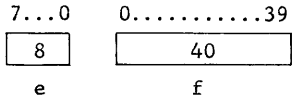


Figure 1. Numerical format

Any sequence $f_0 f_1 \ldots f_{39}$ of bits in the coefficient section represents a number f in the range $-1 \leqslant f \leqslant +1-2^{-39}$, according to the formula

$$f = -f_0 + \sum_{k=1}^{39} f_k \cdot 2^{-k}$$

(see Table 1). This is the familiar scheme in which non-negative numbers are represented in standard binary form (with implicit binary point between f_0 and f_1), while negative numbers are represented in the corresponding (true or two's) complement form. The bit f_0 is called the sign bit of the representation; $f_0 = 0$ when $f \geqslant 0$ and $f_0 = 1$ when $f < 0$. If f is negative, the sequence of bits to the right of f_0 represents (in standard binary form) the positive number f* satisfying $f^* + |f| = 1$, so that $f = -1+f^*$. From this it is easily seen that, as indicated by the table, every positive number f which can be represented has a negative -f which can also be represented, and the only negative number for which the reverse does not hold is f = -1.

Code	Number
10.....00	-1
10.....01	$-1+2^{-39}$
10.....10	$-1+2^{-38}$
⋮ ⋮	⋮
11.....11	-2^{-39}
00.....00	0
00.....01	$+2^{-39}$
⋮ ⋮	⋮
01.....10	$+1-2^{-38}$
01.....11	$+1-2^{-39}$

Table 1. Coefficient coding

For $f > 0$, the number $m \geqslant 0$ of leading digits of f is defined as the number of consecutive bits 0 appearing to the right of the binary point in the representation of f. Evidently $m = 0$ when $1/2 \leqslant f < 1$, $m = 1$ when $1/4 \leqslant f < 1/2$, and so forth. When $f = 0$, $m = 39$ by convention, and for $-1 < f < 0$, m is defined to be the number of leading digits in the positive number -f. For all negative $f \neq -2^{-k}$, this is just the number of consecutive bits 1 appearing to the right of the binary point in the representation of f; if $f = -2^{-k}$, m is one less than this number. If $f = -1$, m is undefined.

Any sequence $e_7 e_6 \ldots e_0$ of bits in the exponent section, except the sequence 00...0,

represents an integer e in the range $-127 \leqslant e \leqslant +127$, according to the formula

$$e = (e_7 - 1)\cdot 2^7 + \sum_{k=0}^{6} e_k \cdot 2^k$$

(see Table 2). This is an "excess-128" scheme, in

Code	Number
00...01	-127
00...10	-126
⋮ ⋮	⋮
01...11	-1
10...00	0
10...01	+1
⋮ ⋮	⋮
11...10	+126
11...11	+127

Table 2. Exponent coding

which each exponent e is represented by the sequence of bits which would represent e + 128 in standard binary form. The bit e_7 is called the exponent sign; $e_7 = 1$ signifies $e \geqslant 0$, while $e_7 = 0$ signifies $e < 0$. For each possible positive exponent e there is a corresponding negative, and vice-versa, without exception. The sequence 00...0, however, is not assigned a numerical value; it is denoted e_Z and called the exponent for "absolute zero," as explained subsequently.

If $e \neq e_Z$, then the pair f,e is taken to to represent a number x,

$$x = f\cdot 2^e.$$

If $f_1 = f_0$, then the same number x is obtained by shifting the coefficient pattern one place left (i.e., replacing f_k with f_{k+1} for $k = 0,1,\ldots,38$, and f_{39} with 0), and subtracting 1 from the exponent, since

$$2f\cdot 2^{e-1} = f\cdot 2^e = x,$$

and the left shift effectively doubles the coefficient. If $f \neq 0$, this process can be repeated until $f_1 \neq f_0$, in which case the number x is said to be in normalized form. In general, m = 0 for a normalized form; the only exception occurs when $x = -2^{-k}$ for some $k > 0$, in which case f = -1 in the normalized form and m is undefined. For f = 0, the normalized form itself is undefined; the number x = 0 is represented by f = 0 with any numerical exponent e whatever. The number x = 0 is also denoted by $e = e_Z$ and any coefficient f whatever; this version of 0 is termed absolute zero, and is subject to different arithmetic rules than the versions with coefficient 0 and $e \neq e_Z$, any of which is termed relative zero.

The foregoing discussion shows that number representation in the coefficient-exponent system is, in general, not unique. It has been customary to take the normalized representation as standard, with special rules to cover the case of zero coefficient. Thus "floating point" arithmetic units are conventionally designed to handle operands in normalized form, and to generate normalized results. In Maniac III, by contrast, the arithmetic unit handles unnormalized as well as normalized operands in a meaningful way, and generates results whose form depends not only on the form of the operands, but on which of several types of arithmetic is called for: Floating Point, Specified Point, Normalized, or Basic. Floating Point results are adjusted according to a "significant digit" criterion; Normalized results are adjusted to normalized form. Specified Point and Basic arithmetic both generate results whose form depends on the operand exponents. More detailed descriptions of the different types of arithmetic are given in subsequent sections. It should be noted here, however, that Maniac III has no "fixed point" arithmetic which operates on numbers in pure coefficient form, as do conventional computers. Hence the usual distinction between "fixed point" and "floating point" numbers need not be made in numerical work, and no formal conversion is necessary to employ the result of one type of arithmetic operation as operand for another.

2. Numerical Results

The Maniac III operations which are relevant to the discussion comprising this and the following three sections are: Add, Multiply, Divide (Floating Point); Add, Multiply, Divide (Specified Point); Normalized Add, Square Root, Sign, Adjust, Scale.* All Add, Multiply and Divide operations can be performed with a sign option for the second operand; hence subtraction is included as a case of addition. Coefficient scaling (shifting), with or without corresponding exponent alteration, is performed by means of the Adjust and Scale operations, while the Sign operation is a conditional negation, from which negatives and absolute values can be obtained as special cases.

In this section some general considerations regarding numerical results are discussed; this serves to provide a context in which detailed features of the individual operations may be better understood.

*This does not include Basic arithmetic, which is discussed in Section 6.

Normally, two words are used to express the complete result of an arithmetic or numerical operation in Maniac III. These are developed in two central registers designated U (for Universal) and R (for Residual); the contents of these registers are denoted u and r, respectively. In some operations the result u,r is to be interpreted as u + r (with r a low order part, small in magnitude compared to u), in others as u with remainder r. In some operations the R register is not used at all, in which case u alone represents the result in its entirety. In any case, u is the approximate result generally used in further computation.

For each of the operations under consideration there is defined a "true" (arithmetically exact) result, expressible as a single number in the case of Add, Multiply, Adjust, Scale and Sign, or as two numbers, one of which is a remainder, in the case of Divide and Square Root. Furthermore, these numbers are all representable with a finite number of binary digits, and their adjustment (binary point positioning as expressed by the exponent) is uniquely determined. Sometimes, however, this "true" result is not produced by the computer. The reasons for this may be categorized as follows:

(a) The exponent of result or remainder is out of the representable range $-127 \leqslant e \leqslant +127$;

(b) The coefficient of the result is out of the representable range $-1 \leqslant f \leqslant +1-2^{-39}$;

(c) The coefficient of the result or remainder is truncated so that some of its rightmost digits do not appear.

The three effects are, of course, not unrelated; in fact, a case of (a) where $e < -127$, if it arises during the course of an operation, is generally converted to a case of (c) by successively halving the coefficient (by right-scaling, so that low order digits are lost), and incrementing the exponent until it reaches the value -127.

During the running of a program, if a final result is generated in which either (a) or (b) obtains, a trap condition is initiated which causes a program interrupt (unscheduled jump), unless specifically overridden by instructions available for the purpose. In case of (c), however, no special computer action is invoked, since the numerical error committed, unlike that of the other two cases, is small. In all three situations, however, the result actually developed by the computer can be interpreted in terms of the "true" one; in (a), all digits of the exponent except the sign are correct, in (b) and (c) the coefficient sign is the actual one, and all other digits are those which would appear in the corresponding part of the coefficient if it were to be developed in full.

A trap condition also occurs in special cases where no "true" result is defined (e.g., because of zero divisor, square root operand negative or scaling parameter out of range). In some such situations a partial or related result is generated, depending on the nature of the anomaly.

In order to discuss the operations in detail, it is convenient to extend the nomenclature as follows: U^E and U^F are the exponent and coefficient sections, respectively, of U; u^E and u^F, consisting of sequences $u^E_7 u^E_6 \ldots u^E_0$ and $u^F_0 u^F_1 \ldots u^F_{39}$, respectively, denote the corresponding contents. Similar conventions apply to R and any other registers which may be introduced. In an operation whose result is defined as u + r, the non-sign coefficient positions of R generally contain the low-order continuation of the coefficient residing in U; that is, the coefficient of the result is represented by the 79-bit sequence $u^F_0 u^F_1 \ldots u^F_{39} r^F_1 \ldots r^F_{39}$. The u + r interpretation is consistent with this if $r^E = u^E - 39$ and $r^F_0 = 0$. This rule for the exponent is the one actually used, unless $r^E < -127$; in this case, r is adjusted to exponent $r^E = -127$ as explained earlier. The u + r interpretation then still holds (although r^F may have lost some non-zero digits at the right), but the 79-bit interpretation must be modified.

The 79-bit interpretation is also affected when $r^F_0 = 1$, which occurs frequently because of the method of rounding employed in the Maniac III system. Rounding is defined as follows: Consider a coefficient sequence $f_0 f_1 \ldots f_n$ of arbitrary length n, and suppose it is desired to replace this with a shorter sequence whose last bit is in place p. The rounded sequence is taken as $f_0 f_1 \ldots f_p$ if (i) $f_p = 1$ or (ii) $f_{p+1} = \ldots = f_n = 0$; if $f_p = 0$, however, and condition (ii) is not met, the rounded sequence is taken as $f_0 f_1 \ldots f_{p-1} 1$. Thus the last bit of the rounded sequence is always 1 unless truncation involves no approximation (i.e., only bits 0 are dropped).*

*This rounding scheme has twice the spread ("variance") of the more conventional one where u^F is altered by addition of 1 in the pth place if the part dropped exceeds 1 in the (p+1)st place; however, the scheme described here also has some advantages, both theoretical and practical, over the traditional one. First, it is symmetric ("unbiased"); the mean of the numbers which round to a particular value u^F is u^F. Second, it is non-propagative; only the rightmost digit of the number u^F is altered in rounding,

It should be noted that under these rules the operations of rounding and negation commute; i.e., the negative of a rounded value is the rounded value of the corresponding negative.

In practice, a rounded result u is computed from a full result u + r by "forcing" u^F_{39} to 1 if any digits 1 have been "shifted off" into the low-order region. If $u^F_{39} = 0$ originally, this amounts to adding 2^{u^E-39} to u; if $r^E = u^E-39$, this incrementation can be compensated for by setting r^F_0 to 1, thereby subtracting 2^{u^E-39} from r and leaving the u + r interpretation valid. This step is always taken before any necessary adjustment of r for $r^E < -127$; thus it can be said that the u + r interpretation always holds, to a suitable approximation, given only that the exponent and coefficient of u have themselves stayed within range.

When r is a remainder instead of a low-order part, the interpretation is always arithmetically exact (i.e., in division, dividend = quotient × divisor + remainder, etc.), unless $r^E < -127$ necessitates adjustment of r. In the case of division, a rounded quotient u is obtained, and the remainder r compensated accordingly; in the square root case, however, no rounding takes place, and the remainder is always positive.

Because there is no normalized standard form, zero results may be expected to occur more often in this arithmetic system. Cases where one or both of u and r are relative zero as the result of the operation arise naturally, and are subject to no conventions additional to those described above. Absolute zero (exponent e_z) can occur as a result only if one or possibly both operands are absolute zero (specifically, both operands in addition, one or both operands in multiplication, one operand (dividend) in division, and the single operand in the adjust, scale, sign and square root operations). In these cases, both u^E and r^E are set to e_z, and both u^F and r^F to 0.

One further property of absolute zero should perhaps be mentioned here. In the natural ordering of computer numbers, absolute zero is taken as greater than any number with negative sign and less than any number with positive sign; the latter category includes relative zero. The effect of this convention is only apparent in the result of the (quasi-arithmetic) instructions Jump on Arithmetic Comparison and Jump on Magnitude Comparison, in which a jump condition is considered to be met if and only if $u \geqslant x$, where x is a number in memory. If u is absolute zero while x is relative zero, the condition is considered not to be met, since by the above definition $u < x$ under these circumstances.

Some of the convenient features of the Maniac III system which can be appreciated even from this preliminary discussion are:

(a) the adjustment for $e < -127$ means that small numbers never get converted into large ones through exponent underflow, which makes it unnecessary to avoid the low end of the numerical scale;

(b) the non-exceptional nature of relative zero for the system, and the consistency of the interpretation of absolute zero, permit results involving zero to be much more naturally defined;

(c) the rounding scheme, in addition to its theoretical advantages, gives rise to results which are simultaneously rounded and correct as two-word representations, thereby making it unnecessary to have an option for rounded arithmetic;

(d) in the majority of cases results have a consistent interpretation, even where they are anomalous, which simplifies the general task of the analysis and control of sequences of arithmetic operations.

3. Fundamental Arithmetic Operations

The operations Add, Multiply and Divide each require two operands; one of these is taken from register U, the other from memory. The operand from memory is first fetched to a register designated S (for Storage). The procedure then implemented can be symbolized by

$$u * (\pm s) \longrightarrow u,r$$

where * is either +, · or /, and the +,- alternative is specified in the instruction calling for the operation.

The operations designated Floating Point are defined in terms of the adjustment of their results, as follows: for Add, the result appears adjusted to the larger of the two operand exponents, while for Multiply and Divide, the result appears adjusted so that its number of leading digits m is given by $m = \max(m_1, m_2)$, where m_1, m_2 are the corresponding numbers associated with the operands.

If the value of each operand is regarded as uncertain in the 39th place, then the transmitted

---

and this both simplifies the mechanization and would seem to be advantageous in situations where rounding error does not behave like a symmetric random variable.

uncertainty in the result, adjusted according to these rules, will also be approximately in the 39th place; hence the informal characterization "significant-digit arithmetic." These adjustment rules are more than mere rules-of-thumb, however; they have desirable characteristics with regard to the propagation of errors of arbitrary magnitude, and their formal consistency also recommends them (see Reference 1).

Exception is made to the rules as described when adjustment is necessary to keep the coefficient within range, or to avoid the exponent condition $u^E < -127$; the required adjustment in both cases involves a right-scaling of coefficient, which is in the direction of decreasing apparent significance. Adjustment for $u^E > +127$ requires a left-scaling of coefficient; even if this would not take the coefficient out of range, it would increase its apparent significance, and so the step is not undertaken. The case therefore causes a trap condition.

Relative zero operands in multiplication and division are handled in a way consistent with the interpretation m = 39; the result is always zero with an appropriate exponent (even in the case of relative zero divisor, which is a trap condition). Absolute zero operands, however, act like the true zero of the number system, obeying the rules

$$x + 0 = 0 + x = x;$$

$$x \cdot 0 = 0 \cdot x = 0;$$

$$0/x = 0;\ x/0 \text{ undefined (trap, no operation).}$$

When absolute zero is produced as a result, it is always with coefficient 0, regardless of the coefficients attached to the absolute zero operands involved.

The multiplication and division operations described above have the property that if they are performed on normalized (non-zero) operands, the results are produced in normalized form also. (An exception, of generally trivial consequence, arises for a result which is exactly a negative power of 2, since the m = 0 form is not normalized in this case.) Hence only one further operation, Normalized Add, is needed to permit calculation to be carried out fully normalized in standard fashion. Normalized Add gives a normalized, rounded result regardless of the original adjustment of the operands. The only exceptions occur when continued adjustment by left-scaling fails to normalize the coefficient before $u^E = -127$ is reached (this includes the case of relative zero result). Such an unnormalizable result always gives rise to a trap condition, which permits contingent procedures to be defined by program. However, an absolute zero result, which can only occur through addition of two absolute zero operands, does not lead to a trap condition.

The third type of arithmetic to be discussed is Specified Point, which supplants the conventional "fixed-point" operation. Here Add, Multiply and Divide are all three subject to the same type of rule: the result is adjusted so that its exponent is equal to the exponent of the first operand (that originally residing in U). The only exceptions occur when absolute zero operands are involved, in which case the behavior is the same as for the floating point operations.

Following the specified point rules obviously never leads to out-of-range exponents, but the required adjustment may cause the coefficient to exceed its permitted magnitude. When this happens the sign and remaining bits which do appear are correct (i.e., are part of the true result) as mentioned earlier; since high order digits have been lost, however, a trap condition obtains.

4. Further Numerical Operations

The operations Square Root, Sign, Adjust and Scale are not explicitly characterized as to arithmetic type, but their definitions reflect the general arithmetic philosophy outlined above.

Square Root is performed on an operand fetched from memory to S, according to the formula

$$\pm\sqrt{s} \rightarrow u,r.$$

The result in U has the same number of leading digits as the radicand, and is not rounded; this allows the remainder in R to be treated as meaningful (i.e., (square root)^2 + remainder = radicand). Square Root thus has the character of a floating point operation. If the operand is absolute zero, both the result and the remainder are absolute zero with coefficient 0. If the operand is negative, a trap condition occurs; in this case, however, a square root is still computed, that of the operand negated.

The Sign operation is defined as follows: an operand is fetched from memory to S, following which one of two alternatives is effected; either

$$\pm s \rightarrow u$$

if $u \geqslant 0$ originally, or

$$\overline{\pm} s \rightarrow u$$

if $u < 0$ originally. The +,- alternative is specified in the instruction, as usual. If the operand is absolute zero, then the result is absolute zero with coefficient 0. One use of this "conditional negation" is to form absolute values; if the operand is taken as u itself (possible because U has a memory address in Maniac III), then the + alternative effectively performs

$$|u| \longrightarrow u,$$

while the - alternative performs

$$-|u| \longrightarrow u.$$

The contents of R are undisturbed by this operation. The result normally takes the exponent of the operand; adjustment is made, however, for out-of-range coefficient (which can only occur on negation of -1). The Sign operation may therefore be classified as floating point.

The Adjust and Scale operations are two versions of "arithmetic shift." They both basically involve a single operand and a parameter n in the range $-128 \leqslant n \leqslant +127$. In Adjust, the operand is fetched from memory to S, transmitted (positively or negatively) to U, and then adjusted by adding n to u^E and scaling u^F (with r^F) accordingly. Thus, if $n > 0$ the coefficient is scaled to the right, (successive halving), and if $n < 0$ it is scaled to the left (successive doubling). The numerical value of the number u is unchanged, except insofar as digits are "shifted off." The full result is u + r, with u rounded in the usual fashion. If the operand is absolute zero, then the final values of both u and r are absolute zero, with coefficient 0.

Exceptions occur when the performance of the operation as defined would force the exponent or the coefficient out of range. In these cases the floating point conventions are followed; the operation is carried to completion in the face of $u^E > +127$, but suspended otherwise; either case gives rise to a trap condition.

Scale is similar to Adjust except that the exponent is not altered; thus the operand is effectively multiplied by a power of 2,

$$2^{-n}(\pm u) \longrightarrow u,r.$$

The scaling process is carried out completely, even if this causes loss of high-order digits at the left. Thus Scale resembles a specified point operation, in contrast to Adjust.

The parameter n is defined in a computer instruction as a 14-bit integer; however, unless the leftmost 7 bits are identical (i.e., unless $-128 \leqslant n \leqslant +127$), no operation is performed and a trap condition results.

With n = 0 and the - alternative, either Adjust or Scale acts as a negation operation; they differ in that the former keeps the coefficient -1 in range by right-adjustment, while the latter does not.

5. Trap Conditions

The trap conditions mentioned in the foregoing discussion are classified into four categories, as follows:

(1) Coefficient Trap--the coefficient has gone out of range, and digits have been lost at the left (specified point type operations).

(2) Exponent Trap--the exponent has gone out of range, causing its sign to change spuriously (floating point type operations, range exceeded positively).

(3) Parameter Trap--the scaling parameter n has been specified outside the range $-128 \leqslant n \leqslant +127$ (on Adjust and Scale).

(4) Operation Trap--the operation has not been completed according to its definition (divisor is zero, or radicand is negative; result is unnormalizable, or adjustment cannot be completed).

These are collectively referred to as arithmetic traps, to distinguish them from other kinds of trap conditions which can occur in the course of computer operation (namely, instruction traps and real time traps). There is a fifth category of arithmetic trap, reflecting a feature of the Maniac III register structure which has not yet been mentioned: Each of the registers U, R and S has one extra digit position, designated U_X^F, R_X^F and S_X^F, respectively, which may be considered to lie immediately to the left of the sign position (coefficient position 0). This means that the actual range of coefficients these registers can hold is $-2 \leqslant f \leqslant +2-2^{-39}$. In the operations discussed thus far, these extra positions are used for internal convenience only, and a result with $u_X^F \neq u_0^F$ (which is the condition for u outside the normal range $-1 \leqslant u \leqslant +1-2^{-39}$) can never be generated. The Basic arithmetic operations discussed in the next section, however, can leave U in the condition $u_X^F \neq u_0^F$, and it is to protect against the initiation of a non-Basic operation under these circumstances that the fifth trap condition is introduced:

(5) Segment Trap--the condition $u_X^F \neq u_0^F$ obtains at the start of a non-Basic operation which uses u as an operand.

Each arithmetic trap condition, when it occurs, causes a corresponding trap toggle (1-bit register) to be set to 1. At the completion of the performance of the instruction which gave rise to the trap condition, the following instruction in sequence is fetched and examined by the computer control. If this one of five special Disable instructions, the corresponding trap toggle is reset to 0, and the next instruction is fetched in turn. This allows the effect of any anticipated arithmetic trap condition to be

nullified by the computer program.

If any trap toggles are still set when a non-disable instruction is finally encountered, that instruction is not performed, but instead the instruction residing in a special trap location is fetched and executed. At this time the remaining trap toggles are all reset to 0. However, the presence of each trap condition, whether nullified or not, is recorded in a corresponding trap indicator (1-bit register), by setting it to 1 if it is not already in that state; this occurs immediately following the original arithmetic instruction, before any Disable instructions have been acted upon.

The instruction in the trap location may be a Jump instruction, specifying a new control address, in which case the trapping has effectively caused the insertion of a special routine into the program sequence. This routine may use the information recorded in the trap indicators to initiate remedial procedures.

If the instruction in the trap location is not a Jump, control reverts back to the original point (i.e., to the instruction about to be performed before the interpolation of the special instruction) after its execution. The trap indicators remain set in whatever configuration they may have achieved, and so are available for use in the main program.

This description of the essential features of arithmetic trapping has ignored complications which arise due to the possible simultaneous occurrence of trap conditions of the other types (i.e., instruction and real time). A full development of these complications is irrelevant to the purposes of the present discussion.

6. Basic Arithmetic

Still another category of arithmetic operation is to be included in the Maniac III repertoire. Basic operations are so called because they employ exponent conventions which are in some sense "natural"--in particular, the exponent of a product is the sum of the exponents of the factors, and the exponent of a quotient is the difference of the exponents of dividend and divisor. However, these operations differ fundamentally from those discussed earlier in that they may involve two-word operands, and that results are defined in terms of the extended register structure (X position included). The operations are intended specifically to expedite the performance of multi-precise arithmetic, although they are useful in other contexts also.

Multi-precise operands are assumed to be defined by a sequence of 39-bit segments, possibly signed (as e.g. are produced by floating and specified point operations). It is anticipated that either of two exponent conventions might be appropriate: each segment might carry the exponent of the entire number, or this exponent might appear only on the highest order segment, with the exponents on lower order segments decreasing by 39 at each stage (results of ordinary arithmetic operations obey the latter convention).

The category of Basic operations may be considered to include Basic Add, Basic Multiply, Basic Divide, and an additional operation designated Round to Nearest Value. The properties of these operations will only be sketched here. In each case, one word is fetched from memory, and this is combined with the contents of U and possibly R in a prescribed way to produce a double-precise result, one word of which is then stored in memory.

In Basic Add, the result is a double-precise sum, the high order part of which is retained in U while the low order part is stored in memory. The effect is to add the memory operand to the operand initially residing in R and produce a sum adjusted to the larger of the two operand exponents (possible overflow being taken care of by the extended register structure); to this sum is added the number originally in U^F, scaled as if it carried an exponent 39 less than that of the intermediate sum. Two multi-precise numbers can be added by a succession of such operations; the final contents of U^F at each stage are carried over and applied to the addition of the next higher pair of segments. (The exponents of the two operands must not differ by more than 39 for this procedure to be successful.) If the operand exponents are identical, no scaling of segment coefficients need take place; under these circumstances, the carry from each segment addition is recorded in U_0^F, and U_X^F records the sign of the sum. The instruction Basic Add includes the usual +,- option, so that it serves for subtraction, and it also has an option for initially clearing U, to be used in cases where no lower order segment is defined.

In Basic Multiply, the product of the operand initially residing in R and the operand from memory is formed, with exponent equal to the sum of the exponents of the factors; to this product is added the number originally in U^F, scaled as if it carried an exponent 39 less than that of the product. The product of a single-precise multiplicand and a multi-precise multiplier can be formed by a succession of such operations, the high order segment of the product at each stage being the increment added to the low order product at the next stage. In combination with addition this procedure can be used to form products of two multi-precise factors. The instruction Basic Multiply includes a +,- option for the multiplier and an option for initially clearing U.

In Basic Divide, the operand initially residing in R serves as divisor, and the dividend is a two-word operand with the contents of U as

high order part and the number from memory as low order part. Before the start of the division proper, however, the contents of R are normalized; the exponent of the quotient is then the difference between the exponent of the dividend and the exponent of the normalized divisor. The quotient of a multi-precise dividend and a single-precise divisor can be formed by a succession of such operations, the remainder at each stage being used as the high order part of the dividend at the next stage. To form quotients with the divisor also multi-precise, the division operation is used to generate quotient digits, but multi-precise remainders are computed by multiplying and subtracting (remainder = dividend - quotient × divisor). To simplify the implementation of this procedure, the instruction for Basic Divide has a +,- option which applies to the remainder, as well as the standard one for the divisor.

The Round to Nearest Value instruction combines a high order segment initially residing in U with a low order segment fetched from memory and introduced into R. If the low order segment is in the range $-1/2 \leqslant f < 1/2$, the high order segment is left unchanged; otherwise 2^{-39} is added to or subtracted from the high order segment, as appropriate, and the sign of the low order segment in R is reversed. This may be regarded as effecting a "traditional" rounding operation on the number in U; in addition to its normal uses, such an operation turns out to be useful in the performance of multi-precise division (applied to the normalized divisor initially, it insures that the quotient segments will be in range).

The definition of the Jump on Arithmetic Comparison and Jump on Magnitude Comparison instructions mentioned in section 2 are framed in terms of the extended register structure, thereby permitting the result of a basic operation (in U) to be arithmetically compared with a number from memory (which, however, is necessarily in the normal range).

7. Computer Implementation

The implementation of the rather elaborate arithmetic system described is facilitated by several basic structural features of the Maniac III computer. Transmission between any pair of the central registers U, R and S is efficiently realized by means of a universal communication bus called the "central distributor," and this bus is partitioned so that the exponent and coefficient sections can be transmitted separately. Scaling of coefficients and/or incrementation or decrementation of exponents by 1 are both processes which can be carried out independently and concurrently in these registers when no transmission among them is called for. Finally, both the coefficient and exponent sections of all three registers have an extra stage (for coefficients, it is the X position already mentioned), which in many cases allows operations leading to out-of-range numbers to be completed, and the necessary corrections or invocation of trap conditions to be made subsequently.

Supplementing the advantages offered by these structural features, considerable effort has been put into organizing the arithmetic procedures into a pattern of uniformity, to permit their more efficient execution. Thus, for example, the three non-Basic Add operations (Floating Point, Specified Point and Normalized) are all realized by a single micro-program in the computer control, in which the variations between arithmetic types are taken into account by making the performance of certain steps conditional. A similar remark holds for the non-Basic Multiply and Divide operations. For all non-Basic arithmetic, a single pattern of end-connections between registers U and R suffices; so, for example, whenever a call is made for both these registers to be shifted right, a connection is automatically set up to effect $u^F_{39} \longrightarrow r^F_0$, and to conditionally effect $1 \longrightarrow r^F_X$ if and only if $u^F_{39} = 1$. The latter step is for the purpose of recording whether or not any digits 1 have been shifted out of U into R, which is the condition for forcing u^F_{39} to the value 1 at the end of an Add or Multiply operation (for rounding).

No attempt will be made here to give details of the procedures for realizing the operations. The interested reader may find some of the notions involved set forth in Reference 2.

References

1. Ashenhurst, R. L., and N. Metropolis, "Unnormalized floating point arithmetic," Journ. ACM, vol. 6, no. 3 (July 1959) pp. 415-28.

2. Metropolis, N., and R. L. Ashenhurst, "Significant digit computer arithmetic," IRE Trans., vol. EC-7, no. 4 (Dec. 1958) pp. 265-7.

AN ORGANIZATION OF AN ASSOCIATIVE CRYOGENIC COMPUTER

Robert F. Rosin

Information Systems Laboratory

University of Michigan

Ann Arbor, Michigan

Prepared under Contract AF 33(657)-7391. Part of the work reported was performed by IBM at the Thomas J. Watson Research Center, Yorktown Heights, New York, in cooperation with the Bureau of Ships under Contract Number 77508 (Project Lightning).

Summary

The design of a total computing system built in the cryogenic medium and with an associative memory is discussed. Topics include instruction sequencing, addressing, indexing, input-output, multi-programming and system programming considerations. The iterative nature of the design of this computer, removing the necessity for much special-purpose hardware, increases its reliability. The associative memory makes it a very flexible and powerful computer.

Introduction

There exist in the literature several articles describing various aspects of computers equipped with cryogenic circuitry and associative memories. It is the purpose of this paper to assemble many of these ideas and several others into a coherent study of the total system organization of an associative cryogenic computer. The ideas are not sufficiently refined to constitute a complete plan for a machine but they do reveal some of the problems and opportunities which would arise in such a project.

The motivation behind this discussion of the organization of an associative cryogenic computer lies in two considerations. Due to the ease of designing relatively simple associative memory cells in the cryogenic medium, the latter appears to be one of the most economical media in which to consider building an associative computer. Correspondingly, as will be discussed more fully later, it is advantageous to build cryogenic computers of uniform logical blocks avoiding special circuitry if at all possible. This is due to the difficulties which will exist in the maintenance of a computer in a cryogenic medium. The design of an associative machine meets this criterion adequately. Thus, the justification of the topic of this paper.

Cryogenics has forced us to reconsider what we currently feel are normal machine functions and control relationships. Primarily, a computer built entirely of cryotrons would exhibit similar cycle times in the storage unit, the arithmetic-logical unit and the control unit. In most contemporary machines, functions such as indexing and operations on special registers take place in a length of time relatively much shorter than memory access and rewrite. Indirect addressing is relatively rapid compared to add time in a contemporary computer. In a cryogenic computer, especially one equipped with associative memory, this is no longer true to the same degree. While retrieval and store times will be slower than most control functions (due to long line length and the accompanying relatively slow switching time) the relative difference will be much less than that which we are used to considering.

The concept of an associative store also implies some new thoughts with respect to classical organization.[1,2,3,4,5] What we shall call an associative memory has been defined as "memory from which a word of data is retrieved on the basis of part or all of the data contents of the word." Thus, in what may be called a fully associative machine, there is no address decoder per se since words are not identified by numeric addresses but by their contents. It is customary to denote part of the contents as the "tag" or name of the word, and the remainder as the contents or value. For example, a 48-bit word might be broken into two 6-bit characters of tag and a 36-bit value. When retrieving a word from the store it is important that the portion of the word not to be used in comparison with the criterion be ignored or masked during the retrieval.

While it is not the purpose of this paper to review previous work in cryogenic memory circuits, a few new concepts should be discussed. Figure 1 shows two sets of memory cells; those of the persistent current loop variety (shown on the right in the figure) have been previously described in their many configurations.[6,7,8,9] The chain bit, shown on the left, was recently disclosed by Seeber.[2] We shall further modify this cell for our own purposes. In principle, this is a two-state device, the present state denoted by that branch which contains current. The source of current is at the upper left end or zero state side of the chain. A resistive (non-superconducting) cross branch anywhere in the chain will force current into the righthand branch from that cell on down; hence these cells are considered to be in the state 1. For consistency, the cross branch will also be considered state 1.

Figure 2 shows an association cell con-

structed from a chain bit. Notice that "read" current will pass through the cell only if there is current in the cross branch. Note also that this case can arise only when this cell is the uppermost with association current passing through. Thus this bit position in memory selects the first of the words positively associated with the criterion to be matched.

A more complicated cell is shown in figure 3. This is the sequence chain bit. It allows sequencing of the zero side down one word upon reading of the cell if desired. This type of bit, also called a "marker" bit, is central to our discussion and will appear in several places throughout this paper.

The criteria which have guided the specification of this machine (and might be applied to the design of any other machine) are the following:

1) it should be able to execute a well-written program efficiently,

2) it should be able to translate a program written in some higher level language into its own machine language efficiently,

3) it should be able to carry out input-output operations efficiently, and

4) it should be able to interrupt normal program control to handle exception conditions.

The nature of associative storage has influenced the application of these criteria.

The Computer

It is appropriate to begin the description of the associative cryogenic machine with the internal word formats, of which there are two: data words and instruction words.(Fig. 4) In this regard, it should be noted that serious design criteria helped determine these formats, but they are open for appraisal and change.

Data words consist of 25 control bits (an arbitrary figure which should be adjusted to fit the final machine design), a 48-bit tag field (six 8-bit characters), and 64 bits of data along with 3 associated flag bits, giving a total of 140 bits. The function of the control bits will be described as the paper proceeds. The instruction word format consists of the same 25 control bits, a 48-bit address part, 2 bits for addressing mode (indirect and immediate), a 15-bit operation code, a 48-bit index address and its 2 associated addressing-mode bits. Each format should carry its appropriate checking information appended to the right-hand end of the word. It will be noted in the figure that the tag field of the data word and the address part of the instruction word occupy the same field of the basic word.

The nature of this machine's design in conjunction with the operational characteristics of chain bits places one restriction on the location of data and instructions in the store. They may be mixed to any degree desired; blocks do not have to be contiguous, but all blocks of data and instructions must occupy sequentially ordered locations in the store. For example, instructions, which appear in a block and are to be executed in a linear sequence, must appear in the order in which they are to be executed, although data words may appear interspersed anywhere in the instruction array. The same rule applies to blocks of data where indexing is to be used for individual word retrieval.

Instruction Sequencing

The six-character tag field seems adequate for all data and storage names. It should be noted that longer names require more bit positions in every word in storage. Examination of the formats presented above shows us that instruction words do not have space for their own tags, that is they have no names. In the average large scientific program, of let us say 32,000 words, about 80% are data and the rest instructions. Of the instructions in the program at most 1/4 are referred to by name. This amounts to, at most, 1600 words. If we were to lengthen the machine word by 48 bits to accommodate a tag part for each instruction, we would be adding over 1.5 million bits to our 32,000 word machine. If, on the other hand, we stipulate that every instruction which we wish to refer to has a data word between it and the previous instruction and that this data word has a unique tag, then as will be shown, control can be transferred to this tag.

Let us discuss instruction sequencing and introduce two of the control bits. They will be called the instruction bit (called the IB and of the persistent current loop type) and the instruction-used bit (called the IUB and of the sequence chain type). Each word in storage which is an instruction will have its IB set to 1, all other words will have their IB set to zero. This will be the duty of the translators and loaders used with this machine. The IUB's of all words above the next instruction to be executed will be set to 0. The IUB's of all words, including the next instruction to be executed and below, will be set to 1. To retrieve the succeeding instruction, the machine retrieves a pattern of 1, 1 for the IB and IUB respectively. The other bits are ignored (masked out) for the retrieval. (Masking will be discussed later.) The first word from the top in storage which has this combination of IB and IUB is the instruction sought, and will be the only one positively associated. This instruction will be read into the decoder to be executed and the IUB's for all words down to and including this word are set to 0. This is well and good for normal processing, but what about transfers both forward and backward in storage, especially when instructions do not have names? To transfer control to an instruction,(Figure 5) the machine retrieves the tag given in the address part of the instruction, and sets all IUB's above the location to 0 and all IUB's below that location to 1. Normal instruction retrieval (IB = 1, IUB = 1) now yields the desired instruction. To address an instruction as data, one of

the group of control bits called "marker bits" is activated and sequenced in the same manner as the IUB, and retrieval is made on the basis of the IB and this bit position being 1. The data word which is used as an instruction transfer pointer can also contain data with no restrictions.

Subroutine processing requires still another chain sequence control bit. We shall call this the subroutine argument bit (SAB). Whenever a subroutine transfer is made, the SAB is set to 1 below this point in the same manner as the IUB is set in an ordinary transfer. (Figure 6) When the subroutine takes over a programming, restriction forces it to retrieve all of its arguments, which will occur as instructions, immediately below the point from which the transfer was made. This can be done easily by retrieving on the basis of the IB = 1 and the SAB = 1. The reason underlying this requirement is that if the subroutine in use calls on another routine, the SAB is then used to retrieve the arguments for the new routine, destroying the previous SAB setting. This scheme allows for saving of subroutine exit histories and intermediate arguments and results so that subroutines defining recursive functions may easily be written. The first argument should always consist of the name of the instruction to which control is to be transferred after completion of the routine just entered. Alternate exits should likewise be included. The arguments for the routine are then picked up and transferred to suitable locations within the routine itself so they may be addressed without need for address modification. Each time an argument is retrieved on the basis of the SAB, the SAB's are reset down through the previously retrieved word. If the instructions contain names and not values, as is likely, then the locations to which they have been taken can be indirectly addressed, allowing very flexible argument addressing.

Interrupts create still another problem. They also imply subroutine action, but they can occur during the execution of any part of the program. To enable the machine to restart after an interrupt, two more bits, called the exception bit (EB) and exception-used bit (EUB), are named in the set of control bits. These are identical in type and operation with the IB and IUB. When an interrupt occurs, control is transferred to the correct routine and instruction sequencing is then carried out based on the EB and EUB, and not the IB and IUB which are preserved so that normal operation may commence from the exact point at which it was stopped. If an interrupt occurs during the execution of an exception routine, it is remembered or acted upon based on the priority it holds over the one being processed. If a more recent interrupt has higher priority, the earlier one is recorded and the exception routine pertinent to the later one takes over.

Addressing

Let us determine what the term indexing means in the context of this machine. Since our blocks of data are not necessarily contiguous, and since we have no address decoder, indexing must be viewed in a different light. An index, i, enables us to retrieve (or store at) the ith occurrence of some tag Y. (The tag could be the tag of the first unused instruction (IUB = 1) so that we can transfer to "this instruction plus five," etc.). There are two convenient schemes. The first is to establish conventional index registers which can be addressed. The number needed is not determined, but they would operate in conventional fashion except that when used to modify an address, the contents of an index register would be loaded into a counter and the tag would be retrieved until the counter hit zero, marking each retrieved word by setting a marker control bit to 0 and using a 1 in that location as part of the tag (not masked out). At final retrieval the marks would be reset to allow future indexing. An alternative scheme is to allow the marker bits to remain, thus allowing relative indexing.

But, instead of using special index registers, retrieving any word in storage and placing its contents in a counter as just described achieves the same result. Modification of the word is achieved by an add (or subtract) one in storage circuit [7] or by performing actual arithmetic on the word in the arithmetic unit. The latter is realistic in this sort of machine. The time needed to access the word can be made to overlap the other accesses and will really not be much slower than retrieving a word from an index register, because access time to registers is not significantly faster than from main store in a cryogenic computer. An enormous advantage gained in the technique just described is that every word in the store is effectively an index register.

As mentioned earlier in our description of word formats both the address part and the index address of any instruction may be interpreted in any one of three modes: direct, immediate and indirect. Direct addressing and indexing are described above. Relative addressing is a form of immediate addressing in which the index is interpreted as immediate while the address part is interpreted directly. Immediate addressing causes no problems since all that is involved is using the literal value of the address and/or index parts of an instruction instead of retrieving on them. An instruction with an immediate address and normal or indirect index field implies the addition of the final index value to the immediate address before the latter is used as an operand value.

Indirect addressing does create a few problems. To allow complete flexibility with respect to indexing, the object of an indirect addressing must be an instruction. Let us consider the case of an instruction with an indirect address and no index. Since the address is marked indirect, the machine retrieves on the tag field finding the desired tagged word and sets a marking bit to one down through that location.

Retrieving on the basis of that marking bit equal to one and the instruction bit equal to one yields the first instruction following the data word with the appropriate tag. The address and index fields of this retrieved instruction now become the addresses to be used with the operation code of the original instruction, provided indirect addressing is not implied by the appropriate bits of the new word. If indirect addressing is implied, then the process described above is repeated.

To avoid complications, the index part may be indirectly addressed down through only one level; that is, if an index part is marked with an indirect bit, the location retrieved indicates the location of the index value to be used and may not be another word of indirect addressing. However, the word retrieved after retrieval based on the address part may initiate another complete indirect retrieval. If this restriction were not made, an arbitrarily large number of special registers would have to be created to retain information pertinent to the various levels of addressing involved in one complex indirect retrieval. It should be pointed out that here, as in indexing, these operations take considerable time, each level of indirect addressing requiring at least one associative retrieval.

Interestingly enough, as one contemplates using such a device as we are describing here, he is confronted with new and strange concepts which for one reason or another never occurred in the use of previous machines. Suppose one has a block of data, tagged by the symbol X, stored throughout the machine, not contiguously, but in some definite order. The operation store in X is to be executed. Which X? The first, the last, a new location? This leads us to the use of two types of store operations; the first uses the tag as the name of the existing location in which to store the information. It can be indexed as can any other tag. The second type of store is different from any ever used in a computer. It says, create a new location with this tag and store the appropriate data in that location. Assuming that the tag is that of a previously created block, which new location is to be used? Can we use one which occurs somewhere in the middle of the existing block? Probably not. It is important to retain the block ordering in most problems even if the elements are not stored in a contiguous manner. For example, if a sentence of English were to be translated into Russian and we stored it under one tag, and then decided to add to its tail, it would be of no use to store the additional material within the limits of the block. Confusion would be the only result. To enable us to get out of this difficulty, we establish a set of rules and a pair of new hardware features. The rules say, load the program and data in one contiguous mass, with no intervening available locations allowed (although empty tagged locations are certainly permitted). Whenever a location has served its usefulness and it is desired to remove it from consideration, set its availability bit (AB) to 1. All retrievals are made with the tag having availability bit set to 0, and all locations which are unused but at the tail end of the store have their availability bits set to 0 as do all initial program and data words.

It is possible to attempt to store information in a new location and find none available. To permit the locations which have their availability bits set to 1 to be used would result in confusion, but the presence of these unused locations is tempting. If storage is attempted and no location is available, we shall determine that this results in an exception condition and an appropriate automatic interrupt. At this time, storage is read out onto some I-O device (hopefully a high-speed disk) being retrieved on the basis of a zero in the AB. Thus only active words will be read out. The device then writes its contents back into storage, but in consecutive locations, so that all unused words effectively disappear. The AB positions are all set to 0 and operation of the program is restarted where it left off. In the event of two consecutive failures in attempting to store in a new location, the program is obviously too long or hardware is at fault and appropriate action is taken.

Sequencing Control Summary

Our machine has no location counter, but substitutes a set of bits in the control part of the word for this feature. The instruction format allows two full addresses: one for the data itself, and one for an index to modify the data tag. Either or both of these may be indirect, immediate or direct, depending upon the setting of four bits associated with each instruction. Indexing is seen in a new light and another control bit (the marker) is provided to facilitate this operation. Blocks of storage cause some problem since their order is important but they have no assigned limits. An additional control bit, the availability bit, is used to control storage availability and prevent the disordering of stored arrays.

Input-Output

Control word I-O as implemented on the 709 and 7090 seems ideal for this type of machine. What is needed is another "counter" which will mark those words "retrieved for output" or "used in input." This will take the form of another chain sequence control bit called the input-output bit (IOB). It will function exactly as does the IUB and its equivalents except that it is under control of the input-output exchange and not the central processor. Using proper tag control, individual words, blocks of data, blocks of instructions or the whole memory can be read or written into. Indexed input and output provide some problems, but they are surmountable. Linear blocks are read in, either in new locations or into words with previously created tags, in a manner analogous to simple store operations. Reading them in backwards provides a problem,

but this can be met by having the control process the IOB sequencing in the inverse direction. Picking up scattered words is messier since the use of a sequence-type bit demands linear order, but subroutines which establish a proper control word sequence can be produced with no exceptional difficulty.

The problem of faulty IO transmission can be met by using the same control marker bits which were used to mark the transmitted words. If a check sum is in error, the routine will have access to the words in storage and can either erase them (make them available) if they were read in, or reread them if they were read out. At the end of a successful IO operation, the bits are set back to 0 before starting a new control word. If, during a write sequence under IO control, there exists a situation in which no locations are available, the machine should interrupt as usual, but retain all of the control information in a frozen state while the remedial read-out write-in routine described earlier is performed. This implies that we have a machine which is at least two machines in one in that either one can interrupt the other in certain situations.

Multiprogramming Considerations

Multiprogramming, which has become a desired feature of new systems, can be accomplished in a very straightforward manner in a machine of our design. Suppose we assign eight of the control bits to a special heading character. Each processing unit which is to have access to the store automatically applies its heading character to the proper area during retrieval. If concurrent operation is desired, extra read and write lines can be provided. Each processor requires its own IB, IUB, etc. Certain routines can be common to several programs and data areas can be also. Data areas can have their heading character changed so they may be, in a sense, transferred to another processor. Interprogram protection is provided by these characters, and words used by several programs can be mixed indiscriminately with no harm. Common subroutines must make themselves available to only one main routine (contained in a separate processor) at one time. The executive routine must provide priority scheduling to determine which routine can force another out of a subroutine temporarily for its own purposes.

In systmms of multiprogramming where simultaneous processing units do not exist, the system proposed above will still suffice. If the executive routine is given a message saying that program A is to supersede program B then the active heading character is changed and processing commences on B while A sits idle. If A is not needed its words may be marked by the AB and the storage can be compressed at a later time.

Table Look-up

Table look-up as a fundamental operation implies the following in a machine of contemporary design. Given an argument, find a corresponding entry in a table (either an equal entry, or one greater than or equal, etc.) and from a parallel table derive a result on the basis of the location of the original argument in the first table.

Central to this concept is the ability to compare an argument with a number held in storage and return a signal depending upon whether the result is high, low or equal. Circuits to accomplish equal comparison exist as part of an associative store.[2] The programmer merely must adjust the retrieval mask to allow for the argument as well as the tag. High and low comparisons are not so easily achieved, but are feasible.

To facilitate the implementation of some of the ideas presented on table look-up and retrieval of data based on complicated criteria, a system of marker bits would be most helpful. For example, to find the first ")" after a letter, it would be sufficient to find each ")", marking it and all words below it, and then do an indexed retrieval based on the marker bit condition desired. A set of machine commands to enable these operations should be investigated.

A look at some of the ideas mentioned above leads one to believe that retrieval might not only be indexed, but might be programmed to set the index counter to the index of the word in which the desired argument is found as in table look-up. This could be a slow operation since retrieval is normally a parallel operation. This looks like a macro which retrieves the tag and then tests the result, adding one to the counter each time the operation is performed.

Mask Control and Retrieval Tags

We have alluded to two types of retrievals; one in which the hardware of the machine determines the fields to be retrieved upon the instruction contains the desired tag allowing the instruction sequencing control to set up the necessary mask and condition, and the other in which the programmer determines the masks and fields to be used. It must be possible for the programmer to have access to any part of a word without too much difficulty, especially since he must be able to program input and output of full words including control and marker bits. This is not so simple as it sounds since ordinary processing is concerned with the right-hand part of the word (excluding the control bits) only, and for this sort of work there is no need for access to these areas.

A solution to this problem is found in the fact that seldom, if ever, is more than the body (less the control bits of a word) used for tagged

retrieval. In our system we shall stipulate that this is never possible. Retrieval based on uncommon tags and masks (such as to find the location of the first word which has a certain marker bit set to 1) can be accomplished in the following way. Three instructions should exist in the machines: one which specifies the word used for retrieval tag, one specifying the word used for masking purposes, and the third commanding the actual retrieval along with an appropriate offset to the left to be applied to the mask tag combination. Bits to the right (shifted out of) are automatically masked out, as are bit positions to the left which are not shifted into. In actual use, these instructions would be combined into macro instructions to enable the programmer to handle special functions based on the control bits as if they were hardware operations built into the device. The operations would also be available for the programmer to use to build up his own macros.

In normal operation, the rightmost 115 bits of the word can be retrieved as a group and can be conveniently checked for errors. If the proper number of checking bits are provided, error correction may be implemented very easily. However, the chain bits create a problem here too. Since the chain bits may change in storage without being brought into the arithmetic unit, forming parity sums, etc., is quite difficult. For this reason, checking and correcting bits would not apply to chain bits. If a word in store is found to be defective, a control bit is set to zero. This bit is called the disabled bit (DB). It consists of a persistent-current loop which always contains a binary 1. If a word is found to be faulty, this bit is set to 0 by forcing enough current through the loop associated with the bit to fuse the metal in the loop so that it may never be reset. All operations in the machine will be masked so that any word used must have a binary 1 in this bit position. Since words need not be stored contiguously in an associative store, the memory is still functionally the same as before except there is one less word available for use. When a memory unit contains too great a number of words which have become marked with an 0 in the DB then the entire segment of the storage must be replaced. A routine which reads this bit and counts the number of occurrences of an 0 in the DB location will determine when this condition is met.

Systems Programming Considerations

The two most important functions which a translator must perform are scanning and table building. This is true independent of the type of translation; assembly, compilation of macro or algebraic language, interpreter translation, etc. Table building is a straightforward task in an associative computer because each table entry is accompanied by the table name. Multiply dimensioned arrays can be built using the list-of-lists technique. The scan is a different matter, but the next few thoughts should explain how it can be easily accomplished in a machine such as we are discussing. A full scan usually implies examining the source input for functional groups, extracting these and adding them to the proper tables, and then performing some sort of translation on all or part of the material and inserting the result into further tables. For example; in a compiler the scan might isolate all variable names, function names, statement labels, operations, etc., tabulate these and then put out a list of intermediate machine operations which would be later translated into machine code. The problem of assigning locations and regions to variables is nonexistent in an associative-store machine, so there is not too much of a problem in putting out machine code after the first scan. The problem is that one must be able to write a program which can isolate functional units in an easy fashion; look for delimiters, other punctuation and operations. Fortunately this provides no great problems.

If the words contain only single characters as suggested then it is a simple task to retrieve not the ith occurrence of X, but the first or ith occurrence of X containing "," followed by ")" is not so simple and must be programmed by finding the index of the comma and determining if the next character is the ")". Other such complicated detections are similarly programmed using marker bits. Finding the first ")" preceded by a letter is not simple if each letter must be compared with the retrieved character. For this reason, a character set might be devised in which letters, numerals, punctuation, and special characters could be distinguished on the basis of two common bits. The mask control and the proper retrieval criterion could enable this task to be programmed and performed easily.

The question arises as to how one would implement a list system in an associative cryogenic store. In one scheme, marker bits are kept to indicate the sequencing of words in a sublist. Tags would have to be provided to serve as branch points. The mode of sequencing would determine the masks to be used during retrieval as affecting flag bits and other parts of the data word. Note that the existence of an availability table is unnecessary.

An alternate scheme is to give each word a distinct tag and to implement a scheme similar to that used by Newell, Shaw and Simon, except in a more general way. In this case a partial mask is used on the tag field during normal retrieval but removed for finding successor and predecessor elements. The words tagged with augmented information contain the tags of the first preferred immediate successor and immediate predecessor elements, the tags of the second preferred, etc., one word for each level of successor and predecessor tags. Alternatively, the words associated with one data word could have the identical tag and the ith retrieval could retrieve the (i-1)th level of successor-predecessor tags when used with proper marking control.

It might be pointed out that common numeric

addressing is possible using associative techniques. In this case a tag field is devoted to the numeric addresses which are fixed for all time and used associatively to find the word with the numeric tag of n. This is a slower scheme than others but provides some generality in an associative store since it is easily simulated in such a device.

Conclusions

There are arguments on both sides of the issue of whether an associative cryogenic computer should be built. The arguments against such a venture center around the fact that perhaps the expense involved in the organization and construction of the memory circuits could best be used more directly in the design of a computer if the memory and hence the entire machine were simpler. However, there are advantages to this design which might be pointed out in order to better evaluate the machine. Let us mention a few.

The complexity in the memory logic as described in this paper diminishes the need for some of the complexity in the control circuits, hence fewer different masks are needed when preparing circuitry. Due to the difficulty expected in maintaining and repairing a cryogenic computer, the iterative character of the circuitry involved becomes advantageous also. This, coupled with the disabled bit (DB) scheme, decreases the problem of maintainability.

The associative property of the memory allows the machine to be used so that symbolic names in source language programs need not be translated into numeric addresses. Table entries are determined by order and relative to a tag, thus these need not be contiguous. This diminishes the storage allocation problem to a very great degree. Moreover, the corresponding pass in an assembler or compiler is not needed, resulting in a real saving of computer time.

The general power and flexibility of an associative memory are not diminished at all in this design. Such operations as table look-up and list processing can easily be achieved in this computer as can compiler statement scanning, and many other functions. Thus, for some purposes associative memories promise to be better organized than conventionally addressed arrays.

The advantages of cryogenic circuitry have not yet been fully demonstrated, however, the prospects for several benefits in this area are quite good. These include relatively fast switching time and compact construction. The cost of a machine built of these components should be low once the masks used for depositing the circuit material have been produced. The fewer the masks, the lower the total cost. The associative cryogenic machine, with a great deal of its logic in memory will benefit from this standpoint.

References

1. Kiseda, J. R., et al, "A Magnetic Associative Memory," IBM Journal, Vol. 5, no. 2, p. 106, 1961.
2. Seeber, R. R., "Associative Self-sorting Memory," Proc. Eastern Joint Computer Conference, 1960.
3. Seeber, R. R. and Lindquist, A. B., "Associative Memory with Ordered Retrieval," IBM Journal, Vol. 6, no. 1, p. 126-136, Jan. 1962.
4. Slade, A. F., McMahon, H. O., "A Cryotron Catalog Memory System," Proc. Eastern Joint Computer Conference, p. 115-119, 1956.
5. Slade, A. E., Smallman, C. R., "Thin Film Cryotron Catalog Memory," Symposium on Superconductive Techniques for Computing Systems, May 1960.
6. Buck, D. A., "The Cryotron - A Superconductive Computer Component," Pro. IRE, Vol. 44, no. 4, p. 482, 1956.
7. Haynes, M. K., "Cryotron Storage, Arithmetic and Logical Circuits," Solid State Electronics Vol. 1, p. 399-408, Sept. 1960.
8. Rosin, R. F., "Cryotron Circuits and Memories," Course on Theory of Computing Machine Design, University of Michigan Engineering Summer Conferences, 1961.
9. Slade, A. E., "A Cryotron Memory Cell," Proc. IRE, Vol. 50, no. 1, p. 81-82, Jan. 1962.

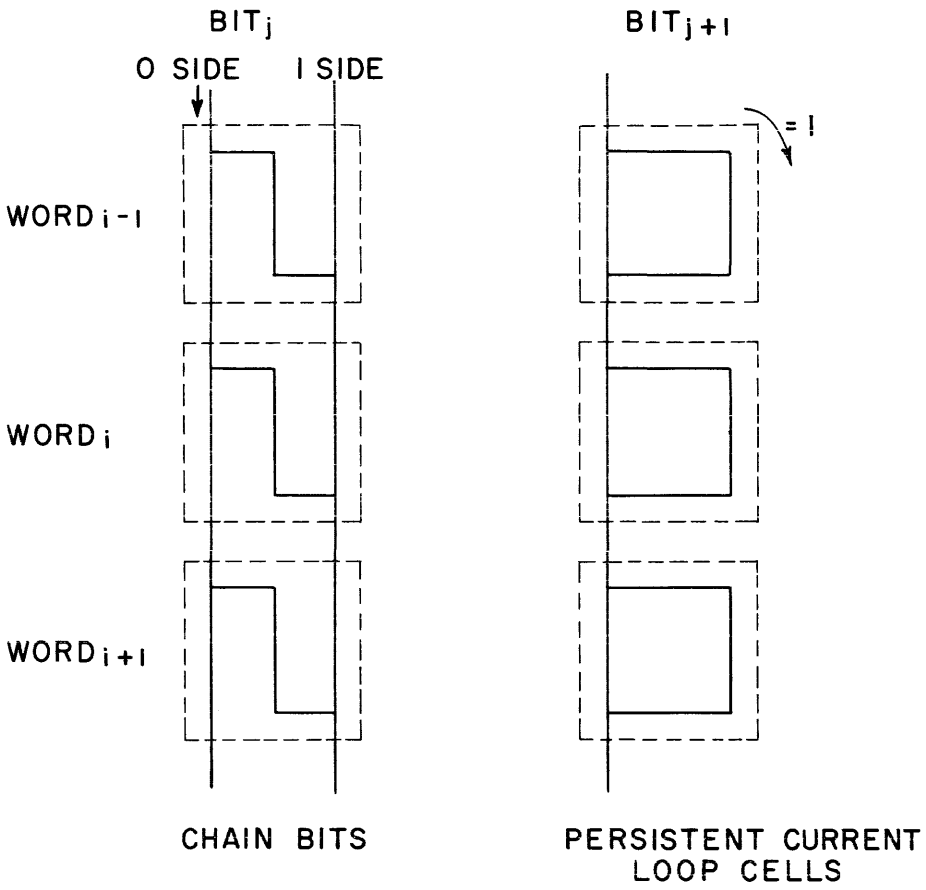


FIGURE 1.
ASSOCIATIVE MEMORY CELLS

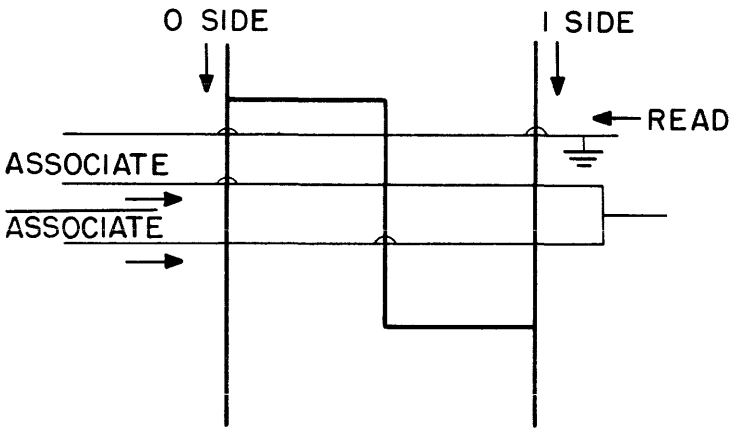


FIGURE 2
ASSOCIATION CHAIN BIT

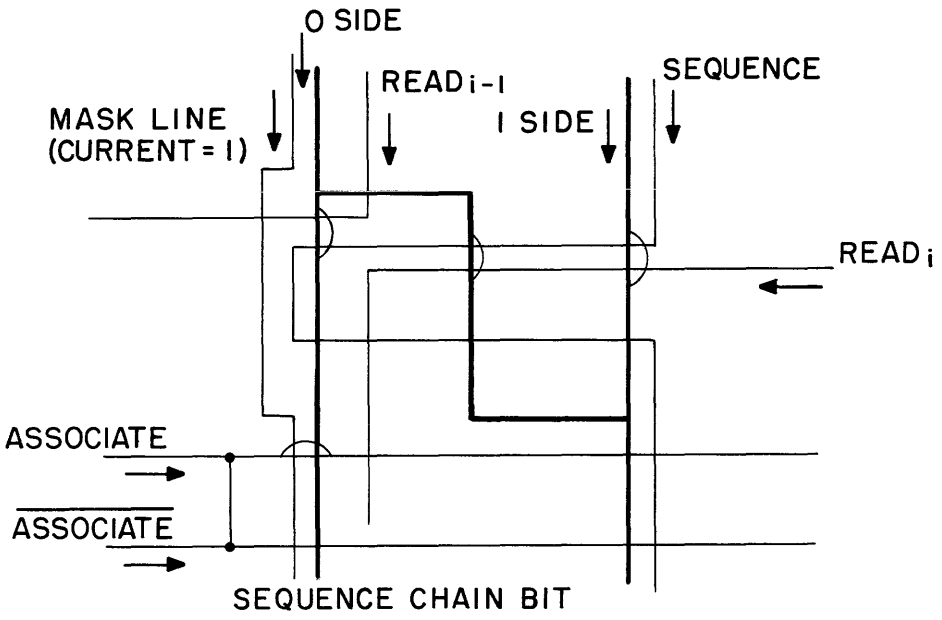


FIGURE 3

SEQUENCE CHAIN BIT

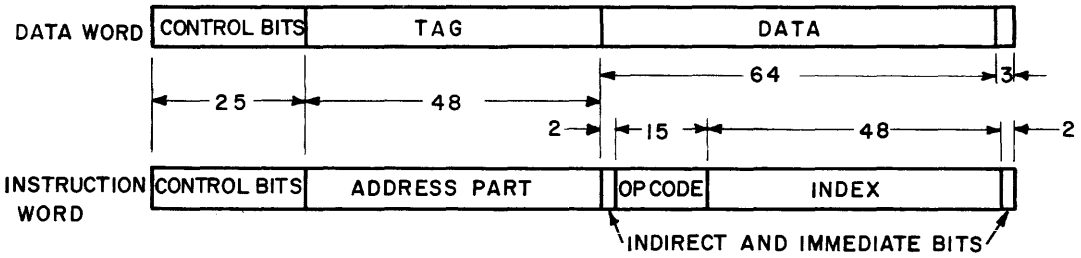


FIGURE 4

WORD FORMATS

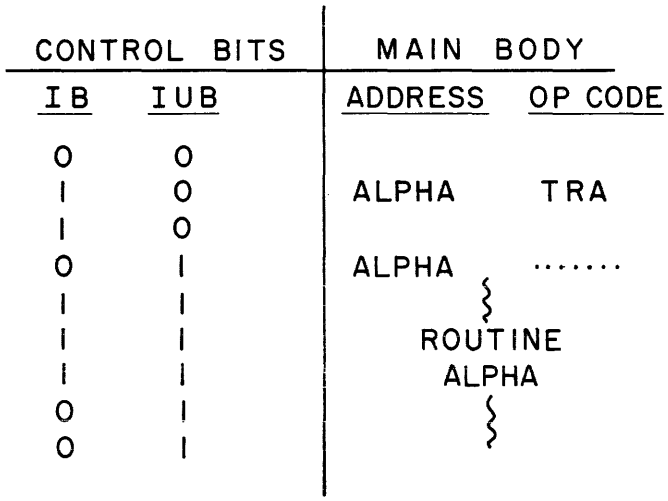


FIGURE 5.
INSTRUCTION SEQUENCING

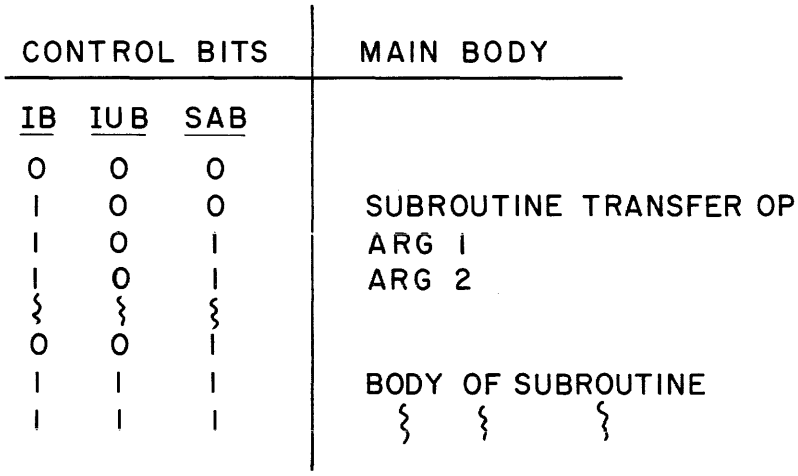


FIGURE 6.
SUBROUTINE TRANSFER &
DATA RETRIEVAL

INTEGRATION AND AUTOMATIC FAULT LOCATION TECHNIQUES IN LARGE DIGITAL DATA SYSTEMS

Donald W. Liddell

U.S. Navy Electronics Laboratory

San Diego, California

Summary. A digital computer, if used with proper programming techniques, can be a powerful tool during the processes of physical integration of complex digital data processing systems. After system integration, as such, has been completed the same techniques may be used to provide performance monitoring and daily calibration status data for all or any part of a system.

Investigation of such programming techniques during system integration of the Developmental Navy Tactical Data System (NTDS) at USNEL produced results which indicated the possibility of using the computer for automatic fault location in the system. Some progress has been made in this area, and a program which allows the NTDS computer to identify a failing logic card associated with its own memory logic and switching circuitry has been successfully demonstrated. The final objectives of this approach are to provide facilities to perform on line performance monitoring and automatic fault location, reduce to a minimum the external test equipment required for a system, and eliminate insofar as possible the high degree of training presently required in the system maintenance technician.

Within the past few years the digital computer has developed from a large, unreliable device into a relatively small package of solid state elements, with fairly good operational reliability. There has, naturally, been a revolution in military real-time data processing systems encouraged by this development. General purpose stored program digital computers and peripheral equipments comprising a system for modern practical data processing are now conceivable in a size suitable for shipboard, aircraft or other vehicle installation. Figure 1 is a block diagram of a typical system.

Along with these new systems a new variety of systems problems has arisen, and one of the first and most important is the fact that the "systems engineer" is suddenly too narrow-minded to cope with the new systems concepts. In the past the systems engineer has been a man with a broad background of knowledge which enabled him to make a complicated complex of equipments function as an integrated unit. However, he is now faced with the fact that this "complex of equipments" is just one black box in a jungle of black boxes which must function together in real time as a new "super system". The real problem here is that he does not recognize this fact. His approach is to make sure that his portion of the system is functioning according to some over-all specifications, then wait for the mysterious process of system integration. When, in fact, an attempt is made to achieve operation with his device in conjunction with other systems units, and complete failure occurs, his reaction is to retire to his own device, check it out thoroughly and proclaim that it is functioning properly while pointing his finger in the approximate direction of the remainder of the system and making disparaging remarks about its obvious inadequacies.

At this point, it should be pointed out that computer programs, although referred to as "software", are in fact hardware in the same sense as any other piece of system equipment. The idea of classifying programs as "software" and placing them in a separate category is repugnant. Programs are indeed among the "hardest" of those devices encountered in system integration. In fact, the computer program is quite analogous to an electronic system unit, i.e., it requires design, performs a function, and requires readjustment after initial design in order to perform in a real-time environment. The term "program debugging" is particularly noxious as it implies the location of minor clerical errors. Nothing could be further from the truth. The programmer vigorously pursues this "debugging function" by placing patch upon patch in the operational program, interrupting these processes frequently with cries of "we've obviously got a hardware problem!" Meanwhile, a final operational program is assembles which will consist of one computer instruction and 30,000 patches. Frequently, when these alleged "hardware problems" have been resolved to be an illegal series of computer program

instructions, it becomes obvious that the computer programmer concerned has been subjected to a gross underexposure to system interface circuitry and timing characteristics.

The foregoing can be described as problem number 1 -- System Integration.

Assuming that system integration has been accomplished -- that is, we have a large group of computer and other system components and a working real-time operational program -- the next phase in a developmental system is one of Operational Evaluation. This is where we find out whether the system performs according to specification and, in fact, to what degree it fulfills desired practical functional requirements. While failures and incompatibilities are noisy and catastrophic during the Systems Integration Phase, they are more subtle but nonetheless catastrophic during the Operational Evaluation Phase. Individual equipments drift out of tolerance, program patches cause nebulous functional changes in the operational program, operators of systems components, bored with long periods of inefficient operation, discover unique ways of causing the system to fault and, as a result, days and in many cases weeks of operational testing are lost or wasted. Quite frequently when magnetic tape recording is the only means of data gathering for analysis, and when analysis of data is performed at a later date, it is obvious that the fact that the equipment in the system was not functioning properly is not discovered until it is too late and tests must be re-run.

The foregoing can be described as problem number 2 -- System Calibration and Performance Monitoring.

Up to this point in development and from this point on, we have and will be faced with the physical failure of sub-system components. Components are defined as cards, modules and other such small replaceable units. It is obvious that even a simple failure in a system such as that shown in figure 1 can be catastrophic. Such a failure in one portion of the system can manifest itself as an ailment apparently in a different part of the system, and many hours of time have been spent by people with intimate knowledge of the circuitry involved and using conventional test equipment before the ailing element was found.

If large scale data processing systems such as NTDS are to operate in real time, and real time includes 24-hour a day operation with no appreciable time out for preventive maintenance, it will be necessary to have facilities available for rapidly locating the failing components in the system while system operation continues.

The foregoing can be described as problem number 3 -- Automatic Fault Location and Diagnostic Techniques.

A general purpose digital computer is the controlling device in a digital data processing system such as has been described and is an excellent tool in solving the problems in the aforementioned three problem categories. During the integration, development and operational testing of the Navy Tactical Data System, efforts in the first two problem areas resulted in computer programs which exhibited characteristics that led to the belief that a solution to the third problem was possible.

During the process of system integration it became obvious that there would be computer failures, interface problems such as timing and logic level differences, incompatibility of different types of circuitry used by various contractors, incompatibility between operational computer programs and equipments, etc. The first effort was in the computer area, various command, memory, arithmetic, input/output, frequency margin, etc., tests were written. Figure 2 shows the original method of taking frequency margins on the computer. Two technicians were required to perform this function; one to operate the computer and the other to make adjustments of the one megacyle clock and observe waveforms on the oscilloscope. As the clock frequency was adjusted to one margin, the computer would fault and the period of the clock would be measured on the oscilloscope. This would be recorded, the program restored by the technician at the console and the process repeated for the other frequency margin. This process quite frequently resulted in the dip stick, held by the technician, destroying several cards in the computer and, as can be imagined, the results are not too accurate. A program was written which allowed the computer to make its own frequency margin measurements. The only thing required of the technician was adjustment of the clock frequency control from one extreme to another and the program would cause the computer to dump its margins on the flexowriter. A sample dump is shown in figure 3. This test resulted in reduction from about 10 to 15 minutes for frequency margin checking to about 1 minute. An executive program was written which allowed all computer maintenance tests to be stored on magnetic tape and automatically called up for execution during preventive maintenance periods. Interference by the technician is necessary only if there is a typeout indicating trouble. Very shortly, as system integration

continued, it became necessary to write simple minded display console and other subsystems computer test programs to assure that these devices were functioning in a gross manner with the computer and with each other. Figures 4 and 5 are PPI presentations of patterns resulting from some of these types of tests. Figure 4 is a category selection test in which symbols appear on the PPI in the same relative position as their selecting buttons appear on the console panel. Obviously, the presence or absence of a symbol at a particular time will pinpoint a fault. The figure shows various combinations of category selections. Figure 5 presents four other system integration type tests. Figure 5A is a function code test where an action button in a system unit is actuated and its associated function code (in octal) is sent to the computer. The function code is checked in the computer and sent back to the system unit where it is displayed for visual check by the unit operator. The spaces between the horizontal row of crosses represents the digits 1 through 6. The function code shown is 56. Figure 5B is a velocity leader test, 5C is a range ring linearity test, and 5D is a primitive position accuracy test. These and other tests have been incorporated under an executive routine which allows them all to be run concurrently in the computer, then selected individually at any system position. This type of test helped immeasurably during the processes of physical system integration. The term "physical integration" is used because the phases of system integration and operational evaluation necessarily overlap somewhat.

Upon entering the operational evaluation phase it became painfully obvious that it would not be possible to do a realistic job of evaluation without some means of system calibration and performance monitoring. Computer calibration type programs were written for various system configurations and were eventually incorporated under an executive routine for use with the complete system. These programs are now written on a magnetic drum for purposes of fast access. Any of the programs may be loaded in the computer in a matter of 1 or 2 seconds, and automatically executed. If a different test is desired, it may be called up by inserting its identifying number into the computer. The currently running test is interrupted; the desired test is automatically loaded and executed.

Figures 6 through 8 describe one of these types of calibrative tests. Figure 6 is a block diagram of the particular system configuration concerned. Included are a video processor, a unit computer, an NTDS display console, and the prime reference in the form of the USM/94 radar simulator. The simulator has two outputs; one is rho theta analog video output, the other is XYH components, in digital format, corresponding to the instantaneous values of rho theta. These XYH components are sent via a digital readout buffer (DRB) to the unit computer. The purpose of this test is to provide positional information calibration for the complex shown. Targets are generated in the USM/94 whose original positions have previously been written in the computer memory. Hence, a comparison with the DRB information (2) will verify whether simulator information is correct or not. The rho theta analog video (1) to the video processor is reported by the processor (3) to the unit computer where, again, a comparison can be made. The rho theta analog video (1) is applied to the NTDS display console and appears as PPI video. A console operator can enter a symbol over what appears to him to be the target position and this information is sent (4) to the unit computer. Each of the aforementioned pieces of information (2, 3 and 4) are caused by the computer to be displayed as distinctive symbols on the NTDS display console PPI as shown in Figure 7. If everything in the complex is indeed in calibration, all symbols and video will be exactly superimposed and the person responsible for operational tests for the day can quickly determine whether system accuracy is adequate for his purposes. The symbols at the twelve o'clock position of Figure 7 are purposely separated to show what erroneous system adjustment would look like, and to identify the various pieces in the arrangement. The half box symbol positioned over the video is the DRB report. The half diamond symbol is the video processor report and the small circle is the operator's interpretation of the target position. The simulator provides for approximately 32 tracks in an infinite variety of configurations. The tracks can, in fact, be moving and a record kept. However, this has not been found to be necessary. Several pieces of video do not have symbols associated with them as this is selectable by program. Once all the information is inserted in the computer, a dump can be made which serves as a calibration curve to show the equipment condition on that particular day. An example of the computer typeout is shown in figure 8. If the system is in perfect calibration the corresponding X and Y values for the three sources of coordinates on the top half of figure 8 should be identical and it follows that the three groups of differences in the bottom half of figure 8 should all be zeros. This particular dump was chosen because it shows some gross discrepancies, and, in fact, these discrepancies tend to point out the areas where there are systems problems.

This type of calibration test has been invaluable during operational testing of the developmental NTDS. In fact without it, evaluation would have been impossible.

The progress made and results obtained from the computer program tests written to help solve the problems of system integration and operational evaluation encouraged a look into the problem of fault location. Fault location and maintenance can be extremely difficult, but conventional methods can be disastrous; fingerpointing runs rampant while the system remains useless. The primary objective is to eliminate insofar as possible the difficulties associated with fault location and maintenance. It is true that a group of highly trained technicians can maintain a large system, provided they have extremely good documentation, good test equipment and good systems engineering leadership. However, if one can designate a single equipment in the system such as the display console as having failed, then a good technician with good documentation and good test equipment can repair it. If indeed one can designate a single drawer of logic as having failed, then a relatively poor technician with good documentation and test equipment can repair it. Further, if one can designate one row of cards as having a failure in it, a technician with a chassis map and a card tester can repair it. Finally if one can designate one card as having failed, then replacement is all that is required. So the trick is to know when a failure is present and to pinpoint its location -- Automatic Fault Location.

There are two ways to automatic fault location. First, the so-called maintenance console which is being pushed by many organizations and indeed may have a place in systems which are not expected to work in real time. However, if they themselves are simple devices, they are inadequate to perform the function of fault location in large scale systems. And, if they are sophisticated enough to perform the required function then they themselves require another maintenance console in order to determine whether or not they are functioning properly. This approach for any system which has a general purpose digital computer, or in fact a wired program computer associated with it, should be opposed. The second way to automatic fault location is to use the digital computer which is the controlling element in a data processing system as the diagnostic and fault locating tool. Assuming that the computer can be used as the standard and the fault locating tool in the system, and since the memory is basic to the computer a computer program should be devised which will allow the computer to find a bad logic card in the logic and switching circuitry associated with its own memory. One method for performing this function with conventional external test equipment is shown in figure 9. The machine in the center of the picture is a memory chassis tester. The device held in the upper half of the machine is a memory chassis from one of the NTDS unit computers. In reality, two technicians are required to perform maintenance and fault location on this device; one, as shown, observing waveforms and the other on the opposite side of the machine pushing logic switches. They go through a logical process, observing waveforms until they find one that does not conform to specifications. This process can take anywhere from several hours to several days for completion. The objective is to eliminate the two technicians, the memory chassis tester and the oscilloscope and to reduce the time factor to a matter of minutes. Figure 10 illustrates the logical processes gone through by the two technicians in the use of the memory chassis tester. The column of X's and 1's on the left indicate the functions of one technician, and the logic tree is followed by the second technician. At each point he reaches a decision and follows an arrow to the next position. Eventually he finds a bad card. This is one of many such logic trees that are used in conjunction with the memory chassis tester. It is obvious that this sort of logical procedure should be adaptable to computer implementation. Since the first NTDS computers were experimental, no physical facilities for implementation of automatic fault location (sensors) was provided. Consequently the approach was to make a complete analysis of all memory logic -- define all possible failures and determine for each failure what the reactions would be for various stimulation patterns. A program was written which analyzes the reactions and points out the failing logic card. This turned out to be approximately an 8-man-month effort; however, when it was successfully demonstrated it was felt that a large step had been taken in the direction of completely automatic fault location in digital data systems. The fault location program itself runs a variety of patterns; all zeros, all ones, alternating twos and fives, and randomly generated words. If there are no faults in the memory logic, the program takes approximately 20 seconds to run. If a fault is encountered the running time is proportionally shorter. When a fault is encountered, the computer will identify it by way of flexowriter output. A typical typeout is shown in figure 11. The typeout indicates in which pattern the test failed; in this case, zeros. The next line indicates a bit picked up. The next line indicates the area in memory where the bit was picked up, and the next three lines offer suggestions for the maintenance technician to

follow. The first four figures in each of these lines indicate the card type that could have failed (021A, 020A, 022B). The next six figures indicate a chassis and the cartesian coordinate location of the card on that chassis (for the first suggestion, J14 = chassis 14, B = row B, and 34 = column 34). Normal procedure would be for the technician to take the first suggestion and interchange these two specified cards. If, upon running the fault location program again, the trouble follows in the typeout as indicated by the arrow on figure 11, it is obvious that there is a bad card in that location, i.e., chassis 14, row E, column 34. If the typeout does not change, the technician takes the second suggestion, and so on, until the trouble does follow card changes in successive typeouts. There are a couple of other suggestions made; however, in just about every instance it will be a card failure. The program will cope with multiple faults. It merely takes them one at a time. When the first fault encountered is corrected the computer then proceeds on to the next. It should be noted that in a tactical situation all computer suggestions would be implemented at once.

What the development of this type of program amounts to is a tremendous job of circuit analysis; however, this could be simplified in future systems if provisions for sensing are included in the design. In the final analysis, what is being done is to let the systems engineer and computer do all of the maintenance technician's thinking for him. This memory test itself actually occupies only a small portion of memory and its one drawback at the present time is lack of mobility. However, it is not felt that it will be difficult to devise a probling program to find a clear space in memory in which to place the test.

If, indeed, there is more than one computer in the system, as is the case in NTDS, we can use this program in one computer to automatically perform fault location in the memory of the other computer. The only requirement here is that there be three functioning addresses in the memory of the ailing computer. At present there is an effort in accomplishing an analysis of the arithmetic section of the computer and writing a program for fault location in this area. Other blocks of the computer will be taken for analysis as time allows.

Concurrently, an effort is being made to use the computer in automatic fault location for the remainder of the system. Two approaches are being pursued; one is to stimulate and observe reactions, as described in the previous test, the other is to duplicate the logic function of the external device in the computer memory and exercise both simultaneously while sensing any difference. The final program will probably be a composite of these two approaches.

It is obvious that the type of person we need to write this type of computer program is not the usual computer programmer. He must be several things. First of all, he must be a good circuit man, he must understand computer operations, he must be an overall excellent systems man of the more broadminded variety, and, incidental to this, he must be an extremely clever computer programmer. There are very few of this type in existence today.

What would be desirable to have as a final product is shown diagrammatically in figure 12. Actually, the computer memory would be filled almost completely with operational programs and the test and executive routine would require a minimum of memory space. The "current test" the majority of the time would be merely a performance monitoring test -- a problem-solving routine. Periodically, the system is asked to solve a problem and, if indeed the answer comes out correct, everything is apparently normal. However, upon receiving an erroneous solution to the problem, the executive routine would cause a diagnostic program to be called in from high-speed storage which asks the question, "Where, on a subsystem basis, is there trouble?" Once this has been defined we can degrade operation of the system; that is, use a simpler operational program which does not include the offending equipment. This will give more space in memory for the more sophisticated automatic fault location program. The fault location program would then be called into the computer and would locate the failing component down to the smallest replaceable unit level. This, of course, assumes maximum system operation requirement all the time. There will, of course, be periods when full operational capabilities are not necessary. During these periods on-line maintenance and diagnostic routines could be performed.

In the afore described test programs probably the most important factor is that they be written in a manner which causes the system equipments to be exercised in precisely the same manner as is done in the system operational program. One or the other must be altered in order to conform. Any other approach obviously makes test results meaningless.

Up until the very recent past, it has been extremely discouraging to talk with various systems people around the country on this subject. There has been very little enthusiasm

for this type of effort. The main concern seems to be with a type of program which is designated as an "exercising program" whose sole function is to exercise a piece of equipment in some fashion for a given period of time. If the equipment indeed passes this test, it is assumed ready for delivery, and acceptable for use in a system. It has only been more recently that it has been realized, or perhaps admitted, that this is not a reasonable way to approach the problem.

There is growing opinion that the types of programming techniques described in this paper are absolutely necessary, if there are to be large scale data processing systems which are capable of working in real time continuously. Results have been encouraging and it is hoped that other systems people will consider their merits.

Reference

"The Development of a Rote Diagnostic Process for the Atlas Guidance Computer System", H. R. Nonken (Burroughs), Vol I, Proceedings of the Fifth AFBMD-STL-Aerospace Symposium, August 1960.

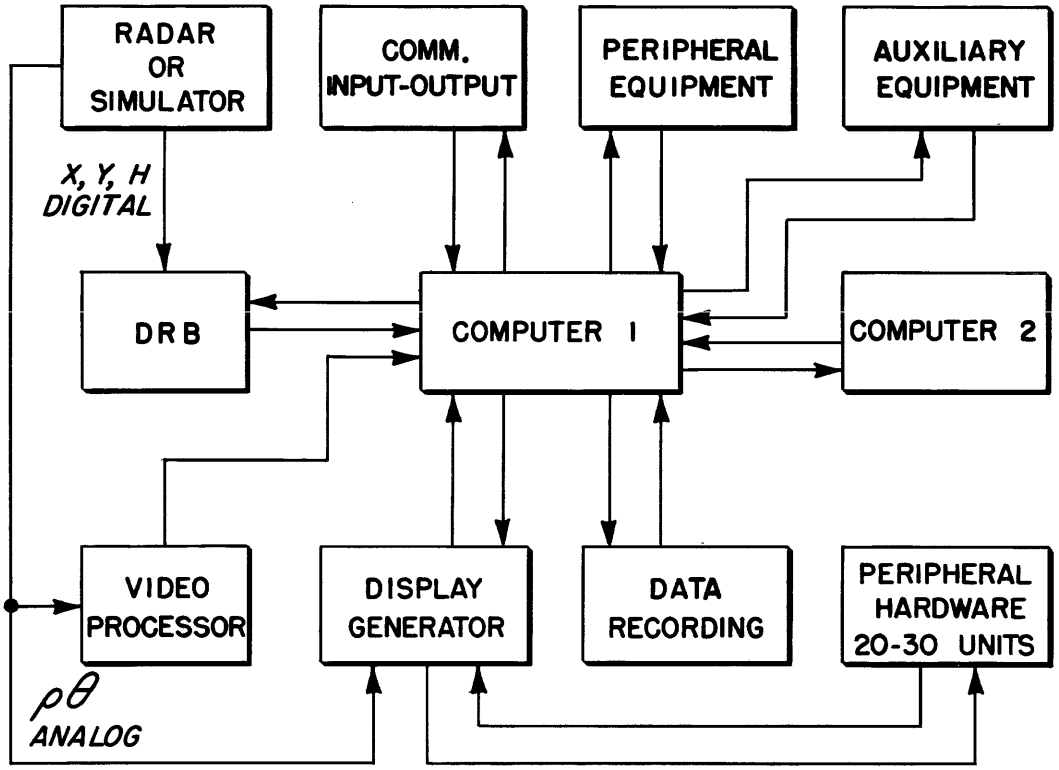


Figure 1 - Typical Data Processing System Configuration

Figure 2 - Frequency Margin Test

NTDS CT

.962 MEGACYCLE 1.040 MICROSEC

1.133 MEGACYCLE .882 MICROSEC

Figure 3 - Flexowriter Typeout of Automatic Computer Frequency Margin Test

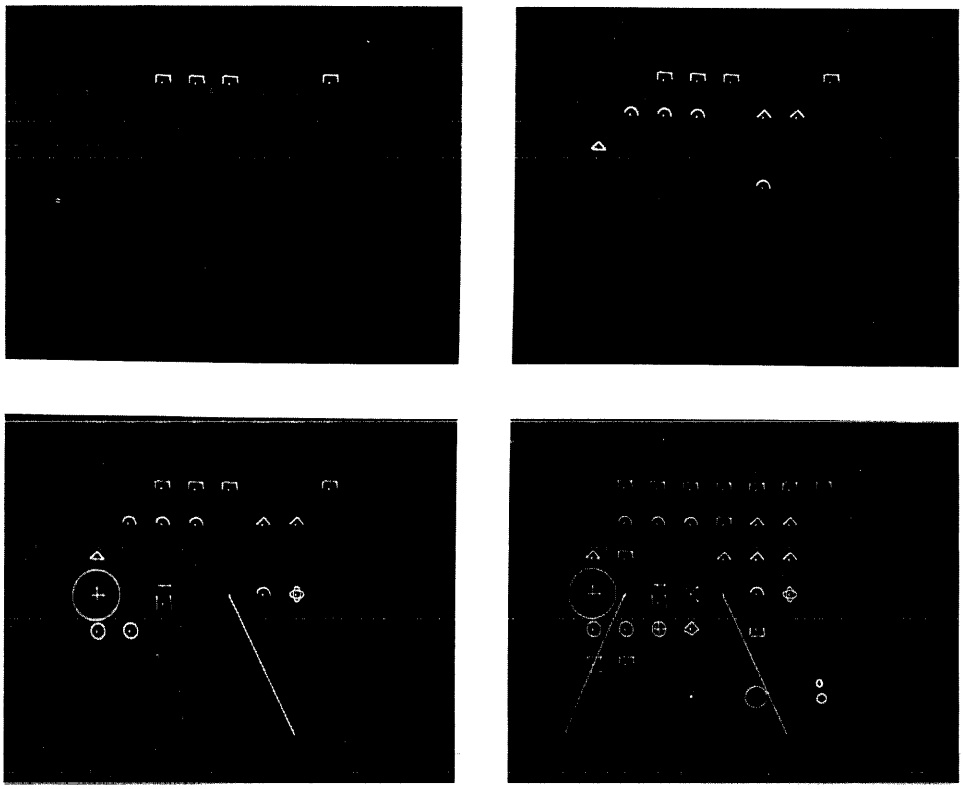


Figure 4 - Category Selection Test

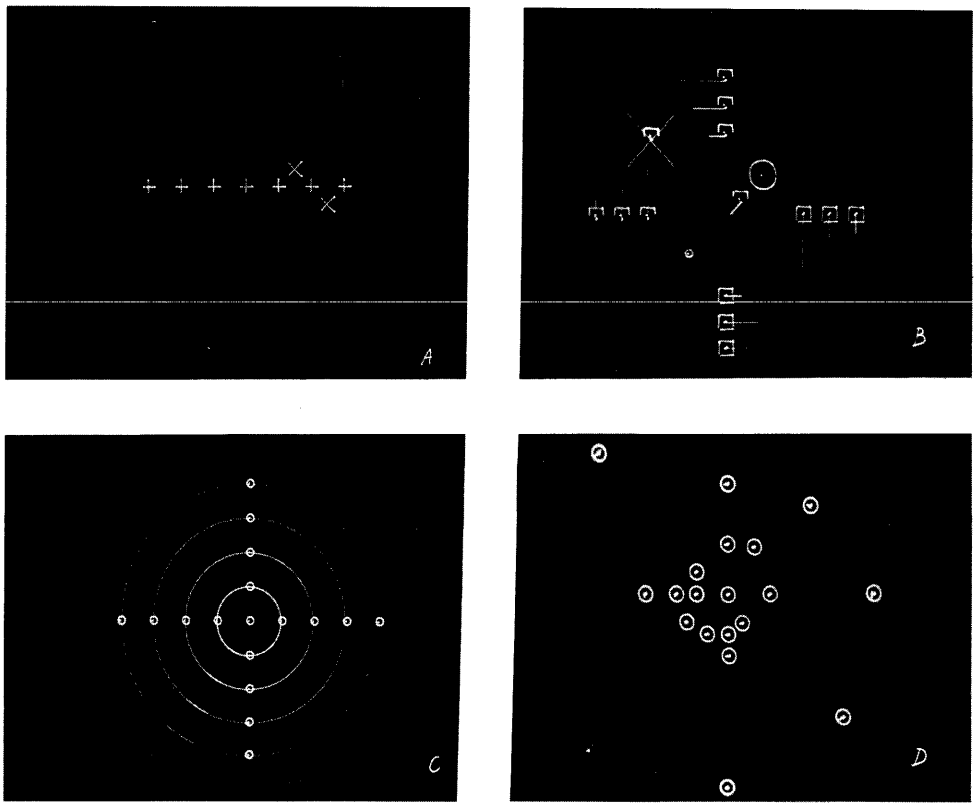


Figure 5 - Miscellaneous Display Tests

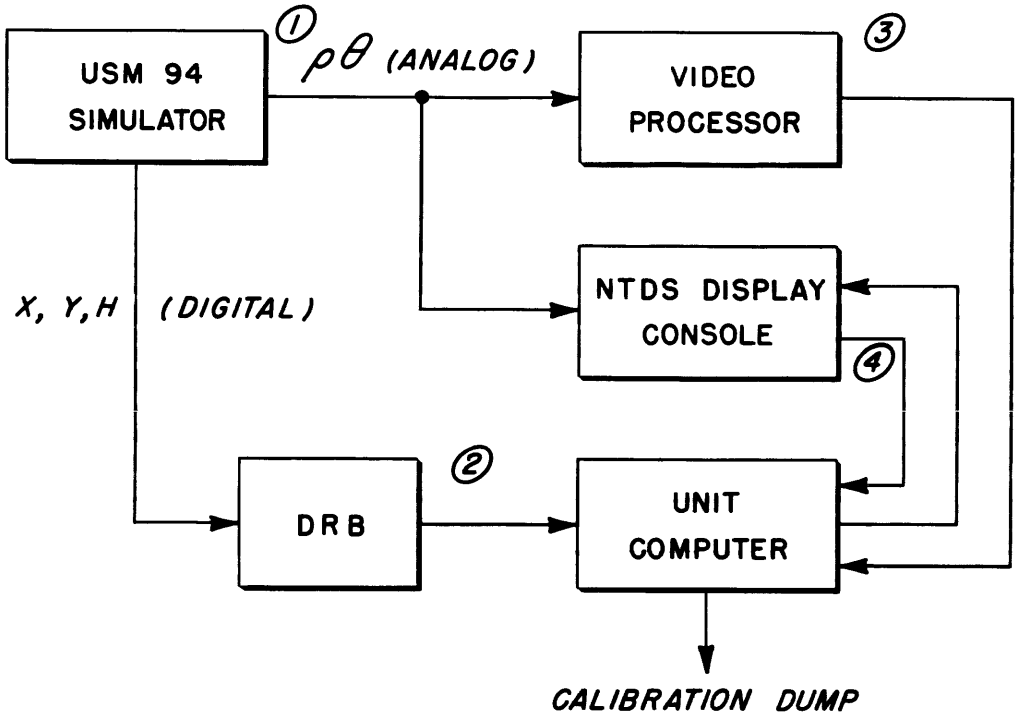


Figure 6 - Typical Sub-System Configuration

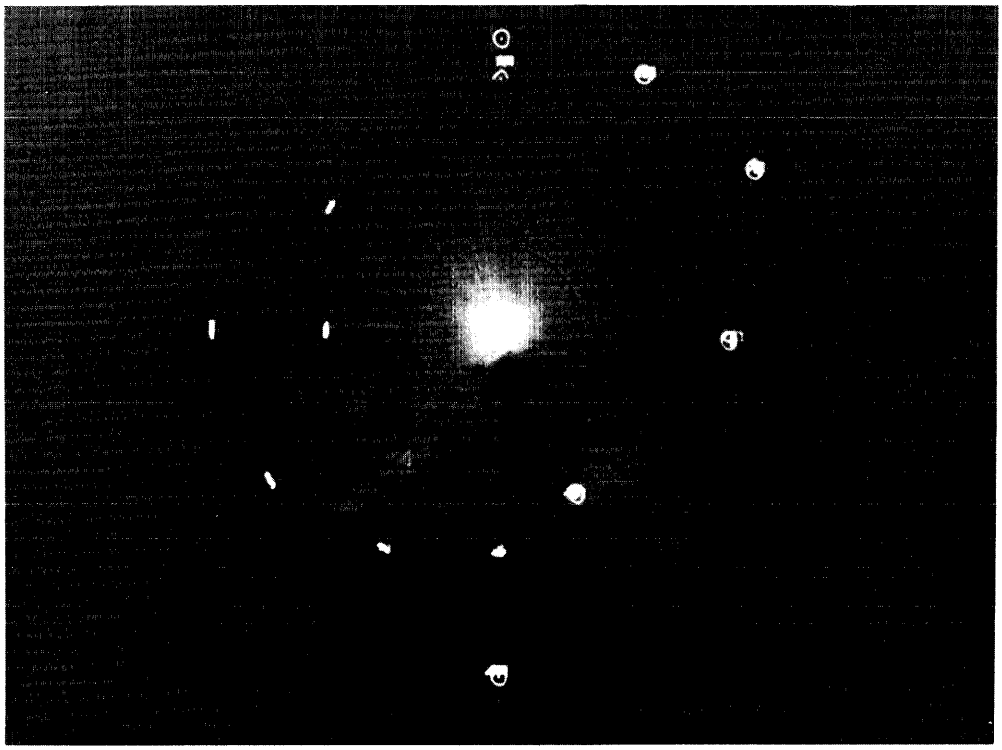


Figure 7 - PPI Presentation of Sub-System Calibration Test

TRACKING ACCURACY-CALIBRATION OF VIDEO PROCESSOR AND NTDS DT/S DATE 5/9/61

TRK NO	X(USM-94) miles	Y(USM-94) miles	X(MAN.) miles	Y(MAN.) miles	X(V.P.) miles	Y(V.P.) miles
13	+ .000	+ 45.375	- .125	+ 44.750	- .125	+ 45.500
14	+ 44.250	+ 27.125	+ 40.250	+ 31.250	+ 40.500	- .125
15	+ .000	- 55.250	- .250	- 55.250	- .375	- 35.375
16	+ 7.000	+330.375	- .125	+ .000	+ 1.875	+200.000
17	+ 40.375	+ .000	+ 40.375	- .250	+ 40.500	- .125
18	+ 13.250	- 25.500	+ 12.625	- 25.625	+ 13.000	- 25.875

DIFFERENCES COMPUTED IN RADAR MILES

TRK NO	USM-94 minus STL		USM-94 minus MANUAL		USM-94 minus V.P.	
	DELTA X	DELTA Y	DELTA X	DELTA Y	DELTA X	DELTA Y
13	+ .000	+ .000	+ .125	+ .625	+ .125	- .125
14	+ .000	+ .000	+ 4.000	- 4.125	+ 3.750	+ 27.250
15	+ .000	+ .000	+ .250	+ .000	+ .375	- 19.875
16	- 18.125	+287.125	+ 7.125	+330.375	+ 5.125	+130.375
17	+ .000	+ .000	+ .000	+ .250	- .125	+ .125
18	+ .000	+ .000	+ .625	+ .125	+ .250	+ .375

VP REPEATED TARGETS

	VP reports within 10 miles of VP matched targets more than once X	Y
13.	+ .375	+ 45.500
13.	- .125	+ 45.500
16.	+ .000	+ 200.000
16.	+ 1.875	+ 200.000

NTDS CN. 1

12 sec. search time and 1 mile subtracted for range compensation on V.P.

Figure 8 - Tracking Accuracy-Calibration of Video Processor and NTDS DT/S

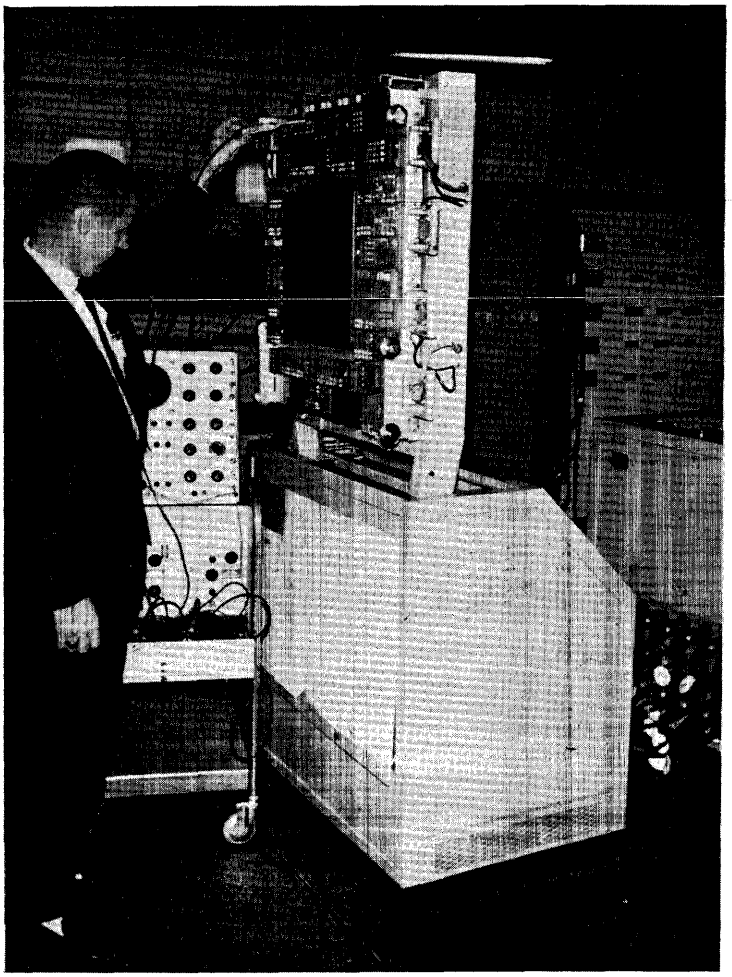
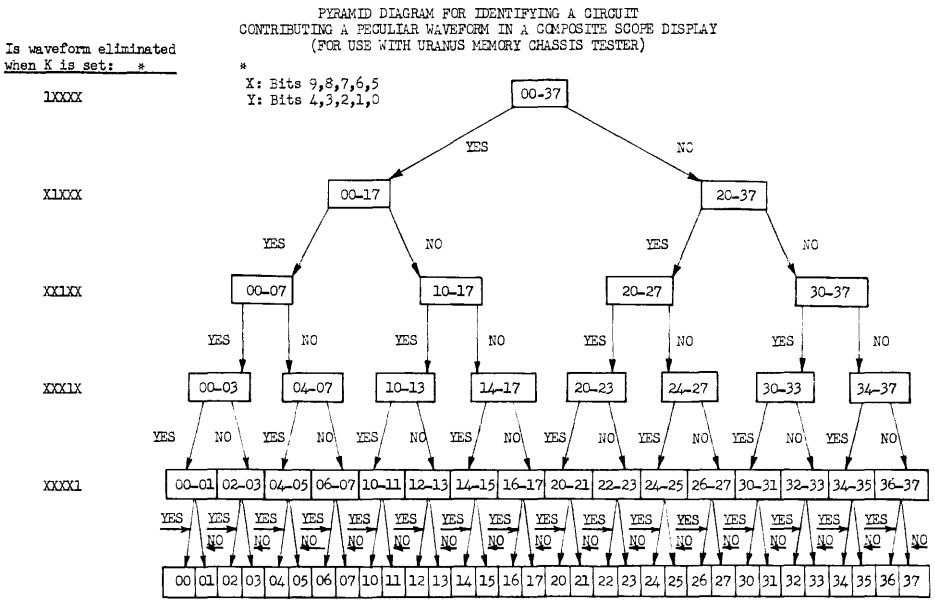


Figure 9 - Memory Chassis Tester



NOTE: Circuit numbers are octal.

Figure 10 - Pyramid Diagram

AUTOMATIC FAULT LOCATION --- MEMORY TEST

```
ZEROS
40000 00000
00004 00000
021A J14B34 EXCHANGE WITH J14E34
020A J14E36 EXCHANGE WITH J14I17
022B J14A09 EXCHANGE WITH J14A19
CHECK LINE VOLTAGE
ADVANCE INHIBIT CURRENT FOR BIT 5


ZEROS
40000 00000
00105 00000
021A J14E34 EXCHANGE WITH J14I33
020A J14E36 EXCHANGE WITH J14I17
022B J14A09 EXCHANGE WITH J14A19
CHECK LINE VOLTAGE
ADVANCE INHIBIT CURRENT FOR BIT 5
```

Figure 11 - Memory Test Typeout

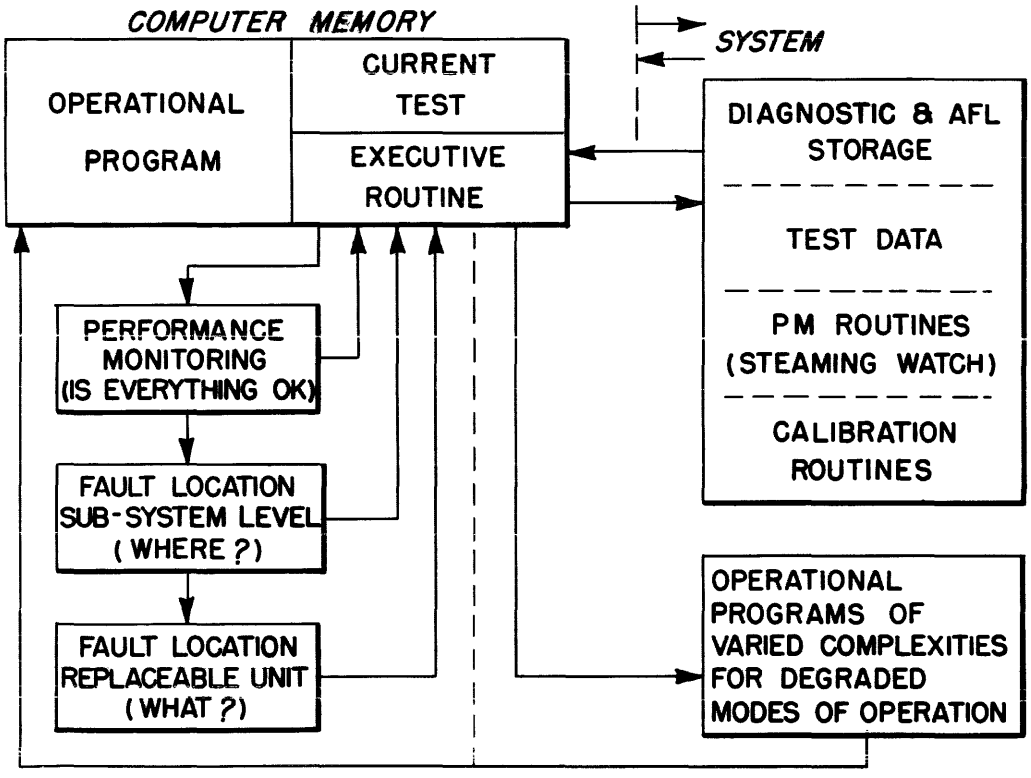


Figure 12 - Proposed Ideal Automatic Fault Location Configuration

THE USE OF COMPUTERS IN ANALYSIS*

Walter J. Karplus and Ladis D. Kovach
Department of Engineering
University of California
Los Angeles, California

Introduction

Automatic computers, both analog and digital, have attained a prominent place in most modern engineering curricula. The computer is recognized as an important engineering design tool permitting the student to test the efficacy of a large number of design hypotheses to determine an optimum design. In these applications the computer is favored because it enables the student to dispense with laborious hand calculations and because it familiarizes the engineering student with techniques which will subsequently be valuable to him in industry. The application of automatic computers to another category of engineering courses - courses in methods of analysis - is more controversial.

Engineering analysis courses include such subjects as electric circuit analysis, dynamics, thermodynamics, fluid mechanics, and field theory. The chief objective of these courses is to instill in the student a profound understanding of fundamental engineering practices and systems and to give him a constructive insight into the response of engineering systems to various excitations. The emphasis in such courses is on generalization rather than on specific solution as in engineering design. Since automatic computers can only be used for numerical analysis - the solution of problems in which all variables and parameters are specified numerically - it has been argued that computers have little or no place in courses in engineering analysis. According to this line of argument the availability of a computer enables the student to "short circuit" the learning process. Homework and classroom problems in courses in analysis are designed ideally in such a way that by carrying out the actual computation, the student gains an insight into the physical system and obtains benefits far exceeding the interpretations that he is able to draw from the final numerical solution. Furthermore, the availability of computers discourages the student from a conscientious effort to find the general and therefore more valuable and elegant solution, rather than a numerical solution applicable only to a specific situation. It has been stated that the availability of analog computers in the late 1940's and early 1950's greatly slowed the orderly development of the field of nonlinear control system theory. Most engineers found it much easier to solve a <u>specific</u> problem on a computer rather than to seek and develop a <u>general</u> theory for handling nonlinear system analyses.

At the other extreme are educators who are so computer-oriented that they recommend that computers be utilized in <u>every</u> course in engineering analysis at least to some extent. They argue that although, as yet, the generalizations which can be drawn from numerical analysis are limited, this is only a reflection of the relative infancy of the computer field. They feel that soon a new generation of automatic digital computers will be available. These new computers will make it possible for an engineering analyst or scientist to "converse" with the automatic brain, that is, to use the computer as an extension of his own thinking processes. In this way the use of computers will permit the drawing of generalizations beyond the scope of the human mind. Accordingly, it is argued that in order to prepare the engineering educational field for this day it is necessary to begin now to reorient thinking in the direction of computers and computing techniques.

It is not the authors' intent in this paper to attempt to resolve these two lines of argument. Along with most other engineering educators we recognize the importance of mathematical sophistication and the fostering of the students' interest and ability to make significant generalizations. At the same time we realize that the computer age is upon us and that no area of technological and scientific endeavor can completely escape its impact. It is the objective of this paper to demonstrate by means of a number of specific examples how the utilization of computers can be compatible with the basic objectives of courses in engineering analysis. Foremost among these examples are two categories of computer utilization:

1. The application of computers to aid the student in the visualization of dynamic or mathematical phenomena.
2. The opening up of new approaches to the explanation of system behavior - approaches which are out of reach of conventional analytical methods.

Monte Carlo Methods and Random Walk Phenomena

The analysis of fields as they occur in such engineering disciplines as electrostatics, heat transfer, and fluid mechanics generally follows a deductive approach. By applying basic physical laws such as the law of conservation of mass and energy and the principle of continuity to a small bounded region of the field (the differential element) the basic partial differential equations characterizing field problems are derived. These

*This publication was sponsored by the Educational Development Program, Department of Engineering, with financial support from the Ford Foundation.

equations include, among others, Laplace's equation, Poisson's equation and the wave equation. Mathematical techniques such as separation of variables, conformal transformations, etc. are then applied to adapt the general equations to the two specific boundary and/or initial conditions. This finally results in two mathematical expressions characterizing the equipotential and stream lines within the field and specifying transient variation of the field potential at specific points. Although this approach is of unquestioned value and utility both to physicists and engineers, many students will find an additional alternative approach both interesting and stimulating.

The second approach is governed by the recognition that the uncertainty principle applies to any specific particle under the influence of the potential field. In other words, the classical equations apply only to the statistical average. A specific particle (fluid particle in a fluid field, electron in electrodynamics, etc.) will not follow the stream line which constitutes the solution to the classical problem, but will rather describe a "random walk". Accordingly, to solve a field problem, i.e., determine the potential at some point within the field, by this approach it is necessary to perform a series of consecutive random walks all commencing from the same point and to tabulate the results of these walks.

While this technique will rarely be able to compete with the more conventional techniques in terms of efficiency, an important educational advantage may be gained in familiarizing the student with the field process as it actually occurs in nature. By permitting the student to compare the results of classical field problem solutions with random walk solutions he will gain a deeper insight into the significance of the mathematics, that is, generalizations which he draws from a mathematical analysis will be more meaningful.

A practical method for performing such random walk solutions, originally presented by Chuang, Kazda, and Windeknecht[1] will now be described. Analog computing techniques are employed since it is desired to present the data visually and to facilitate manipulation of system parameters.

A technique which is well-established in digital computation involves the use of a sequence of random numbers as computer inputs. By making a sufficiently large number of computer runs, and by taking a weighted average of the results of these runs, convergence to the desired solution can be obtained. This technique can be applied in analog computation by exciting the computer system with a random noise generator - a voltage generator whose output amplitude is governed by a stochastic expression. The repetitive mode of analog computer operation is useful in performing such calculations because a large number of separate runs, each with a random noise input, can be completed within a reasonably short time.

The boundary value problem for which the Monte Carlo method is applicable belongs to a family of generalized Dirichlet problems of the form

$$D_1 \frac{\partial^2 \phi}{\partial x_1^2} + D_2 \frac{\partial^2 \phi}{\partial x_2^2} + K_1 \frac{\partial \phi}{\partial x_1} + K_2 \frac{\partial \phi}{\partial x_2} = 0 \qquad (1)$$

$$\phi = F(x_1, x_2) \text{ on the boundary C}$$

where K_1 and K_2 are arbitrary functions of the independent variables x_1 and x_2 respectively, while D_1 and D_2 are constants. The boundary C is an arbitrary finite closed curve - a Jordan curve. The Monte Carlo method provides a solution in the form of the field potential existing at some point within the field. It does not provide equipotential curves as do the more conventional analytical and analog simulation techniques.

In principle the Monte Carlo method involves selecting a point of interest within the field and from this point commencing a random walk. That is, a sequence of small steps are taken such that each step begins at the end of the preceding step but proceeds in a random direction but always parallel to the two perpendicular coordinate directions. Eventually each such random walk will reach a point on the field boundary C. The potential at this boundary point is recorded and a new random walk is commenced. Provided that enough random walks are taken, and provided that each step in the walk is sufficiently small, it can be shown that the weighted average of the boundary intersections will converge to the potential existing at the point in question within the field.

Referring to Figure 1, assume that the solution of Laplace's equation

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0 \qquad (2)$$

is required at some point P inside a region bounded by C upon which the values of ϕ are prescribed. Starting at P a random walk is taken in which steps of equal length are made in either a positive or negative direction parallel to the x- or y-axis. The walk continues until it reaches C when it is terminated and the value of ϕ on C, ϕ_{C_1} say, is recorded. This process is repeated n times and it may be shown that

$$\phi(P) = \lim_{n \to \infty} \sum_{i=1}^{n} \phi_{C_i} / n \qquad (3)$$

In the computer method to be described, the random walk is actually performed by the beam of

a cathode-ray oscilloscope. When this beam contacts the field boundary as defined by a mask placed over the face of the oscilloscope, the random walk is terminated and a new one begins.

Some auxiliary circuitry is required to cause the oscilloscope beam to describe the random walk. It has been shown that when an electrical circuit is subjected to an exciting input voltage of Gaussian white noise, the response current of the circuit is a Markov stochastic process and that consequently, the conditional probability density function of the current satisfies the equations governing random walk phenomena. For the solution of two-dimensional generalized Dirichlet problems two independent resistance-inductance circuits with nonlinear resistance are required. The equations describing the response of these circuits subjected to two independent Gaussian white noise sources $F_1(t)$ and $F_2(t)$ are

$$\frac{dy_1}{dt} + K_1 = F_1(t)$$
$$\frac{dy_2}{dt} + K_2 = F_2(t) \qquad (4)$$

where K_1 and K_2 are functions of y_1 and y_2. The voltages generated by solving Equations (4) on the computer are employed to drive the oscilloscope beam in the x and y directions. Accordingly, the solution of a Dirichlet problem of the type described by Equation (1) takes the following steps:

1. Two resistance-inductance circuits governed by Equation (4) are simulated. In general these governing equations will be nonlinear.
2. Two independent voltage sources of white noise F_1 and F_2 with adjustable spectral-density D_1 and D_2 respectively, are employed to drive the two resistance-inductance circuits.
3. The voltage outputs y_1 and y_2 of the two nonlinear circuits are used to drive the horizontal and vertical inputs respectively of a cathode-ray oscilloscope. The prescribed boundary C is introduced in such a way as to make detectable the impingement of the oscilloscope beam on the boundary.
4. A value of F(Q) must be generated to correspond to any point Q on C on which the beam makes contact with the boundary.
5. A continuous record of successive generations of F(Q) must be kept or, alternatively, successive values of F(Q) must be continuously recorded. In either event the mean value of all the individual results must be determined.
6. At each occurrence of incidence, the oscilloscope beam must be reset to the initial point inside the boundary C, the point at which the solution of Equation (1) is desired and at which each random walk of the oscilloscope beam must begin.

This cycle of operation must be repeated continuously until such time as the value of the mean of the summed boundary values F(Q) ceases to change. A block diagram and some physical detail of a computer system for the solution of Laplace's equation

$$\frac{\partial^2 \phi}{\partial x_1^2} + \frac{\partial^2 \phi}{\partial x_2^2} = 0 \qquad (5)$$

is shown in Figure 2. In this case the potential distribution along the boundary was such that C could be divided into two parts by a straight line so that the arc length (which is a function of x_1 and x_2) of each part was a single-valued function of the length of the dividing line. In field problems of heuristic interest it is nearly always possible to find such a dividing line. Under these conditions two diode function generators suffice to obtain the value F(Q). The incidence of the beam on either of the two sections of the boundary is detected by means of one of two phototubes which trigger a gate. This in turn permits the output of the two diode function generators to apply voltages determined by the boundary point coordinates to the F(Q) register. By suitable timing circuits, each run can be made to last from one to 3 seconds. In this manner a reasonably large number of runs can be completed in a very few minutes; at the same time the trajectory of the oscilloscope beam during each run will be clearly visible to the observer. The opaque mask shown in the figure is opaque only to the detecting photocells. It transmits sufficient light to enable the observer to see the movement of the oscilloscope beam quite clearly.

The generation of the variables y_1 and y_2 in Equation (4) is effected by means of a circuit of the type shown in Figure 2. This circuit requires two analog summers, an analog integrator and a function generator. Low frequency noise sources of Gaussian amplitude distributions having the necessary power spectra are commercially available. Such devices generally employ a gas tube such as a thyratron as the basic noise source.

Experiments have shown that good convergence of the random walk solutions to the analytical solutions in one- and two-dimensional boundary value problems are obtained if approximately 300 random walks are carried out.

Other Analytical Techniques

There are a number of other ways in which computers may be used effectively to bring a deeper understanding of some phase of analysis.

In finding the complex roots of a polynomial equation, for example, the method of steepest descent provides a useful technique. In this method a related function is minimized in a continuous and automatic manner. If the function

has a minimum at only those points where the polynomial equation has roots, then finding the minimum is equivalent to locating a root.

A description of the automatic nulling process associated with the method of steepest descent on an analog computer has been recently described.[2]

A simple variation of this method has even wider application. Once the computer has obtained the root of the polynomial equation by the nulling process it may be operated in an automatic tracking mode. Thus, if one or several of the coefficients of the polynomial are varied, the computer will follow the root to its new location. This technique is extremely useful to control-system engineers who use this root-locus tracking to obtain information about the stability of control systems.[2]

Another analytical approach which can be clarified for the student by the use of computers is conformal mapping. Any attempt to map the z-plane into the w-plane by means of the function $w = f(z)$ must necessarily be done point by point. With an analog computer, however, a curve in the z-plane can be continuously mapped into the w-plane in a matter of seconds. Tomlinson[3] has described the circuitry required to map a circle into an airfoil shape.

Both the Fourier transform and the inverse Fourier transform of a function can be obtained on an analog computer.[4] A by-product of this method are the Fourier coefficients in the expansion of the function. Thus Fourier analysis can be done on either a digital[5] or an analog computer. Again, the advantage of the latter is in helping to visualize the process and making it easy to see the result of changes in the function being analyzed.

Finally, mention should be made of the work that Bellman[6] has done in dynamic programming. This technique was developed to solve variational problems which arise in economic, industrial, engineering, and biological contexts and for which the classical calculus of variations methods are inadequate. By utilizing the capabilities of the digital computer Bellman has developed a new technique which has the advantage that it handles linearities or nonlinearities, constraints or not, implicit or explicit functionals, in precisely the same manner, and with the same basic equation.

Nonlinear Analysis

In this "space age" more attention is being given to nonlinear analysis. As it becomes increasingly important for mathematical models to match physical situations more closely, more thought must be given to topics like nonlinear differential equations, both ordinary and partial. As we dispense with incompressible fluids, perfect gases, isotropic solids, frictionless materials, weightless bars, etc. we are forced into the realm of the nonlinear. Unfortunately this is unexplored territory because the analytical techniques presently in use are linear in nature. This is true of such giants in analysis as the Laplace transform, the Fourier transform, the Cauchy residue theorem and others. Attempts to extend these tools to nonlinear problems have met with little success.

Thus, for lack of adequate analytical techniques for solving nonlinear problems, the use of computers becomes mandatory. The analog computer, in particular, is useful for studying nonlinear problems. Such studies can help the student visualize nonlinear system behavior and may even lead to important generalizations and new analytical methods. Certain modifications of the analog computer are necessary, however, in order to adapt it to nonlinear studies. These consist of repetitive operation, automatic programming, and means for changing the degree of nonlinearity.

Repetitive operation is well known by now as most modern computers have provisions for it. Older installations can be easily converted by the addition of an integrator whose output ramp is used to energize relays which in turn clamp the integrators in the problem. The repetition rate is controlled by the rate at which the ramp is generated. Care must be exercised so that the repetition rate is not so high that the problem frequencies exceed the frequency-response limitations of the amplifiers.

Automatic programming[7] consists of changing problem parameters by predetermined increments. This is done by generating a monotonically increasing step function. The circuit for this is energized by the repetitive reset pulses and is shown in Figure 3. The step function generated can be used in a number of ways in parameter studies.

One of the most common changes that has to be made in a problem is changing a potentiometer. This can be done by applying the output of the step-function circuit to a multiplier which then multiplies a variable by a constant. Figure 4 shows the circuit for accomplishing this.

Automatic changes in initial conditions can produce a family of solutions in a short time. A typical example of this is shown in Figure 5.

The analog computer is basically a device for solving linear problems. The introduction of nonlinearities increases the complexity of the equipment which often results in decreased reliability. It is now possible, however, to represent a wide range of nonlinearities in a simple manner. This is done by using Quadratrons,[8] nonlinear components which supply the square of an input voltage but which can provide other nonlinearities by simple changes in the auxiliary circuitry. Figure 6 shows the basic circuit with the Quadratron denoted by "P".

The characteristic of the Quadratron is a result of the silicon carbide varistor it contains. In this type of varistor the current varies as some power of the voltage. By combining the varistor with ordinary linear resistance it is possible to make this power very nearly two. Current is ideally supplied to the Quadratron by an operational amplifier. Thus the Quadratron-amplifier combination of Figure 6 forms the basic circuit.

The basic circuit is useful, for example, in representing square-law damping in a mechanical system. Note especially that the symmetry about the origin of the circuit's transfer function is exactly what is required in this type of simulation. There has been a tendency in the past on the part of some educators to feel that they have reached a high level of sophistication when they introduce their classes to second-order differential equations containing square-law damping. We can, however, study a large number of systems each having a different degree of damping quite easily.

Consider the nonlinear differential equation

$$m\ddot{x} + \mu \dot{x}^n(\operatorname{sgn} \dot{x}) + k x = f(t)$$
$$\ddot{x}(0) = \dot{x}(0) = 0 \qquad (6)$$

The signum function, sgn $\dot{x}$, is defined by

$$\operatorname{sgn} \dot{x} = \begin{cases} -1 \text{ if } \dot{x} < 0 \\ 0 \text{ if } \dot{x} = 0 \\ +1 \text{ if } \dot{x} > 0 \end{cases} \qquad (7)$$

Figure 7 shows the circuit for obtaining $\dot{x}^n(\operatorname{sgn} \dot{x})$ for $\frac{1}{2} \leq n \leq 2$. The change in the exponent n is accomplished by means of the potentiometer in the circuit. Solutions to Equation (6) are obtained by combining a number of auxiliary circuits as shown in Figure 8.

Some of the advantages of studying a nonlinear differential equation in this manner are as follows:

1. The student is made to realize that the linear case is a special case -- it is one case out of an infinite number of nonlinear cases. This helps to present linear and nonlinear problems in the proper perspective.
2. The student obtains an idea of how much the exponent can differ from unity before the results change appreciably from the linear case. This gives him a feel for the amount of linearization that can be applied to a given problem.
3. The student can examine the family of output curves and relate the changes to the changes in the exponent. He may thus be able to generalize the results.
4. The student becomes familiar with nonlinear systems in general through the visual presentation from the analog computer. This point is discussed later in connection with limit cycles.

More complex equations may be studied by an extension of the methods described. In this way greater insight can be gained into the nonlinear realm which so far has not yielded to general analytical techniques.

A classical nonlinear differential equation that has been widely studied is the van der Pol equation, usually written

$$\frac{d^2x}{dt^2} - \epsilon(1 - x^2)\frac{dx}{dt} + x = 0 \qquad (8)$$

This equation describes an oscillatory system having variable damping. The damping depends not only on the displacement but also on the parameter ϵ. Different types of solutions are obtained depending on the magnitude of ϵ in comparison with unity. It is a simple matter to solve Equation (8) on an analog computer with the techniques already described.[9]

One of the most important features of the study of nonlinear differential equations on an analog computer is the insight it gives the student into limit cycles. These are closed curves representing a steady-state periodic oscillation determined only by the characteristics of the equation itself, and independent of the way the oscillation is started. If the initial point is inside the limit cycle, the solution curve will spiral outward until it reaches the limit cycle. If the initial point is outside, the solution curve spirals inward. It would be too time-consuming to see this phenomena if the solution curve had to be obtained and plotted point by point.

Network Topology

Since 1954 much work has been done in the study of electrical networks from a topological viewpoint. Any network diagram represents two independent aspects of an electrical network: the interconnection between components and the voltage-current relationships of each component. Network topology is concerned with the former aspect. Essential to this type of analysis is graph theory, a fairly recent branch of algebra.

These new techniques have led to the synthesis of conventional electrical networks and combinational switching networks by algebraic methods. When the network has a large number of vertices (or nodes), however, the computational work becomes prohibitive and the use of a computer becomes mandatory.

The analysis and synthesis of electrical networks by topological methods with a digital computer has been described by various authors. One of the first of these was Van Valkenburg.[10] More recently Watanabe[11] has given a computer method for finding all possible trees and/or multitrees in a network.

With the advent of parametric amplifiers there has been a renewed interest in the analysis of networks containing periodic parameters. When even one periodic circuit element is introduced in a lumped, linear, finite, passive, bilateral, time-invariant network, the analysis of the resulting network becomes a formidable task. Leon and Bean[12] have shown how a digital computer may be useful in this case.

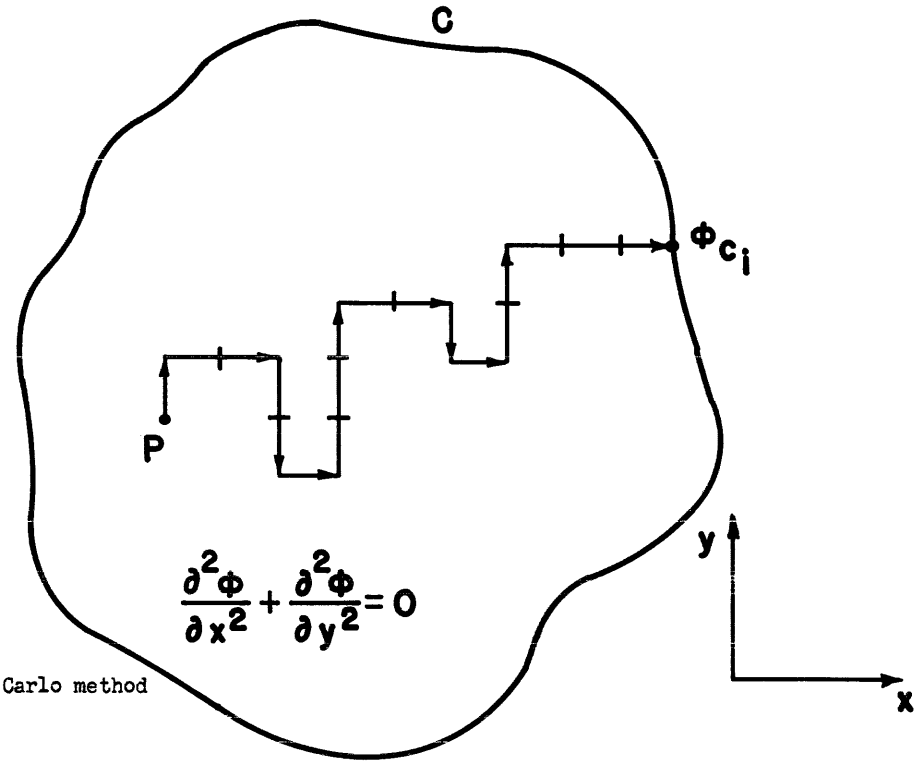
Conclusion

Automatic computers have a definite place in engineering analysis. Analog computers, in particular, are useful in helping the engineer visualize dynamic phenomena and in giving him a feel for the effect that various parameters have on the solution. The use of automatic programming techniques are helpful in making generalizations which may be impossible to achieve without the time-saving feature of the computer.

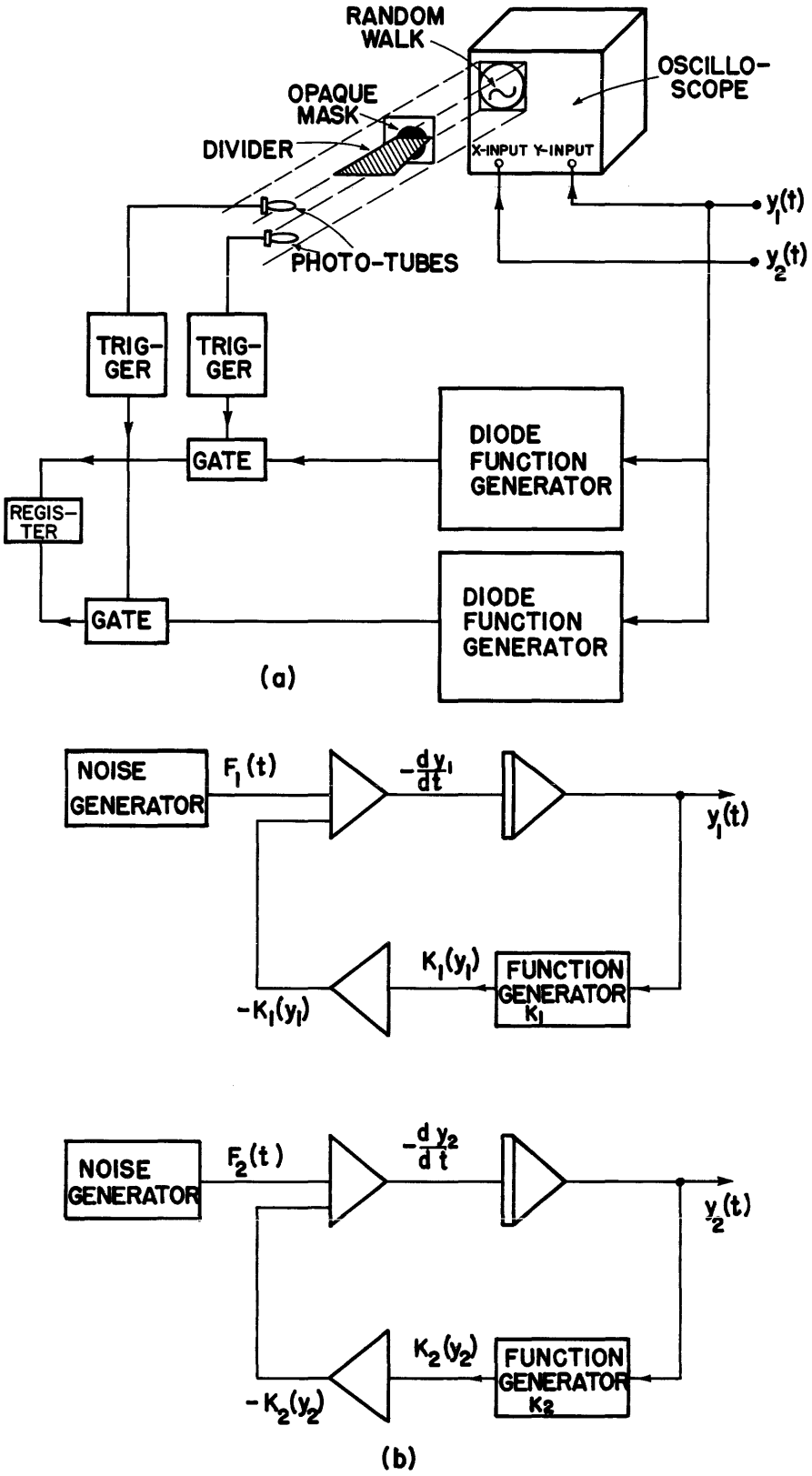
While the use of computers in design may be important to the engineer (especially from an economic standpoint), the real contribution of the computer will come from its use in analysis. With the assistance provided by the computer the analyst can make significant progress in extending man's knowledge of his environment.

References

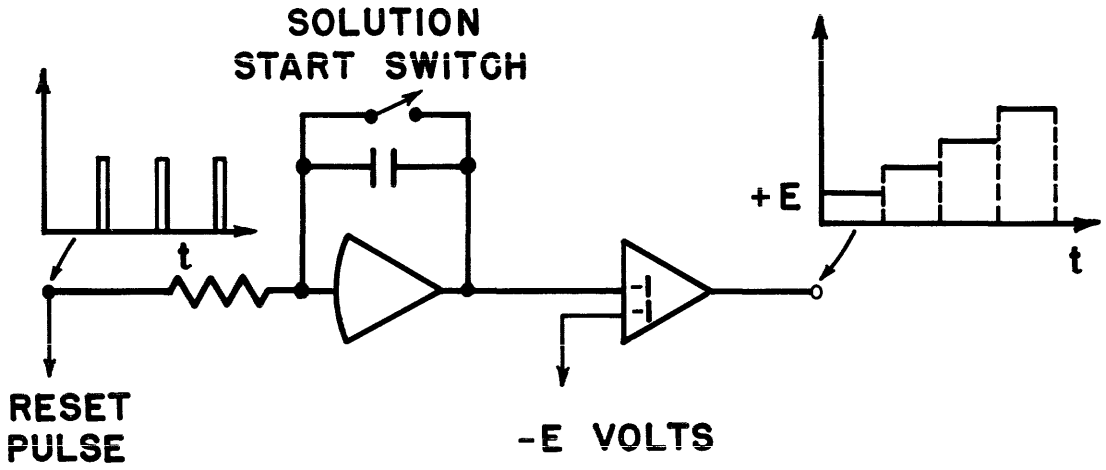
1. Chuang, K., L.S. Kazda, and T. Windeknecht: A Stochastic Method of Solving Partial Differential Equations Using An Electronic Analog Computer, Project Michigan Report 2900-91-T, Willow Run Labs., Ann Arbor, Michigan, June 1960.
2. Kovach, L. D. and H. F. Meissinger: Solution of Algebraic Equations, Linear Programming, and Parameter Optimization, Part 5.11 of Computer Handbook edited by Huskey and Korn, McGraw-Hill Book Co., Inc., New York, 1962.
3. Tomlinson, N.P., et al: Analog-computer Construction of Conformal Maps in Fluid Dynamics, J. Appl. Phys., 21:307 (1950).
4. Kovach, L.D.: Miscellaneous Techniques, part 6.6 of Computer Handbook edited by Huskey and Korn, McGraw-Hill Book Co., Inc., New York, 1962.
5. Szymanski, E.: Analysis of Non-Sinusoidal Voltage Wave-Forms, article in Electronic Computers in Engineering Education, Univ. of Mich., Aug. 26, 1960.
6. Bellman, R.: Some New Techniques in the Dynamic Programming Solution of Variational Problems, Quart. Appl. Math.,vol. 16, pp. 295-305, 1958.
7. Douglas Aircraft Co. Report E.S. 40427, Automatic Analog Mechanization of the Shock Spectrum Problem, Aug. 18, 1961.
8. Kovach, L.D. and W. Comley: A New, Solid-State, Nonlinear Analog Component, IRE Trans. on Elec. Comp., vol. EC-9, no. 4, pp. 496-503, Dec., 1960.
9. Kovach, L.D. and W. Comley: Simple Analogs for Useful Nonlinearities, Cont. Eng. 7(11), Nov. 1960, 135-137.
10. Van Valkenburg, M.E.: Topological Synthesis, IRE Wescon Record (part 2) 2,3-9, (1958).
11. Watanabe, H.: A Method of Tree Expansion in Network Topology, IRE Trans. on Circ. Theory, vol. CT-8(1), 4-10, March, 1961.
12. Leon, B.J. and C.A. Bean: Analysis and Design of Parametric Amplifiers with the Aid of a 709 Computer, IRE Trans. on Circ. Theory, vol. CT-9(3), 210-215, Sept., 1961.



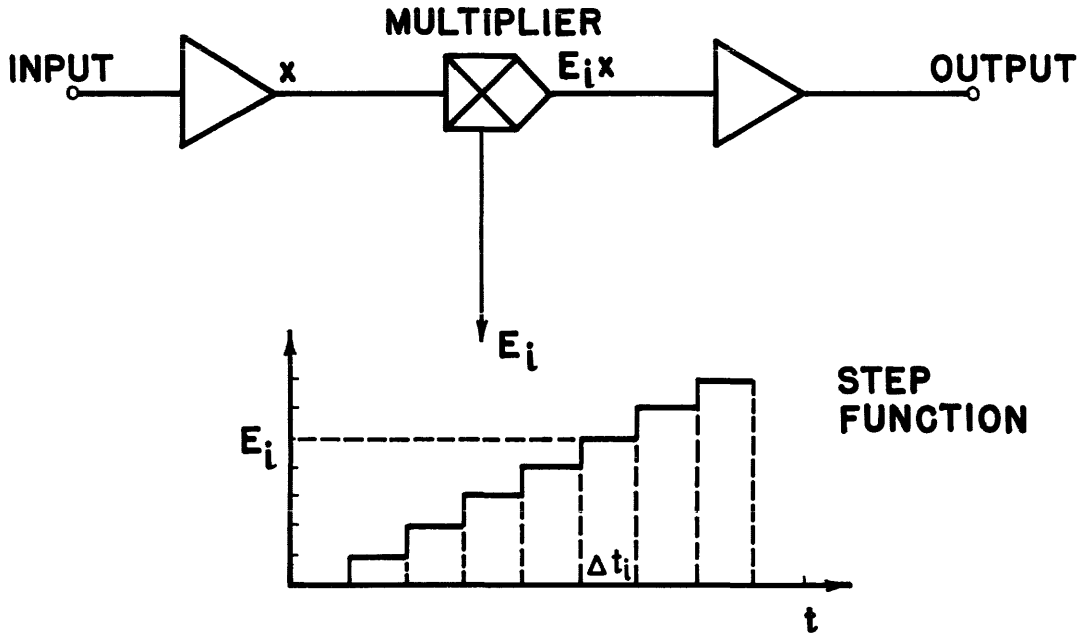
1. A random walk in the Monte Carlo method



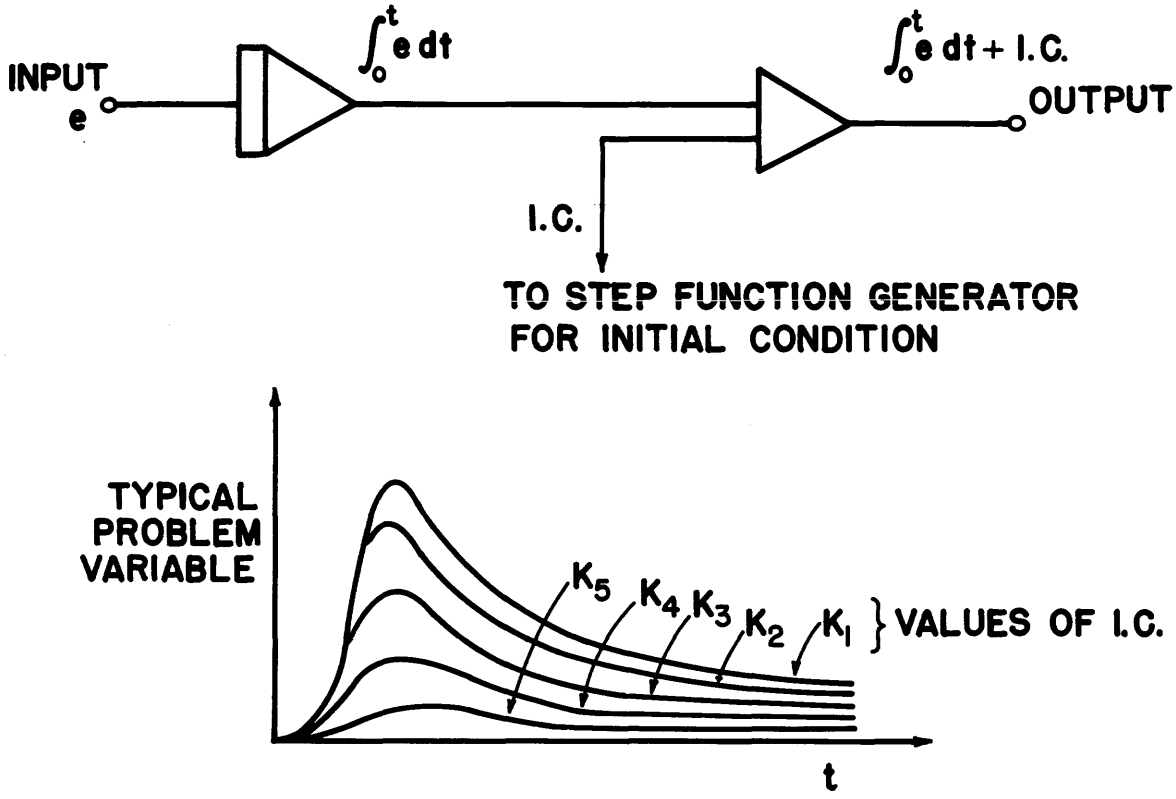
2. Solving Laplace's equation by the Monte Carlo method (after Chuang, et al[1])
   a) Block diagram of system
   b) Circuit for generating random walks



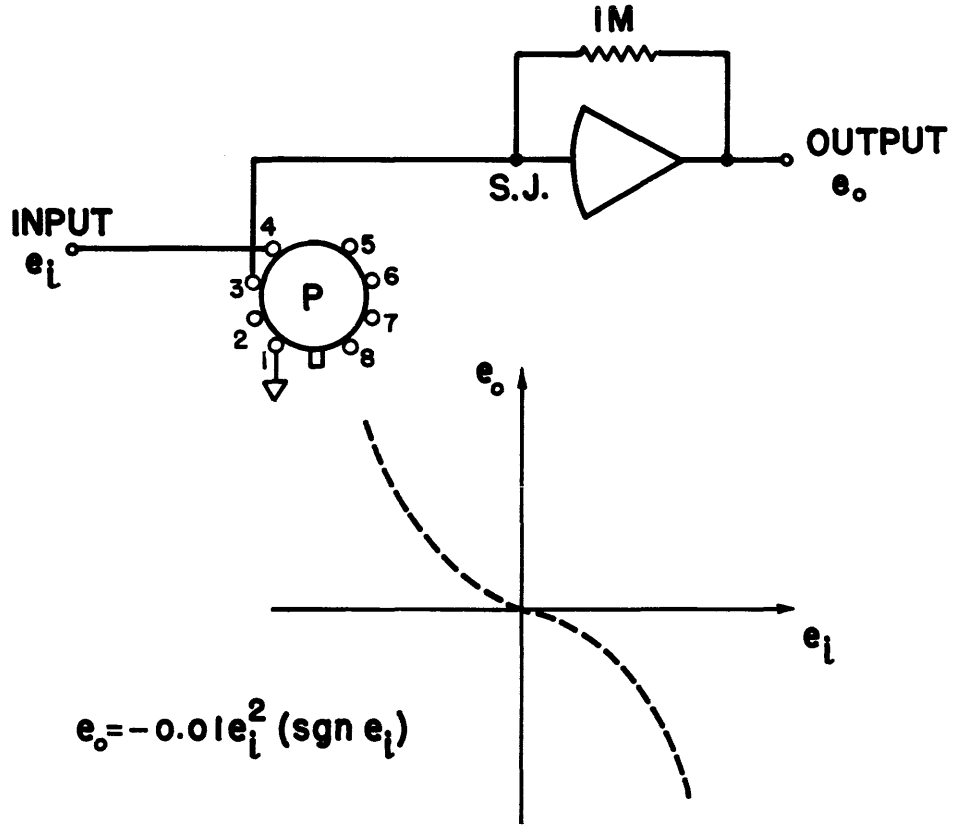
3. Step-function generator



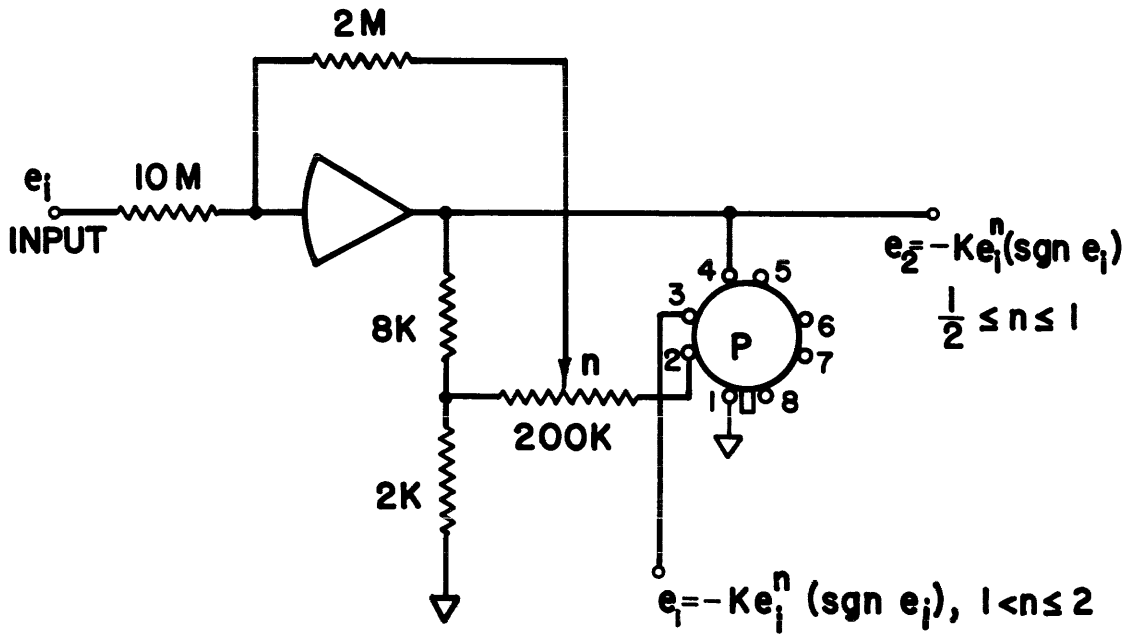
4. Automatic changing of a constant coefficient



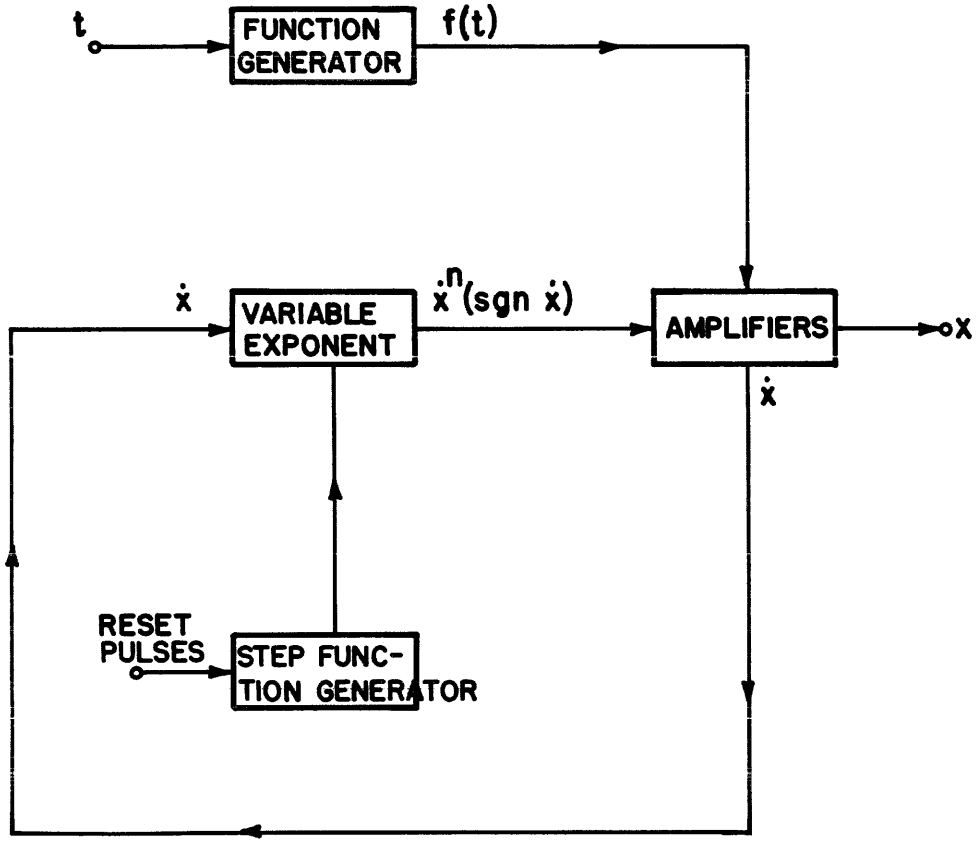
5. Automatic changing of initial conditions and the resulting solutions



6. Basic circuit and transfer function of the Quadratron



7. Circuit for obtaining an arbitrary exponent



8. Block diagram of system for solving a nonlinear differential equation

ANALOG SIMULATION OF PARTICLE TRAJECTORIES IN FLUID FLOW

Vance D. Norum
Space-General Corporation
Glendale, California

Marvin Adelberg and Robert L. Farrenkopf
Space Technology Laboratories, Inc.
Redondo Beach, California

Abstract

This paper presents a detailed account of the analog simulation of particle trajectories in a two-dimensional fluid flow field governed by Laplace's equation. A conductive surface is used as a direct analog of the two-dimensional fluid flow field in conjunction with an electronic analog computer to determine the trajectories of particles in the presence of fluid flow. Emphasis is placed on the concept of accuracy of the particle trajectories as well as error criteria by which trajectory accuracy can be judged; and on the sources of error inherent in their determination.

A detailed error analysis is presented in which a suitable error model is derived and certain inaccuracies in the computing equipment are assumed in order to predict their effect on the particle trajectories. An example is presented to illustrate the types and magnitudes of errors that exist in a typical problem. The analog simulation is also used to obtain trajectories in a potential flow field distorted by the presence of a cylinder and the results are then compared to a similar case obtained by other authors using a different approach. These results were comparable, with suitable explanations for the differences.

I. Introduction

The problem concerns the determination of the trajectory of a particle (or group of particles) in a flowing gas stream. This problem was of particular interest 15 years ago with regard to impingement of rain and ice formation on leading edges of aircraft, and is currently of interest in solid propellant rocket engine design where the products of combustion are 10 to 40 percent in the condensed phase (liquid or solid).[1] Impingement of these particles upon the interior surfaces of a rocket engine could seriously impair engine performance by imparting intolerable heat transfer rates, by causing serious erosion, and by converting the kinetic energy of the particles to thermal energy. Also of interest is the influence of the particle trajectory upon the thrust vector angle of the exhaust products.

A fundamental assumption commonly made in formulating the mathematical model for this problem is that the fluid or gas flow can be adequately represented as potential flow satisfying Laplace's equation. For simple flow field geometry this equation can be solved analytically. Otherwise, any of the well known, long established analog techniques for simulating Laplace's equation can be used.[2,3,4] Although a three-dimensional simulation is feasible by use of an electrolytic tank,[5,6,7,8] discussion will be limited to the simulation of Laplace's equation in two dimensions by means of the commonly employed conducting surface known as "Teledeltos paper."[9] By present day standards of analog simulation this approach is actually quite crude due to the lack of uniformity, isotropicity, and homogeneity in the paper, qualities which are highly desirable in any conductive solid used for analog simulation purposes. There has, however, been some activity in the development of such conductive surfaces.[10,11,12,13,14]

Previously reported particle trajectory analyses[15,16] assumed simple flow field geometry, thus enabling Laplace's equation to be solved analytically. An analog computer was then employed in these analyses to solve the highly nonlinear differential equations characterizing the dynamics of the particle within the flow field.

The analyses of particle trajectories in electronic vacuum tubes having arbitrary flow field geometries have been reported in References 13, 18, and 19 using electrolytic tanks and conducting surfaces to simulate Poisson's as well as Laplace's equation, thus incorporating space charge effects into the simulation. Items of critical importance that have either been totally neglected or treated very lightly in all previous work on the analog simulation of particle trajectories in flow fields have been detailed error analyses and even suitable definitions of error to support the claims of two percent accuracy, etc.* It is the intent of this paper to emphasize these long neglected items and to focus particular attention on those aspects of particle trajectory simulation using conductive surface analogs where future efforts along these lines might be most profitably placed.

---

* Space General Corp.
Glendale, California

** Space Technology Laboratories, Inc.
Redondo Beach, California

* As mentioned in Reference 3 the concept of accuracy and a suitable definition of error in simulations of this type is far from obvious.

A formal statement of the problem and a detailed discussion of the analog simulation will be followed by a detailed error analysis which will include: formulation of error criteria for simulations of this type, derivation of an error model together with a discussion of inputs to the error model (i.e., the sources of error) and finally an example illustrating the types and magnitudes of the various errors that exist in a typical problem. The error analysis of this hypothetical case will be followed by an actual example, simulated on the computer, for which results are compared with those obtained by other authors considering similar cases but using different techniques.

II. Particle Trajectories in Fluid Flow

A. Problem Statement

The fluid dynamics problem of determining the trajectories of small particles in two dimension fluid flow can be solved analytically only in the simplest of cases. In general, the fluid flow is governed by the highly nonlinear Navier-Stokes partial differential equations. Simplifying assumptions can often be justified to reduce these equations to Laplace's equation, which can then be solved analytically, in some cases, for the fluid velocity as a function of position coordinates. Unfortunately, even when this latter step is accomplished the particle dynamical equations must be integrated to yield the coordinates of a particle trajectory as a function of time. Except in the most trivial cases these equations are always nonlinear and a closed form solution is impossible. Consequently, ultimate solution of the problem must be effected either numerically via a digital computer or else simulated using analog computer techniques.*

A number of assumptions have been made in the analysis of this problem and these include:

Two independent space variables (two dimensional flow).

The presence or motion of the particle does not influence the flow of the gas (action of the gas on the particles is considered but not action of the particles upon the gas).

Motion of one particle is independent of the motion of any/all neighboring particles.

A unique value of drag coefficient for the particle exists as a function of its Reynolds number and each particle is spherical in nature.

The gas flow is potential flow (satisfies Laplace's equation).

The only force acting on these particles is the drag force (gravitational and other effects are neglected).

The particle has constant mass and density.

The gas is incompressible.

A derivation of the mathematical model characterizing particle trajectories under the above assumptions is presented in Appendix A together with a brief discussion of the physical significance of the various normalizations and parameters. For simulation purposes, the equations are most simply represented as

$$\frac{\partial u(x, y)}{\partial y} - \frac{\partial v(x, y)}{\partial x} = 0 \tag{1}$$

$$\dot{x} = \frac{C_D R_e}{24K}\left[u(x, y) - \dot{x}\right] \tag{2}$$

$$\dot{y} = \frac{C_D R_e}{24K}\left[v(x, y) - \dot{y}\right] \tag{3}$$

where u, x, v, y represent normalized gas and particle velocities in the x and y directions respectively and C_D is, in general, some nonlinear function of R_e where R_e is related to system dependent variables by*

$$R_e = k\left\{\left[u(x, y) - \dot{x}\right]^2 + \left[v(x, y) - \dot{y}\right]^2\right\}^{1/2} \tag{4}$$

**

The physical significance of the coefficient K is discussed in Appendix A. The simultaneous solution of Equations (1) to (4) for x and y as functions of time represents the desired result.

There are two major problems which must be solved simultaneously to achieve the above mentioned desired result, namely the solution of partial differential Equation (1) for a particular flow field configuration and then using this result to solve nonlinear ordinary differential Equations (2) to (4) for x(t) and y(t). The method of analog simulation employing a conductive surface affords flexibility in varying both particle parameters and flow field geometry. Consequently this approach was used in obtaining the results reported in this paper. It should be noted that very little experimental work has been done to corroborate the results obtained by use of the above mentioned mathematical model. Reference 16 reports experimental results for particle trajectories in flow over a cylinder, the same case used as the second example of this paper.

*Both computational approaches to different aspects of this problem have been treated at STL as indicated by References 20 and 22.

*See Equation (A-8) in Appendix A.

**See Equation (A-7) in Appendix A.

B. The Analog Simulation

In the preceding section we have considered the particle equations of motion as a function of the gas velocities which reflect their driving forces. Moreover, these gas velocities were related to Laplace's equation under one of the assumptions. This section will consider the mechanization of Equations (1) to (4) on the analog computer and the sources of error inherent in this mechanization.

Figure 3 shows an overall schematic of the problem simulation along with the location of possible error sources. The analog computer simulates the particle equations of motion and provides the desired output, namely the x and y coordinates of the particle as a function of time. An X-Y plotter, on which is mounted an electrically energized conducting surface similar in shape to the physical problem (thus satisfying the required boundary conditions), receives x and y as inputs and positions a voltage sensing probe to the proper coordinates. The probe voltages at (x, y) are related to the driving forces acting on the particle and thus form the inputs for the equations mechanized on the computer which in this case are identical to Equations (1) to (4). All normalization coefficients have been taken as unity as indicated in Appendix A. The term $C_D R_e$ is unfortunately a nonlinear function of the magnitude of the difference between the normalized gas and particle velocities. The actual simulation of Equations (2) and (3) is straightforward (the use of the nonlinear equipment being indicated in the simulation schematic of Figure 3) and need not be discussed further except in regards to the forthcoming error analysis.

It remains to discuss the simulation of Equation (1), the solution of which is used as an input to the simulation of Equations (2) and (3). Since the potential function satisfying Laplace's equation is a harmonic function, the conducting surface used in representing this function can be energized either of two ways per the Cauchy-Riemann conditions. In one arrangement voltages are imposed along two surface boundaries which are the direct reflection of two fluid flow potential lines, ϕ, in the physical problem. In the alternate arrangement the surface is energized along boundaries that are the direct analogs of two stream lines in the physical problem. The latter approach was adopted in this investigation. This flexibility is due to the orthogonality of the fluid flow potential lines and stream lines. Since in the actual physical problem, the components of gas velocity u and v are related to the gas potential and stream functions ϕ and ϕ' by the previously mentioned Cauchy-Riemann conditions, it follows that in this simulation

$$u(x, y) = \frac{\partial V(x, y)}{\partial y} \tag{5}$$

$$v(x, y) = -\frac{\partial V(x, y)}{\partial x} \tag{6}$$

where $V(x, y)$ is the conducting surface voltage at the point (x, y).

Finite difference approximations to these partial derivatives can be obtained by using a multipronged probe which measures the potential at several points in the vicinity of (x, y). As an example of one possible arrangement, consider that indicated in Figure 1. The symmetrical probe consists of four "points"

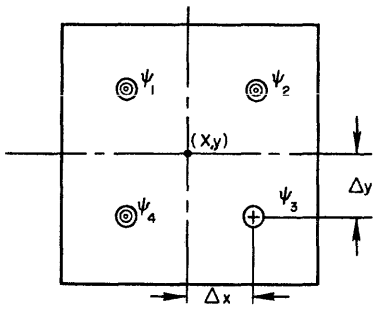


Figure 1. Orientation of Four-Element Probe

which measure respectively the voltages ψ_1, ψ_2, ψ_3, and ψ_4 near (x, y). The following relations can then be written

$$\frac{\partial \psi}{\partial x} = \frac{\psi_2 - \psi_1 + \psi_3 - \psi_4}{4\Delta x} \tag{7}$$

$$\frac{\partial \psi}{\partial y} = \frac{\psi_1 - \psi_4 + \psi_2 - \psi_3}{4\Delta y} \tag{8}$$

The determination of voltage derivatives by averaging two finite voltage differences in this way (instead of using only one difference) has certain advantages which will be discussed later.

In view of Equations (5) to (8) it is apparent that the driving voltages for the analog computer mechanication can be obtained directly from ψ_1 to ψ_4 taking into account proper polarities and certain constant scale factors. Figure 3 schematically indicates the required signal flow lines.

C. Sources of Simulation Errors

Let us now give a brief qualitative description of the major sources of simulation errors. A quantitative discussion of the errors appears in Appendix B. These errors can be conveniently broken down into conducting surface errors, probe errors, and analog computer component errors. It is assumed that the X-Y plotter, upon which the conductive surface analog is carefully oriented, can be adjusted to have sufficient sensitivity and that the effects of plotter backlash can be minimized.

1. Computer Errors. Servo Multiplier Errors. Since servos are used to perform the required multiplications, the effects of misalignment of the various servo potentiometers must be considered. Other servo error sources will be assumed negligible.

Probe Element Gain Errors. If each probe element is not multiplied by precisely the same computer gain, significant errors in the simulated particle trajectory will result.

Other computer errors will be assumed to have negligible effect on the simulated particle trajectories although it should be pointed out that it is extremely important to accurately balance each amplifier to have zero output for a grounded input.

2. Conducting Surface Errors. Misalignment of the conducting surface coordinates with respect to the coordinate axes of the X-Y plotter.

Nonhomogeneity of the conducting surface resistivity.

Computer loading effects on the conducting surface.

3. Probe Errors. Errors caused by approximating surface voltage derivatives by the differences of finite surface voltages.

Angular misorientation of the probe on the carriage of the X-Y plotter.

Probe elements becoming dirty or jammed in their housing causing nonrepeatable solutions.

III. Error Analysis

A. Error Criteria

The errors mentioned in Section II-C will cause errors in the x and y components of the particle trajectory. As will be shown subsequently, a quantitative measure of this error in a given particle trajectory can be obtained from a suitable error model having the aforementioned errors as "inputs." However, even when the errors in a given particle trajectory can be determined it is necessary to be able to evaluate the tolerability of these trajectory errors by some standard criterion.

In order to illustrate the difficulty associated with a comparison of trajectories by an error criterion in problems of this type consider the exact and actual trajectories in Figure 2. At a specific value of time, say t', the coordinates of the exact trajectory are x(t') and y(t') whereas the actual simulated trajectory has coordinates x'(t'), y'(t'). Therefore the coordinate errors are

$$\delta x(t') = x'(t') - x(t') \tag{9}$$

$$\delta y(t') = y'(t') - y(t') \tag{10}$$

and the magnitude of the trajectory error is

$$\delta r(t') = \left[\overline{\delta x(t')}^2 + \overline{\delta y(t')}^2\right]^{1/2} \tag{11}$$

While $\delta r(t')$ is the actual trajectory error, it alone may not be a sufficient criterion for evaluating the usefulness of this analog simulation for a particular problem since it takes no account of the potential flow field's geometry. To illustrate this point consider a specific value of error, δr, at both x_1 and x_2 at which the cross-sectional dimensions are respectively AA and BB. Now δr is apt to be far more critical in the latter case and thus, perhaps, δr should be weighted by some geometrical surface dimension. No dimension stands out as the obvious choice. The one selected here is the distance, D, between boundaries normal to the particle velocity vector. A typical D is illustrated in Figure 2. Thus the error criterion becomes

$$\epsilon = \frac{\left[\overline{\delta x(t)}^2 + \overline{\delta y(t)}^2\right]^{1/2}}{D(t)} \tag{12}$$

This is probably the most useful criterion since it deals with a weighted error magnitude. However, $\epsilon(t)$ constantly changes with time and is thus only a local figure of merit rather than a figure of merit for the entire trajectory. To establish such a criterion for the entire trajectory the weighted rms trajectory $\bar{\epsilon}$ is adopted.

$$\bar{\epsilon}(t) = \left[\frac{1}{r}\int_r \epsilon^2\, dr\right]^{1/2} \tag{13}$$

Equations (12) and (13) represent useful error criteria. However, the nature of the error equations (whose derivation follows) causes ϵ and $\bar{\epsilon}$ to be extremely sensitive to trajectory length $\int_r dr$ due to the double integrations of particle accelerations (in which the error sources occur) in the determination of δx and δy. Thus the expected trajectory error at x_2 exceeds that at x_1, for example. While ϵ and $\bar{\epsilon}$ are not necessarily linearly related to $\int_r dr$, a more useful set of criteria for comparing simulations of this type is

$$\epsilon_r = \left[\delta x^2 + \delta y^2\right]^{1/2}\left[D\int_r dr\right]^{-1} \tag{14}$$

$$\bar{\epsilon}_r = \left[\frac{1}{r}\int_r \left(\frac{\delta r}{D}\right)^2 dr\right]^{1/2}\left[\int_r dr\right]^{-1} \tag{15}$$

The example considered in part C of this section specifies trajectory errors in terms of ϵ_r and $\bar{\epsilon}_r$. It is, of course, an easy matter to convert back to ϵ and $\bar{\epsilon}$.

B. General Trajectory Equations

The previous section was devoted to qualitative aspects of error in a particle trajectory. It is the intent of this section to derive a mathematical model that will adequately characterize the behavior of this error in a quantitative fashion once the inputs to the error model have been specified. The structure for deriving the error model is indicated schematically in Figure 3.

The pertinent equations are easily derived from the basic system equations, namely

$$\ddot{x} = \frac{s}{K} w_x \qquad \text{(horizontal particle acceleration)} \qquad (16)$$

$$\ddot{y} = \frac{s}{K} w_y \qquad \text{(vertical particle acceleration)} \qquad (17)$$

$$s = s(w^2) \qquad (C_D R_E/24) \qquad (18)$$

$$w^2 = w_x^2 + w_y^2 \qquad \text{(relative velocity squared)} \qquad (19)$$

$$u = \frac{\partial V}{\partial y} = V_y \qquad (20)$$

$$v = -\frac{\partial V}{\partial x} = -V_x \qquad (21)$$

(gas velocity relations in terms of surface voltage derivatives) *

$$V_{xx} + V_{yy} = 0 \qquad \text{(Laplace's equation)} \qquad (22)$$

Here (x, y) are the particle coordinates, u and v the corresponding components of gas velocity at (x, y) and w_x and w_y are defined by

$$w_x = u - \dot{x} \qquad (23)$$

$$w_y = v - \dot{y} \qquad (24)$$

These equations are mechanized on the analog computer to yield the particle trajectory coordinates (x, y) as continuous functions of time. However, the independent additive errors discussed in Section II-C and indicated in Figure 3, namely δV_x, δV_y, ϵ_f, ϵ_{11}, ϵ_{12}, ϵ_{21}, and ϵ_{22}, give rise to trajectory errors δx and δy and hence rise to gas velocity errors δu and δv due to mispositioning of the X-Y plotter. Thus, instead of solving the desired Equations (16) to (22), the computer solves these equations with the dependent variables replaced by the following terms

$$u \longrightarrow u + \delta u + \delta V_y \qquad (25)$$

$$v \longrightarrow v + \delta v - \delta V_x \qquad (26)$$

$$\dot{x} \longrightarrow \dot{x} + \delta\dot{x} \qquad (27)$$

$$\dot{y} \longrightarrow \dot{y} + \delta\dot{y} \qquad (28)$$

$$\ddot{x} \longrightarrow \ddot{x} + \delta\ddot{x} + \epsilon_{12} \qquad (29)$$

$$\ddot{y} \longrightarrow \ddot{y} + \delta\ddot{y} + \epsilon_{22} \qquad (30)$$

$$s \longrightarrow s + \delta s + e_f \qquad (31)$$

$$w^2 \longrightarrow w^2 + \epsilon_{11} + \epsilon_{21} + \delta w^2 \qquad (32)$$

These values replace the original terms in Equations (16) to (22). δw^2, δu, δV, δs are replaced in the resultant equations by noting that*

$$\delta w^2 = \delta\left(w_x^2 + w_y^2\right) = 2w_x\delta w_x + 2w_y\delta w_y \qquad (33)$$

$$\delta u = V_{yx}\delta x + V_{yy}\delta y \qquad (34)$$

$$\delta v = -V_{xx}\delta x - V_{xy}\delta y \qquad (35)$$

$$\delta s = \frac{ds}{dw^2}\delta w^2 \qquad (36)$$

This leads to two lengthy equations in $\ddot{x} + \delta\ddot{x}$ and $\ddot{y} + \delta\ddot{y}$, which, upon subtracting respectively Equations (16) and (17), yield the desired differential equations

$$\left[a_1\frac{d^2}{dt^2} + a_2\frac{d}{dt} + a_3\right]\delta x + \left[a_4\frac{d}{dt} + a_5\right]\delta y = E_1 \qquad (37)$$

$$\left[b_4\frac{d}{dt} + b_5\right]\delta x + \left[b_1\frac{d^2}{dt^2} + b_2\frac{d}{dt} + b_3\right]\delta y = E_2 \qquad (38)$$

where

$$a_1 = b_1 = K \qquad (39)$$

$$a_2 = s + 2\frac{ds}{dw^2}w_x^2 \qquad (40)$$

$$b_2 = s + 2\frac{ds}{dw^2}w_y^2 \qquad (41)$$

$$a_3 = -\left[s + 2\frac{ds}{dw^2}w_x^2\right]V_{yx} - 2\frac{ds}{dw^2}w_x w_y V_{yy} \qquad (42)$$

$$b_3 = \left[s + 2\frac{ds}{dw^2}w_y^2\right]V_{xy} + 2\frac{ds}{dw^2}w_x w_y V_{xx} \qquad (43)$$

$$a_4 = b_4 = 2\frac{ds}{dw^2}w_x w_y \qquad (44)$$

$$a_5 = \left[s + 2\frac{ds}{dw^2}w_x^2\right]V_{xx} + 2\frac{ds}{dw^2}w_x w_y V_{xy} \qquad (45)$$

---

* $V_x = \frac{\partial v}{\partial x}$, In this section an alphabetic subscript denotes a partial derivative with respect to the subscripted variable.

* All items greater than first order are neglected.

$$b_5 = -\left[s + 2\frac{ds}{dw^2}w_y^2\right]V_{yy} - 2\frac{ds}{dw^2}w_x w_y V_{yx} \quad (46)$$

$$E_1 = K\epsilon_{12} + \left[s + 2\frac{ds}{dw^2}w_x^2\right]\delta V_y - 2\frac{ds}{dw^2}w_x w_y \delta V_x + w_x\left[\epsilon_f + \frac{ds}{dw^2}(\epsilon_{11} + \epsilon_{21})\right] \quad (47)$$

$$E_2 = K\epsilon_{22} - \left[s + 2\frac{ds}{dw^2}w_y^2\right]\delta V_x + 2\frac{ds}{dw^2}w_x w_y \delta V_y + w_y\left[\epsilon_f + \frac{ds}{dw^2}(\epsilon_{21} + \epsilon_{11})\right] \quad (48)$$

The a_i and b_i above are time varying coefficients available from the simulation. E_1 and E_2, the inputs to Equations (37) and (38), are generated by the error sources δV_x, δV_y, ϵ_i, ϵ_f and certain other obtainable time varying coefficients as indicated in Equations (47) and (48). δV_x and δV_y are the accumulation of errors inherent in the analog determination of u and v.

$$\delta V_x = \delta_1 V_x + \delta_2 V_x + \delta_3 V_x + \delta_4 V_x + \delta_5 V_x \quad (49)$$

$$\delta V_y = \delta_1 V_y + \delta_2 V_y + \delta_3 V_y + \delta_4 V_y + \delta_5 V_y \quad (50)$$

where

- δ_1 refers to voltage error sources caused by conducting surface misalignment
- δ_2 refers to voltage error sources caused by angular misalignment of probe
- δ_3 refers to voltage error sources caused by unequal probe element gains
- δ_4 refers to voltage error sources caused by finite difference approximation to voltage gradient components
- δ_5 refers to voltage error sources caused by conducting surface irregularities

ϵ_{11}, ϵ_{12}, ϵ_{21}, ϵ_{22} are errors sources caused by inaccuracies in the computer servo multipliers and ϵ_f refers to errors in the function generator. A detailed discussion relating these voltage error sources to their physical causes is presented in Appendix B.

C. Example

The preceding error analysis is now applied to the simple problem of determining the particle trajectory error in a fluid flow field characterized by parallel stream lines. This problem is considered since the error equations simplify to the extent of permitting a solution without the use of elaborate computing equipment. One shortcoming of this example is that many of the errors become negligible and thus a more optimistic answer will result than if a more complex problem had been considered. Nevertheless, this example should lend a deep insight into the more general problems and permit the use of engineering judgment to predict errors for more complex cases. Also the results of this example establish a lower limit on expected trajectory errors for a problem more involved than this one. The flow field configuration and trajectory chosen for this example are indicated in Figure 4 which represents the conducting surface analog of the fluid flow field. A voltage difference of V_s is applied across two parallel boundaries of the field as indicated in Figure 4. The particle (represented by the probe) begins at point A ($x = o$, $y = y_o$) with the same velocity as the fluid U, and follows trajectory A-C up to some final point. This trajectory differs from the ideal trajectory A-B due to errors in the simulation. The object is to determine the magnitude of these errors and weigh them by the aforementioned criteria. For this case, Equations (37) and (38), the error model equations, simplify to

$$K\frac{d^2}{dt^2}\delta x + s\frac{d}{dt}\delta x = E_1 \quad (51)$$

$$K\frac{d^2}{dt^2}\delta y + s\frac{d}{dt}\delta y = E_2 \quad (52)$$

where K and s are constants and

$$E_1 = \left(G_{1y} - G_{3y}\right)\left(\frac{Y}{4\Delta y}\right)\left(\frac{V}{V_s}\right) + \left(G_{2y} - G_{4y}\right)\left(\frac{Y}{4\Delta y}\right)\left(\frac{V}{V_s}\right) + 2\ell_{12} + \delta_5 V_y \quad (53)$$

$$E_2 = \alpha + \beta + \left(-G_{1x} + G_{3x}\right)\left(\frac{Y}{4\Delta x}\right)\left(\frac{V}{V_s}\right) + \left(G_{2x} - G_{4x}\right)\left(\frac{Y}{4\Delta y}\right)\left(\frac{V}{V_s}\right) + 2\ell_{22} + \delta_6 V_y \quad (54)$$

A derivation of the above expressions for E_1 and E_2 is included as Appendix C.

All other terms in the general error equations can be neglected since they involve either products of small terms such as w_x, w_y, ϵ_{12}, etc., and thus introduce only second order effects or because they involve V_{xx}, V_{yy}, V_{xy}, and $ds/d(w^2)$ which are zero for this problem. The vanishing of the second order partials of surface voltage is a characteristic of parallel stream lines; $ds/d(w^2)$ equals zero, since for low Reynolds Numbers s is approximately constant, this constant being unity.

To evaluate the error of the trajectory let the error indices defined by Equations (14) and (15) be computed for this example (where D is unity).

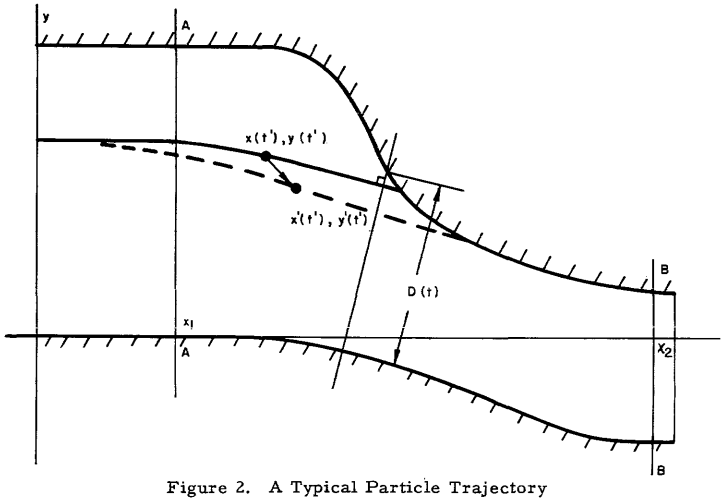


Figure 2. A Typical Particle Trajectory

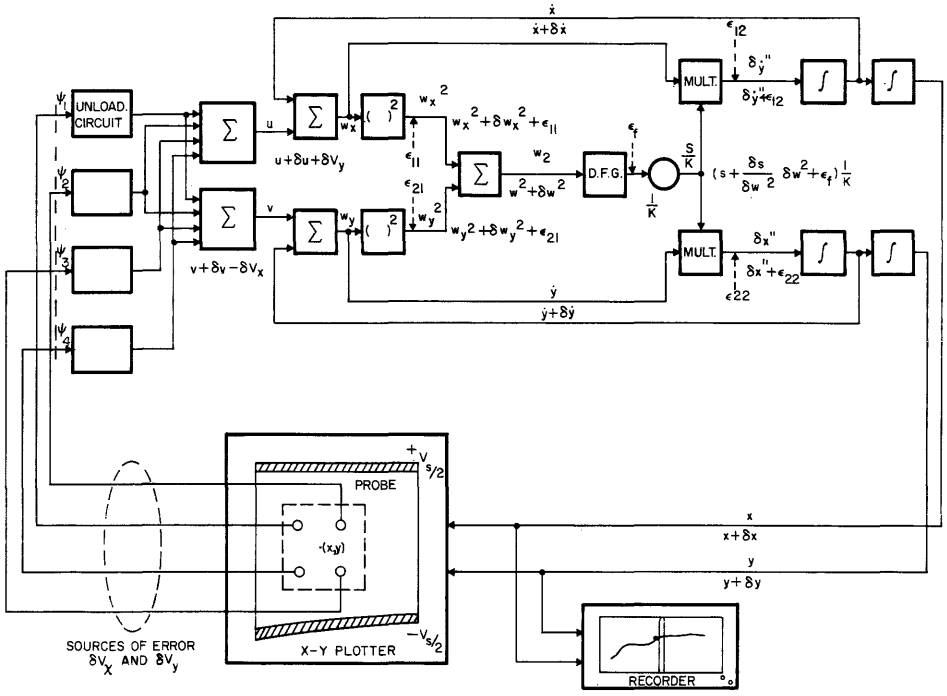


Figure 3. Simulation Schematic

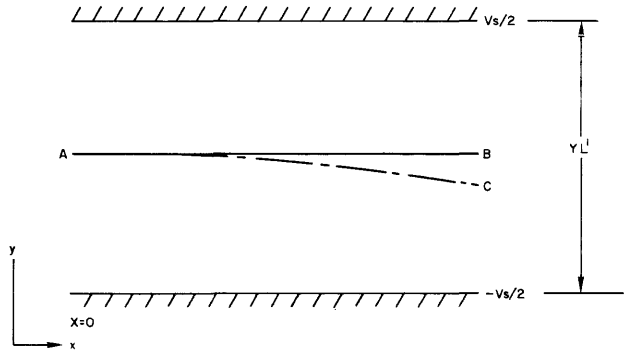


Figure 4. A Particle Trajectory in a Potential Flow Field with Parallel Stream Lines

$$\epsilon_r = \frac{\delta r}{r} \tag{55}$$

$$\overline{\epsilon}_r = \frac{1}{r}\left[\frac{1}{r}\int_r (\delta r)^2\, dr\right]^{1/2} \tag{56}$$

$$r = \int_r dr \quad \text{where} \quad (dr)^2 = (dx)^2 + (dy)^2 \tag{57}$$

and where

$$(\delta r)^2 = (\delta x)^2 + (\delta y)^2 \tag{58}$$

Observing Equations (53) and (54) it is noted that all terms except $\delta_5 V_x$ and $\delta_5 V_y$ remain constant for the duration of a trajectory run. It therefore becomes convenient to reduce E_1 and E_2 to the following forms:

$$E_1 = f_1 + g_1 \tag{59}$$

$$E_2 = f_2 + g_2 \tag{60}$$

where*

$$f_1 = (G_{1y} - G_{3y})\left(\frac{Y}{4\Delta y}\right)\left(\frac{V}{V_x}\right) + (G_{2y} - G_{4y})\left(\frac{Y}{4\Delta y}\right)\left(\frac{V}{V_s}\right) + 2\ell_{12} \tag{61}$$

$$g_1 = \delta_5 V_y \tag{62}$$

$$f_2 = \alpha + \beta + (G_{3x} - G_{1x})\left(\frac{Y}{4\Delta x}\right)\left(\frac{V}{V_s}\right) + (G_{2x} - G_{4x})\left(\frac{Y}{4\Delta x}\right)\left(\frac{V}{V_s}\right) + 2\ell_{22} \tag{63}$$

$$g_2 = \delta_5 V_x \tag{64}$$

In Equations (59) and (60), f_1 and f_2 will assume some constant value throughout the determination of a given trajectory whereas g_1 and g_2 will each be a two-parameter family of random variables with the coordinates x and y as parameters. Since Equations (51) and (52) are linear, the total solutions δx and δy can be decomposed as follows

$$\delta x = \delta x_1 + \delta x_2 \tag{65}$$

$$\delta y = \delta y_1 + \delta y_2 \tag{66}$$

* $\frac{Y}{V_s} = \frac{1}{V_y} = \frac{1}{u} = 1$ for this simple problem.

where δx_1 is the solution of (51) when E_1 is replaced by f_1, δx_2 is the solution of (51) when E_1 is replaced by g_1, δy_1 is the solution of (52) when E_2 is replaced by f_2, and δy_2 is the solution of (52) when E_2 is replaced by g_2.

Consider now the evaluation of $(\delta r)^2$ and $\overline{\delta r} = 1/t \int_o^t (\overline{\delta r})^2\, dt$.

$$(\delta r)^2 = \left[(\delta x_1)^2 + (\delta y_1)^2\right] + 2\left[\delta x_1 \delta x_2 + \delta y_1 \delta y_2\right] + \left[(\delta x_2)^2 + (\delta y_2)^2\right] \tag{67}$$

$$\overline{\delta^2 r} = \frac{1}{t}\int_o^t \left[(\delta x_1)^2 + (\delta y_1)^2\right] dt' + \frac{2}{t}\int_o^t (\delta x_1 \delta x_2 + \delta y_1 \delta y_2)\, dt' + \frac{1}{t}\int_o^t \left[(\delta x_2)^2 + (\delta y_2)^2\right] dt' \tag{68}$$

It is apparent that once $(\delta r)^2$ and $\overline{\delta^2 r}$ have been determined, the chosen criteria ϵ_r and $\overline{\epsilon_r}$ defined in Equations (55) and (56) respectively follow immediately

$$\left[\text{note: } \overline{\delta^2 r} \equiv \overline{(\delta r)^2}\right]$$

Unfortunately $(\delta r)^2$ cannot be computed unless δx_2 and δy_2 are first determined as functions of time. However, this involves complete knowledge of the conducting surface characteristics as a random function of x and y which is difficult to obtain. Therefore the best that can be done in evaluating $(\delta r)^2$ is to determine the effects of all errors except those originating from the conducting surface. To this end define

$$(\delta r')^2 = (\delta x_1)^2 + (\delta y_1)^2 \tag{69}$$

Since f_1 and f_2 are constant throughout the determination of any one trajectory, $(\delta x_1)^2$ and $(\delta y_1)^2$ are easily determined by solving Equations (51) and (52); the results, for zero initial conditions and s = 1, are:

$$(\delta x_1)^2 = f_1^2\left\{t - K\left[1 - \exp(-t/K)\right]\right\}^2 \tag{70}$$

$$(\delta y_1)^2 = f_2^2\left\{t - K\left[1 - \exp(-t/K)\right]\right\}^2 \tag{71}$$

The relationship between time t and trajectory length r for the trajectory shown in Figure 4 can easily be obtained by integrating the equations of exact particle motion: $\ddot{x} = \ddot{y} = 0$ subject to

$\dot{x}(o) = 1$, $\dot{y}(o) = 0$, $x(o) = 0$, and $y(o) = y_o$. The result for this trajectory is that $r = x = t$. Substituting this result in Equations (70) and (71) enables $\delta r'$, defined by Equation (69), to be evaluated as

$$\epsilon'_r = \frac{\delta r'}{r} = \Omega_1\left(\frac{K}{r}\right) f_3 \tag{72}$$

where

$$\Omega_1\left(\frac{K}{r}\right) = 1 - \frac{K}{r}\left(1 - e^{-\frac{r}{K}}\right) \tag{73}$$

and

$$f_3 = + \left(f_1^2 + f_2^2\right)^{1/2} \tag{74}$$

Although it is known that f_1 and f_2 (and thus f_3) are constant for the entire run, it is not known with certainty what value the constant will assume. At best, one can only assume probability density functions for each of the individual terms of f_1 and f_2 [see Equation (61) and (63)] and then the density functions for f_1, f_2, and f_3 follow. Each of the individual terms in f_1 and f_2 will be assumed to possess a uniform probability density as sketched in Figure 5.

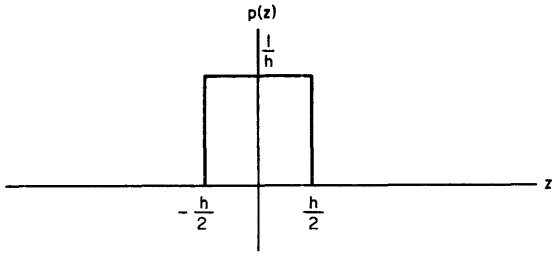


Figure 5. Uniform Probability Density Function

The maximum limits of each of the terms in f_1 and f_2 must now be discussed. The limits assumed are

$$|\alpha| \leq 0.00312 \text{ rad} \tag{75}$$

$$|\beta| \leq 0.5 \text{ degree} = 0.0087 \text{ rad} \tag{76}$$

$$\left.\begin{matrix} |2\ell_{12}| \\ |2\ell_{22}| \end{matrix}\right\} \leq 0.002 \tag{77}$$

$$\left.\begin{matrix} |G_{1x} - G_{3x}| \frac{Y}{4\Delta x}\frac{V}{V_s} \\ |G_{2x} - G_{4x}| \frac{Y}{4\Delta x}\frac{V}{V_s} \\ |G_{1y} - G_{3y}| \frac{Y}{4\Delta y}\frac{V}{V_s} \\ |G_{2y} - G_{4y}| \frac{Y}{4\Delta y}\frac{V}{V_s} \end{matrix}\right\} \leq 0.0025 \tag{78}$$

Implied in Equation (78) is $(Y/4\Delta y) = (Y/4\Delta x) = 50$ which corresponds to the value used in actual simulations; and $V/V_s = 0.5$. The probability density function for f_1 and f_2 can be determined by convolution of the density functions of the terms that compose them. Given the density functions for f_1 and f_2, the density function for f_3 follows. The details are given in Reference 23. This function is presented in Figure 6 where the abscissa values correspond directly to values of f_3. By Equation (72), the probability density function for ϵ'_r, $p_{\epsilon' r}(z)$ versus z, can be obtained from the same figure by noting that $f_3 = z\Omega_1^{-1}(K/x)$. Thus a rather complete picture of ϵ_r is obtained for the effects of all errors except the random conducting surface irregularities.

Consider now the evaluation of $\overline{\delta^2 r}$ in Equation (68). Since f_1 and f_2 are constant, the first integral can be solved using Equations (70) and (71); realizing that $r = t$ yields

$$\overline{\epsilon'} = \frac{\sqrt{\overline{\delta^2 r}}}{r} = \Omega_2\left(\frac{K}{x}\right) f_3 \tag{79}$$

where

$$\Omega_2\left(\frac{K}{x}\right) = \frac{1}{3} - \left(\frac{K}{x}\right) - 2\left(\frac{K}{x}\right)^2 e^{-\frac{x}{K}} + \left(\frac{K}{x}\right)^2 - \frac{1}{2}\left(\frac{K}{x}\right)^3 e^{-2\frac{x}{K}} + \frac{1}{2}\left(\frac{K}{x}\right)^3 \tag{80}$$

Equation (79) is of the same form as Equation (72) and thus Figure 6 can be used to predict the rms trajectory errors due to all causes except surface noise.

The last integral in Equation (68) can be broken into two parts. Considering the first part, namely

$$\frac{1}{t}\int_o^t (\delta x_2)^2 \, dt'$$

it follows that by the convolution integral

$$\delta x_2 = \int_o^t h(\tau)\, g_1(t - \tau)\, d\tau \tag{81}$$

where h(t) is the impulse response of the linear system defined by Equation (51). Thus it follows that

$$\frac{1}{t}\int_o^t (\delta x_2)^2 dt' = \int_o^t dt' \int_o^{t'} d\tau h(\tau) g_1(t' - \tau) \cdot \int_o^t d\sigma h(\sigma) g_1(t' - \sigma)$$

interchanging order of integration yields:

$$\frac{1}{t}\int_o^t (\delta x_2)^2 dt' = \int_o^t d\tau h(\tau) \int_o^t d\sigma h(\sigma) \cdot \left[\frac{1}{t}\int_o^t g_1(t' - \tau)\, g_1(t' - \sigma)\, dt'\right] \quad (82)$$

The bracketed term is a psuedo autocorrelation function of random surface irregularities.* This function can be experimentally determined for a given straight line trajectory on a given conducting surface. Considering a different straight line trajectory displaced somewhat from the first generally results in an autocorrelation function of surface irregularities that is different from the original.

These considerations make it most reasonable to compute the expected value of $\delta^2 r$, namely, $E(\delta^2 r)$. From Equations (68), (79), and (82) the result will follow.

$$E(\overline{\delta^2 r}) = r^2 \Omega_2^2 E(f_3^2) + \frac{2}{t}\int_o^t E(\delta x_1 \delta x_2 + \delta y_1 dy_2) + \frac{1}{t}\int_o^t E\left[(\delta x_2)^2 + (\delta y_2)^2\right] dt \quad (83)$$

$E(f_3^2)$ can be obtained simply by taking the second statistical moment using the distribution function in Figure 6. The second term in Equation (83) vanishes since δx_1, δy_1, δx_2, and δy_2 are all uncorrelated and assumed to have zero mean values. The evaluation of the first integral of the third term can be obtained directly from Equation (82)

$$\frac{1}{t}\int_o^t E\,(\delta x)^2\, dt = \int_o^t d\tau h(\tau) \int_o^t d\sigma h(\sigma)\, \frac{1}{t} \cdot \int_o^t E\, g_1(t' - \sigma)\, g_1(t' - \sigma)\, dt' \quad (84)$$

*A detailed treatment of these concepts may be found in Reference 25.

An average autocorrelation function was experimentally determined by averaging the results of several trajectories and found to be of the form

$$\gamma_x(\eta) = K_x e^{-k_x|\eta|} = 9 \times 10^{-4} e^{-0.46|\eta|} \quad (85)$$

where η is a surface distance argument in the x direction measured in inches. Converting η to a time argument [as required by Equation (84)] Equations (85) and (84) can be solved to yield

$$\begin{aligned}\overline{\delta^2 x_2} = {} & \frac{2K_x x}{\xi_x} - 2\left[\frac{K_x K^3 \xi_x}{K^2\xi_x^2 - 1} + \frac{K_x K}{\xi_x}\right]\left(1 - e^{-\frac{x}{K}}\right) \\ & + \frac{K_x}{\xi_x^2(K\xi_x - 1)}\left(1 - e^{-\xi_x x}\right) \\ & + \frac{K_x}{\xi_x}\left[\frac{Ke^{-\frac{x}{K}}}{K\xi_x + 1} - \frac{1}{\xi_x}\right]\left(1 - e^{-\xi_x x}\right) \\ & + \frac{K_x K^3 \xi_x}{K^2\xi_x^2 - 1}\left(1 - e^{-2\frac{x}{K}}\right) \\ & - \frac{K_x K}{\xi_x}\left[\frac{1}{K^2\xi_x^2 - 1}\right]\left[1 - e^{-x\left(\xi_x + \frac{1}{K}\right)}\right] \\ & + \frac{K_x K}{K\xi_x - 1}\left[\frac{Ke^{-\frac{x}{K}}}{K\xi_x - 1} - \frac{1}{\xi_x}\right]\left(e^{-\xi_x x} - e^{-\frac{x}{K}}\right)\end{aligned} \quad (86)$$

where

$$\xi_x = k_x L' \quad (87)$$

An identically similar result follows for the evaluation of $\delta^2 y_2$ except that K_y replaces K_x and ξ_y replaces ξ_x where

$$\gamma_y(\eta) = K_y e^{-k_y|\eta|} = 1.7 \times 10^{-4} e^{-1.42|\eta|} \quad (88)$$

$$\xi_y = k_y L' \quad (89)$$

All the error sources have now been considered in the evaluation of $\overline{\delta^2 r}$. The total expected mean squared error is then obtained using Equation (83). The square root of this equation divided by r yields $\overline{\epsilon_r}$. The results are summarized in Table 1 below.

Table 1.

		Surface Noise Errors	Expected Fixed Trajectory Errors Caused by f_1, f_2		Expected Total rms Error
K/r	L'	$\overline{\delta r/r}$	$\overline{\delta r/r}$	$\delta r/r$	$\overline{\delta r/r}$
1/10	5 in	0.0114	0.0033	0.0059	0.0119
1/10	10 in	0.0083	0.0033	0.0059	0.0089
4/5	5 in	0.0058	0.0014	0.0028	0.0060
4/5	10 in	0.0042	0.0014	0.0028	0.0044

Note: r = 5 in all cases.

Note that errors increase with an increase in the K/r factor and decrease with an increase in L'. Thus inertia parameter K of 0.5, corresponding to a characteristic dimension L that reflects to an L' of 5 inches in the simulation, will result in 1.19 percent normalized rms trajectory error for the conditions assumed in this example. The above table also contains the expected trajectory error due to the fixed errors f_1 and f_2 considered in the first part of this example. The surface noise rms error is also indicated. It should be noted that errors several times the expected values are possible but not very likely.

IV. Simulation Example

Of the many different configurations for which particle trajectory simulations were performed by the authors[22, 23] one was that of determining the particle trajectories in an infinite potential flow field distorted by the presence of a cylinder (Figure 7). This particular problem is interesting in that it has been solved differently by others*[16] and thus provides a good test case for comparing results obtained by the technique described in this paper. Trajectories were obtained using the same values of inertia parameter K and free stream Reynolds number R_{eo}** employed in Reference 16.

The conducting surface used in the problem is illustrated in Figure 7. By imposing voltages along two stream lines a field of infinite extent is simulated. A-A' represents a typical particle trajectory which at coordinates (-5L', yL') has the same velocity as the gas. The results of the simulation are shown in Figure 7 (K = 4, R_{eo} = 63.25) and Figure 8 (K = 0.5, R_{eo} = 70.75) in which the ordinate scale has been expanded to twice that of the abscissa. The corresponding solutions of Reference 16 are also plotted for comparison, and are comparable but not identical to those obtained using the conducting surface to simulate Laplace's equation and a multipronged probe to obtain a finite difference approximation to the components of the potential gradient.

If the solutions of Reference 16 are accepted* as ideal, it is still not possible to determine the trajectory errors δx and δy in the conducting surface simulation directly from Figures 8 and 9, since corresponding points at various instants of time t on the two trajectories are not known. In order to get a qualitative error measure it is assumed that δx and δy are approximately equal, in which case the errors appear to be about one to two percent of the trajectory lengths. This seems consistent with the results obtained with the simple problem considered in Section III-C.

Since each of the trajectories in Figures 8 and 9 seem to be in error by about the same amount it is probable that the conducting surface irregularities (which generally vary with position) are not the prime source. A prime suspect is a possible probe angular misalignment since a misalignment of one degree or so would be sufficient to cause the observed errors.

V. Conclusions

Based on the results obtained in the error analysis of particle trajectories in a parallel flow field and on experimental results of trajectories in the vicinity of a cylinder it can be concluded that conductive surface analogs, used in conjunction with an analog computer, can provide answers that are sufficiently accurate for a wide variety of problems. It is hard to specify a confidence level for results in the general case since this is so dependent upon the shape and size of the conducting surface, Reynolds number R_{eo}, the trajectory length r, and the particle inertia parameter K. One can merely apply judgment based on the results obtained to predict errors in the general case. Let us review these results and attempt such a judgment.

The error magnitude decreases with an increase in inertia parameter K and increases with an increase in trajectory length r. For the case of a particle trajectory in a flow field with parallel stream lines, the trajectory error is solely a function of K/r once the error sources and initial particle position have been specified. The worst case considered in this investigation considered a value of K/r = 0.1 and V/V_s = 0.5. For this case the maximum rms trajectory error due to all effects is about 3.5 percent of the trajectory

*The technique used by the authors in Reference 16 was to first obtain a closed form analytic solution for Laplace's equation and then use this result in formulating the nonlinear equations governing particle motion which were subsequently solved on the analog computer.

**This is identical to R_e except that the velocity term in the equation defining Reynolds number is replaced by the free stream velocity U.

*This assumption has not been verified.

length with an "expected" error of about 1.2 percent. The error magnitude at a particular point on the trajectory cannot be computed directly since the effect at a specific point due to conducting surface irregularities is not known. If the conducting surface error has a mean value close to zero (which is often the case), then the expected error at any point for which K/r = 0.1 is about 1/2 to 1 percent of the trajectory length. Higher values of K/r would, of course, result in lower error values. In the experimental data obtained in simulating conditions of flow in a field distorted by a cylinder, the particle trajectory error in the vicinity of the cylinder's surface was assumed to be about 1 to 2 percent of the trajectory length as described previously.

Extrapolating the above remarks to the general case, it seems probable that for K/r values greater than 0.1, the expected trajectory errors should be less than 2 to 3 percent with errors of as much as 5 percent possible, but unlikely. It is obvious that if greater care is taken to reduce the error sources, the trajectory accuracy will improve. There is a limit to which this can be done and as this limit is approached the simulation becomes so sensitive that it loses its basic advantages and makes other means of solving the problem more attractive.

Future efforts to improve the accuracy of this method of analog simulation should be directed toward the development of a more homogeneous conducting surface than Teledeltos paper; and toward the construction of voltage sensing probes that are simple, reliable, aligned properly, and which accurately form the components of voltage gradient that are required. It would be desirable to determine what sort of tests should be performed on the conducting surface in the "as received" condition in order to estimate what its trajectory error effects will be once it is shaped to simulate the required potential field.

Aside from the mechanical design problems of probe construction, some effort might be directed towards determining an optimum probe configuration employing, perhaps, the use of more than four elements. The ideal or "optimum" probe element spacing is also an unknown and will depend on the surface noise characteristics and the problem simulated. It is possible that different probes should be used for different classes of problems.

Regarding the estimation and analysis of errors in particle trajectories, it might be desirable to formulate different "figures of merit" or error criteria than those advanced in this paper in order to get a more meaningful characterization of the error in a given problem.

Future attempts to increase the accuracy beyond the present state-of-the-art should consider some of the aforementioned problems.

Thus, the method of particle trajectory simulation described in this paper which employs an electrical conducting surface is worthwhile in all cases except those requiring extremely high accuracy. The simulation is conceptually simple and provides the investigator with a deep insight to the physical problem that would not ordinarily be obtained in as simple a fashion by other means. In cases where the accuracy is insufficient as determined perhaps by an error analysis, a more sophisticated method of solving the problem might be preferable; the use of a digital computer might be an alternate means in this case.

Appendix A

Derivation of Mathematical Model

This appendix will present a brief derivation of the mathematical model used in the simulation. The eight assumptions cited in Section II are assumed to hold throughout the derivation.

The motion of a particle in fluid flow is governed by the conventional equation for the drag force of a body in a fluid. Throughout the analysis, particles will be assumed to be spherical in shape. From the definition of drag coefficient C_D, the only force acting on the particle is

$$F = \frac{1}{2}\rho_g \pi a^2 C_D \left|\vec{w}'\right| \vec{w}' \qquad \text{(A-1)}$$

where a is the radius of the particle, ρ_g is the density of the gas, and w' is the relative velocity between the gas and particle, viz*

$$w' = \left\{\left[u'(x', y') - \dot{x}'\right]^2 + \left[v'(x', y') - \dot{y}'\right]^2\right\}^{1/2} \qquad \text{(A-2)}$$

where $u'(x', y')$ and $v'(x', y')$ represent the components of the gradient of some potential function $\phi(x', y')$ which satisfies Laplace's equation

$$\frac{\partial^2 \phi'(x', y')}{\partial x'^2} + \frac{\partial^2 \phi'(x', y')}{\partial y'^2} = 0 \qquad \text{(A-3)}$$

in every region of some irrotational, incompressible, and nonviscous potential flow field free of sources or sinks. The components of the gradient $\nabla\phi'$ may be written

$$\frac{\partial \phi'(x', y')}{\partial y'} = u'(x', y') \qquad \text{(A-4)}$$

$$-\frac{\partial \phi'(x', y')}{\partial x'} = v'(x', y') \qquad \text{(A-5)}$$

Hence (A-3) can be expressed as

$$-\frac{\partial v'(x', y')}{\partial x'} + \frac{\partial u'(x', y')}{\partial y'} = 0 \qquad \text{(A-6)}$$

*w', u', v', x', and y' represent the non-normalized components of the gas and particle velocities in the x' and y' directions, respectively.

If a Reynolds number R_e is defined as

$$R_e = \frac{2a\rho_g |\vec{w}'|}{\mu} = k|\vec{w}'| \qquad (A\text{-}7)$$

Equation (A-1) can be expressed as*

$$\vec{F} = \frac{\pi}{4} a\mu R_e C_D \vec{w}' \qquad (A\text{-}8)$$

Resolving this force into x and y components and setting the rate of change of momentum equal to this force, we obtain for the x and y directions respectively

$$\frac{4}{3}\pi a^3 \rho_p \ddot{x}' = \frac{\pi}{4} a\mu C_D R_e \left[u'(x', y') - \dot{x}'\right] \qquad (A\text{-}9)$$

$$\frac{4}{3}\pi a^3 \rho_p \ddot{y}' = \frac{\pi}{4} a\mu C_D R_e \left[v'(x', y') - \dot{y}'\right] \qquad (A\text{-}10)$$

where $4\pi a^3\rho_p/3$ is the mass of the particle and (x', y') are the position coordinates of the particle within the flow field. Equations (A-9) and (A-10) can easily be rendered to dimensionless form through some suitable normalization.

$$\ddot{x} = \frac{C_D R_e}{24K}\left[u(x, y) - \dot{x}\right] \qquad (A\text{-}11)$$

$$\ddot{y} = \frac{C_D R_e}{24K}\left[v(x, y) - \dot{y}\right] \qquad (A\text{-}12)$$

where

$$K = \frac{2\rho_p a^2 U}{9\mu L} \qquad (A\text{-}13)$$

U and L represent a characteristic velocity and characteristic length respectively. The problem time τ, the independent variable in Equations (A-9) and (A-10) is related to the dimensionalized independent variable t, in the normalized Equations (A-11) and (A-12) by the following equation

$$t = (\tau U)/L \qquad (A\text{-}14)$$

When the Reynolds number, R_e, is low (say less than unity) then the particle is said to be in the Stokes regime and in this case, we can write:

$$\frac{C_D R_e}{24} = 1 \qquad (A\text{-}15)$$

When outside the Stokes regime ($R_e > 1$), Equation (A-15) is not valid. We define a function $f(R_e)$:

$$\frac{C_D R_e}{24} = f(R_e) \qquad (A\text{-}16)$$

such that it is valid for the range of R_e of interest and determined by experimentally reported results.[16]

Thus we may conclude that there are two parameters which must be considered when scaling or analogies are employed: K, and R_{eo}.* Only one parameter, K, is important when the particle is in Stokes flow. Langmuir and Blodgett[14] have shown that K, often referred to as an inertia parameter, is the ratio of two lengths L and ℓ. L is some characteristic length in the flow field, say the diameter of a cylinder upon which the particles impinge (or flow around). ℓ is the distance a particle will travel in a quiescent medium having a viscosity μ if the particle is in Stokes flow with initial velocity U. K may be considered a parameter which is a measure of the degree to which a particle will respond to a signal (relative velocity of the gas to the particle) under a given set of conditions. Large values of K correspond to particles which will not alter their direction (due to their inertia) when the gas does (in order to flow around some object having a characteristic dimension, L). Conversely, particles having a low value of K associated with them will not deviate much from the gas streamlines of the flow field. Working with large values of K will not prove interesting since such particles will travel in essentially straight lines. Conversely, small values of K correspond to particles which behave like gas molecules. Thus, the interesting cases correspond to $K \approx 1$.

Appendix B

Details of Error Sources

The error sources which form the driving functions E_1 and E_2 for Equations (37) and (38) will now be considered. As mentioned in Section II-C, δV_y and δV_x will be made up of a sum of several terms.

1. Conducting Surface Misalignment

Consider Figure B-1 in which the conducting surface coordinate axes (x', y') are misaligned with respect to the X-Y plotter axes (x, y) by an angle α. For small α it follows that for any fixed point on the surface

$$x - x' = -y\alpha \qquad (B\text{-}1)$$

$$y - y' = +x\alpha \qquad (B\text{-}2)$$

$$V_x = V_{x'} - \alpha V_{y'} \qquad (B\text{-}3)$$

$$V_y = V_{y'} + \alpha V_{x'} \qquad (B\text{-}4)$$

*C_D is usually defined for an object which is not accelerating. In the case treated here, the particle is accelerating and we will assume that the same results are applicable.

*$R_{eo} = \frac{2a\rho_g U}{\mu}$

The computer positions the plotter to coordinate (x, y) where the surface voltage is V(x, y). If OA = OA' and OB = OB' in Figure B-1, then the errors introduced are

$$\delta_1 V_x = V_x - V'_{x'} \tag{B-5}$$

$$\delta_1 V_y = V_y - V'_{y'} \tag{B-6}$$

Expanding V' in a Taylor's series about its operating point for the deviations given by Equations (B-1) and (B-2) (for which case V = V'), taking derivatives with respect to x' and y' and combining the results with Equations (B-1) to (B-6) yields (neglecting high order effects)

$$\delta_1 V_x = \left[-V'_{y'} + V'_{x'y'}x - V'_{x'x'}y\right]\alpha \tag{B-7}$$

$$\delta_1 V_y = \left[V'_{x'} - V'_{x'y'}y + V'_{y'y'}x\right]\alpha \tag{B-8}$$

2. Probe Angular Misalignment

Consider Figure B-2. To approximate V_x and V_y, the computer forms

$$V_x = \frac{\psi_2 - \psi_1 + \psi_3 - \psi_4}{4\Delta x} \tag{B-9}$$

$$V_y = \frac{\psi_1 - \psi_4 + \psi_2 - \psi_3}{4\Delta y} \tag{B-10}$$

To compute the values of the ψ_i's, the surface voltage is expanded in a bivariate Taylor's series about the coordinates (x, y) and the deviations of the various probe elements from (x, y) is inserted in the resulting expression under the condition that there is a probe rotation β. For example, the normalized coordinates of the probe that measures ψ_1, are $[-\Delta x + \beta\Delta y, \Delta y + \beta\Delta x]$. Having obtained the ψ_i's, Equations (B-9) and (B-10) are used resulting in

$$\delta_2 V_x = -\beta V_y \tag{B-11}$$

$$\delta_2 V_y = \beta V_x \tag{B-12}$$

3. Unequal Probe Element Gains

The right hand sides of Equations (B-9) and (B-10) are only ideally formed. Each ψ_i in each equation is multiplied by a computer gain which is close to but not exactly equal to one. Thus it follows that

$$\delta_3 V_x = \frac{1}{4\Delta x}\left[(G_{2x} - 1)\psi_2 - (G_{1x} - 1)\psi_1 + (G_{3x} - 1)\psi_3 - (G_{4x} - 1)\psi_4\right] \tag{B-13}$$

$$\delta_3 V_y = \frac{1}{4\Delta y}\left[(G_{1y} - 1)\psi_1 - (G_{4y} - 1)\psi_4 + (G_{2y} - 1)\psi_2 - (G_{3y} - 1)\psi_3\right] \tag{B-14}$$

where G_{ix} and G_{iy} are the gains on the i^{th} probe in forming V_x and V_y. Once again a Taylor's Series can be used to determine the ψ_i's, and neglecting higher order terms the result is

$$\delta_3 V_x = \left(\frac{1}{4\Delta x}\right)\left[-G_{1x} + G_{2x} + G_{3x} - G_{4x}\right]V + \frac{1}{4}\left[-G_{1x} + G_{2x} - G_{3x} + G_{4x}\right]V_y \tag{B-15}$$

$$\delta_3 V_y = \left(\frac{1}{4\Delta y}\right)\left[G_{1y} + G_{2y} - G_{3y} - G_{4y}\right]V + \frac{1}{4}\left[-G_{1y} + G_{2y} - G_{3y} + G_{4y}\right]V_x \tag{B-16}$$

4. Finite Difference Approximation to Voltage Gradient Components

Here

$$\delta_4 V_x = \frac{\psi_2 - \psi_1 + \psi_3 - \psi_4}{4\Delta x} - V_x \tag{B-17}$$

$$\delta_4 V_y = \frac{\psi_1 - \psi_4 + \psi_2 - \psi_3}{4\Delta y} - V_y \tag{B-18}$$

Again a bivariate Taylor's Series is used, and neglecting terms higher in order than the cubics results in

$$\delta_4 V_x = -\frac{1}{3} V_{yyy}\Delta y^2 \tag{B-19}$$

$$\delta_4 V_y = -\frac{1}{3} V_{xxx}\Delta x^2 \tag{B-20}$$

where Δx and Δy are the normalized (with respect to L') distances defined in Figure (B-2). Equations (B-19) and (B-20) show a significant result in that no linear terms in Δx and Δy appear. Thus $\delta_4 V_x$ and $\delta_4 V_y$ contain no first order error effects and in most cases can be ignored. Only in the immediate vicinity of gas flow stagnation points will there be any significant error. This elimination of first order effects resulted from using a four element probe and averaging the voltage derivatives. It is not a characteristic of arbitrary probe designs.

5. Conducting Surface Irregularities

This error, denoted by $\delta_5 V_x$ and $\delta_5 V_y$, is unavoidable and does not readily lend itself to analysis in the general case of arbitrary conducting surface shapes. A simple example is

considered in Section III-C which predicts particle trajectory errors statistically given a statistical description of the surface voltage errors. This lends insight to the more general case.

6. Errors Caused by the Computer Loading the Conductive Surface

These errors are negligible in using an uncoated Teledeltos surface since its source resistance was experimentally determined to be about 3K ohms and an unloading circuit pictured in Figure B-3, into which each ψ_i is transmitted can be adjusted for an input impedance in excess of 1000M ohms.

Adding all the separate error sources the inputs to Equations (37) and (38) are

$$\delta V_x = \delta_1 V_x + \delta_2 V_x + \delta_3 V_x + \delta_4 V_x + \delta_5 V_x \quad \text{(B-21)}$$

$$\delta V_y = \delta_1 V_y + \delta_2 V_y + \delta_3 V_y + \delta_4 V_y + \delta_5 V_y \quad \text{(B-22)}$$

7. Errors Caused by Misaligned Servo Multiplier Potentiometers $\epsilon_{11}, \epsilon_{12}, \epsilon_{21}, \epsilon_{22}$

Consider the derivation of ϵ_{11}, the output error in the formulation of ω_x^2. In Figure B-4 θ_1 and θ_{11} represent the ratios of pot rotation above the center tap to the total rotation above this point. If pot 1 is misaligned by 100 ℓ_{11} percent full scale with respect to the feedback pot then

$$2\ell_{11} = \theta_{11} - \theta_1 \quad \text{(B-23)}$$

also

$$w_x = \theta_1 \quad \text{(B-24)}$$

$$w_x \theta_{11} = w_x^2 + \epsilon_{11} \quad \text{(B-25)}$$

Combining these equations yields

$$\epsilon_{11} = 2\ell_{11} w_x \quad \text{(B-26)}$$

Similarly it can be shown that

$$\epsilon_{12} = 2\ell_{12} \frac{s}{K} \quad \text{(B-27)}$$

$$\epsilon_{21} = 2\ell_{21} w_y \quad \text{(B-28)}$$

$$\epsilon_{22} = 2\ell_{22} \frac{s}{K} \quad \text{(B-29)}$$

8. Function Generator Error ϵ_f Is Already in the Desired Form and Is a Function of w^2

$$\epsilon_f = \epsilon_f(w^2) \quad \text{(B-30)}$$

The derivation of error sources for the general error Equations (37) and (38) is now complete. Equations (B-20), (B-22), and (B-26) to (B-30) form the driving forces in the determination of δx and δy.

Appendix C

Details of Error Sources for Example

This appendix presents a description of the individual errors comprising the terms in the "inputs" E_1 and E_2 to the error model.

Approximation of V_x and V_y by finite differences of surface voltages. This error is nonexistant since the third order partials of V are zero for parallel stream lines.

Conducting surface misalignment. In this case the terms $\delta_1 V_x$ and $\delta_1 V_y$ reduce to

$$\delta_1 V_x = -V'_{y'}, \alpha = -\alpha \quad \text{(C-1)}$$

$$\delta_1 V_y = V'_{x'}, \alpha = V_x \alpha = 0 \quad \text{(C-2)}$$

Equation (57) reduces to $-\alpha$ since $V'_{y'}$ is normalized to equal unity in magnitude. It will be assumed that the surface can be aligned on the X-Y plotter so that there is less than 1/16 inch offset at the end of a 20-inch surface edge. This corresponds to $|\alpha| \leq 0.18$ degree.

Probe angular misalignment. Here the general equations reduce to

$$\delta_2 V_x = -\beta V_y = -\beta \quad \text{(C-3)}$$

$$\delta_2 V_y = \beta V_x = 0 \quad \text{(C-4)}$$

It is assumed that $|\beta| \leq \pi/360^\circ = 0.5^\circ$.

Errors caused by unequal multiplication of probe element voltages. Here the general equations reduce to

$$\delta_3 V_x = \left(\frac{-G_{1x} + G_{2x} + G_{3x} - G_{4x}}{4}\right)\left(\frac{Y}{\Delta x}\right)\left(\frac{V}{V_s}\right) + \left(\frac{-G_{1x} + G_{2x} - G_{3x} + G_{4x}}{4}\right) V_y$$

$$= \left(\frac{-G_{1x} + G_{2x} + G_{3x} - G_{4x}}{4}\right)\left(\frac{Y}{\Delta x}\right)\left(\frac{V}{V_s}\right) \quad \text{(C-5)}$$

since the gains can usually be controlled so that the second term is negligible with respect to other system errors, and

$$\delta_3 V_y = \left(\frac{+G_{1y} + G_{2y} - G_{3y} - G_{4y}}{4}\right)\left(\frac{Y}{\Delta y}\right)\left(\frac{V}{V_s}\right) \quad \text{(C-6)}$$

For subsequent convenience Y/V_s (the parameters being defined in Figure 4) has been introduced into these equations. This is valid since $Y/V_s = 1/V_y' = 1$ due to the normalization of V_y' referred to previously. These probe element gains are best adjusted by using an amplifier as a comparator. It is then the difference between gains that is controlled rather than the values of the gains themselves. Experience indicates that the difference between two such gains can be held well below 0.01 percent. Therefore

$$\left|G_{1i} - G_{3i}\right| \leq 0.0001, \quad \left|G_{2i} - G_{4i}\right| \leq 0.0001$$

for i = x or y.

Conducting Surface Irregularities. These errors, denoted by $\delta_5 V_x$ and $\delta_5 V_y$, are some random function of the probe coordinates (x, y) and thus do not remain fixed along a given trajectory like all the other errors considered in this analysis. As mentioned in Section III-C, further detailed discussion on the statistical description of these errors may be found in Reference 23.

Servo Potentiometer Misalignment Errors. From Appendix B, it follows that

$$\epsilon_{12} = 2\ell_{12}\frac{s}{K} \qquad (C\text{-}7)$$

$$\epsilon_{22} = 2\ell_{22}\frac{s}{K} \qquad (C\text{-}8)$$

The servos are assumed sufficiently accurate so that no potentiometer is misaligned more than 0.1 percent of full scale, therefore:

$$\left|\ell_{12}\right| \leq 0.001, \quad \left|\ell_{22}\right| \leq 0.001$$

Summing all the aforementioned errors yields the following expression for E_1 and E_2:

$$E_1 = \left(G_{1y} - G_{3y}\right)\left(\frac{Y}{4\Delta y}\right)\left(\frac{V}{V_s}\right) + \left(G_{2y} - G_{4y}\right)\left(\frac{Y}{4\Delta y}\right)\left(\frac{V}{V_s}\right) + 2\ell_{12} + \delta_5 V_y \qquad (C\text{-}9)$$

$$E_2 = \alpha + \beta + \left(-G_{1x} + G_{3x}\right)\left(\frac{Y}{4\Delta x}\right)\left(\frac{V}{V_s}\right) + \left(G_{2x} - G_{4x}\right)\left(\frac{Y}{4\Delta x}\right)\left(\frac{V}{V_s}\right) + 2\ell_{22} \qquad (C\text{-}10)$$

Bibliography

1. Hoglund, R. F., Recent Advances in Gas-Particle Nozzle Flows, ARS Paper No. 2331-62, presented at the Solid Propellant Rocket Conference; Waco, Texas, January 1962, 43 pp.
2. Malavard, L. C., "The Use of Rheoelectric Analogies in Aerodynamics," AGARDograph 18, August 1956.
3. Logan, B. F., G. R. Welti, and G. S. Sponsler, Analog Study of Electronic Trajectories, J. Assoc. Comp. Mach., vol 2, pp. 28-41, 1955.
4. Karplus, W. J., Analog Simulation: Solution of Field Problems, McGraw Hill Publishing Co., New York, New York, 1958.
5. Einstein, P. A., Factors Limiting the Accuracy of Electrolytic Plotting Tanks, Brit. J. Appl. Phys., vol. 2, pp. 49-55, 1951.
6. Sander, K. F. and J. G. Yates, A New Form of Electrolytic Tank, Proc. Inst. Elec. Engrs., vol. 104, pt. C, pp. 81-86, 1957.
7. Boothroyd, A. R., E. C. Cherry, and R. Makar, An Electrolytic Tank for the Measurement of Steady State Response, Transient Response and Allied Properties of Networks, Proc. Inst. Elec. Engrs., vol. 96, pt. I, pp. 163-177, 1949.
8. Softky, S., and J. Jungerman, Electrolytic Tank Measurements in Three Dimensions, Rev. Sci. Instr., vol. 23, pp. 306-307, 1952.
9. Kallis, C. J., N. H. Freed, and C. L. Suzman, Conducting Sheet Analogues for the Solution of Field Problems, Report 1/60, Department of Mechanical Engineering, University of Witwatersrand, Johannesburg, South Africa, October 1960.
10. Norum, V. D., Conductive Solid Analogs in the Determination of Particle Trajectories in Potential Flow Fields, Unpublished term paper for Course No. ENGR 213 B, University of California, Los Angeles, 22 pp., January 1961.
11. Hicks, H., and A. Permoda, Uniformly Conductive Surfaces, Icing Research Staff, University of Michigan, Engineering Research Institute, Ann Arbor, Mich., 1953.
12. Winiarski, F. J., An Electronic Analog Computer Input Device for Functions of Two Variables, M. S. thesis, University of California, Los Angeles, 1955.
13. Martin, R. J., N. A. Masnari, and J. E. Rowe, Analog Representation of Poisson's Equation in Two Dimensions, IRE Trans. on Electronic Computers, vol. EC-9, No. 4, pp. 490-496, December 1960.
14. Larrowe, V. and M. Spencer, Use of Semi-Conducting Surfaces in Analog Function Generation, presented at the National Simulation Conference; Dallas, Texas, October 1958, 6 pp.

15. Langmuir, I., and K. B. Blodgett, A Mathematical Investigation of Water Droplet Trajectories, AAF TN 5418, ATI No. 25223, 1946.

16. Brun, R. J., W. Lewis, P. J. Perkins, and J. S. Serafini, Impingement of Cloud Droplets on a Cylinder and Procedure for Measuring Liquid-Water Content and Droplet Sizes in Supercooled Clouds by Rotating Multicylinder Method, NACA Report No. 1215, Lewis Flight Propulsion Lab., Cleveland, Ohio, 1955.

17. Rowe, J. E., and R. J. Martin, An Electron-Trajectory Calculator and its Component Poisson Cell, Proc. IEE, vol. 105, pt. B (Supplement), pp. 1024-1032, 1958.

18. Hollway, D. L., The Determination of Electron Trajectories in the Presence of Space Charge, Australian J. Phys., vol. 8, pp. 74-89, 1955.

19. Brewer, G. R. and T. VanDuzer, Space-Charge Simulation in an Electrolytic Tank, J. Appl. Phys., Vol. 30, pp. 291-301, March 1959.

20. Kliegel, J. R., One Dimensional Flow of a Gas Particle System, Journal of Aero/Space Sciences, March 1960.

21. Adelberg, M. and S. Lafazan, Analog Simulation of Two Dimensional Uncoupled Particle Trajectories in a Potential Flow Field, Unpublished STL Internal Document (Unclassified) No. 7620.3-2, January 1960.

22. Norum, V. D., Analog Simulation of Particle Motion in a Potential Flow Field, Unpublished STL Internal Document (Unclassified) No. STL/TM-60-0000-09030, April 1960.

23. Farrenkopf, R. L., Simulation Error Analysis of Particle Trajectories in a Potential Flow Field, Unpublished STL Internal Document (Unclassified) No. 9313.8-99, August 1961.

24. Beers, Y., Introduction to the Theory of Error, Addison-Wesley Publishing Co., Reading, Mass., 1957.

25. Davenport, W. B. and W. L. Root, An Introduction to the Theory of Random Signals and Noise, McGraw Hill Book Co., New York, New York, 1958.

26. Malavard, L., On a New Technique for Experimental Investigations by Means of Rheoelectric Analogs, Recherche Aeronaut, vol. 20, pp. 61-68, 1951.

27. Kriebel, A. R., Particle Trajectories in a Gas Centrifuge, Transactions of the ASME Journal of Basic Engineering, Paper No. 60-WA-158.

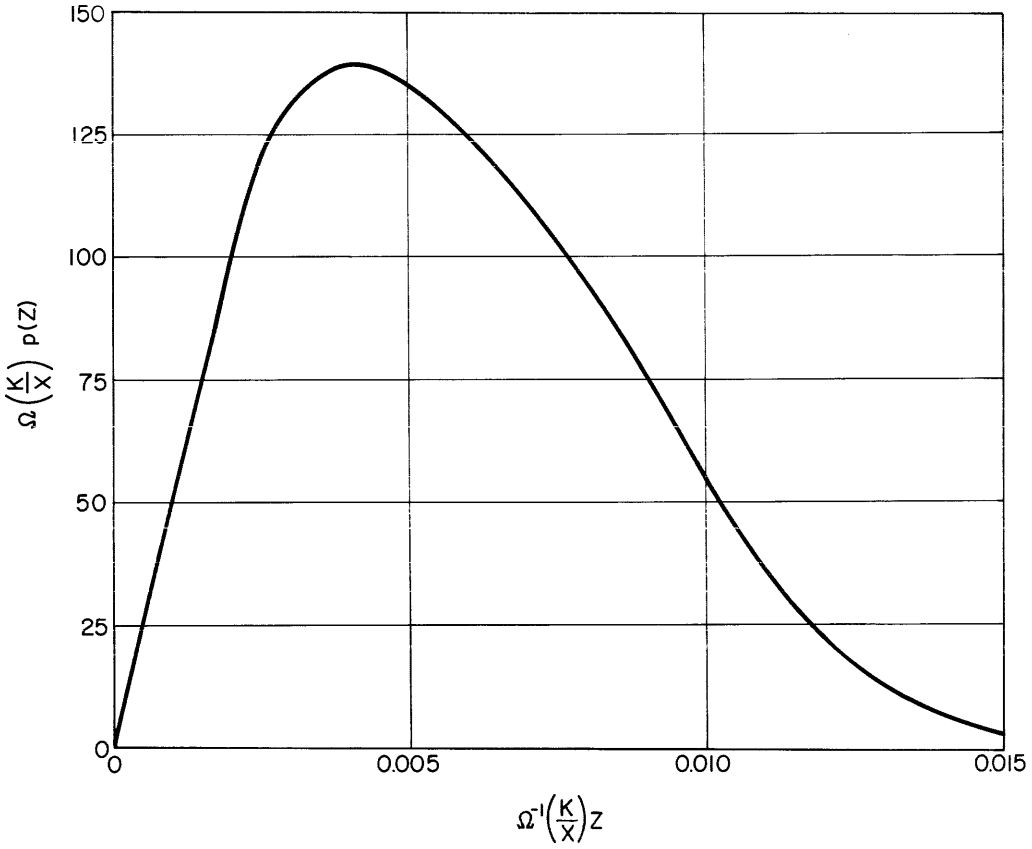


Figure 6. Probability Density Function of Trajectory Error

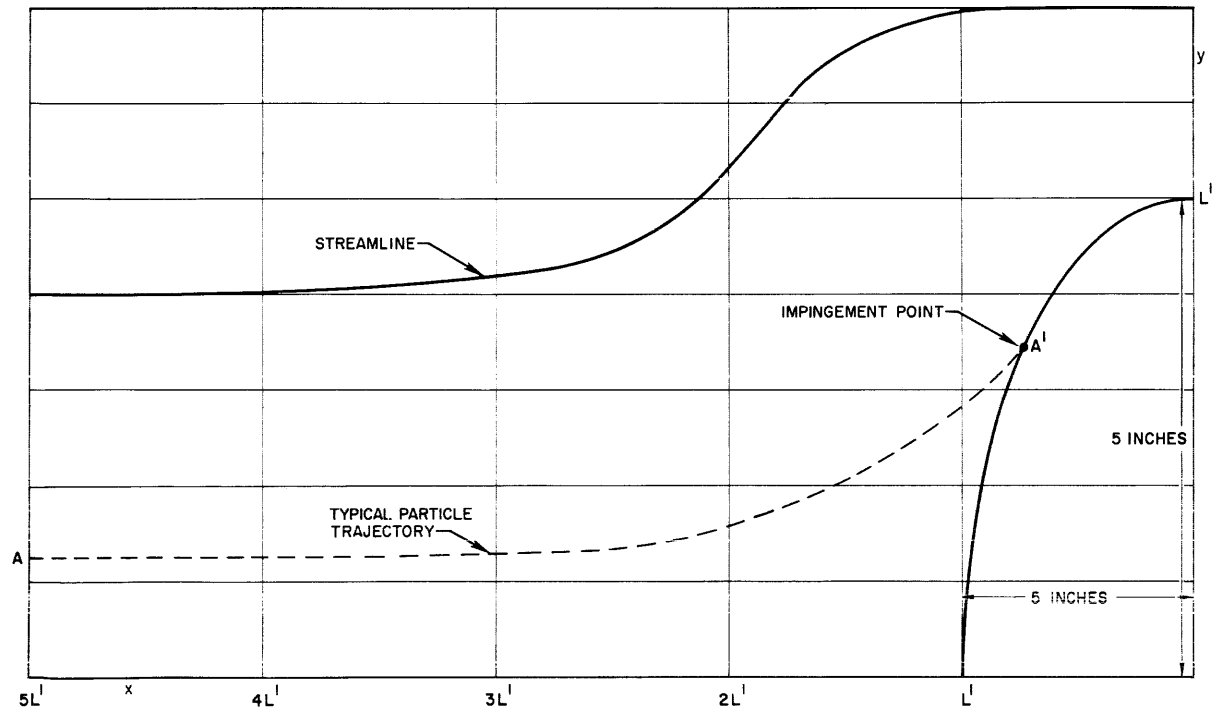


Figure 7. Quarter Section of Infinite Flow Field Distorted by Cylinder

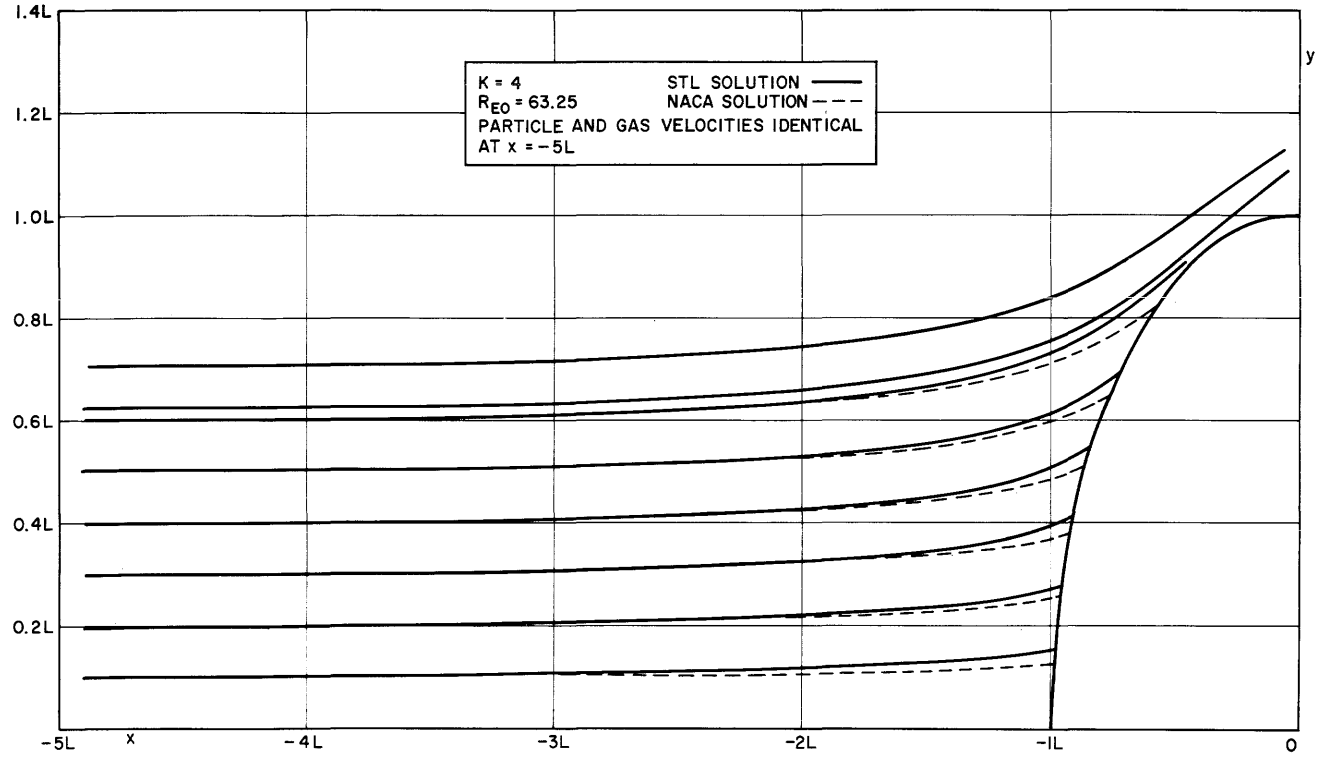


Figure 8. Particle Trajectories in the Vicinity of a Cylinder of Radius L

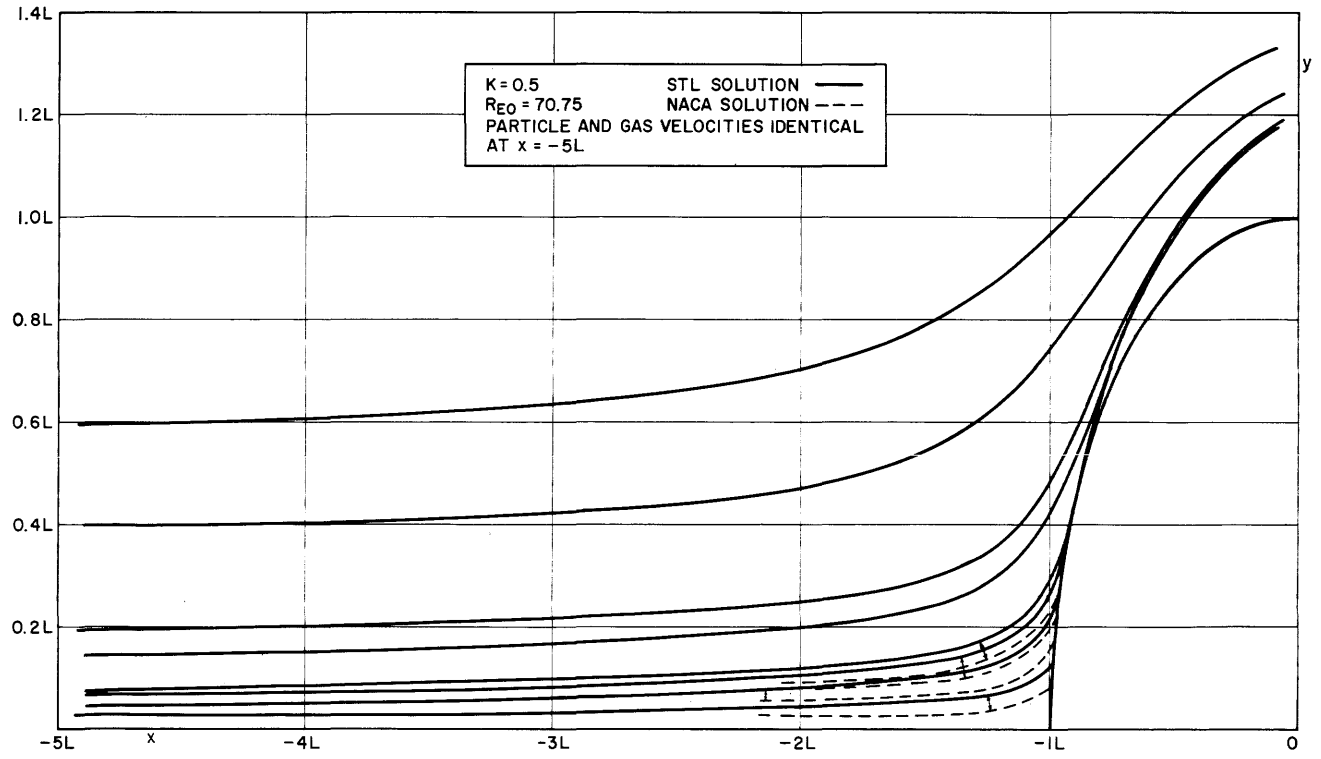


Figure 9. Particle Trajectories in the Vicinity of a Cylinder of Radius L

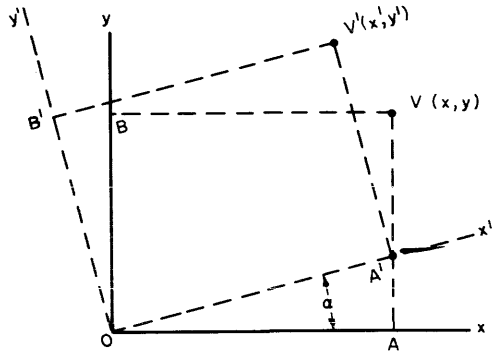


Figure B-1. Conducting Surface Misalignment

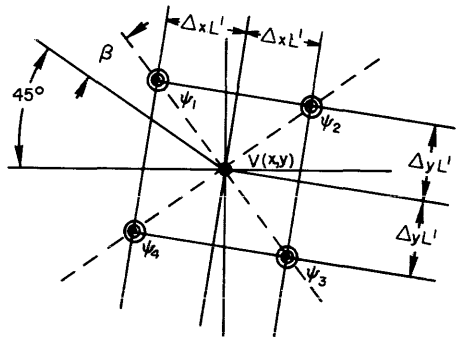


Figure B-2. Angular Misalignment of Probe

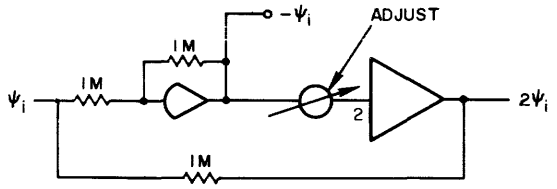


Figure B-3. Unloading Circuit

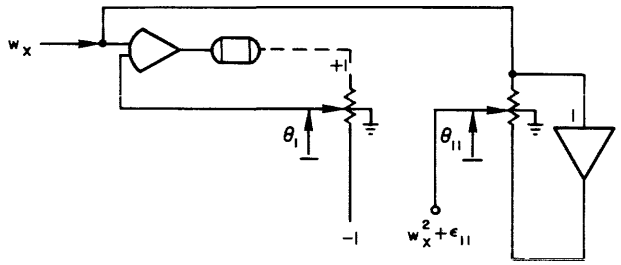


Figure B-4. Servo Multiplier

THE APPLICATION OF FINITE FOURIER TRANSFORMS TO ANALOG COMPUTER SIMULATIONS

Eric Liban

Grumman Aircraft Engineering Corp.

Bethpage, New York

Summary

An Analog Computer technique for the solution of certain classes of boundary value problems of partial differential equation based on Finite Fourier Transforms is presented, which requires considerably less computer components than conventional finite difference methods. The derivation of the Finite Fourier Transform method is briefly stated and then applied to analog computer simulations of heat transfer equations with linear and nonlinear boundary conditions.

Introduction

A well known method for the solution of ordinary linear differential equations with constant coefficients is the Laplace transform method which reduces the differential equation to an algebraic equation. The solution of the latter is a function of a parameter "s" and given initial conditions, and its inverse Laplace transform is the solution of the differential equation. This method is easily extended to partial differential equations in two independent variables. A Laplace transform with respect to one of the independent variables gives an ordinary differential equation, whose solution is the Laplace transform of the function satisfying the partial differential equation and its initial and boundary conditions. The inapplicability of this method for analog computations arises from the fact that the inverse Laplace transform, a necessary step to extract the solution, is an integration over an infinite path in the complex domain. To solve partial differential equationson an Analog Computer by a transform method the inverse transform must be obtainable by operations in the real domain only. The particular transform discussed in this paper is the Finite Fourier Transform, which is applicable to equations in which only the even order derivatives (of the function) with respect to the transformed variable appear. The heat equation, wave equation and bending beam equation are of such nature. Many other transforms exist which may be used for other types of equations. (Refs. 1, 2, and 3).

A feature, which makes the Finite Transform a very economical method for analog computers, is that the inverse transform may be solved only for regions of interest. The truncation error in the analog simulation may be made smaller than the error in the amplifiers themselves, and therefore the solution obtained by the Finite Fourier Transform method is the exact solution within the accuracy of the Analog Computer; and the bounds on the deviation from the true solution can be stated precisely in terms of the usually small amplifier noise and integrator drifts.

Finite Fourier Transforms

The transform exists for all bounded, piecewise continuous functions over a finite interval. The extension of a continuous function $F(x)$ defined for $0 \leq x \leq \pi$, into an odd periodic function $\tilde{F}(x)$ with period 2π may be expressed by the Fourier series

$$\tilde{F}(x) = \frac{2}{\pi} \sum_{n=1}^{\infty} f_s(n) \sin nx \ . \qquad (1)$$

The coefficients are given by

$$f_s(n) = \int_0^{\pi} F(x) \sin nx \, dx \ , \qquad (2)$$

$$n=1, 2, \ldots \ .$$

This set of coefficients, defined by Eq. (2), is the Finite Fourier Sine Transform of $F(x)$, and the inverse Sine Transform is $\bar{F}(x)$ as defined by Eq. (1). Note that $\bar{F}(x) = F(x)$ for $0 < x < \pi$. But $\bar{F}(0)$ and $\bar{F}(\pi)$ will always equal zero, whatever $F(0)$ and $F(\pi)$ may be. This follows from the extension of $F(x)$ into an odd function and the property of Fourier series to converge at finite discontinuities to the average value of the limits the function approaches from the right and left respectively.

The following expression is obtained by two successive integrations by parts and a substitution using Eq. (2):

$$\int_0^{\pi} \frac{d^2F(x)}{dx^2} \sin nx \, dx = -n^2 f_s(n) + n[F(0) - (-1)^n F(\pi)] \, . \tag{3}$$

Expanding $F(x)$ into an even function $\tilde{\bar{F}}(x)$ of period 2π leads to the Fourier series

$$\tilde{\bar{F}}(x) = \frac{1}{\pi} f_c(0) + \frac{2}{\pi} \sum_{n=1}^{\infty} f_c(n) \cos nx \tag{4}$$

where the set of coefficients $f_c(n)$ is the Finite Fourier Cosine Transform (FFCT) given by

$$f_c(n) = \int_0^{\pi} F(x) \cos nx \, dx \, , \qquad n=0, 1, 2, \ldots \, . \tag{5}$$

The inverse FFCT is the function $\tilde{\bar{F}}(x)$ as given by Eq. (4). Here $\tilde{\bar{F}}(x) = F(x)$ for the closed interval $0 \leq x \leq \pi$; while $d\tilde{\bar{F}}/dx = dF/dx$ holds generally only for $0 < x < \pi$, since the derivative of $\tilde{\bar{F}}$ may be discontinuous at $x = n\pi$. The following relations are obtained for the FFCT:

$$\int_0^{\pi} \frac{d^2F}{dx^2} \cos nx \, dx = -n^2 f_c(n) - F'(0) + (-1)^n F'(\pi) \tag{6}$$

by integrating the left side by parts twice and substituting Eq. (5), $F'(0)$ and $F'(\pi)$ are the values of the derivative of $F(x)$ at $x = 0$ and π, respectively.

Two other useful Finite Fourier Transforms are stated below:

Finite λ-Transform

$$f_\lambda = \int_0^{\pi} F(x) \sin \frac{2n-1}{2} x \, dx \, , \qquad n=1, 2, \ldots \, . \tag{7}$$

Inverse λ-Transform

$$\bar{F}(x) = \frac{2}{\pi} \sum_{n=1}^{\infty} f_\lambda(n) \sin \frac{2n-1}{2} x \tag{8}$$

and

$$\int_0^{\pi} \frac{d^2F}{dx^2} \sin \frac{2n-1}{2} x \, dx = -\left(\frac{2n-1}{2}\right)^2 f_\lambda(n) + \left(\frac{2n-1}{2}\right) F(0) - (-1)^n F'(\pi) \, . \tag{9}$$

Finite μ-Transform

$$f_\mu(n) = \int_0^{\pi} F(x) \cos \frac{2n-1}{2} x \, dx \, , \qquad n=1, 2, \ldots \, . \tag{10}$$

Inverse μ-Transform

$$\bar{\bar{F}}(x) = \frac{2}{\pi} \sum_{n=1}^{\infty} f_{\mu}(n) \cos \frac{2n-1}{2} x \qquad (11)$$

and

$$\int_0^{\pi} \frac{d^2F}{dx^2} \cos \frac{2n-1}{2} x \, dx = - \left(\frac{2n-1}{2}\right)^2 f_{\mu}(n) - F'(0) + (-1)^n F(\pi) \; . \qquad (12)$$

Only the relations expressing the transforms of the second derivative of a function, in terms of the transform of the function and its boundary values, were shown. However, relations may be obtained for all even order derivatives. For an elementary discussion of the Finite Fourier Transform see Churchill, Modern Operational Mathematics in Engineering.[4] For a more advanced treatise on the Finite Fourier Transform and other Integral Transforms see bibliography.

Analog Simulations of the Heat Equation

The Re-entry Problem

The Finite Fourier Transform method is applied first to the one-dimensional heat equation with boundary conditions encountered in the "re-entry problem". Let ξ be the space coordinate of a one-dimensional slab of length L, which is the idealization of a longitudinal section of the nose cone of a missile, τ the time and $T(\tau, \xi)$ the temperature. At $\xi = 0$ a heat input or heat flux is given, while at $\xi = L$ the slab is assumed to be insulated, that is, the temperature gradient is zero at $\xi = L$. In this problem the main interest is in the temperature of the face $\xi = 0$, to determine whether it rises above or remains below the melting point. Also of concern is the temperature at $\xi = L$, which must remain below the melting temperature of the back-up structure. The mathematical statement of this problem for homogeneous slabs with temperature independent thermal constants and uniform initial temperature T_0 is given by

$$\frac{\partial T}{\partial \tau} = \alpha \frac{\partial^2 T}{\partial \xi^2} \; , \quad 0 \le \xi \le L \; , \quad \tau > 0 \; . \qquad (13)$$

Initial Condition:

$$T(0, \xi) = T_0 \; . \qquad (14)$$

Boundary Conditions:

$$k \frac{\partial T}{\partial \xi} = - q(\tau) \quad \text{at} \quad \xi = 0$$

$$\frac{\partial T}{\partial \xi} = 0 \quad \text{at} \quad \xi = L \; . \qquad (15)$$

The symbols used in the above equations are defined in the following table:

$T \equiv$ temperature - °R

$\tau \equiv$ time - sec

$\xi \equiv$ space coordinate - in

$\alpha \equiv$ thermal diffusivity - in^2/sec

$k \equiv$ thermal conductivity - BTU/sec-in-°R

$q \equiv$ heat input rate per unit area - $BTU/sec\text{-}in^2$

The linear heat equation may be solved for the temperature deviation from the initial temperature, expressed by

$$u(\tau, \xi) = T(\tau, \xi) - T_0 \; . \qquad (16)$$

Equation (16) and the substitutions

$$x = \frac{\pi}{L} \xi \; , \qquad t = \alpha \frac{\pi^2}{L^2} \tau \; , \qquad (17)$$

transform Eqs. (13), (14), and (15) to the equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \; , \quad 0 \le x \le \pi \; , \quad t > 0 \; . \qquad (18)$$

Initial Condition:

$$u(0, x) = 0 \ . \tag{19}$$

Boundary Conditions:

$$\frac{\partial u}{\partial x} = - \frac{L}{\pi k} q(t) \equiv - Q(t) \quad \text{at} \quad x = 0$$
$$\frac{\partial u}{\partial x} = 0 \quad \text{at} \quad x = \pi \ . \tag{20}$$

Equation (18) and boundary conditions as shown in Eq. (20) indicate the use of the Finite Fourier Cosine Transform with respect to x. The FFCT of u(t, x) is given by

$$f_c(t, n) = \int_0^{\pi} u(t, x) \cos nx \, dx \ , \qquad n=0, 1, \ldots \ . \tag{21}$$

It is easily seen that

$$\int_0^{\pi} \frac{\partial u}{\partial t} \cos nx \, dx = \frac{d}{dt} f_c(t, n) \equiv \dot{f}_c(t, n) \ , \qquad n=0, 1, \ldots \ . \tag{22}$$

By multiplying both sides of Eq. (18) with cos nx dx and integrating from 0 to π using the notation defined by Eqs. (21) and (22), the differential equations for the terms of the FFCT are formed. Equation (6) with the given Boundary Conditions as shown in Eq. (20) gives:

$$\dot{f}_c(t, n) = - n^2 f_c(t, n) + Q(t) \ , \qquad n=0, 1, 2, \ldots \tag{23}$$

with initial conditions:

$$f_c(0, n) = 0 \ , \qquad n=0, 1, 2, \ldots \ . \tag{24}$$

The initial condition of $f_c(t, n)$ equals the FFCT of the initial condition of u(t, x), which for the case under discussion, equals zero. If the problem were solved for the temperature T with initial conditions $T(0, x) = T_0$, then the initial conditions for Eqs. (23) would still all be zero with the exception of $f_c(t, 0)$, since the integral of a constant times cos nx between 0 and π always vanishes when $n \neq 0$. When the initial temperature distribution is a function of x, then the initial conditions for $f_c(t, n)$ may be either readily computable by hand or, easily evaluated on the analog computer. The set of solutions $f_c(t, n)$ of Eqs. (23) form the FFCT of u(t, x); i.e., they are the Fourier coefficients as a function of time t of the temperature u(t, x). The inverse FFCT, as given by Eq. (4), for x = 0 and $x = \pi$ respectively, becomes:

$$u(t, 0) = \frac{1}{\pi} f_c(t, 0) + \frac{2}{\pi} \sum_{n=1}^{\infty} f_c(t, n) \tag{25}$$

$$u(t, \pi) = \frac{1}{\pi} f_c(t, 0) + \frac{2}{\pi} \sum_{n=1}^{\infty} (-1)^n f_c(t, n) \ . \tag{26}$$

Clearly, it is not possible to solve Eqs. (23) for all n from 0 to ∞. However, the inverse Finite Fourier Cosine Transform is a fairly rapidly convergent series, because it is a continuous function for all x. The coefficients behave like $1/n^2$ for increasing n. When $Q(t) \geq 0$ for all t, it can easily be seen from Eqs. (23) that $f_c(t, n) \geq 0$ for all n and t. Hence, there exists an N for any ϵ, no matter how small, such that

$$\max \left| u(t, 0) - \frac{1}{\pi} f_c(t, 0) - \frac{2}{\pi} \sum_{n=1}^{N} f_c(t, n) \right| < \epsilon \ . \tag{27}$$

That is, there exists an N such that the truncation error

$$E_0 = \max \frac{2}{\pi} \sum_{n=N+1}^{\infty} f_c(t, n) \qquad (28)$$

will be smaller than the accuracy of the computer or some other desired accuracy compatible with analog computer characteristics. The maximum absolute errors in the computer solutions of

$$u(t, x) = \frac{1}{\pi} f_c(t, 0) + \frac{2}{\pi} \sum_{n=1}^{N} f_c(t, n) \cos nx \qquad (29)$$

for $x \neq 0$, will all be smaller than E_0, since for positive $f_c(t, n)$

$$\max \frac{2}{\pi} \sum_{n=N+1}^{\infty} f_c(t, n) \cos nx \leq \max \frac{2}{\pi} \sum_{n=N+1}^{\infty} f_c(t, n) \; . \qquad (30)$$

The number of Fourier coefficients to be generated on the analog computer may therefore be obtained by an a priori estimate or by increasing N until $f_c(t, N) \approx 0$ for all t. Figure 1 shows the analog computer setup for the Finite Fourier method solution of Eqs. (18), (19), and (20) with $N = 7$. For test purposes the heat input $Q(t)$ was a steep saw tooth and it was found that the temperature at $x = 0$ for all time t differed always by less than 1% from the analytically obtained exact solution. The slab assumes a uniform steady state temperature when the heat input is of finite duration. The Fourier Transform method of simulation will always give the correct steady state temperature regardless of the number of Fourier coefficients used. This can easily be seen from the fact that the output voltages of all the integrators will eventually be zero, when $Q(t) \equiv 0$, except the output of the integrator generating $f_c(t, 0)$. When the slab is at uniform constant temperature, then the Fourier series of the temperature distribution is simply the constant $(1/\pi) f_c(t, 0)$.

Convective Boundary Condition

The boundary with convective heat transfer may be either $\xi = 0$ or $\xi = L$, or both. In addition a heat input $q(\tau)$ may be given. In this section the analog simulation of the heat equation is shown for the following boundary conditions:

$$k \frac{\partial T}{\partial \xi} = - q(\tau) \qquad \text{at} \quad \xi = 0 \qquad (31)$$

$$k \frac{\partial T}{\partial \xi} = h(T(\tau, L) - T_i) \qquad \text{at} \quad \xi = L \; , \qquad (32)$$

where

$h \equiv$ convective heat transfer coefficient - BTU/in^{2}sec °F.

The boundary conditions, stated in words, mean that the bar is heated at the rate $q(\tau)$ at the face $\xi = 0$ while the face $\xi = L$ is in contact with a substance at temperature T_i, which may be a function of time, and heat is transferred from the bar to this substance according to Eq. (32). Let the bar be initially at a uniform constant temperature $T(0, \xi) = T_0$. With the substitutions Eqs. (16) and (17), $u_i = T_i - T_0$, and $(L/\pi k) q(t) = Q(t)$ the problem is stated:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \; , \quad 0 \leq x \leq \pi \; , \quad t > 0 \; . \qquad (33)$$

Initial Condition:

$$u(0, x) = 0 \; . \qquad (34)$$

Boundary Conditions:

$$\frac{\partial u}{\partial x} = - Q(t) \qquad \text{at} \qquad x = 0 \qquad (35)$$

$$\frac{\partial u}{\partial x} = \frac{L}{\pi} h\left[u(t, \pi) - u_i\right] \quad \text{at} \quad x = \pi . \quad (36)$$

Since the boundary conditions prescribe the derivatives at $x = 0$ and $x = \pi$, the use of the Finite Fourier Cosine Transform is indicated; and from Eq. (6) and the above boundary conditions is obtained the set of differential equations

$$\dot{f}_c(t, n) = - n^2 f_c(t, n) + Q(t)$$

$$+ (-1)^n \frac{L}{\pi} h\left[u(t, \pi) - u_i\right] \quad (37)$$

$$n=0, 1, 2, \ldots .$$

The essential difference between Eqs. (37) and the equations for the Fourier Transform of the heat equation with insulated boundary Eqs. (23) is that now the boundary condition is a function of the temperature at $x = \pi$. But $u(t, \pi)$ is given by Eq. (26), as in the previous example and may be generated continuously. Hence the last term of Eq. (37) is easily formed and fed back to the appropriate integrators in the simulation. The analog circuit for the heat equation with a heat input on one face and a convective heat transfer at the other face is shown in Fig. 2. The FFCT was solved for $n = 0$ to $n = 7$.

Radiation Heat Transfer[5]

Examples chosen for this type of boundary conditions are a flat plate and a sphere radiating in a vacuum. The boundary condition for radiation heat transfer at the face $\xi = L$ is given by

$$k \frac{\partial T}{\partial \xi} = - \sigma\epsilon(T^4(\tau, L) - T_a^4) \quad (38)$$

where

$\sigma \equiv$ Stefan-Boltzmann's constant - BTU/sec in^2 °R^4

$\epsilon \equiv$ emissivity - dimensionless

$T_a \equiv$ temperature to which surface radiates - °R.

The Flat Plate Radiating into Vacuum

The term "flat plate" is taken here to mean a solid slab bounded by a pair of parallel planes $\xi = - L$ and $\xi = L$. The initial temperature T_0 is uniform and Stefan-Boltzmann radiation into vacuum takes place at the two faces $\xi = \pm L$. Clearly, the temperature distribution will be symmetrical about $\xi = 0$ and therefore it is more convenient to state the problem as follows:

$$\frac{\partial T}{\partial \tau} = \alpha \frac{\partial^2 T}{\partial \xi^2} \quad 0 \leq \xi \leq L , \quad \tau > 0 . \quad (39)$$

Initial Condition:

$$T(0, \xi) = T_0 \quad (40)$$

Boundary Conditions:

$$\frac{\partial T}{\partial \xi} = 0 \quad \text{at} \quad \xi = 0 \quad (41)$$

$$k \frac{\partial T}{\partial \xi} = - \sigma\epsilon[T(\tau, L)]^4 \quad \text{at} \quad \xi = L \quad (42)$$

Equation (41) follows from the symmetry condition, and Eq. (42) is obtained from Eq. (38) by letting $T_a = 0$ for radiation into vacuum. Since Eq. (42) is nonlinear it can no longer be solved for the temperature deviation from the initial temperature T_0. It is probably just as simple to simulate the original equations, and certainly the results would be more easily interpreted; however, for simplicity of presentation let

$$x = \frac{\pi}{L} \xi \quad \text{and} \quad t = \alpha \frac{\pi^2}{L^2} \tau \quad (43)$$

and set

$$g = \frac{L}{\pi} \frac{\epsilon\sigma u_0^3}{k} \quad (44)$$

and

$$u = \frac{T}{T_0} . \quad (45)$$

The problem may now be stated

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \quad 0 \leq x \leq \pi , \quad t > 0 . \tag{46}$$

Initial Condition:

$$u(0, x) = 1 . \tag{47}$$

Boundary Conditions:

$$\frac{\partial u}{\partial x} = 0 \qquad \text{at} \quad x = 0 \tag{48}$$

$$\frac{\partial u}{\partial x} = - g u^4(t, \pi) \qquad \text{at} \quad x = \pi . \tag{49}$$

The differential equations for the FFCT are derived in the same way as in the previous examples and are given by

$$\dot{f}_c(t, n) = - n^2 f_c(t, n) - (-1)^n g u^4(t, \pi) \tag{50}$$

$$n=0, 1, 2, \ldots .$$

The initial conditions of $f_c(t, n)$ are the FFCT's of $u(0, x)$, which for $u(0, x) = 1$ equals π when $n = 0$, and equals zero for all $n \neq 0$. The last term in Eq. (50) is generated continuously from

$$u(t, \pi) = \frac{1}{\pi} f_c(t, 0) + \frac{2}{\pi} \sum_{n=1}^{N} (-1)^n f_c(t, n) \tag{51}$$

raised to the 4^{th} power, which in the simulation indicated in Fig. 3 was performed by two successive servo-multiplications. The simulation proved to be stable and in agreement with the solution obtained by Abarbanel.[5] If boundary Eq. (42) were replaced by Eq. (38), the simulation would differ only slightly.

Sphere Radiating Into Vacuum

The heat equation for a sphere with temperature distribution a function of time τ and distance ρ from the center of the sphere, expressed in polar coordinates is given by

$$\frac{\partial T}{\partial \tau} = \frac{\alpha}{\rho^2} \frac{\partial}{\partial \rho} \left(\rho^2 \frac{\partial T}{\partial \rho} \right) . \tag{52}$$

By the change of variables

$$r = \frac{\pi}{R} \rho , \quad t = \alpha \frac{\pi^2}{R^2} \tau , \quad u = r \frac{T}{T_0} \tag{53}$$

where R is the radius of the sphere, Eq. (52) becomes

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial r^2} \tag{54}$$

for $0 < r < \pi$ and $t > 0$.

Initial Condition from Eq. (53):

$$u(0, r) = r . \tag{55}$$

The radiating boundary condition as given in Eq. (38) becomes

$$\frac{\partial u}{\partial r} = u(t, \pi) - g u^4(t, \pi) \quad \text{at} \quad r = \pi \tag{56}$$

where

$$g = \frac{R}{\pi} \frac{\epsilon \sigma T_0^3}{k} . \tag{57}$$

Boundary Condition:

$$u(t, 0) = 0 \quad \text{at} \quad r = 0 . \tag{58}$$

Equation (58) insures that the temperature $T(\tau, 0)$ remains finite. In this case the temperature is prescribed on the boundary $r = 0$ while at the boundary $r = \pi$ the temperature gradient $\partial u/\partial r$ is a given function of the temperature. The Finite λ-Transform, as given by Eq. (7), is to be used for these boundary

conditions. From Eqs. (9), (56), and (58) one obtains the following set of differential equations

$$\dot{f}_\lambda(t, n) = -(n - \tfrac{1}{2})^2 f_\lambda(t, n) - (-1)^n\left[u(t, \pi) - g u^4(t, \pi)\right] \qquad (59)$$

$$n = 1, 2, \ldots .$$

From Eq. (8)

$$u(t, r) = \frac{2}{\pi}\sum_{n=1}^{\infty} f_\lambda(t, n)\sin(n - \tfrac{1}{2})r \qquad (60)$$

and

$$u(t, \pi) = \frac{2}{\pi}\sum_{n=1}^{\infty} -(-1)^n f_\lambda(t, n) . \qquad (61)$$

Equation (61) is used in the simulation to form the last term in Eq. (59). From Eq. (53) the temperature at $\rho = R$ is given by

$$T(t, R) = \frac{T_0}{\pi} u(t, \pi) . \qquad (62)$$

The initial conditions $f_\lambda(0, n)$ are obtained from Eqs. (55) and (7) by

$$f_\lambda(0, n) = \int_0^{\pi} r \sin(n - \tfrac{1}{2})r \, dr , \qquad (63)$$

$$n=1, 2, \ldots, N$$

which upon integration becomes

$$f_\lambda(0, n) = -\left(\frac{1}{n - \frac{1}{2}}\right)^2 (-1)^n , \qquad (64)$$

$$n=1, 2, \ldots, N .$$

Since initially $u(t, \pi) = \pi$ from Eq. (53), one can perform a steady state check on the computer setup and on the convergence of the truncated series; namely, from Eqs. (61) and (64)

$$\frac{2}{\pi}\sum_{n=1}^{N}\left(\frac{1}{n - \frac{1}{2}}\right)^2 \approx \pi . \qquad (65)$$

On the analog computer (as well as numerically) 2% accuracy may be achieved with eight coefficients. The analog simulation is shown in Fig. 4.

Concluding Remarks

Problems with convective as well as radiating boundary conditions prescribed may be solved by this method by the obvious extension of combining the convective term $h(u - u_i)$ and the radiation term $g(u^4 - ua^4)$ in the differential equations generating the Finite Fourier Transforms. The temperature at points other than the boundary, if they should be needed, may be readily obtained by summing the Fourier coefficients weighted by their proper trigonometric terms. This simply requires a potentiometer for each term and an inverter when necessary. For convection and for radiation the temperature at the boundary should be as accurate as possible, because the boundary conditions themselves are given by them. The Finite Fourier Transform method which gives the exact boundary temperature within the computer accuracy can be very successfully applied.

The most widely used methods for the solution of partial differential equations are based on finite differences. These methods require certain assumptions about where the finite difference equals the derivative which by necessity have to be most loosely made on the boundaries; the very points where the greatest accuracy is required. One must form many grid points, at least, near the boundary in order for the solution of the difference equation to come close to the solution of the differential equations. For substances with low conductivity, as used for nose cones in missiles, the spacing of the grid points become even more

critical.

The Finite Fourier Transform method may also be used for equations with more than one space variable. Stephens and Karplus[6] have employed this transform for analog simulations of two-dimensional diffusion equations. Two approaches are given in their paper. In the first only one space variable is transformed, which reduces the given problem to a partial differential equation with one independent space variable. This they solve then by a finite difference method. In the second approach the dependent function is transformed with respect to both space variables, thus producing a system of ordinary differential equations. In principle, this method may be extended to analog simulations of heat equations in three space variables. The number of ordinary differential equations, however, becomes quite large when Fourier transformations are performed with respect to three variables. Their analog simulation would be practical only if the analog computer is capable of time sharing or repetitive modes of operation, and possesses dynamic storage elements. The Finite Fourier Transform method may also be a very efficient technique for the solution of multidimensional heat equations on a linked Analog-Digital computing system.[7]

The number of amplifiers required for the solution of the heat equation by the Finite Fourier Transform method is about one-half to one-fifth of the number of amplifiers needed for a finite difference simulation of equal accuracy. The ratio depends on the shape of the heat input curve, the thermal constants, and the physical dimensions.

References

1. Sneddon, I.N., "Fourier Transforms," McGraw-Hill, New York, 1951.

2. Tranter, C.J., "Integral Transforms in Mathematical Physics," Wiley & Sons, Inc., New York, 1956.

3. Leonard, J.L., "Integral Transforms for Application to Partial Differential Equations," Grumman Aircraft Engineering Corporation, Research Department Report RE-103, September 1958.

4. Churchill, "Modern Operational Mathematics in Engineering," McGraw-Hill, New York, 1944.

5. Abarbanel, P.S., "Time Dependent Temperature Distribution in Radiating Solids," Journal of Mathematics and Physics, Vol. 30, 1960.

6. Stephens, P.A., Jr. and Karplus W.J., "Application of Finite Integral Transforms to Analog Simulations," AIEE Transactions, Vol. 78, Part I, 1959.

7. Burns, A.J. and Kopp, R.E., "Combined Analog-Digital Simulation," Proceedings of the Eastern Joint Computer Conference, December 1961.

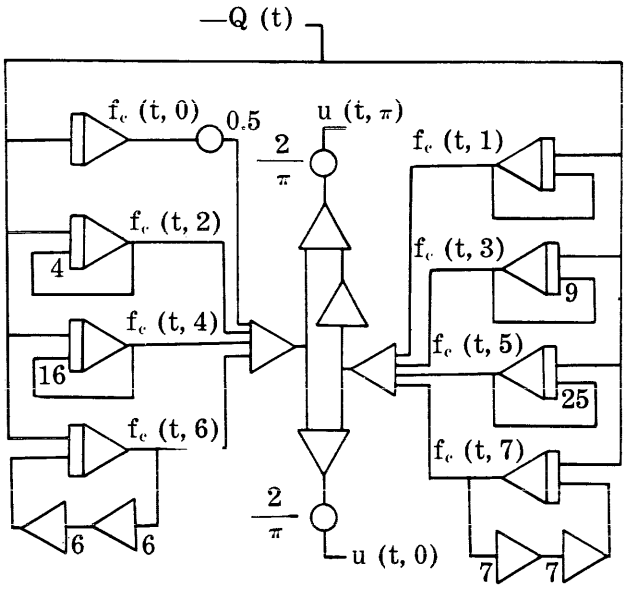


Fig. 1 - Analog Simulation Of The Re-entry Problem

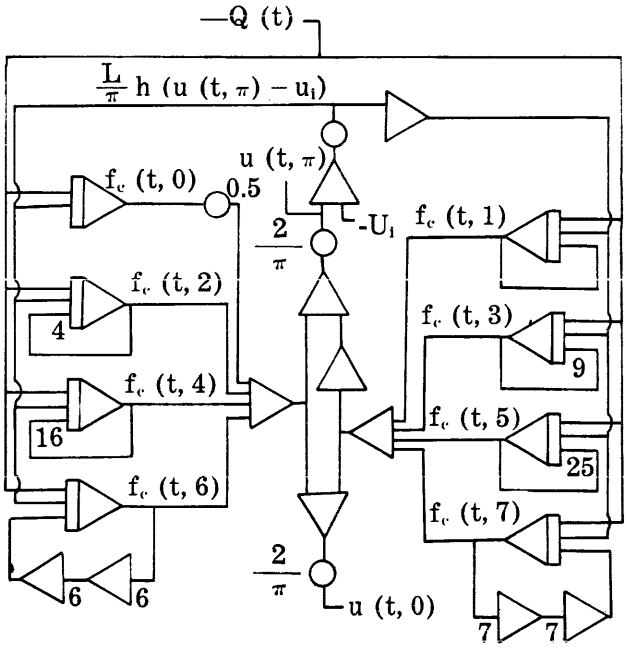


Fig. 2 - Heat Equation With Convective Boundary Conditions

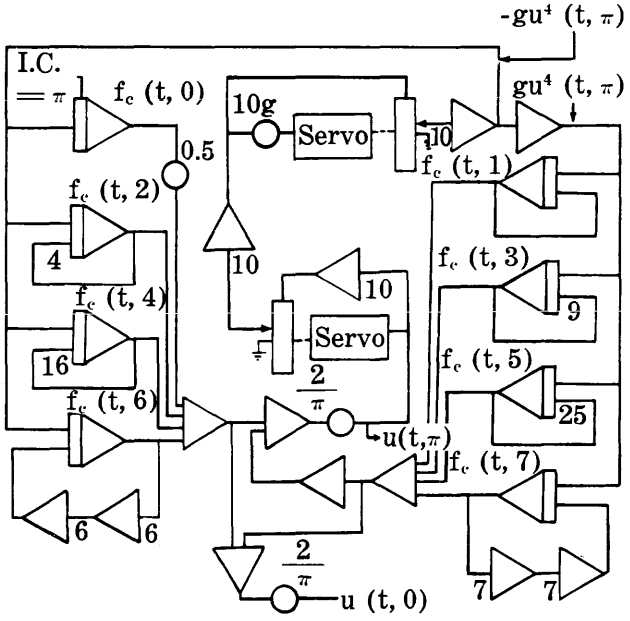


Fig. 3 - Flat Plate Radiating Into Vacuum

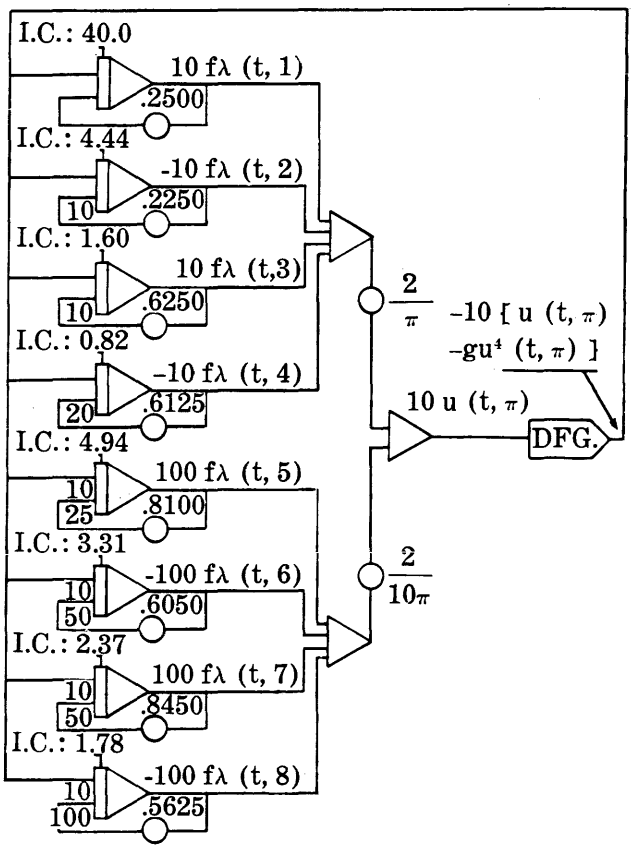


Fig. 4 - Sphere Radiating Into Vacuum

ANALOG SIMULATION OF THE RE-ENTRY OF A BALLISTIC MISSILE WARHEAD AND MULTIPLE DECOYS

L. E. Fogarty and R. M. Howe

University of Michigan

Ann Arbor, Michigan

Summary

The basic problem considered here is the computation of the reentry trajectory of a single ballistic missile warhead as well as the trajectories of a number of decoys which originate from the warhead trajectory. Suitable three dimensional equations of motion are presented for a reentry vehicle with arbitrary drag coefficient, mass, and area, and the analog computer circuit for solving these equations in real time is given. Then a method of using several such circuits to compute simultaneously the trajectories of multiple targets with variations in all three initial velocity components as well as variations in ballistic coefficient is presented.

Introduction

One of the more interesting current problems in simulation involves the computation of the trajectories of a very large number of reentry vehicles, each starting with the same position coordinates and approximately the same velocity coordinates, but with the possibility of widely varying ballistic coefficients. For example, this is the problem presented in the simultaneous simulation of a reentering ballistic missile warhead and a large number of decoys. In this paper we will show first how each trajectory can be computed in real time using a modest amount of analog computing equipment. If we compute decoy trajectories for the extreme upper and lower limits of velocity perturbations and ballistic coefficient variations, then simple linear or second order interpolation can be used to obtain the trajectory for any decoy with intermediate velocity or ballistic coefficient variation. The required averaging and summing operations can be accomplished using simple passive resistors. The net result is that a reasonable amount of analog equipment, as needed to generate the limiting trajectories, can be used as a basis for simultaneous generation of hundreds of decoy trajectories

First we will consider the appropriate equations of motion for a single reentering particle. The analog-computer solution of these equations will then be presented using a mechanization with solid-state nonlinear components. This adds the possibility of high speed repetitive operation, although for simulation purposes real time computation would be more appropriate. Finally, the technique of generating a very large number of trajectories from several limiting trajectories will be described.

Equations of Motion in the Trajectory Plane

It will be assumed that for purposes of this simulation the rotation of the earth can be neglected. Although this assumption is not in any way necessary to implement a practical analog solution, it simplifies the equations and hence the computer mechanization of the problem. The actual errors in trajectory as viewed from a position near the impact point as a result of this assumption will be quite small compared with the total trajectory distance traversed in the simulation.

With the above assumption the trajectory for a pure drag vehicle will lie in a vertical plane through the center of the earth. We can use polar coordinates r, θ in this plane to describe the position of the warhead, which we will assume is a point mass m (see Figure 1). We denote the horizontal velocity component by U_h and the vertical velocity component by W_h (positive downward). Summing forces in the horizontal and vertical directions, we obtain

$$m(\ddot{r}\dot{\theta} + 2\dot{r}\dot{\theta}) = X_h \qquad (1)$$

$$m(\ddot{r} - r\dot{\theta}^2) = -\frac{mg_o r_o^2}{r^2} - Z_h \qquad (2)$$

Here X_h and Z_h are horizontal and vertical forces due to aerodynamic drag, and g_o is the acceleration due to gravity at a fixed distance, r_o, from the center of the spherical earth. The term $2\dot{r}\dot{\theta}$ in Eq. (1) is the Coriolis acceleration, whereas $r\dot{\theta}^2$ in Eq. (2) is the centrifugal acceleration.

The terms $r\ddot{\theta} + 2\dot{r}\dot{\theta}$ in Eq. (1) can be rewritten as $1/r \; d/dt(r^2\dot{\theta})$. Thus the equation can be

integrated to give

$$mr^2\dot{\theta} = \int_o^t r\, X_h\, d\tau + mr^2\dot{\theta}]_{t=0} \tag{3}$$

This is the well known angular momentum integral. We note that

$$U_h = r\dot{\theta}, \qquad W_h = -\dot{r} \tag{4}$$

Thus Eqs. (2) and (3) can be rewritten as

$$W_h = \int_o^t \left(\frac{g_o r_o^2}{r^2} - \frac{U_h^2}{r} + \frac{Z_h}{m}\right) d\tau + W_h]_{t=0} \tag{5}$$

and

$$U_h = \frac{1}{r}\int_o^t \frac{rX_h}{m}\, d\tau + \frac{rU_h]_{t=0}}{r} \tag{6}$$

The aerodynamic drag force D will be directed opposite to the warhead total velocity vector, as shown in Figure 2, and is given by

$$D = \frac{1}{2}\rho_a V_a^2 C_D A \tag{7}$$

where ρ_a is the atmospheric density, V_a is the total velocity, C_D is the drag coefficient, and A is the characteristic area on which the drag coefficient is based. C_D will in general be a function of Mach number M, although it may be sufficiently accurate for purposes of this simulation to assume that C_D is constant. The drag components X_h and Z_h along the horizontal and vertical (downward) directions are given, respectively, by

$$X_h = -D\frac{U_h}{V_a} = -\left(\frac{1}{2}\rho_a C_D A\right) U_h V_a \tag{8}$$

$$Z_h = -D\frac{W_h}{V_a} = -\left(\frac{1}{2}\rho_a C_D A\right) W_h V_a \tag{9}$$

where the total velocity V_a is given by

$$V_a = (U_h^2 + W_h^2)^{1/2} \tag{10}$$

Finally, we will assume an exponential atmospheric model. Thus let

$$\rho_a = \rho_{ao} e^{-h/h_o} \tag{11}$$

where the altitude h is given by

$$h = r - R \tag{12}$$

R is the radius of the spherical earth. Eqs. (5), (6), and (8) through (12) are the basic equations for motion in the plane of the trajectory. Scaling of these equations for the computer is simplified if we introduce a dimensionless radial distance ρ and a perturbation $\delta\rho$ given by the formulas

$$\rho = r/r_o, \qquad \delta\rho = \rho - 1 \tag{13}$$

From equation (5) with $W_h = W_h]_{t=0} = Z_h = X_h = 0$ we note that a satellite in a zero-drag circular orbit at radial distance r_o will have a velocity U_{h_o} given by

$$U_{h_o} = \sqrt{r_o g_o} \tag{14}$$

It is convenient to define dimensionless velocities in terms of U_{h_o}. Thus let

$$u_h = U_h/U_{h_o}, \quad w_h = W_h/U_{h_o}, \quad v_a = V_a/U_{h_o} \tag{15}$$

In terms of the dimensionless variables of Eqs. (13) and (15), Eqs. (5) and (6) become

$$w_h = \int_o^t \left[\frac{1}{1+\delta\rho} - u_h^2\right) + \frac{Z_h}{mg_o}\right] \frac{d\tau}{T} + w_h]_{t=0} \tag{16}$$

and

$$u_h = \frac{1}{1+\delta\rho}\int_o^t (1+\delta\rho)\,\frac{X_h}{mg_o}\, d\frac{\tau}{T} + \frac{\rho_o u_{h_o}}{1+\delta\rho} \tag{17}$$

where ρ_o and u_{h_o} are the initial values of ρ and u_h and where T is a time constant equal to the reciprocal of the circular-orbit frequency and is given by

$$T = \sqrt{r_o/g_o} \tag{18}$$

For typical ballistic reentry trajectories the altitude at initiation of trajectory simulation may range as high as 200 statute miles. If we choose 100 statute miles as the altitude of the circular reference orbit, then $r_o = 3950 + 100 = 4050$ statute miles and r ranges between 4150 miles initially and 3950 miles at impact. Thus $\delta\rho$ in Eq. (13) ranges over $\pm\, 100/4050 = \pm\, 0.025$. Therefore we will neglect $\delta\rho$ everywhere in Eqs. (16) and (17) as being small compared with unity except in the term $[(1+\delta\rho)^{-1} - u_h^2]$, where we approximate $(1+\delta\rho)^{-1}$ by $1 - \delta\rho$, and the term $u_{h_o}(1+\delta\rho)^{-1}$ which we also approximate as $u_{h_o}(1-\delta\rho)$. Then Eqs. (16) and (17) become

$$w_h = \int_o^t \left[1 - \delta\rho - u_h^2 + \frac{Z_h}{mg_o}\right] \frac{d\tau}{T} + w_h]_{t=0} \tag{19}$$

and

$$u_h = \int_o^t \frac{X_h}{mg_o}\frac{d\tau}{T} + \rho_o u_{h_o} - (\rho_o u_{h_o})\delta\rho \qquad (20)$$

In addition, we note that

$$\frac{d(\delta\rho)}{d(t/T)} = -w_h \qquad (21)$$

For r_o = 4050 statute miles the time constant T = 837 seconds.

Let us assume that we desire to measure the position of the warhead in the plane of the trajectory using dimensionless rectilinear coordinates x and y, as shown in Figure 3. If 1000 statute miles is approximately the maximum value for horizontal target distance $r_o x$, then the maximum value of θ in the figure is approximately 0.25 radians or 14 degrees, in which case small-angle geometry can be used to compute x and y. Thus if we integrate local horizontal velocity $U_{h_o} u_h$ directly, we obtain, approximately, the distance coordinate xr_o. Therefore, we can write

$$x_o - x \cong \frac{U_{h_o}}{r_o}\int_o^t u_h d\tau = \sqrt{\frac{g_o}{r_o}}\int_o^t u_h d\tau = \int_o^t u_h \frac{d\tau}{T} \qquad (22)$$

where x_o is the initial x.

From Figure 3 it is apparent that

$$r_o y = r\cos\theta - R \cong r\left(1 - \frac{\theta^2}{2}\right) - R \cong r - R - r_o\frac{\theta^2}{2} \qquad (23)$$

Clearly $\theta \cong x$, and Eq. (23) becomes

$$y \cong \delta\rho + \frac{r_o - R}{r_o} - \frac{x^2}{2} \qquad (24)$$

From Eqs. (22) and (24) we can compute the dimensionless coordinates x and y, having solved Eqs. (16) and (17) for the dimensionless velocities u_h and w_h, and the dimensionless radial perturbation $\delta\rho$. To define completely the equations to be solved with the analog computer we need only rewrite Eqs. (8) and (9) for the horizontal and vertical aerodynamic forces in terms of the dimensionless variables. It is convenient for scaling purposes to compute in each case the logarithm of the horizontal and vertical aerodynamic acceleration in units of g_o. Thus we obtain

$$\log\left(-\frac{X_h}{mg_o}\right) = B - 0.434\frac{r_o}{h_o}\delta\rho + \log u_h + \log v_a \qquad (25)$$

and

$$\log\left(-\frac{Z_h}{mg_o}\right) = B - 0.434\frac{r_o}{h_o}\delta\rho + \log w_h + \log v_a \qquad (26)$$

where

$$B = \log\left[\frac{1}{2}\rho_o e^{-\frac{r_o - R}{h_o}}\frac{C_D A}{mg_o}U_{h_o}^2\right] \qquad (27)$$

and is a constant if C_D is a constant, or at worst a simple function of Mach number M for a Mach-dependent drag coefficient.

We have now developed the equations for obtaining the x and y coordinates of a warhead or decoy in the plane of the trajectory, and the equations are scaled conveniently for computer solution. We will see later that displacement at right angles to the plane of the nominal trajectory for decoys with initial side-velocity components is directly proportional to horizontal-distance traveled, $x_o - x$, and can be so computed.

Analog Computer Circuit for Obtaining Trajectories in the Plane of the Motion

The analog computer circuit for solving the two-dimensional equations of motion given at the end of the previous section is shown in Figure 4. The circuit has been scaled with unity equal to 100 volts, the computer reference, and uses the dimensionless velocity and displacement variables presented earlier. The circuit as shown allows horizontal velocities up to 25,600 ft./sec., altitudes up to 200 statute miles, initial horizontal down range distance from impact of 1000 statute miles, and maximum vertical and horizontal acceleration components of 50 g's. The time scale of this particular circuit is approximately 42 times real time (T = 20 seconds instead of 837 seconds, as given in Eq. (18) for a mean reference altitude of 100 miles). This was chosen to allow convenient recording of solutions. Extension of the circuit to real-time operation involves reduction of the gain of each integrator by a factor of 42, which because of our previous experience on real-time analog simulation of orbital and reentry vehicles is known to be quite feasible.

Note that all nonlinear operations in the cir-

cuit of Figure 4 are performed using solid-state components, quarter-square multiplier elements for squaring and square-root operations, and logarithm generators for computing aerodynamic acceleration components in accordance with Eqs. (25) and (26). The logarithm generators allow particularly favorable scaling to be employed in what is normally a very difficult problem for analog mechanization. The diodes around amplifiers B8 and C6 limit the amplifier outputs in the negative voltage direction to about -0.7 volts. This prevents amplifier saturation whenever the log generators go outside their useful range (in this case approximately 2 1/2 decades). Thus aerodynamic accelerations can be computed with reasonable accuracy from 50 g_o down to 0.2 g_o. The diode circuits around amplifiers C1 and C2 prevent the vertical and horizontal drag accelerations from taking on negative values. Thus as a reentry simulation takes place the drag acceleration is held to zero within the offset of amplifiers C1 and C2 (equivalent to less than $\pm 2.5 \times 10^{-6} g$) at high altitudes. As soon as the more dense atmosphere is encountered, the drag acceleration increases smoothly from zero, with the computation accurate to the order of several percent of the value itself for 0.5 g or greater acceleration. The diodes around amplifier A1 prevent horizontal velocity u_h from going negative.

This problem was set up on an AD-1-32PB* electronic differential analyzer using quarter-square multipliers with an accuracy specification of ± 0.035 percent (± 35 millivolts) for the squaring operation and fixed diode log generators with an accuracy specification of ± 0.2 percent over two decades of input voltage. Actual recordings of computer solutions are presented in a later section.

Note in Figure 4 that a single potentiometer, C4, sets the logarithm of C_DA/mg_o. For a Mach-dependent drag coefficient it is only necessary to add a function of mach number M to the input of amplifier B8. Note also that single potentiometers set perturbations δu_{h_o} and δw_{h_o} in initial horizontal and vertical velocity.

Generation of the Third Dimensional Coordinate for Multiple Trajectories

In all of the discussion up to now we have assumed a two-dimensional trajectory in a nominal reference plane. If the warhead or any decoy has an initial small velocity component v_{h_o} at right angles to the nominal trajectory plane, the net effect will be to yaw the trajectory plane for that particle through a small angle $\psi = v_{h_o}/v_{a_o}$, where v_{a_o} is the initial total velocity. This will generate a dimensionless side displacement $z \cong \psi(x_o - x)$, where $x_o - x$ is the dimensionless horizontal distance traveled by the particle from the start of the reentry trajectory simulation. Using the voltage output $4(x_o - x)$ from the circuit of Figure 4, the circuit of Figure 5 can be utilized to compute the side displacement z for any number of targets, each with arbitrary initial side velocities and hence trajectory yaw angles ψ of either polarity. Note that only n + 1 passive resistors are required for n targets.

*Applied Dynamics, Inc., Ann Arbor, Michigan

Generation of the Two-Dimensional Coordinates for Multiple Trajectories

In the previous section we saw how the side displacement z could easily be generated for any arbitrary number of particles from the downrange distance $x_o - x$. Assume next that we wish to compute the trajectory variables x and y for a large number of particles, each with different initial horizontal velocity u_{h_o}, vertical velocity w_{h_o}, and ballistic coefficient C_DA/mg_o. As a specific example suppose we choose for a nominal trajectory that of a particle with an initial altitude of 200 statute miles, $u_{h_o} = 0.8$ (initial horizontal velocity of 0.8(25,600) = 20,500 ft./sec.), $w_{h_o} = 0.25$ (initial flight path angle $\gamma = \tan^{-1}(-.25/.8) = -17.3$ degrees), and $C_DA/mg_o = 0.1$. Next assume that the maximum deviation δu_{h_o} in horizontal velocity corresponds to ± 512 ft./sec. In Figure 6b are shown the analog computer trajectory solutions in this case, where the center trajectory is the nominal one, and where the trajectories on either side represent the particle path for + 512 ft./sec. and -512 ft./sec. horizontal velocity deviations, respectively. Here the nominal trajectory impact point is approximately at the origin of the xy coordinate system of Figure 3.

Next consider the trajectories for initial vertical velocity deviations of ± 512ft./sec., as shown in Figure 6a. Again the center trajectory is the nominal one whereas the trajectories on either side are the perturbed ones. For orientation purposes the outline of the surface of the earth is shown at the bottom of the figure.

Finally, consider the trajectories for $C_DA/mg_o = 1$ and 0.01 in addition to the nominal C_DA/mg_o of 0.1. These results are also shown in Figure 6a, where the trajectories for $C_DA/mg_o = 1$ and 0.01 do not depart from the nominal trajectory until near the end of the trajectory, since this is where the large aerodynamic forces are encountered.

From the results of Figure 6 it is evident that reasonably accurate trajectories for any initial velocity components between the two limiting

extremes can be obtained by linear interpolation from the extreme trajectories. For different C_DA/mg_o it appears that a logarithmic interpolation is preferable. This conclusion is substantiated in Figure 7, where vertical target displacement y and horizontal target displacement x are recorded as a function of time for $C_DA/mg_o = 1$, 0.1, and 0.01. In each case points which would have been obtained for $C_DA/mg_o = 0.1$ from logarithmic interpolation between the trajectories for $C_DA/mg_o = 1$ and 0.01 are shown. Note that they fall almost exactly on the true trajectory for $C_DA = 0.1$.

Similarly, in Figures 8 and 9 are shown time-history recordings of y and x for horizontal velocity deviations of ± 512 ft./sec. from the nominal trajectory and vertical velocity deviations of ± 512 ft./sec. In each case it is evident that the nominal trajectory would have been obtained accurately by linear interpolation from the extreme trajectories.

In the previous section we established the method for generating lateral displacement z for any arbitrarily large number of targets with different initial side velocities. We have now established the method for a similar calculation of the x and y coordinates for any arbitrarily large number of targets with different initial horizontal and vertical velocities, and ballistic coefficient, i.e., compute the limiting trajectories at both extremes of each initial variable and use linear or logarithmic interpolation to obtain the individual trajectories.

The above interpolation computations can be implemented with simple passive resistor networks. Assume that we have six analog computer circuits similar to that shown in Figure 4, each capable of generating a separate reentry trajectory. Let the first two circuits employ nominal initial vertical velocity w_h and ballistic coefficient C_DA/mg_o, but the extreme initial limits of initial horizontal velocity, i.e., $u_{h_o} + \delta u_{h_o}$ and $u_{h_o} - \delta u_{h_o}$, respectively. In each case designate the output coordinates x_{u_1}, y_{u_1} and x_{u_2}, y_{u_2}, respectively.

Similarly, let the second pair of circuits compute the trajectory coordinates x_{w_1}, y_{w_1} and x_{w_2}, y_{w_2}, respectively, for the extreme limits of initial vertical velocity, and the third pair of circuits compute the trajectory coordinates x_{D_1}, y_{D_1} and x_{D_2}, y_{D_2}, respectively, for the extreme limits of C_DA/mg_o.

Finally, assume that a seventh analog circuit generates the nominal trajectory coordinates x_{ref}, y_{ref}. Then the interpolations described previously can be implemented to compute x(t) by means of the circuit shown in Figure 10, which is actually a mechanization of the formula

$$x = x_{ref} + \frac{\partial f}{\partial u_h}\Delta u_h + \frac{\partial f}{\partial w_h}\Delta w_h + \frac{\partial f}{\partial \log \frac{C_DA}{mg_o}}\Delta \log \frac{C_DA}{mg_o} \qquad (28)$$

This formula simply consists of the zeroth and first order terms in the Taylor series expansion for x about the nominal trajectory x_{ref}. The difference voltages $x_{u_2} - x_{ref}$ and $x_{ref} - x_{u_1}$ are applied across an array of series resistors as shown at the top of Figure 10. From these resistors we can obtain voltages $x_1'(t), x_2', \ldots$ representing the term $\partial f/\partial u_h \Delta u_h$ in Eq. (28) for targets number 1, 2, 3, ... The distribution of series resistor values corresponds to the distribution in initial horizontal velocity perturbations for all the targets. Similarly the second and third terms on the right side of Eq. (28) are generated for different targets as voltage outputs of series resistor arrays at the middle and bottom of Figure 10. The distribution of series resistor values in the middle array corresponds to the distribution in initial vertical velocity perturbations for all the targets, while the distribution of series resistor values in the bottom array corresponds to the distribution in the logarithm of C_DA/mg_o for all the targets. For a given target the horizontal displacement coordinate x is obtained by summing the voltages representing each of the terms in Eq. (28) as implemented with the passive-resistor network shown in Figure 10.

The circuit for obtaining vertical displacement y for each of the targets is similar to that for x in Figure 10, except that the y voltage outputs from each trajectory simulation are used instead of the x outputs. Note that the circuit of Figure 10 uses linear interpolation between the nominal trajectory and the perturbed trajectory on either side, whichever is appropriate. If less accuracy is required or if the maximum perturbations that need to be considered are small enough, the nominal or reference trajectory can be selected with the minimum initial horizontal velocity, vertical velocity, and C_DA/mg_o to be considered. Then all perturbations are positive and only four trajectories (x_{ref}, y_{ref}; x_{u_2}, y_{u_2}; x_{w_2}, y_{w_2}; x_{D_2}, y_{D_2}) need be computed instead of the seven shown inf Figure 10.

In Figure 5 we showed the circuit used to generate different side-displacement z for various targets using horizontal displacement $x_o - x$ as measured from the initial position x. In Figure 10 each target has a different horizontal displacement x, from which it is possible to generate a lateral displacement -z proportional to $x_o - x$ by

a passive-resistor network which adds a voltage proportional to x to a constant negative bias voltage representing x_o. Here both signs of initial side-velocity perturbation can be taken into account by using for the reference-trajectory the plane corresponding to the maximum negative initial side velocity.

Conclusions

It is concluded that the reentry trajectory in two dimensions can be simulated with a modest amount of analog-computing equipment over a wide range of initial conditions and ballistic coefficients. The third dimension is obtained as a coordinate displacement proportional to horizontal-distance traveled. Reasonably accurate trajectories for a large number of particles with different initial velocities and ballistic coefficients can be obtained with passive resistor circuits by linear interpolation from limiting trajectories. A minimum of four two-dimensional trajectory circuits are required for this, although the use of seven trajectory circuits improves the accuracy considerably. In either case the amount of analog equipment required is not formidable. It should be noted that if accurate simulation of target velocities or accelerations is required (particularly in the case of accelerations) the simple linear interpolation formulas proposed in this paper would result in large errors in that portion of the trajectory wherein the maximum accelerations occur. Perhaps computation of each individual two dimensional trajectory is needed in this case.

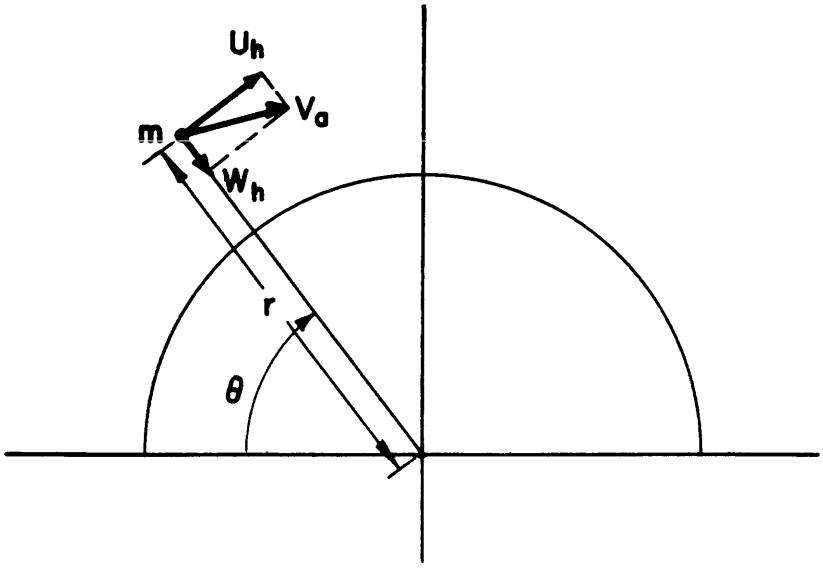


Figure 1. Basic Geometry in the Plane of the Trajectory

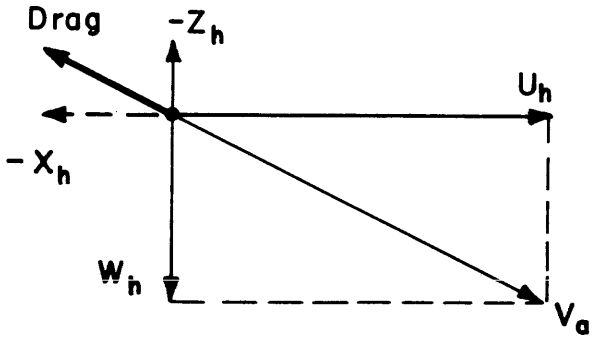


Figure 2. Geometry of the Drag Forces

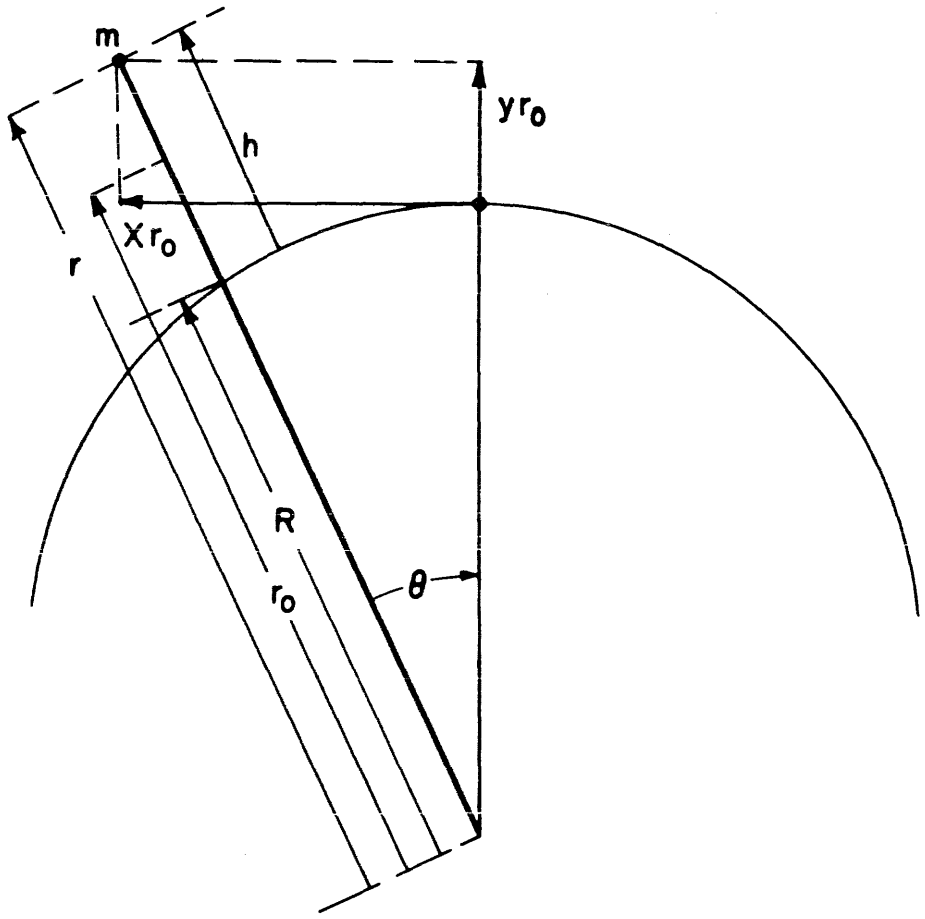


Figure 3. Coordinates in the Plane of the Trajectory

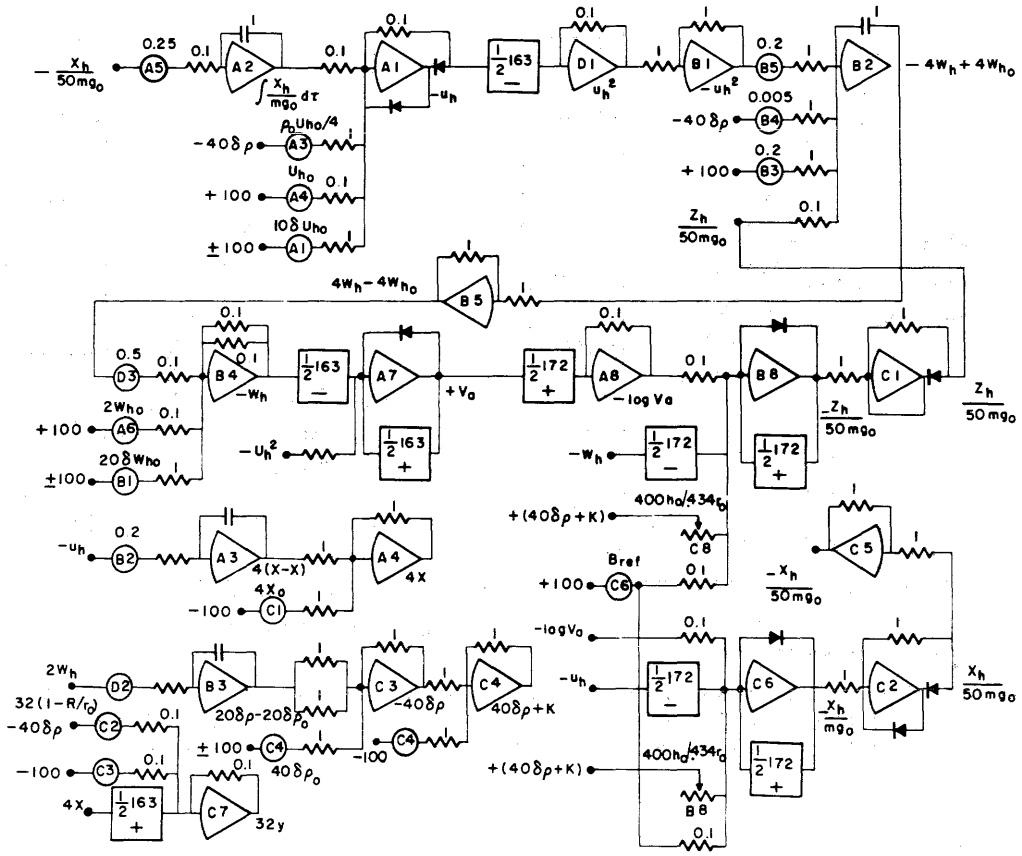


Figure 4. Computer Circuit for the Two-Dimensional Trajectory Equations

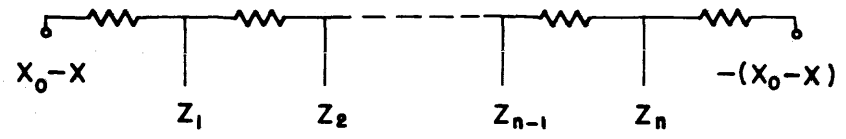


Figure 5. Circuit for Generating Side Displacement Z for n Targets

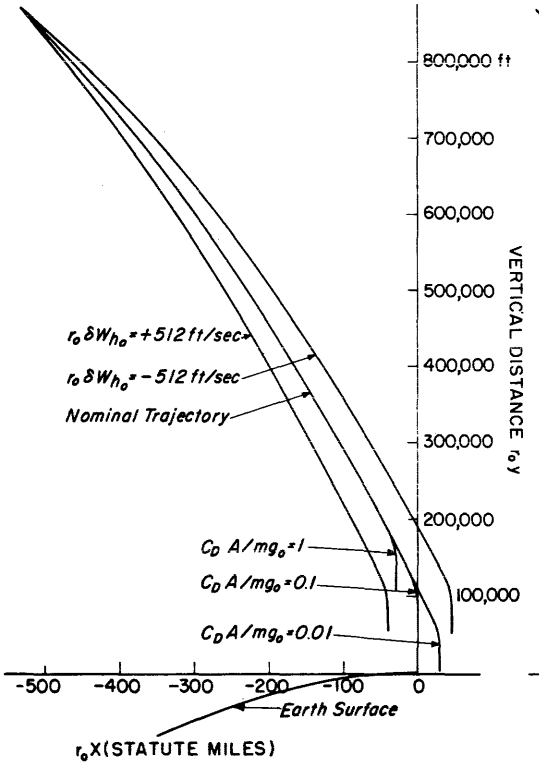


Figure 6a. Effect of Initial Vertical Velocity Perturbation and Different $C_D A/mg_o$ on an ICBM Trajectory

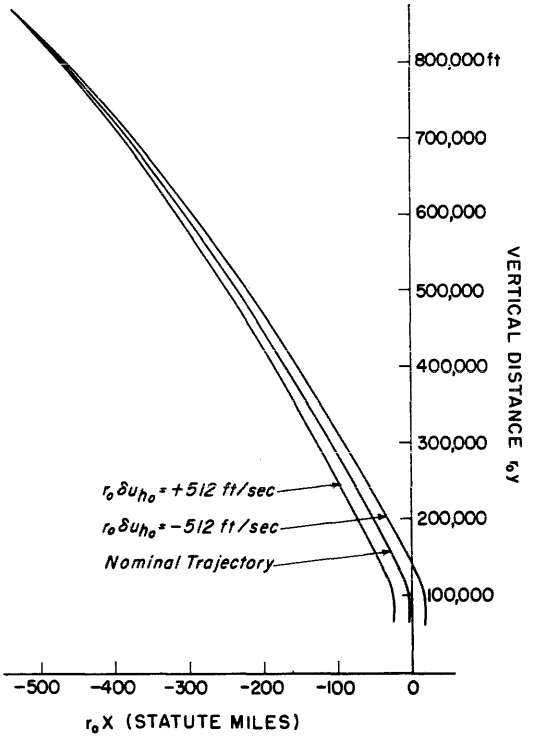


Figure 6b. Effect of Initial Horizontal Velocity Perturbation on an ICBM Trajectory

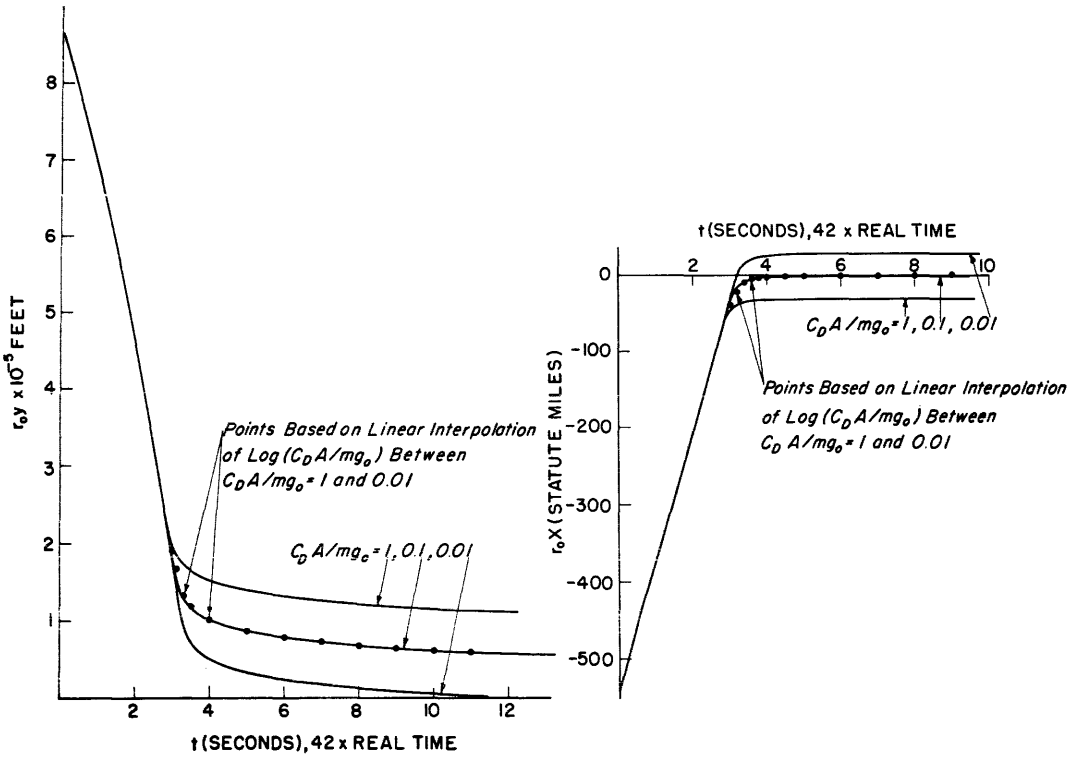


Figure 7. Effect of $C_D A/mg_o$

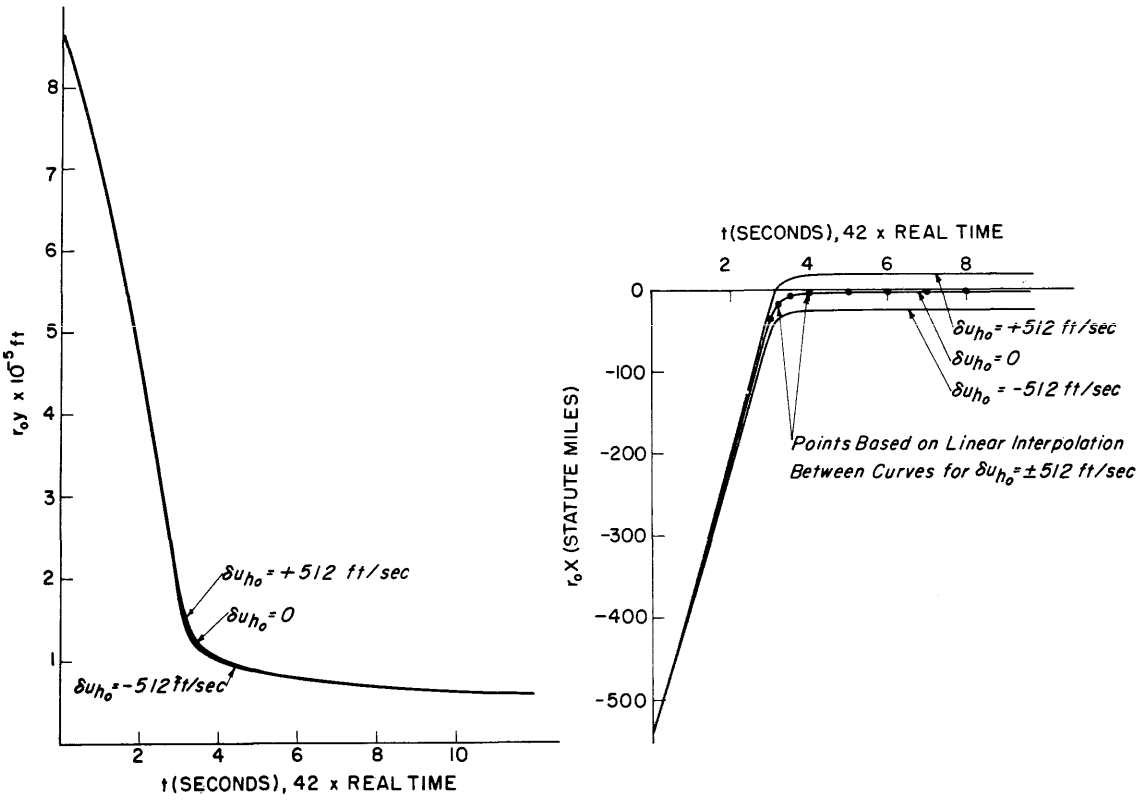


Figure 8. Effect of Initial Horizontal Velocity Perturbation

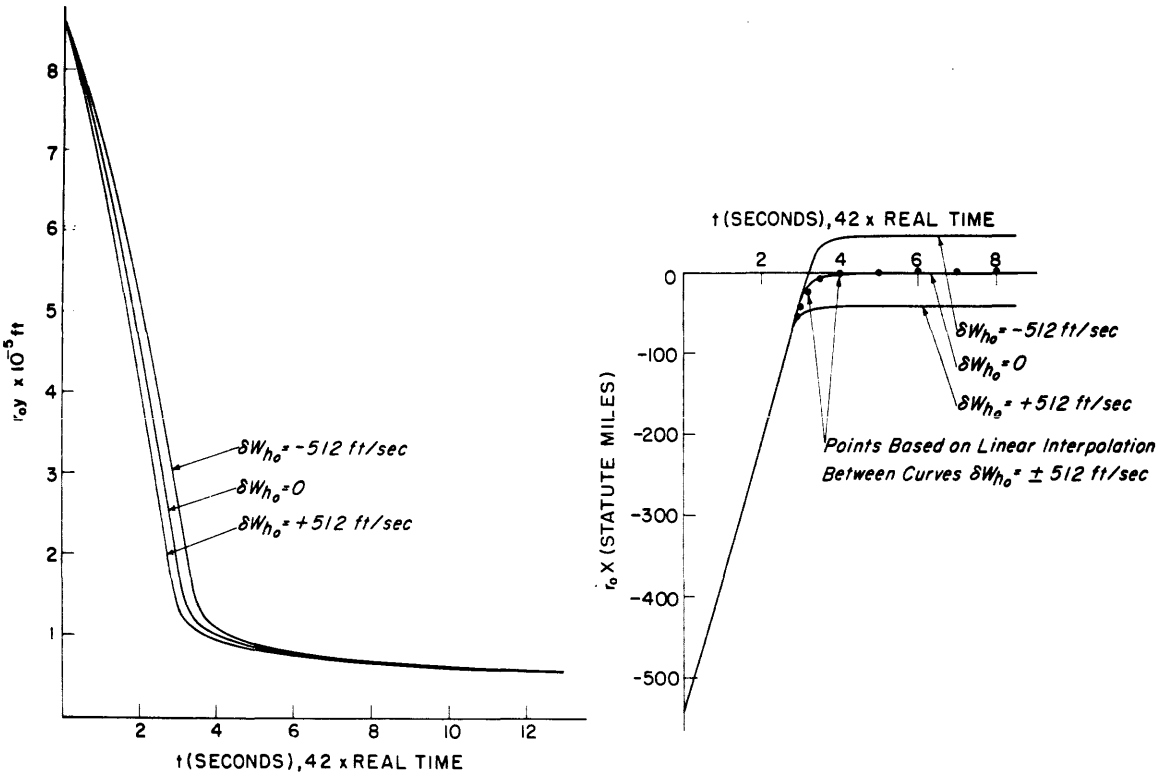


Figure 9. Effect of Initial Vertical Velocity Perturbation

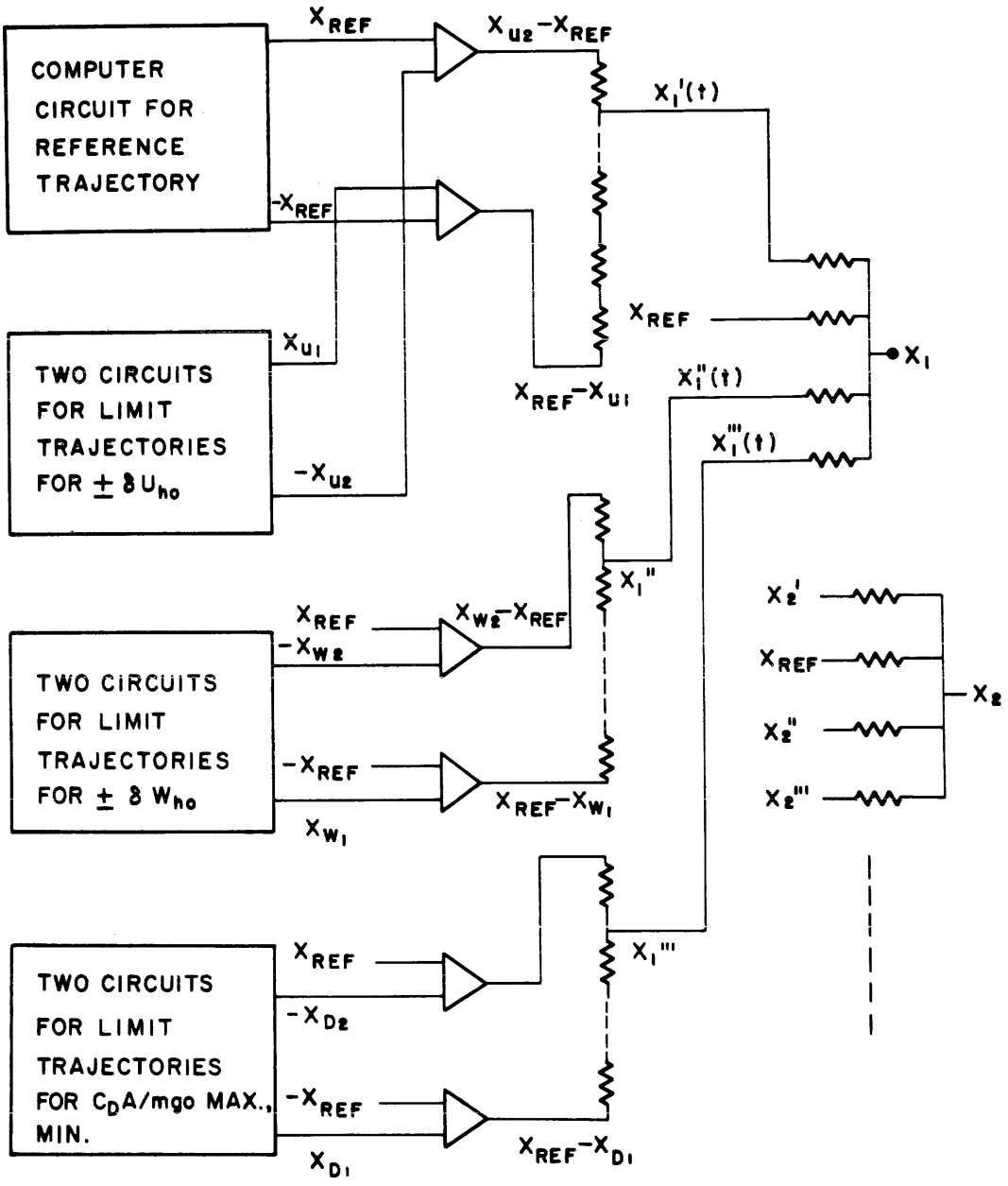


Figure 10. Circuit for Computing Horizontal Distance Coordinate for Multiple Trajectories

THE CONSTRUCTION OF AN EMPIRICALLY BASED MATHEMATICALLY DERIVED CLASSIFICATION SYSTEM

Harold Borko

System Development Corporation

Santa Monica, California

Summary

This study describes a method for developing an empirically based, computer derived classification system. 618 psychological abstracts were coded in machine language for computer processing. The total text consisted of approximately 50,000 words of which nearly 6,800 were unique words. The computer program arranged these words in order of frequency of occurrence. From the list of words which occurred 20 or more times, excluding syntactical terms, such as, and, but, of, etc., the investigator selected 90 words for use as index terms. These were arranged in a data matrix with the terms on the horizontal and the document number on the vertical axis. The cells contained the number of times the term was used in the document. Based on these data, a correlation matrix, 90x90 in size, was computed which showed the relationship of each term to every other term. The matrix was factor analyzed and the first 10 eigenvectors were selected as factors. These were rotated for meaning and interpreted as major categories in a classification system. These factors were compared with, and shown to be compatible but not identical to, the classification system used by the American Psychological Association. The results demonstrate the feasibility of an empirically derived classification system and establish the value of factor analysis as a technique in language data processing.

1. Traditional Methods of Classification

The purpose of classification is to organize and systematize a set of events, and classification in science provides a means by which knowledge can be divided into groups of related phenomena. An organized system of knowledge usually, although not necessarily, implies a hierarchical arrangement. In zoology, for example, animals may be classified by:

a. phyluum
b. class
c. order
d. family
e. genus
f. species
g. subspecies

The value of such a classification system is obvious, for it helps organize knowledge on a functional basis from the general to the specific.

Libraries, which are the store houses of human knowledge, use classification systems in order to arrange books and documents into groups according to subject matter. There are two principle systems of library classification--namely, the Dewey Decimal System and the Library of Congress System. These and similar schemes are based upon the theory that knowledge can be arranged into a small number of groups in a logical manner. It can be assumed that the categories selected were adequate for classifying existing documents, but certainly Dewey was not able to anticipate the advent of television, computers, and the entire electronics industry. However, this does not mean that new knowledge cannot be fitted into the Dewey Decimal System, new topics can be made into subdivisions of an existing category. The system is expandable, but at a price.

By way of an example: If we were to classify a document about angels, we would have no trouble fitting it into the category of theology with the classification number 235.5. Angels, you see, were in existence at the time the system was organized. If, however, we were to try to classify a document on tunnel diodes, we would find that there is no such category. If we were fortunate enough to have a librarian who was aware that tunnel diodes were circuit components used by electronic engineers, she would classify the document as 621.38151 which deals with electronic engineering, circuit components, vacuum tubes. Obviously, tunnel diodes are not vacuum tubes, but this is the closest approximation available, and the document will actually be classified in this manner. Even if we were to grant that an engineer would look for a document dealing with tunnel diodes under the classification of vacuum tubes--and this is highly unlikely--the classification would still be inefficient. Twice as many digits were required to classify this item as to classify "angel," and it is safe to assume that many more articles are being written on tunnel diodes then on angels these days. The point being made is that any classification system based upon a priori notions of hierarchy is bound to break down in spite of the fact that the system has the capacity to expand.

With the recognition of these limitations, the pendulum has swung to the opposite extreme--to the avoidance of all classification and the use of uniterms, descriptors, or subject headings. These procedures have many advantages, for one can create new index terms and these may be listed, usually alphabetically, and with no regard to a pre-ordered hierarchial arrangement. But there are also disadvantages. After the system has been operating for some time, and four or five thousand terms have been accumulated, the listing becomes difficult to use. To make the system more comprehensible, the human mind seeks to organize and group the related terms in some logical manner,

and we are back to a type of classification system with its multiplicity of problems.

How does one break out of this vicious cycle? Some form of classification seems to be necessary and desirable, but yet all a priori classification systems, no matter how logical they may appear to be, are doomed to failure. The solution must necessarily involve the development of a classification system based on empirical data and derived by some mathematical means. Can this be done? I believe that the answer to this question is yes, and a procedure for deriving such a classification system is described in this paper.

Science is based on the conviction that a given domain of events is not as chaotic as it would appear; that underlying a complex domain are a number of concepts which, if isolated and understood, could explain the occurrences of the events. The scientist attempts to discover these concepts and their relationships. Factor analysis is a mathematical tool which enables one to isolate the underlying variables in a domain of events. This method has been applied with a fair degree of success by psychologists in their attempts to determine the underlying variables of intelligence, personality, creativity, ability, etc. In this study, we are applying the technique of factor analysis to discover the relationship of key content words as they are used in the psychological literature. The concepts, derived in this manner, will form the basis for an empirical classification system.

The remainder of this paper will describe the experimental design and the methodology employed in the research.

2. Experimental Design

The basic concept of document classification or library organization is to group together those documents which deal with the same topic. As was pointed out earlier, one can start with a field of knowledge--broad or limited--and can, on an a priori logical basis, divide this field into various categories and subcategories. In the present study, we will avoid this traditional approach to classification, and we will seek to determine on an experimental basis whether documents can be grouped according to subject matter without any preconceived notion of logical categories.

The subject matter of a document is determined by its contents, and the contents are conveyed by the words in the document. It follows that documents on different topics will use a different set of words to express their ideas. Therefore, it is hypothesized that documents can be classified on the basis of the words they contain.

To experimentally test this hypothesis, the following procedures will be followed:

a. A set of documents will be selected and coded into machine language for computer processing.

b. The content words will be arranged by frequency of occurrence and 90 index terms will be selected by the experimenter from among the high frequency content words.

c. A data matrix will be prepared with the index terms on the horizontal axis and the document numbers on the vertical axis. Each individual cell will contain a number signifying the frequency of occurrence of the term in the given documents.

d. Based upon the numbers in the data matrix, each index word would be correlated with every other index word and the results arranged in a 90x90 correlation matrix.

e. The correlation matrix will be factor analyzed, and the resulting factors rotated and interpreted.

These steps in the experimental procedure will be discussed in detail.

2.1 Selection of Data

The selection of data for analysis is crucial to the study, and a number of topics including the size of the library, the specificity of the collection, its use, etc., must be considered in making the selection. It is obvious that if the library contained documents dealing with art, literature, psychology, physics, chemistry, etc., these documents could be grouped by subject with relatively little overlap. Such a collection would not provide a very rigorous test of the mathematical procedures for deriving a classification system. Consequently, it was decided to restrict the library to one field, and psychology was selected as the discipline to be investigated. The choice of psychology has many advantages. It has a rich literature, the American Psychological Association (APA) publishes twelve different journals, and there are in addition many other related periodicals. Among the journals published by the APA is the Psychological Abstracts which includes an index to the literature. It would, therefore, be possible to check the index derived by this study with the existing psychological index.

To construct an adequate classification system, approximately 500 documents would be needed. The average length of a journal article in psychological publications is between 3,500 and 5,000 words. A corpus of 500 articles would therefore contain between 1,750,000 and 2,500,000 words. This is an extremely large amount of text when one considers that each word must be keypunched for computer analysis. Since we do not wish to reduce the number of articles, it was decided to work with their abstracts rather than with the complete article. An average abstract, as it appears in the Psychological Abstracts, contains 100 to 200 words, and this is a reasonable number for computer analysis. Also, since by its very nature an abstract must contain the basic ideas expressed by the author, it should also contain the important words used to express these ideas.

1. Ability
2. Achievement
3. Activity
4. Analysis
5. Anxiety
6. Areas
7. Aspects
8. Attitude(s)
9. Behavior
10. Boys
11. Brain
12. Case(s)
13. Cerebral
14. Child(Children)
15. Clinical
16. College
17. Community
18. Concepts
19. Correlation
20. Counseling
21. Data
22. Development
23. Education(al)
24. Ego
25. Emotional
26. Evidence
27. Experimental(s)
28. Factor(s)
29. Family
30. Field
31. Frequency
32. Function
33. Girls
34. Group(s)
35. Guidance
36. Information
37. Intelligence
38. Job
39. Knowledge
40. Learning
41. Level(s)
42. Life
43. Light
44. Literature
45. Man(Men)
46. Mental
47. Nature
48. Normal
49. Organization
50. Parents
51. Patient(s)
52. Perception(ual)
53. Performance
54. Personal
55. Personality
56. Problem
57. Procedure(s)
58. Program
59. Psychiatric
60. Psychoanalysis(tic)
61. Psychology(ical)
62. Psychotherapy
63. Reading
64. Reinforcement
65. Research
66. Response(s)
67. Role
68. Scale
69. Schizophrenia
70. School(s)
71. Science
72. Selection
73. Social
74. Speech
75. Status
76. Stimulus(i)
77. Structure
78. Student(s)
79. Symptoms
80. System
81. Teaches
82. Technique(s)
83. Test(s)
84. Theory
85. Therapy
86. Training
87. Treatment
88. Value
89. Visual
90. Workers

Figure 1. Index Terms

Words		Case(s)	Child(Children)	Factor(s)	Level(s)	Psychology(ical)	School(s)
Abstract	# 74	0	0	1	1	1	0
	# 307	1	2	0	0	0	1
	# 321	0	2	1	0	1	0
	# 575	1	2	0	0	1	0
	# 626	0	1	0	1	0	2
	# 647	1	2	0	0	1	0
	# 653	4	1	0	0	1	0
	# 674	1	3	0	1	0	0
	# 796	1	1	1	0	0	0
	# 857	0	4	1	1	0	4
	# 905	0	8	0	0	0	3
	# 911	0	1	1	0	1	0
	# 917	0	2	0	1	0	1

Figure 2. A Portion of the Data (Document-Term) Matrix

As a result of the above reasoning, the Psychological Abstracts, volume 32, no. 1, 1958 was selected for study. This journal contains 1,037 abstracts of which we needed a representative sample of approximately 500. A quick glance through the abstracts reveals that some articles are very short, containing little more than the title and the author's name, and others are comparatively long. In order to insure that the sample will be relatively uniform and the selection unbiased, all articles between one and two inches in length were selected for inclusion in this study. This resulted in a sample of 618 abstracts.

2.2 Computer Analysis of the Text and Selection of Index Terms

Having selected the documents which were to make up the experimental library, the next step consisted of keypunching the entire body of text preparatory to computer processing. The program used for this analysis was FEAT (Frequency of Every Allowable Term)(3). It was written by John C. Olney of the System Development Corporation and is designed to perform frequency and summary counts of individual words and of word pairs. The output of this program is an alphabetical listing, by frequency of occurrence, of all word types which appeared in the text. Certain function words such as and, the, at, a, etc., were placed in a "forbidden word list" table, and the frequency of these words was recorded in a separate listing.

The FEAT program performs a number of analyses, but for the purpose of this experiment we are only concerned with the listings of each word type in order of frequency of occurrence and alphabetically within the frequency. This list provides a basis for selecting the index terms, for it is assumed that good index terms are words which occur often in the text. Therefore, all words which appeared twenty times or more were considered as possible index terms. From this list 90 terms were chosen for further analysis. Ninety terms were selected because the largest matrix that our computer program can handle is of rank 90.

Approximately 200 words in the text occurred 20 or more times. Words like "did," "however," "many," "compared," etc., were eliminated. Most of the terms selected were nouns. In the future it may be possible to define the criteria for good index terms rigorously enough so that it would be possible to make the selection automatically by means of a computer program. But for the present this was not possible, and so the experimenter made the final selection.

The 90 index terms used in the study are listed in Figure 1. Note that some words like child-children were combined into one index term.

2.3 Data Matrix, Document-Term

The next step in the analysis required a record of the documents in which each of the 90 terms occurred. A special computer program, called the Descriptor Word Index Program, was written to provide this information and to prepare a document-term matrix in a form suitable for input to the Factor Analysis Program. The Descriptor Word Index program was prepared by Eileen Stone of the System Development Corporation. Since the data consisted of 90 terms, a set of two 80-column cards was produced for each of the 618 documents. Every term was assigned a unique column on the card, and the number of times a word occurred in the document was punched in the proper column.

A portion of the document-term matrix is reproduced in Figure 2. The complete matrix is 90x618, 90 terms and 618 documents. The number in each cell represents the number of times a given word occurred in a particular document. Referring to Figure 2, it can be seen that the term "child (children)" did not occur in document number 74, occurred four times in document number 857 and eight times in document number 905. The term "level(s)" occurred once in documents numbers 74, 626, 674, 857, 917, and did not occur in the other documents in the example.

2.4 Correlation Matrix, Term-Term

The data matrix contains data on the number of times each term occurred in the various documents. With this information one can compute the degree of association among terms based upon their occurrence in the same set of documents. A measure of this association is the correlation coefficient. This is a decimal number which varies from +1.000 to -1.000. +1.000 would mean a perfect correlation, namely, that every time the word A occurred, word B appeared in the same document. A zero correlation would indicate no relationship between the occurrence of words in documents, and a negative correlation would mean that if word type A occurs, then word type B is not likely to occur in that document.

To compute the correlation coefficient from raw score data, (and this is the form of the Document-Term Matrix) we used the following formula:

$$r_{xy} = \frac{N\Sigma XY - (\Sigma X)(\Sigma Y)}{\sqrt{[N\Sigma X^2 - (\Sigma X)^2][N\Sigma Y^2 - (\Sigma Y)^2]}}$$

when N is equal to the number of documents (618) and X and Y are the terms being correlated.

A computer program exists for computing these correlations. The result of these calculations is a 90x90 matrix in which each term is correlated with every other term (see Figure 3). Each cell contains a number, a decimal fraction, which expresses the extent to which the two variables (i.e., terms) are related. Thus we see that "ability"--term 1--is correlated with "achievement" --term 2--with a coefficient of 0.272, while "ability" is correlated with "activity"--term 3-- with a correlation of 0.028.

	Terms				
	Ability	Achievement	Activity	Analysis	Anxiety
Ability		.27245	-.02846	.04804	.07999
Achievement	.27245		-.02563	-.04055	.11925
Activity	-.02846	-.02563		-.00207	-.02511
Analysis	.04804	-.04055	-.00207		.03022
Anxiety	.07999	.11925	-.02511	.03022	

Figure 3. A Portion of the Correlation (Term-Term) Matrix

Note that the diagonals in Figure 3 are blank. The diagonal value expresses the correlation of the term with itself, i.e., the communality of the term. By communality is meant the portion of the term's variance which is shared with other terms and not its unique component. There are a number of methods which can be used to estimate the communality. The simplest procedure is to choose the highest correlation coefficient from among the other correlations in that set (p. 86-87[1]). Although this method is crude and tends to underestimate the communality, it has been shown to be an adequate method when dealing with a large number of variables. In this study, the highest correlation coefficient in the column was used as the estimate of the communality.

It is worthwhile to re-emphasize at this point that the correlation matrix is completely determined by the empirical data and not by any a priori assumptions on the point of the investigator. "Ability" and "achievement" occur relatively frequently in the same document and so are highly correlated. Clearly, if we were able to group together the related terms, we would have a type of classification system, for we would be able to state that, for example, documents dealing with ability and documents dealing with achievement are likely to be in the same area of discourse. However, this is not a task that can be done by inspection. In the 90x90 matrix, which is symmetrical, we have 4,005 correlations $(\frac{N(N-1)}{2})$. In order to analyze the data in a precise fashion, we employ the technique of factor analysis

2.5 Factor Analysis

The technique of factor analysis is based upon matrix algebra. The procedures are described by Thurstone [4] and more recently by Harman[1] who also discusses the use of electronic computers as a computational aid. In this study, the calculations were performed on the IBM 7090 using programs available at the System Development Corporation.

The purpose of factor analysis is to reduce the original correlation matrix to a smaller number of factors. A factor corresponds to the eigenvector. The size of the eigenvector, i.e., the eigenvalue, is equal to the contribution of the variance made by that factor. The first eigenvector, or factor, accounts for a relatively large proportion of information and each succeeding factor accounts for less. In factor analysis it is not necessary to account for the total variance of the correlation matrix, for we know that a certain proportion of the variance is unique or specific to the given set of documents in the experimental situation. We are only interested in the common variance, that portion of the variance that is due to the relationship among the terms and which would continue to be true for all sets of documents. The problem, of course, is to determine the proportion of the total variance which is common. Another way of stating this problem is to pose the question as to when to stop factoring.

Unfortunately, there is no exact answer to this question, but a number of empirical rules have been developed as guides for determining the number of factors to be extracted in any given instance. In essence, these rules are based upon (a) some previously acquired knowledge of the domain, and (b) the amount of variance accounted for by the factors. Since this study is an exploratory one and we had no previous knowledge of the domain, a conservative approach was taken. First we extracted factors (eigenvectors) until the eigenvalues became negative. This resulted in 53 factors which accounted for all the common variance among the terms as indicated by their correlation coefficients and with the highest correlation value in the diagonals. From these we selected the first four factors which accounted for approximately 30 per cent of the variance; the first ten factors, which accounted for about 62 per cent; and the first 18 which accounted for somewhat more than 90 per cent of the variance.

This gave us three sets of factors. We interpreted all three, compared all three, and made the final selection on the basis of which set provides the most meaningful framework for classification.

The first step in interpretation consisted of rotating these factors mathematically. By rotation is meant the process of moving the factor axes and their hyperplanes in order to allow more points to fall on these hyperplanes. It is obvious that the location of axes by which we describe points in space is arbitrary. Take, for example, the measurement of latitude and longitude which is a grid system and one which could be started anywhere. In longitude, we begin at Greenwich, which is of historical importance but has no real significance. Latitude, on the other hand, is fixed with reference to real features--the poles and the equator. The scientist using factor analysis cannot be content with factors that are merely mathematical conveniences. He wants each factor to correspond to some meaningful dimension of his problem space. Therefore, he rotates the factors, i.e., he moves the origin and angle of the grid system so as to obtain a meaningful position. Meaningfulness is determined by the criteria of simple structure of the factor domain. In recent years, a number of analytic methods of rotation have been developed which approximate simple structure. In this research project, the orthogonal varimax solution was used. Three separate rotations were made; one for 4 factors, one for 10, and one for 18 factors.

3. Interpretation of the Factors

Factor rotation is designed to locate the reference axes in a grid system such that each factor can be meaningfully interpreted. The interpretation must be made by the investigator and is based upon his knowledge of the analytic procedures and the subject matter. There is, therefore, a degree of subjectivity in the names

selected for each factor. These names may be regarded as hypotheses about the factor meaning.

In the 4-factor system, each factor was bipolar making a total of eight dimensions. The last factor was poorly differentiated. In general, it was felt that this was not an adequate set of dimensions.

On the other extreme, the 18-factor system resulted in a number of factors based on a close association of a few words, but which could not be interpreted as underlying dimensions of psychological literature. So that, for example, Factor No. 1 was highly loaded on the two terms--girls and boys--and Factor No. 4 pointed out the relationship between college and student. In addition, there seemed to be more than one factor dealing with the same subject matter. The 18-factor system leads to too fine a discrimination for this first exploratory study. After reviewing all three sets of factors, it was felt that the most reasonable interpretations could be made for the 10-factor system.

Insofar as the actual mechanics of interpreting factors are concerned, the first problem that one faces is to decide the value of the factor loadings which are to be considered. The loadings were arranged in order of descending magnitude and therefore on a descending scale of significance. The vast majority of the loadings were below .10 and these were considered as not significant. We tried to restrict our interpretations to factors that had loadings greater than .20, but in some few instances we did look below that point. In those cases where the high loadings were negative, we reflected the factor so as to make them positive. There were no true bipolar factors; although some factors had one or two moderately significant negative loadings. These were not interpreted.

Listed below are the significant loadings for each factor, together with its name and interpretation.

3.1 Interpretation of Each Factor

3.1.1 Factor No. 1 Academic Achievement

Term No.	Word	Factor Loading
33	girls	.74
10	boys	.73
70	school	.30
2	achievement	.20
63	reading	.18

Five terms had significant loadings on this factor. Underlying these five terms is a concept dealing with the achievement of boys and girls in school. The fact that "reading" is on the list while "college" is not, leads one to infer that this factor deals with elementary school achievement.

3.1.2 Factor No. 2. Experimental Psychology--Perception and Learning

Term No.	Word	Factor Loading
52	perception(ual)	.46
40	learning	.36
27	experimental	.29
84	theory	.25
26	evidence	.24
89	visual	.23
30	field	.21

This factor is concerned with experimentation, theories, and evidence, particularly in the areas of perception and learning.

3.1.3 Factor No. 2. Social Psychology and Community Organization

Term No.	Word	Factor Loading
49	organization	.67
17	community	.54
77	structure	.38
90	workers	.22
30	field	.15
4	analysis	.15
73	social	.11
67	role	.10
38	job	.10

The high loadings deal with community structure and organization. Looking further down the list of significant terms we see that this factor is also concerned with social roles, work, field analysis, etc., all of which are involved in social psychology.

3.1.4 Factor No. 4. Studies of College Students

Term No.	Word	Factor Loading
78	student(s)	.71
16	college	.70
34	group(s)	.17
46	mental	.16
28	factor(s)	.15
81	teacher	.14
37	intelligence	.11
55	personality	.10

The terms that define this factor are the words "student" and "college." The factor apparently groups together all studies involving college students whether they deal with teacher training, intelligence, personality, etc. In contrast with Factor No. 1 which has to do exclusively with school achievement of boys and girls, this factor (No. 4) has a more varied subject matter as indeed it should, for the interests of college students are more diverse than academic achievement.

3.1.5 Factor No. 5. School Guidance and Counseling

Term No.	Word	Factor Loading
58	program	.42
23	education(al)	.36
14	child (children)	.33
50	parents	.29
35	guidance	.29
81	teachers	.28
36	information	.27
70	school(s)	.25
20	counseling	.20

It can be seen that while school and education have significant loading on this factor, the emphasis is on parent-teacher-child guidance and counseling.

3.1.6 Factor No. 6. Clinical Psychology and Psychotherapy

Term No.	Word	Factor Loading
87	treatment	.44
59	psychiatric	.35
15	clinical	.32
62	psychotherapy	.22
12	case(s)	.16
69	schizophrenia	.16
85	therapy	.16
34	group(s)	.12
60	psychoanalysis	.12
20	counseling	.11

This factor is clearly concerned with the treatment of mental disorders which is part of clinical psychology.

3.1.7 Factor No. 7. Educational Measurement

Term No.	Word	Factor Loading
2	achievement	.46
1	ability	.36
19	correlation	.35
68	scale	.32
34	group(s)	.22
63	reading	.20
37	intelligence	.20
83	test(s)	.20
70	school(s)	.19

The common element present in the terms with high factor loadings is the measurement of academic ability. The presence of the words "correlation," "scale," and "test" helps define this factor and differentiate it from Factor No. 1 which we labeled "Academic Achievement."

3.1.8 Factor No. 8. General Psychology--Psychology as a Science

Term No.	Word	Factor Loading
73	social	.42
65	research	.32
71	science	.31
61	psychological	.25
75	status	.24

This factor is best labeled "General Psychology," for it deals with the broad general areas of social science and psychological research.

3.1.9 Factor No. 9. Developmental Psychology

Term No.	Word	Factor Loading
25	emotional	.32
22	development	.32
13	cerebral	.23
14	child (children)	.22
84	theory	.19
42	life	.18
47	nature	.18
28	factor(s)	.18

High loadings on "development," "emotional," "cerebral," and "child (children)" help define the discipline with which this factor is concerned as developmental psychology.

3.1.10 Factor No. 10. Therapy: Case Studies

Term No.	Word	Factor Loading
54	personal	.56
12	case(s)	.55
85	therapy	.42
41	level	.21

While there appears to be some similarity between this factor and Factor No. 6 which was labeled "Clinical Psychology and Psychotherapy," the distinctive feature in this factor is the emphasis upon the words "personal" and "case." These terms imply an emphasis on case studies, as part of therapy, and with a rich literature of their own.

4. Discussion

Figure 4 lists, in summary form, the names applied to each of the ten factors, and compares these to the subject classification used by the American Psychological Association. There is a high degree of similarity, as logically there should be, but there are also some very significant differences. Educational psychology, as a classification category, seems to be quite broad and includes four of the ten factors. The factors also cut across categories, so that Factor No. 2, "Experimental Psychology--Perception and Learning," is covered by two subject classifications, namely, "Complex Processes," and "Receptive and Perceptual

Factor Name	Subject Classification--Psychological Index
1. Academic Achievement	A. Educational Psychology--School Learning
2. Experimental Psychology--Perception and Learning	B. Complex Processes and Organizations--Learning and Memory
3. Social Psychology and Community Organization	C. Social Psychology--Social Institutions
4. Studies of College Students	(Educational Psychology)
5. School Guidance and Counseling	D. Educational Psychology--Educational Guidance
6. Clinical Psychology and Psychotherapy	E. Clinical Psychology, Guidance, Counseling--Diagnosis and Evaluation, Treatment Methods
7. Educational Measurement	F. Educational Psychology Educational Measurement
8. General Psychology --Psychology as a Science	G. General Psychology--Theory and Systems, Methods, Professional Problems
9. Developmental Psychology	H. Developmental Psychology
10. Therapy: Case Studies	I. Behavior Deviations
	Unmatched Headings
	J. Physiological Psychology
	K. Personnel Psychology
	L. Industrial and other Applications

Figure 4. Comparison of Factor Names and APA Subject-Heading Index

Processes." On the other hand, there are some classification headings which do not have comparable factors. The most important of these are "Personnel Psychology," and "Industrial and Other Applications." Why this is so is not clear, but one can hypothesize two explanations: either the writings on these topics are not unique, and so can be better subsumed under one or more of the categories, or the sample of documents was neither adequate enough nor representative enough to bring out this factor. In our present state of knowledge, we do not have sufficient information to choose between these alternatives.

This research study has demonstrated a procedure for grouping documents on the basis of their similarity as determined by the words in the text and not on any a priori logical hierarchical scheme. This does not mean that these ten factors are the best categories for grouping psychological literature. Before this can be determined, additional studies of a similar nature would have to be undertaken and the reliability of the factors established. This study was exploratory in nature and was primarily concerned with procedural problems. The obtained results establish the efficacy of the methodology.

5. Implications For Future Research

5.1 Replication

While the usefulness of the method of factor analysis for empirically deriving a classification system has been demonstrated, the reliability of the results needs further testing. It is proposed that an additional sample of data be factor analyzed and the factors compared. Actually, three or four such replications would be necessary before a consistent set of factors emerges.

5.2 Efficiency of the Factor Derived Classification System

Even assuming that the factors are stable, it is necessary to compare the value of a classification scheme based upon factor dimensions with the more traditional subject classification categories. An experimental test of comparative efficiency would have to be devised using as a criterion the information retrieval effectiveness of the two systems.

5.3 Probabilistic Indexing

Maron[2] described the concept of probabilistic indexing. As an extension of his work, it is proposed that the factor loadings of terms be used as probability measures for determining the category to which the document belongs. For example, the term "social" has a factor loading of .11 on Factor No. 2, "Social Psychology and Community Organization," while the same term has a factor loading of .42 on Factor No. 8. "General Psychology." It is suggested that if the word "social" occurs as a key word in an article, that article be given a .4 probability in "General Psychology," and a .1 probability in "Social Psychology." It is hypothesized that this type of probabilistic indexing will result in increased information retrieval effectiveness. This is an area which warrants serious investigation.

6. Summary And Conclusions

The purpose of this study was to describe and utilize an empirically based mathematical procedure for classifying documents. An experimental library of 618 psychological abstracts was selected and coded into machine language for computer processing. The words in these documents were ordered according to frequency of occurrence, and 90 index terms selected from among those words which occurred 20 or more times. Each term was correlated with every other term, and the resulting correlation matrix factor analyzed. Ten factors were extracted and rotated for meaning. These were then interpreted and the results compared with, and shown to be similar to, the existing subject classification system employed by the American Psychological Association. The study demonstrates the feasibility of using factor analysis as a method for determining the basic dimensions of a classification system

References

(1) Harman, H. H., Modern Factor Analysis, University of Chicago Press, Chicago, Illinois, 1947.

(2) Maron, M. E., Kuhns, J. L., and Ray, L. C., "Probabilistic Indexing," TM No. 3, Ramo-Wooldridge, Los Angeles, California, June 1959.

(3) Olney, J. C., "FEAT, An Inventory Program for Information Retrieval," TM-4018, System Development Corporation, Santa Monica, California, July 1960.

(4) Thurstone, L. L., Multiple-Factor Analysis, University of Chicago Press, Chicago, Illinois, 1947.

THE STORAGE AND RETRIEVAL OF PHYSIOLOGICAL AND MEDICAL DATA IN A MODERN HOSPITAL

Paul C. Tiffany

Aerospace Corporation

El Segundo, California

(Formerly with System Development Corporation, Project Medic)

Abstract

As an introduction this paper considers some of the problems in data handling in a modern hospital. Next, the needs of the users of the data are considered. The principle area of interest is indicated as that hospital function dealing with the storage and retrieval of the clinical record after the patient's hospitalization. The types of terms used in medicine are examined and two currently used schemes for the indexing of diseases are described. Next a storage and retrieval system that permits the medical researcher to examine or browse through clinical records or abstracts of the records is described. The paper concludes with observations on the need for applied research and system development to acquire pilot systems for the storage and retrieval of physiological data.

Introduction

A system study[1] describing a General Medical and Surgical Hospital of the Veterans Administration indicated that faster and higher-capacity storage and retrieval systems--possibly computer-based--were needed to manage technical and clinical data.

Problems of Data Handling in the Hospital

Clinical records inherently defy logical control. In addition to sheer volume, the following are typical complications:

1. "Summaries" of hospital stays often contain lengthy "soft" literary descriptions that are difficult to abstract and "harden."

2. Medical and surgical entries in "Progress Notes," often lack the parameters necessary to describe many complications.

3. Concrete detail may not be available for the physician's or nurse's first encounter with the unusual situation.

4. Soft literary comments are considered necessary to supplement hard parametric values.

5. Only detailed research for an extended period over the range of the many diseases and almost infinite number of disease combinations will yield a pattern of knowledge such as employed by an experienced doctor when he examines a clinical record.

Two Assumptions Regarding the Storage and Retrieval of Clinical Data

1. If a general solution exists for storage and retrieval problems, this general solution will be found most readily by applying the hard practical techniques used to solve storage and retrieval problems for particular systems.

General techniques, common characteristics, laws of behavior, and formulae will emerge from the solutions of the problems of different systems. In some respects, each system will be unique, as determined by the nature of the data to be handled and the needs of the user. The common problems will be solved by deduction; esoteric and inductive reasoning may only serve to fog the issues.

2. If existing concepts on automatic abstracting, indexing, pattern recognition, parsing of sentence structure, translation from physiological language to a mathematical logical language, teaching machines, heuristic programming, etc., are collected and ordered, a storage and retrieval system will then exist for the management of data in the clinical record.

It is possible to assemble such a system and it should be attempted soon. This paper will suggest a broad configuration omitting details that only can be acquired through extensive research. Hopefully, this description will prompt other researchers to collect, refine, and integrate existing concepts, and from this, evolve an effective storage and retrieval system for medical and physiological data.

Consideration of the Needs of the User of Physiological Data

"Real Time" Data Storage in the Clinical Chart

The capability of storing data in "real time" as it is received asynchronously from the varied hospital services (wards, clinics, laboratories, x-ray, conferences, operating rooms, etc.) and then retrieving part or all of it when requested by users, clearly, will aid in satisfactorily caring for patients. An experiment is in progress, conducted by a joint team of researchers from the Veterans Administration and the System Development Corporation of Santa Monica, to demonstrate this capability in a simulated hospital ward environment, using an IBM 1401 with RAMAC. The project will handle those forms found in most hospital clinical records as suggested by the Joint Commission on Accreditation of Hospitals.

Storage of the Clinical Record After Hospitalization

Access to records stored after the patient's discharge is a major need (and is the main interest of this paper). According to Huffman[2], the Medical Record Library (MRL) function is to provide medical records containing sufficient data to verify the diagnosis and treatment and document the results for all patients treated in the hospital. These records should be readily available for:

1. The patient, as busy physicians cannot remember the details of all their cases; also, if the patient is readmitted, accurate records save time and effort.

2. The hospital, to review the competency of their staff and the results of treatment, and to compile statistics; also, for medico-legal purposes.

3. The physician, to review his cases and consultations, and to treat patients who do not recall previous illnesses and hospitalizations; also, for medico-legal purposes.

4. Medical research and teaching, since many observations can only be made clinically.

The Storage and Retrieval of Documents Written in the Medical and Surgical Field

The field of medical documentation is burdened by the problem of how to store documents so that they can be recalled readily. To solve the problem concisely, the computer must be "taught" the language of "physiology." Hans Selye and Miklos Nadasdi in their book on a "Symbolic Shorthand System" discuss the possible adaption of their system to electronic machines, and state, "Perhaps the mechanical memories of the future will not take this particular form, but whatever we use, punch-card systems, electronic machines, or any other physical aid to classification, the first problem is to create a simple, mnemonic language, which can easily be fed into such machines."[3]

Although it was originally designed for processing documents on stress and endocrinology, the "Symbolic Shorthand" technique appears to be directly adaptable to a library system for all types of medical documents. Consideration is given to the possibilities of generating an abstract of the clinical record, and of generating an index based on the abstract. This can be compared to the techniques used at Western Reserve for document storage.

Some Concepts of the Size of the Medical and Physiological Language

A computer system that can communicate and handle storage and retrieval problems using medical and physiological data must be programmed to recognize the terms and words of medicine and physiology.

Medicine is a science which is concerned with the abilities of recognizing, preventing, curing, and alleviating diseases and their causes. Surgery is considered a medical science that manually corrects, aids deformities and defects, and repairs injuries and is, therefore, a sub-category of this definition of medicine.

Physiology is a science dealing with the functions of organs and physical parts during life. By way of contrast, "physiology" is dynamic, and "anatomy," static.

A cursory examination reveals the large size of the task of developing descriptor terms for the data encompassed by these sciences. Skeptics may weigh "Stedman's Medical Dictionary"[4] or merely leaf through its 1656 pages of definitions.

A Smaller Manageable Area for Examination

The immense field of medical terminology may be bounded somewhat by selecting a subclassification of the entire science, specifically, endocrinology. Endocrinology deals with the glands and their secretions. It is further concerned with the movement of those glandular secretions that pass directly into the blood or lymph. In addition, the science is interested in the activation of cells of other parts of the body by organic substances called autacoids. An example of an autacoid is the hormone thyroxine, produced by the thyroid gland, which considerably influences growth.

Why was endocrinology selected? After attending conferences, visiting wards, examining the data in the charts of the patients, and discussing the question with various doctors, it appears that endocrinology involves a larger volume of objective data than other services and would benefit most directly from computer handling. Much work has already been accomplished to firm these data and,therefore, facilitate this study.

To reduce the problem to a level suitable for this paper, one small area of the entire subject of endocrinology was selected, namely, the Hypophysis. A study of this gland, though quite complex, is easily bounded, and appears to afford hard and finite data. The following material is presented to indicate how large the terminology and indexing problem is for a single gland.

The Hypophysis is the pituitary body. It consists of two lobes and is located at the base of the brain in a depression of the sphenoid (wedge shaped) bone called the pituitary fossa. The smaller posterior lobe is developed from and attached to the brain by the infundibulum, the name given to the stalk attachment of the hypophysis. The larger anterior lobe is developed from the buccal cavity, the cavity of the mouth. (See Figure 1.)

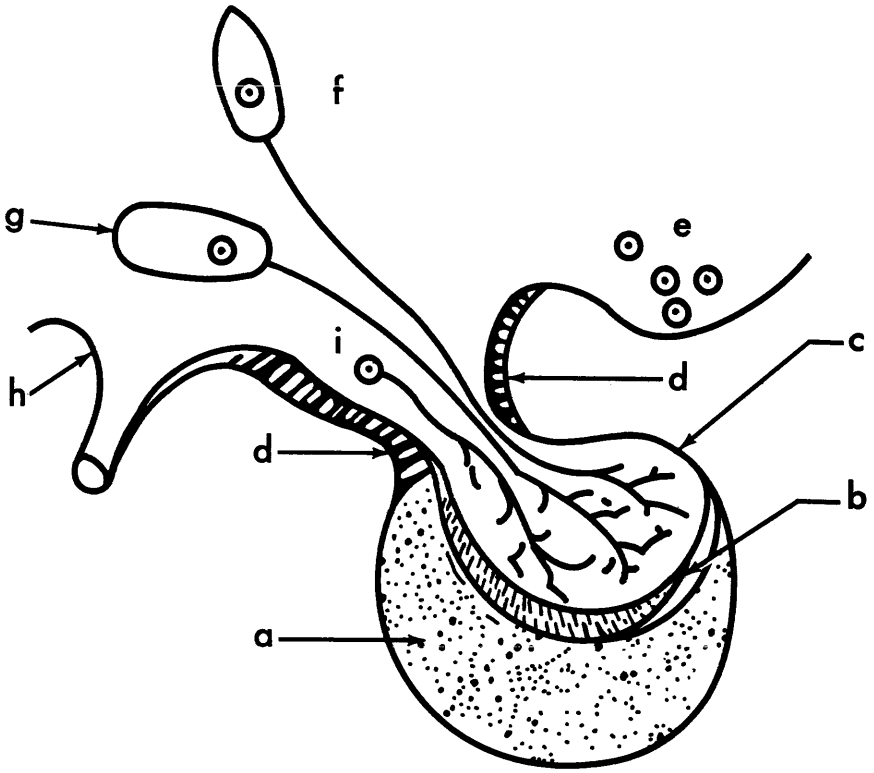


DIAGRAM OF THE HYPOPHYSIS

a. Anterior pituitary. b. Pars intermedia. c. Neural lobe. d. Pars tuberalis. e. Mammillary body. f and g. Hypothalamic nuclei. h. Optic chiasma. i. Tuber cinereum.

FIGURE 1

The foregoing serves to introduce the wide range of data used to describe this gland and its functions. Hans Selye in his "Symbolic Shorthand System," conceived of the following eight major categories.

1. Actions upon the hypophysis. The following types of actions are viewed as interventions upon the gland and its system:

a) Extirpation, complete removal of the gland

b) Partial extirpation

c) Exposure to ionization or x-rays

d) Activation, excitation, or depression of gland by drugs, medication, etc.

2. Morphology. The features, form, and structure of the organism are covered by the following two subjects:

a) Anatomy--the structure and parts of the gland.

b) Histology--the minute structure of the tissues. For example, the posterior lobe is composed of neuroglia-like cells (like the brain cells). These spindle-shaped cells are known as pituicytes. These cells are filled with a clear homogeneous material known as hyaline bodies,also Hering's bodies. This smaller lobe also consists of connective tissue, blood vessels, and numerous nerve fibers. The larger anterior lobe consists of cords of cells which are separated by very small channels, microscopic vessels. There are three types of cells in this lobe, chromophobes which do not stain easily, acidophils which stain with acid dyes, and basolphils which stain with basic dyes.

3. Chemistry. The general chemistry of the structure of the gland is divided into two sections.

a) Organic compounds of the gland

b) Chemistry of the hormones, including such terms as:

- Trade names
- Extraction
- Synthesis
- Chemical activation and inactivation
- Preservation, etc.

4. Humoral Physiology. This classification deals with the functions of the blood and lymph ducts in relation to functions of the gland:

a) Mechanism of secretion, including pathways and flow

b) Mechanism of action

c) Metabolism of the hormones, their fate in the body

d) Utilization of the hormones for the performance of their functions in the body.

5. Pharmacology. This deals with the drugs used in treating the hypophysis, and includes such items as:

a) Application

b) Dosage, effect data between dosage and results

c) Relationships between chemical and pharmalogic effects

d) Sensitization and desensitization

e) Pharmacological resistance

f) Immunity and allergy

g) Toxic effects

6. Hypophyseal Hormones. This subject deals with the secretions of the gland:

a) Crude hypophyseal extracts

b) Crude anterior lobe extracts

c) Adrenocortocotrophic hormone

d) Somotrophic hormone

e) Gonadotrophic hormone in general

- Follicle stimulating hormone
- Luteinizing hormone
- Luteotrophic hormone

f) Thyrotrophic hormone, TSH

g) Crude middle lobe extracts

h) Melanophorotrophic hormone, intermedin

i) Other middle lobe hormones

j) Posterior lobe extracts

k) Vasopressin, antidiuretic hormone

l) Oxytocin

m) Other posterior lobe hormones.

More terminology must be introduced to understand the functions of some of these pituitary

hormones. The smaller posterior lobe furnishes a secretion that stimulates contraction of unstriped muscular tissue; raises blood pressure; and is oxytocic (hastens childbirth), antidiuretic (decreases the secretion and discharge of urine), and melanophore (control of skin pigmentation) dispersing. The larger anterior lobe furnishes a gonadotrophic hormone (affecting the sex glands), thyrotrophic hormone (affecting the development and activity of the thyroid), and an adrenotrophic hormone (abbreviated as ACTH, influencing the growth and activity of the adrenal cortex); prolactin (stimulating the secretion of milk); and generally contributes to growth promotion. Ketogenic (producing acetone bodies in the blood), diabetogenic (causing diabetes), and insulin-antagonizing effects are obtained by injection of the extracts of this anterior lobe.

7. Pathology of Hypophysis. This classification deals with the condition of the gland or fluid produced by disease, and considers such items as:

a) Aplasia--the congenital absence of an organ.

b) Hyperplasia--the increase in tissue elements such that the bulk of the organ is increased, excluding tumor formations.

c) Hypoplasia--the defective development of an organ.

d) Accessory glands.

e) Hemorrhages and other vascular lesions.

f) Any other malformations.

g) Inflamations.

h) Tumors.

8. Diseases of the Hypophysis. Diseases of this gland are defined as any malady, ailment, or maladjustment that causes a condition in which bodily health is impaired. Classifications include:

a) Diseases with underactive pituitary, either lobe.

b) Diseases with overactive pituitary, either lobe.

c) Tumors of the hypophysis.

d) Injuries and operations.

e) Others.

As an examination of the broad classifications of diseases used in a system for the storage and retrieval of documents dealing with studies of stress and endocrinology, the following diseases of the hypophysis are listed:

- Hypopituitarism in general
- Simmonds disease
- Sheehan's syndrome
- Hypophoseal dwarfism
- Hypophoseal infantilism
- Hyperpituitarism in general
- Acromegaly
- Cushing's syndrome
- Pseudo Cushing's syndrome
- Berardinelli's syndrome
- Diabetes insipidus
- Antidiabetes insipidus
- Inflammation of the hypophysis
- Non-endocrine hypophoseal tumors
- Cranio-pharyngioma

A Look at Two Coding Schemes that are Currently Used to Index Hospital Records

Two well developed schemes exist for classifying diseases and for assigning number codes to identify the diseases. The classification and coding are described for the pituitary gland. The first system is that listed in the ICDA, "International Classification of Diseases, Adopted for Indexing of Hospital Records."[5] The second is the SN, the "Standard Nomenclature of Diseases and Operations,"[6] published for the American Medical Association.

Description of Diseases of the Hypophysis as Listed in the ICDA

The International Classification creates a "separate title or code number in the classification only when a separation is warranted because of the frequency of its occurrence."[5] This system is based therefore on the frequency with which diseases occur and is a statistical model of public health information. If the user of this index requested data that occurred most frequently, then this system might satisfy his needs. The medical research worker, however, is not always interested in data primarily because of its frequency of occurrence (for example the common cold).

The ICDA states, "For hospital indexing purposes, the most efficient classification system is one which permits the location of a maximum number of pertinent records with the review of the least number of records." ICDA adherence to its own definition of efficiency is covered in a study reported in the Journal of the American Association

of Medical Librarians,[7] which contrasts this indexing system with the SN.

In Appendix 1 some disease symptoms and terms are described to clarify the classifications; however, these descriptions do not appear in the International Classification. The major category dealing with diseases of the endocrine system is Number III: Allergic, Endocrine System, Metabolic and Nutritional Diseases. The next lower level of classification is Diseases of Other Endocrine Glands (270-277). Under sub-class 272 are most of the items that are of interest to this paper. For a more complete treatment of indexing in the ICDA refer to Appendix 1.

Description of Diseases of the Hypophysis as Listed in the Standard Nomenclature

The technique of indexing in the SN is more highly ordered than in the ICDA and would appear to lend itself more readily to automatic programming.

In general, the plan of the SN is to code the disease as a two part index. The first part indicates topographically the system of the body that is infected; the second indicates the etiology, the cause of the disease. Refer to Appendix 2 for an example of the SN applied to diseases and operations of the hypophysis gland.

Appendices 1 and 2 suggest the complicated thesaurus that must be considered for any data retrieval system. A set of uniterms as advocated by Dr. Mortimer Taube, or descriptors as promoted by Calvin Mooers, would be extensive even for so limited an area of the human body as the pituitary gland. An extrapolation to the complete body and its systems shows the enormity of the complete storage and retrieval problem. Perhaps, retrieval systems should be specialized and deal only with individual medical areas in the beginning.

A "Storage and Retrieval" System Description

The system illustrated in Figure 2 was evolved by synthesizing a number of independent concepts. Many of these concepts will require considerable development and study before firm operational techniques may be recommended; however, rudimentary and perhaps even somewhat crude methods are available presently to set up a prototype system.

"In the majority of cases, the problem is not that our existing technology is insufficiently advanced to provide the mechanisms we need for effective handling. In short, the practical problem we face in this area is the proper selection and application of the systems and equipment now available to us, and to do this properly we must understand clearly the elements of our own individual working environment."[8]

The purpose of this proposed system is to provide high-speed storage and retrieval for the user of medical data. The part of the system that abstracts the clinical record for indexing and storage also abstracts the users' request. The system compares the request with stored information and then displays the matched abstract, indirectly asking if it is what the user desired. The user then either accepts what is offered or modifies his request. This inter-communication between the system and the user permits the retrieval of a single abstract, several abstracts, a solitary clinical record, selected parts of a record, selected parts of several clinical records, or several complete clinical records.

The presentation that follows covers three areas:

1. The Movement of Data in Processing of Clinical Charts for Storage.

2. The Movement of Data in Retrieving Abstracts and Charts.

3. Communications of the User with the System.

The Movement of Data in Processing of Clinical Charts for Storage

This concept is illustrated in Figures 2a and 2b. Figure 2a is the legend for the flow diagram; the flow of the data in processing the clinical charts as they come from the ward is shown in Figure 2b. The clinical charts (input data) are shown entering the system at the base of the diagram.

Read-in of Clinical Charts of Discharged Patients to be Abstracted. Clinical charts are entered into the system to be abstracted (line A of Figure 2a) by the Automatic Abstractor and classified by the Index Classifier. In the early stages of system development the process would be manual; however, efforts should be extended to computerize this function. The efforts of Zellig S.Harris[9] and others to parse the English sentence indicate a possibility of selecting key words in the doctor's summary of the clinical record and automatically abstracting them. The abstracts would be indexed according to the primary disease as coded by the standard nomenclature with next level storage indicated by secondary illnesses. Further and deeper indexing should now be made by abstracting the clinical chart. The resulting abstract used as an index will attain a level of storage and retrieval far beyond the capabilities of modern medical record libraries, particularly for research purposes. The system will also allow for browsing by the user. An existing manual system is the Symbolic Shorthand System of Hans Selye.[3]

Read-in of Clinical Charts to be Reduced. The read-in of clinical records of discharged patients for reduction to micro-image is handled at the same time that the clinical chart is being abstracted and indexed. Line B indicates the movement of the chart to the micro-image producer. Several devices exist for this function including

Magna card, Minicard, File search, "Walnut," Recordak, and other micro recording and display systems.

Storage of Abstract Classification and Index for Retrieval of Abstract. Line C indicates the movement of the coded abstract to the Request and Abstract Index Matcher. In a study made for the National Science Foundation, and in a course on Information Storage and Retrieval,[10] Dr. Robert M. Hayes presented a mathematical model in which an association matrix was used to indicate the relevancy between documents.

"Of particular mathematical interest is the fact that powers of the association matrix provide a method for determination of its eigenvalues and thus represent an approximation to the methods of eigen-value analysis and factor analysis."

This technique suggests that an abstracted request can be treated as a vector and multiplied by the association matrix some "n" times to attain an approach to an "eigen" vector, thereby processing the original request to attain a close fit to the stored file of documents. This matching technique affords the requester considerable freedom in forming his request, using key words as a basis. A researcher could draw as many relevant cases from the stored data as he desired in either direction in the file from the closest match.

Storage of Abstract in Appropriate Classification as Determined by Automatic Abstraction. Line C leads to a storage of abstracts. The first level of classification or index is made by use of the SN number of the primary diagnosis, and then by the SN number of the secondary diagnosis. The next level would be stored as the coded abstract itself, possibly as a dynamic file of "agents" acting on "targets". Associated with each filed abstract would be a stored serial index to retrieve in part or completely the clinical charts of interest.

Transfer to Storage of Complete Charts. Line E indicates the movement of the micro-miniaturized data into storage. This storage is arranged as a serial file, with the most recent information most accessible. The file would be culled constantly, and charts grouped by admission dates. A serial number assigned to a patient would be preserved for that patient, permitting grouping all his hospital stays under his most recent admission.

Movement of Data in Retrieving Abstracts and Charts

The following flow of data is presented to indicate how the system retrieves data for the user.

Read-in of Text of Request. The input of the user is indicated by line 1 (Figure 2b), and represents "n" possible iterations of user requests as he modifies them to retrieve the clinical data he wants.

Request Index as Generated. Line 2 indicates the movement of the generated request index. This index is to be developed by the same abstracting technique (key words) as used on the clinical charts.

"The inquirer's document would then be coded in exactly the same manner as the documents of the collection have been encoded, and the resulting notional abstract would be set up as a question pattern in a data-processing machine of adequate functional ability."[11]

As described previously, a correlation matrix could be used to obtain the chart with the closest match to the generated request.

Temporary Storage of Request Index. Line 3 shows the movement of the index with the closest match into a storage register that will hold the data, pending an examination of the matched abstract by the user of the system.

Abstract Index with Closest Match to Request Index. The abstract is now moved via line 4 to a device that will print out, or display by some other technique (such as a cathode ray tube), the matching abstract, for the user's examination. The diagram indicates "n" possible feedbacks to the user/researcher until he is satisfied that the abstract contains the material that he desires. This feedback technique is similar to automated teaching practices and permits communication between the system and the user.

Activation Switch Indicating Agreement of Requestor with Match. Line 5 carries the input of the user indicating his satisfaction with the matched abstract. By switch insertion the user will modify his request, e.g., requests for the quantity of abstracts desired. Related abstracts will be stored near the matched abstract.

Index Used to Activate Output of Abstracts. The indication of acceptance of the match will move the index, illustrated on line 6, to activate the output of the abstracts from storage as per the user's request.

Output of Abstracts. The abstracts requested will now be produced, probably via on-line printer, for the user, as indicated by line 7. Each abstract, as described previously, will carry a serial index as part of the data to enable direct retrieval of clinical charts, if desired.

Entry of Request Using Hospital Serial Number. At this point, by examining the abstracts the user will determine which clinical records or forms or parts of records he desires. For example, the researcher may only be interested in the laboratory reports, or the medications used for a certain classification of diseases. Selectively using the serial indices and guides to specific parts of the charts the user inserts a request (as illustrated by line 8).

COMMUNICATIONS OF THE USER WITH THE SYSTEM

(I) Written requests for retrieval of clinical data stored in discharged patients' charts.

(II) Image of abstract data with closest match to request for examination by requestor.

(III) Activation switches to retrieve abstracts in classification which agrees with request.

(IV) Printout of abstracts of clinical records in classification that matches request.

(V) Keyboard for insertion of serial number to request whole or that part of clinical record which is desired. Insertion will also indicate method of output.

(VI) Display output of selected parts of data as indicated in retrieval request.

(VII) Hard copy output of selected clinical data on printer.

MOVEMENT OF DATA IN PROCESSING OF CLINICAL CHARTS FOR STORAGE

(A) Read-in of clinical records of discharged patients to be abstracted.

(B) Read-in of clinical records of discharged patients to be reduced to micro images.

(C) Storage of abstract classification and index for retrieval of abstract.

(D) Storage of abstract in appropriate classification as determined by automatic abstraction.

(E) Transfer to storage of complete charts. Charts are now stored in agreement with hospital serial index. Data are now micro-miniaturized.

MOVEMENT OF DATA IN RETRIEVING ABSTRACTS AND CHARTS

(1) Read-in of text of request describing the type and number of clinical charts desired to be retrieved.

(2) Request index as generated by automatic abstractor.

(3) Temporary storage of index generated for this request.

(4) Abstract index with closest match to request index.

(5) Activation switch indicating agreement of requestor with match.

(6) Index used to activate output of abstracts in this classification for examination.

(7) Output of abstracts with hospital serial number to be used to retrieve charts as desired.

(8) Entry of request using hospital serial number for retrieval of complete records or parts of charts as desired.

(9) Output of charts or selected parts as requested.

FIGURE 2a.
LEGEND FOR INFORMATION STORAGE AND
RETRIEVAL SYSTEM SHOWN IN FIGURE 2b

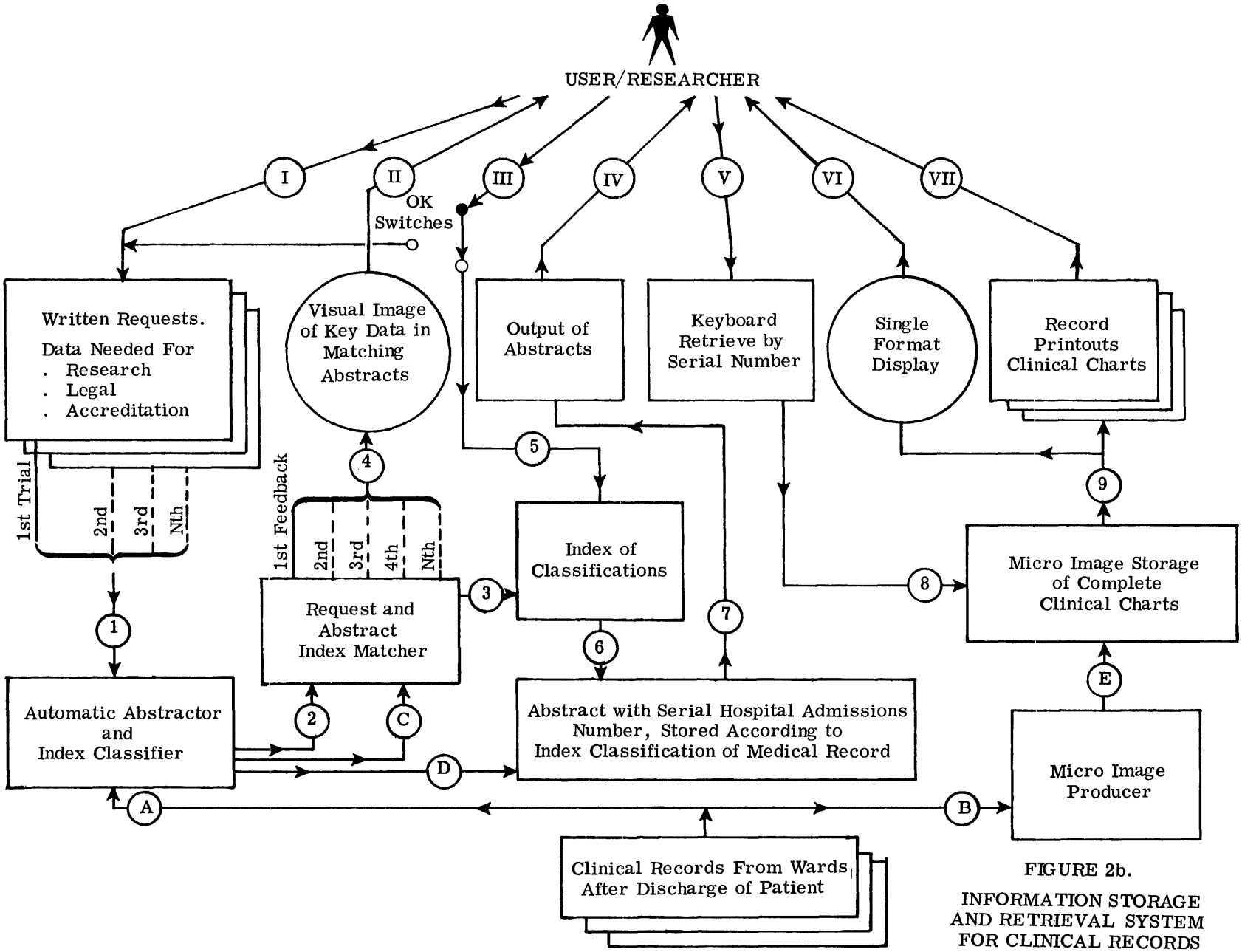


FIGURE 2b.

INFORMATION STORAGE AND RETRIEVAL SYSTEM FOR CLINICAL RECORDS

Output. The output as indicated by line 9 can be of two types: a display of individual bits of data for visual examination, such as a display of the doctor's orders which had been given for a patient; or a printout of partial or complete clinical records that had been stored as micro images.

Communicating with the System

This section briefly describes the inputs (requests) from the user and outputs (retrieved data) to the user.

Written Requests. For retrieval of the data stored in the clinical charts of discharged patients, the user will write (typewrite) an input (I) to the system using an SN number and a written description of his request. The request may be dictated by various requirements such as the need of data to defend the hospital against malpractice suits, the retrieval of records for statistics to support research endeavors, the gathering of figures for hospital management to compile accreditation reports, etc.

Image of Abstract Data with Closest Match. This output (II) from the system makes it possible for the researcher to see the abstract that is the closest match to his request. If the abstract does not meet the needs of the requestor, he can reframe his request and the system will display a new matched abstract.

Activation Switches. This input (III) provides the requestor with the capability of retrieving as many abstracts as he may wish to look at. These abstracts are stored in the immediate environment of the abstract that best fit the user's request. This satisfies two needs of the user: the desire to browse through the file, and the need to retrieve a quantity of abstracts to obtain additional material to support research and teaching projects. By switch insertion the user will indicate to the system how much material he wishes to examine, and what type of output (soft visual-display or hard printed-copy) he desires.

Printout of Abstracts. The retrieval system will supply the output (IV) in the form of a printout of the abstracts the user requested. The storage location of the abstracts has been supplied by the matching techniques and the storage plan. No provision would be made to exceed the boundaries of the SN classification in supplying the quantity of abstracts desired.

Keyboard. After examining the abstracts and deciding that the complete clinical charts or parts of the charts are necessary, the requestor will key into the system the serial number in the abstract, the format number for the part or complete clinical record, and the method of output desired. This input (V) directly addresses the micro-image storage system and provides by switch-insertion the serial index to retrieve the required data.

Display Output. The system can display a visual-type image of individual pages of the clinical chart (Output VI). This capability will enable the research worker to examine a part of a clinical chart directly if the data in the abstract were insufficiently detailed.

Hard Copy Output. The printed output (VII) supplies the user with a "hard copy" of the retrieved clinical chart, or section of the chart exactly as it was stored when transmitted from the ward.

Conclusion

Economic considerations of this system configuration appear quite extensive; however, this configuration is suggested for the purpose of demonstrating and developing the feasibility of such systems. Simplification of functions and equipment should lead to practical operational concepts.

The level of retrieval for research, the rapidly expanding field of medical knowledge, the mounting heap of clinical charts in hospital storage vaults, and the increasing demands upon the hospitals because of our rapidly growing population indicate a need for increased research and development on storage and retrieval systems.

This paper concludes with a series of observations drawn from the "Proceedings of the San Jose Conference on Health Information Retrieval, 1959."[12]

"1. Health and medical information is the essential requisite for effective care of individual patients and one of the essential needs for sound research but its volume and complexity has become so great that it can no longer be effectively stored, codified, retrieved and transmitted by conventional methods.

"2. The problems and needs of health information retrieval are of critical importance to all professional disciplines related to health, to all agencies concerned with health, rehabilitation, welfare and correction. These problems and needs are nation-wide in scope.

"3. The technology of information storage and retrieval is already in a state of development where it may be applied to this problem.

"4. To accomplish this objective, intensive studies are necessary in order to define precisely the information retrieval requirements of health and medical practice and research.

"5. The benefits to the individual and social health of man to be derived by successfully doing so are considered to be very great and it is therefore recommended by the conference that the development of definitive studies directed toward solution of this problem be encouraged by every means possible."

Appendix 1

The ICDA Index

The following listing is in the format of the ICDA. Comments and descriptions not a part of the ICDA are placed in parentheses.

272. DISEASES OF PITUITARY GLAND

272.0 Anterior pituitary hyperfunction

(The diseases in this general group are due to over-activity of the anterior lobe of the hypophysis.)

Includes: Acromegaly (also known as Marie's disease. This disease has the following symptons: enlargement of head and face, hands and feet, and thorax. This growth is due to excessive secretion of the acidophil cells of the anterior lobe.)

hyperpituitarism (A condition, which is marked by acromegaly as described above and hypertrichosis, which is the abnormal growth of hair such as on the face of women or on the back in both men and women.)

hypophyseal gigantism

premature puberty

(At this point, the ICDA, to prevent hospital personnel from making mistakes when using this coding index, lists three diseases that should not be listed under 272.0.)

Excludes: Basiophilic adenoma (277.0)

Cushing's syndrome (277.0)

pituitary basophilism (277.0)

(To show how much is excluded, a description of Cushing's syndrome in Stedman's Medical Dictionary lists the following group of symptoms for these diseases: abnormal carbohydrate metabolism; obesity of head, neck, and trunk; decalcification of the bones; and increased blood pressure. It has been suggested that the syndrome points to the presence of a basophil adenoma of the hypophysis cerebri, pituitary basophilia. This describes a pituitary tumor made up of basophil cells.

Now the listing proceeds with the next classification of pituitary diseases.)

272.1 Anterior pituitary hypofunction

(Hypofunction is the opposite of hyperfunction. Instead of overactive the anterior lobe is underactive.)

Includes: adiposogenital distrophy (This is a disease of the type of Froehlich's syndrome. The condition is caused by a deficiency of the anterior lobe and often combined with a deficiency of the posterior lobe. Symptoms of the disease are: increase in fat, loss of sexual power, atrophy of the external genitals, and loss of hair.)

dwarfism

(of several types)

-hypophyseal

Loraine type

(Idiopathic infantilism)

pituitary

Froehlich's syndrome

hypopituitarism

(juvenile)

pituitary

cachexia

dwarfism

infantilism

obesity

Simond's disease

(This disease is marked by premature aging, rapid atrophy of the genital organs with repression of secondary sexual characteristics, and loss of hair. Disease is usually noted in young women due to atrophy or diminished functioning of hypophysis.)

(Again to prevent errors the ICDA advises against the inclusion of the following diseases.)

Excludes: dwarfism:

achondroplastic (758.0)

(occasioned before birth)

congenital (277.0)

NOS (277.1)

pancreatic (587.9)

dwarfism and infantilism (277.1)

infantilism NOS (277.9)

(Then the ICDA goes on to the next listing.)

272.2 Chromophobe adenoma, pituitary

(A tumor in the chromophobe cells of the anterior lobe.)

272.9 Other and Unspecified

Includes: abcess

adenoma NOS (tumor)

cyst

diabetes insipidus

(A disease characterized by the excretion of large amounts of pale urine of low specific gravity, and caused by a deficiency in the antidiuretic action of the gland.)

Dyspituitarism

(A complex of symptoms due to either excessive or deficient pituitary action.)

eosinophilic adenoma

(A tumor of the chromaphil cells of the anterior lobe associated with gigantism and acromegaly.)

(This concludes diseases of the pituitary in the ICDA, however, to be complete,classification 277 must also be considered.)

277.0 Pituitary basophilism

Includes: basophilic adenoma

Cushing's syndrome

(also under major classification II dealing with neoplasms the ICDA lists data of importance.)

195. MALIGNANT NEOPLASM OF OTHER ENDOCRINE GLANDS

195.3 Pituitary gland and craniopharyngeal duct (pouch)

Includes: malignant craniopharyngioma,

craniobuccal (Rathke's) pouch

hypophysis cerebri

224. BENIGN NEOPLASM OF ENDOCRINE GLANDS

224.2 Pituitary gland and craniopharyngeal duct (pouch)

Includes: tumor unspecified whether benign or malignant

(Again in the section on OPERATIONS AND TREATMENTS the following classification is found.)

2. OPERATIONS ON ENDOCRINE SYSTEM (08-09)

09 OPERATIONS ON OTHER ENDOCRINE GLANDS

09.2 Hypophysectomy (pituitectomy)

Includes: complete partial

excision of pituitary transfrontal

neoplasm transphenoidal

09.3 Other Operations on pituitary gland

Includes: aspiration or drainage biopsy

craniobuccal pouch incision and drainage

Rathke's pouch

Appendix 2

The SN Index

The material below will illustrate the coding of the SN by showing how it is applied to coding diseases and operations of the hypophysis. Figure 3 shows the word structure of this index. The broadest classifications are made topographically by systems of the body. System 8 is the class dealing with the Endocrine system. The Endocrine system classification is divided into 9 categories, each coded by a two-digit number as follows:

80 Endocrine System

81 Thyroid Gland

82 Parathyroid Glands

83 Thymus Gland

84 Pituitary Gland

85 Pineal Gland

86 Adrenal Glands

87 Pancreas

88 Gonads

89 Carotid Gland

Classification 84, the Pituitary Gland, has 6 subdivisions coded as follows:

840 Pituitary gland, generally

841 Anterior lobe

842 Posterior lobe

843 Pars intermedia

844 Pituitary stalk

845 Craniobuccal (Rathke's) pouch

Subsequent digits, separated from the topographical classification by a dash, define the etiological classification (the causes of the disease). The first digit after the dash indicates the cause. For example, in the case of the pituitary gland, the causes are listed as follows:

- 0 Diseases Due to Prenatal Influence

- 1 Diseases Due to Infection with Lower Organism

- 4 Diseases Due to Trauma or Physical Agent

-50 Diseases Due to Circulatory Disturbance

- 6 Diseases Due to or Consisting of Static Mechanical Abnormality

- 7 Diseases Due to Disorder of Metabolism, Growth or Nutrition

- 8 New Growths

- 9 Diseases Due to Unknown or Uncertain Cause with the Structural Reaction Manifest

- X Diseases Due to Unknown or Uncertain Cause with the Functional Reaction Alone Manifest

- Y Diseases Due to Cause not Determinable in the Particular Case.

As this paper is more interested in the plan and logic of the classification than the entire list of diseases, all diseases are not presented; however, as most troubles of the pituitary gland are due to over and under activity an examination of the scheme of - 7, Disorder of Metabolism, is presented.

- 77 and - 78 Disturbances of Specific Endocrine Organs or Hormones

-776 Pituitary gland, anterior lobe, increased or perverted function

-777 Pituitary gland, anterior lobe, decreased function

-778 Pituitary gland, posterior lobe, increased or perverted function

-779 Pituitary gland, posterior lobe, decreased function.

By this scheme, the classification of diseases due to metabolic disturbance is as follows:

- 7 Diseases due to Disorder of Metabolism, Growth or Nutrition

841 - 776 Anterior pituitary hyperfunction

841 - 7761 Hypophyseal gigantism

841 - 7762 Acromegaly

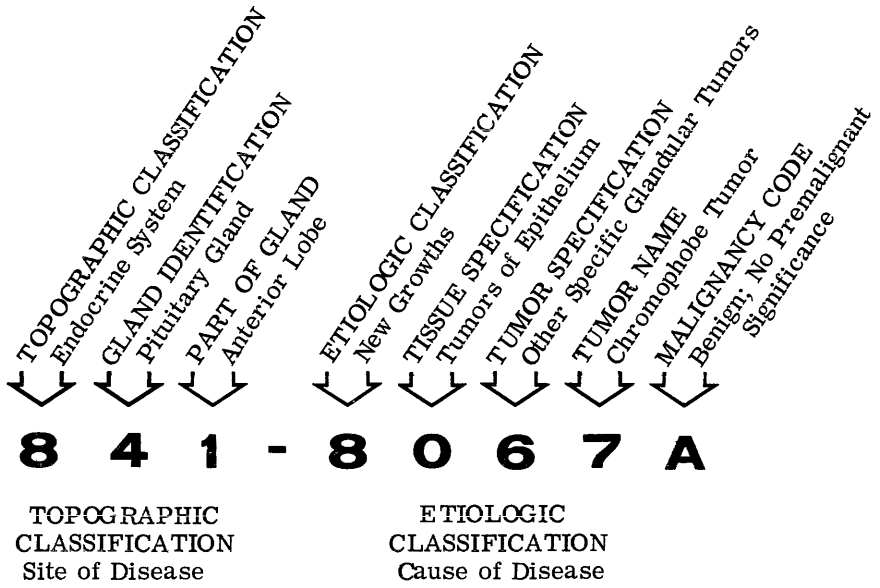
841 - 7763 Pituitary basophilism

841 - 7764 Premature puberty

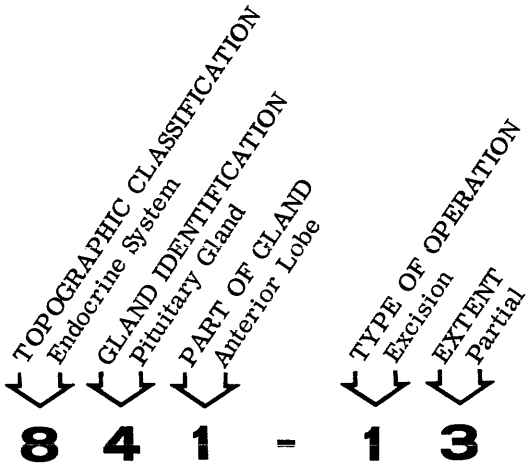
841 - 777 Anterior pituitary hypofunction

841 - 7771 Pituitary dwarfism

841 - 7772 Juvenile Hypopituitarism



"CHROMOPHOBE CARCINOMA OF PITUITARY GLAND"
For Diseases of Hypophysis



"HYPOPHYSECTOMY"
For Operations on Hypophysis

FIGURE 3.
EXAMPLE OF STRUCTURE OF WORD CODES

841 - 7773 Hypopituitary cachexia

841 - 7774 Sex infantilism

841 - 7775 Sex infantilism with obesity

841 - 7776 Dwarfism and infantilism

842-779 Diabetes insipidus due to unknown cause

Under the classification - 8 for NEW GROWTHS a letter, A through I, may be added to specify the Malignancy Code for the growth.

"A" Benign - no premalignant significance

For example:

841 - 8091A Adenoma of pituitary gland

Coding for Operations. For coding surgery of the pituitary gland, the following plan is used. See Figure 3.

- 0 Incision (to cut open)

8.. - 01 Exploration of endocrine gland (The periods..are to be filled-in with digits to indicate the site. For example 841 - 01 is an exploration of the anterior lobe.)

- 1 Excision (to open and remove, cutout)

820 - 1. Parathyroidectomy

830 - 1. Thymectomy

84. - 1. Hypophysectomy (The first period is to be filled-in to indicate the site. For example, 842 - 1. is the removal of the posterior lobe.)

850 - 1. Pinealectomy

860 - 1. Adrenalectomy (The final period is to be filled-in with a 2 or 3 to indicate complete or partial. For example, 841 - 13 is the partial removal of the anterior lobe.)

811 - 11 Local excision of lesion of endocrine gland (Periods are replaced by digits to indicate the exact part of the hypophysis.)

811 - 16 Biopsy of endocrine gland

References

1 MEDIC Branch--Special Development Division of SDC. Patient Data Processing in the Hospital--System Description, TM(L) 566/001/00, Jan. 1, 1961.

2 HUFFMAN, EDNA K., R.R.L. Manual for Medical Record Librarians, Physicians Record Co., fourth edition, 1955.

3 SELYE, HANS and MIKLOS NADASDI. Symbolic Shorthand System for Physiology and Medicine, ACTA Inc., Montreal, second edition, 1958.

4 TAYLOR, NORMAN BURKE, V.D., M.D. (ed.). Stedman's Medical Dictionary, The Williams & Wilkins Company, 1957.

5 International Classification of Diseases Adapted for Indexing of Hospital Records and Operation Classification, U.S. Department of Health, Education, and Welfare, Public Health Service, December 1959.

6 PLUNKETT, RICHARD J., M.D. (ed.) and ADALINE C. HAYDEN, R.R.L. (assoc. ed.). Standard Nomenclature of Diseases and Operations, The American Medical Association, McGraw-Hill Book Company, Inc., 1952.

7 "Efficiency in Hospital Indexing of the Coding Systems of the International Statistical Classification and the Standard Nomenclature of Diseases and Operations," Journal of the American Association of Medical Librarians, Vol. 30, No. 3, Part 1, June 1959, pp. 95-111, 129.

8 MEYER, ROBERT S. "Specialized Planning for Information Retrieval--Modern Trends in Documentation," Symposium, University of Southern California, edited by Dr. Martha Boaz, Pergamow Press Ltd., April 1958

9 HARRIS, ZELLIG I. "Transformations and Discourse Analysis Project," Department of Linguistics, University of Pennsylvania.

10 HAYES, DR. ROBERT M. "Information Storage and Retrieval" (X459CD), U.C.L.A., October 1961.

11 LUHN, H. P. "A Statistical Approach to Mechanized Encoding and Searching of Literary Information," I.B.M. Journal of Research and Development, Vol. 1, No. 4, October 1961.

12 Proceedings of the San Jose Conference on Health Information Retrieval, 1959.

FACT SEGMENTATION

Martin N. Greenfield

Minneapolis-Honeywell EDP Division

Wellesley Hills, Massachusetts

The FACT Compiler is Honeywell's English language narrative compiler used for commercial data processing applications. The program segments created by FACT have their position in memory dynamically relocated in order to make the most efficient use of the available core storage. The methods employed in this operation will be described as they are general in scope and of sufficient merit to find use in other applications.

Segmentation

Segmentation is the process of dividing a single program into pieces. This is done to permit the operation of programs that are too large to completely fit into memory. The pieces of the program are loaded only when needed. By having these segments time share areas of memory, the program may be executed.

The importance of segmentation has grown with the size of programs being produced. This has been accentuated by the popularity of compiler usage. It has become easier and as a result practical to write larger programs attacking larger problems. Divorcing the compiler user from machine considerations tends to have him create larger programs. The compiler tends to generate code that is more general and requires more space than the human created codes. The result is that every major compiler must give careful consideration to segmentation provisions.

Static and Dynamic Allocation

The customary practice of assigning memory in segmentation operations is static allocation. At compilation, assembly, or scheduling time, the area available to each segment is determined. The space is assigned and the segment always occupies the same locations whenever it is loaded. During execution, the flow of the program determines when each segment is loaded.

More recent developments have led to the usage of dynamic allocation of segments. At execution time both when a segment is to be loaded and where it is to go is determined. Allocation is determined by the monitor and can vary for subsequent loadings of the same segment.

Dynamic allocation requires a much more complicated loading and control routine than its static counterpart. More time is required to load a segment because it must have its contents conditioned to the positions it is to occupy. Inter segment communication is more involved because the positions of the segments are not predetermined.

Despite these penalties, the FACT Compiler design uses dynamic allocation for its object program segmentation primarily because it is so much more flexible and economical in its use of space. Applications such as data processing and information retrieval are characterized by handling large volumes of input having widely varying processing requirements. It is generally not known at the time a program is generated what the mix of processing segments and storage areas should be during each period of program execution. These requirements are dependent on the input. Only a data sensitive segmentation control can precisely determine what minimum portions of a routine must be present. Static allocation must compensate for this by keeping segments present for the worst case. The more flexible dynamic allocation conserves space by reacting only as needed. Segments that are unneeded are not kept in memory. Segments are loaded into what space is available at the time they are entered rather than requiring a specific area be designated for them. Because of this conservation of space, object code placing emphasis on execution speed at the expense of memory may be generated.

Characteristics of FACT Segmentation

Generally segments are composed of all of the instructions, constants and data storage required for their execution. In order to take the fullest advantage of the dynamic control, FACT segments are subdivided into smaller units. The smaller the unit, the greater can be the control over what must be present in memory.

Each major procedure, input edit procedure, report procedure, or internal or external file area is a segment. The advantage in separating segments from their file areas can best be clarified by example. A procedure may be operating on File A and File B. Upon terminating the operations on File B, the space used by File B may be made available to some other segment while operations continue on File A.

The mix of segments present during any period of execution is dictated by the flow of the program and is in turn sensitive to the processing requirements of the input. The storage occupied by a segment is released as soon as it is no longer required and this storage becomes available to any other segment.

Customarily, reloading a segment requires that it always be obtained from the secondary storage (program tape, disc, drum). In FACT, a check is made to determine whether a called segment that has been released is still intact in memory. This allows the operation to tend to release segments as soon as possible. If they are required a short time thereafter, they may be activated quickly without paying for a secondary storage access. Coupled with this feature is the tendency built into the control of keeping released segments intact as long as possible.

FACT segments are generated as relocatable blocks of code. Every segment must be operated on at the time it is loaded to make it executable. The location into which a segment is loaded is determined at the moment it is to be loaded. This location depends on the available configuration of space and will be described in more detail. Each segment is relocatable within memory. Segments are frequently moved and rearranged in memory in order to accomodate the loading of other segments.

The Segmenter

The routine that controls the segmentation operations is called the Segmenter. It is the initial segment of each program. It operates with either the Honeywell production monitor (Executive System) or checkout monitor (Program Test System) in activating, loading, releasing and relocating segments.

The Segmenter uses two tables. The Segment Control Table has an entry for each segment indicating its size, its base location (current address of the first word of the segment), its status, and its follower. The status can be active (A), inactive but intact in core (I), or out of core (O). The follower is the number of the adjacent segment in higher addressed core.

The Bank Control Table has an entry for each memory module (2048 words). The entry indicates the first and last segment numbers in the bank, the currently available inactive storage, the highest and lowest locations scheduled for usage in this bank, and the lowest location that has never been loaded. The highest and lowest scheduled locations are carried because the Honeywell 800 has multiprogramming capability. A scheduler defines the areas of memory available to each program. A FACT program must be scheduled for a consecutive set of registers. Other programs may be in parallel operation in the same bank but outside the scheduled area for the FACT program. Figure 1 shows both the Segment Control Table and the Bank Control Table.

The Segmenter normally relocates within the confines of a bank. No Segment is located such that it occupies portions of two banks. Because FACT segments tend to be small, this is not a restriction.

Segmenter Operation

By example using figures 1-6, typical Segmenter operations will be described.

Each FACT program initially loads the Segmenter as its first segment. After initialization, the first segment actually generated from source code is loaded. When other segments are addressed, the Segmenter will control their entry. Segments no longer needed will be released or deactivated by the Segmenter.

Following a short execution period, a typical memory layout and the corresponding tables in the Segmenter are pictured in Figure 1. The Executive System Monitor and the Segmenter are in bank 0. Active segments 3 and 5 are in bank one. Segments 1 and 6 in bank 0 and segment 8 in bank 1 were loaded and subsequently released. They are in an inactive status as indicated by the cross-hatching.

Examining the Segment Control Table entry for segment 5 shows that it starts in bank 1, location 200. It is 600 registers long, is in active status (A), and is followed in core by segment 8. The Bank Control Table shows that segment 0 (the Segmenter segment) is the first and segment 6 is the last segment in bank 0. An area from bank 0 location 500 to bank 1, location 1799 has been scheduled for this program. 1000 registers of inactive storage are available in bank 0. Location 1800 is the lowest location in bank 0 that has never had a segment loaded into it.

Activate Inactive Segment (Figure 2)

At this point if segment 1 is addressed, the Segmenter will discover from the Segment Control Table that is is intact in core. Segment 1 status is changed to active and the available storage in bank 0 is reduced by its length. Since this process is rapid and required no accessing secondary storage, it is attractive to release segments as soon as possible knowing that they may be reactivated rapidly.

Load Segments into Unused Storage (Figure 3)

When segments 9 and 10 are addressed, they must be loaded from secondary storage. The Segmenter will load them into areas of unused storage if possible so as to avoid destroying inactive segments. Preserving inactive segments will save accesses to secondary storage should they be required again. Adding the segment length and the field in the Bank Control Table designating the lowest location unloaded permits the Segmenter to determine that segments 9 and 10 will not fit into the end of bank 0, but will fit into the end of bank 1. The loader in the

monitor will search for the segments and load them where the Segmenter has specified. The Segmenter tables are then updated (base, status and follower fields of segments, last segment, unused storage, lowest location unloaded of bank 1).

Release Segment (Figure 4)

Releasing segment 5 has the Segmenter change the segment's status and add its length to the available inactive storage in the Bank Control Table. Segment 5 is still intact and may be reactivated easily if required again.

Load Segment into Opening (Figure 5)

Scanning the Bank Control Table will indicate that segment 7 will not fit at the end of any bank, but can fit into the inactive storage of bank 1. There is no inactive contiguous storage large enough to contain segment 7. The Segmenter must collect an adequate area by relocating the active segments in bank 1.

The relocation process can best be visualized by picturing the active segments in bank 1 as loosely strung beads. The gaps of string between the beads represent the inactive storage. The bead following the bead at the beginning of the bank is moved adjacent to it. The gap created below the moved bead is examined to see if the called segment will fit. If it will not, another bead is moved adjacent to the moved bead until an adequate gap is developed. The called segment will then be loaded into the gap.

To perform this operation it is necessary to know the sequence of segments in the bank. This is found starting with the first segment as indicated in the Bank Control Table (segment 3). In each Segment Control Word, the segment number that follows is shown in the follower field. Thus, segment 3's follower is segment 5, 5's follower is 8 and so forth. (This refers to the segment control table indicated in figure 4).

Scanning down the chain of segments, the last of the first group of active segments can be found. This segment is the head bead of the analogy, (segment 3) Following it will be a group of inactive segments (5 and 8). The bead to be first moved is the first active segment (segment 9) following the group of inactive segments . Segment 9 is moved next to segment 3. By doing this, inactive segment 5 is overlaid. Since segment 5 can no longer be directly activated, its status is changed to out of core (0). Segment 10 will also have to be moved before segment 7 will fit. After this, segment 7 is loaded into the bank. Segment 7 will overlay segment 8, so it is necessary to change segment 8's status to out of core (0).

All of the above operation seems to be quite involved and yet its operation is barely perceptible when watching it at the H-800 console.

Squishing (Figure 6)

Segmenter operations described so far have been confined to action within one bank. Should a segment be called that is larger than the inactive storage in any one bank, the inactive storage in all of the banks is accumulated in an attempt to fit the segment. To do this, the Segmenter employs a subroutine called the Squisher. This routine, whose name sounds like a fugitive from a science fiction movie, will coagulate the active segments into the lower addressed banks and locations available to the program. Where necessary segments will cross into other banks. Like all reputable science fiction creatures, the Squisher is destructive in that it destroys all of the inactive segments in memory. In return it collects all of the inactive space together into upper memory. This should enable the loading of the called segment that otherwise could not have fit.

Another use of the Squisher is in preparation for executing a sort. Sorts are amorphous by nature and operate faster if given more space. The Squisher operation gathers all available inactive space for the sort and informs the sort generator of the amount of space available.

Relocation, Communication, Linkage

The explanation thus far has been about when and why segments are relocated. The next sections will explain how the relocation is performed, how the migrant segments can successfully communicate with each other, and what the linkage is that brings the Segmenter into action.

Relocation

The execution of any code is sensitive to its location in storage. When segments are moved or loaded, their code must be adjusted to the new environment. When loading a routine from a secondary storage as tape, it is possible to have information on how to relocate each word delivered to the loader. This relocation information is used during loading and does not permanently reside in memory. Unavailability of relocation information for the memory to memory relocation of FACT requires another method of determining the relocation rules.

A Segment is generated as a block of code that must be moved together. When a segment is moved, the incremental address is the same for every word in the segment. The segments are constructed in the structured format shown in Figure 8. Each word of the segment that follows the same relocation rules is assembled into the same region. One word in the segment defines the length of each of these regions. It then becomes possible to have a relocation

routine operate on each of the regions and make the appropriate modifications.

Each segment is kept on the program tape though it were to be loaded into locations starting at zero. The relocation increment for a program being loaded is the intial loading address (or base). The memory to memory relocation subroutine is used to perform the relocation. Relocation data need not be kept on the program tape nor must the loader be concerned with this process.

Intersegment Communication

Segments must be able to operate on information in other segments, transfer information between segments, and transfer control to other segments. The mobility of FACT segments requires a communication device for these functions. This centers about a table called a Reference Region that is part of each segment.

The Reference Region is a table of addresses of locations in other segments that are addressed by the segment. These entries contain initially the number of the segment in one field and the location relative to address zero within that segment that is being addressed in another field. Whenever a segment is loaded or moved in memory, its relocation increment is used to update the Reference Region tables of all other segments of that program that are in memory. Whenever a segment is loaded, its Reference Region table is updated to reflect the positions of those segments that it addresses that are in memory. It is possible to rapidly scan and update the Reference Region tables by using the Segment Control Table. The segments may freely communicate using the information kept current in their Reference Regions.

The updatings of the Reference Region of segment 3 of the example used before is shown in Figure 7. Addressing between segments is always done using either indexed or indirect addressing features of the Honeywell 800. The entries in the Reference Region are arranged so they may be loaded into the special registers for this mode of addressing.

Linkage to the Segmenter

The Segmenter is dormant until called to load, activate, or release segments. When the Segmenter is active, the program it controls is kept dormant. (This need not be the case because of the parallel processing capability of the Honeywell 800). Other programs running in parallel do keep operating. It is necessary to stop the operation of the program calling the Segmenter because some of its segments in memory may be relocated, a segment may have to be loaded from secondary storage, and other calls to the Segmenter must be inhibited.

Calls to the Segmenter to release a segment occur at points in the code that can be defined a compilation time. The calling linkage is a simple direct entry similar to those used in entering monitors.

Linkage to activate or load a segment are quite different. The system is designed to allow segments to easily address each other and at the same time have the segments activated and released frequently. It would be most inefficient in time and storage if each instance of intersegment addressing had to require testing to determine whether the segment was active in memory.

The method employed relies on the fact that intersegment communication always uses indirect or index addressing that employs the words in the Reference Region table to load these special registers. Whenever a segment is inactive, all Reference Region words pertaining to it are forced to bad parity. When a segment loads the special register with the Reference Region word, the Honeywell 800 forces an automatic unprogrammed transfer of control to a specified location. The location transferred to is the start of the linkage to enter the Segmenter.

The hardware is such that no sequence counter or any other settings are disturbed by the unprogrammed transfer. After the Segmenter operation, it becomes very easy to restore control to the calling segment as none of its settings have been changed by the linkage. Thus, a hardware feature of the Honeywell 800 is used as an automatic test sensitive to the absence of an addressed segment.

Since a hardware error detection feature is being used in this linkage, the Segmenter must verify that this is a legitimate call and not a machine malfunction. The Segmenter reduces the probability of a mistaken call by verifying that the location of the word of bad parity resides in a Reference Region table and the Segment number referred to is inactive. The reliability of the H-800 memory further reduces the probability.

Operation with the Monitors

The Segmenter has been designed to work with both the Program Test System and the Executive System monitors.

The Program Test System, Honeywell's checkout system, allows dynamic or snapshot dumping of memory areas and tapes during the program operation. The dumping occurs at points where the programmer requests derail linkages to the monitor. The requested dumping is identified by the location of the derail. The mobility of the segments created

a problem that was handled by including the segment number in the derailing instruction. Use of the Segment Control Table allowed a routine to convert the derail location to a location relative to location zero. The particular derail could then be identified. The instructions dumped are converted via use of the Segment Control Table to locations relative to zero. Thus, the programmer need not be hindered by the actual position of the segment while it was being dumped.

The Executive System will be the primary monitor working with the Segmenter. Each FACT program is given an area of memory to operate in at scheduling time. The full power of the Executive System Monitor is available to the FACT programs. It is used to set restart points, restart individual programs, execute a full schedule of parallel operating routines, test the validity of the schedule, etc. Loading of segments is done by the Run Supervisor after receiving the information from the Segmenter. The search for a new segment on the Production Run Tape is always optimized to the proper direction.

Operating Experience

That described in this paper is not a proposed or theoretical system but one that has been in daily operation since August 1961. There have been embellishments to improve the Segmenter, but it has not changed in its basic operation.

In September 1961, the Segmenter was tied into the Executive System. Runs have been scheduled and executed involving combinations of two FACT programs and an assembly language program all operating in parallel. In that operation there are periods where five of the eight independent sequence counters in the H-800 are active. All eight of the sequence counters were used at some time during this operation.

Future Modifications

A number of ideas have been suggested to add to the flexibility and attractiveness of the Segmenter. Currently the Segmenter requires about 500 memory cells. There is a separate segmenter for each FACT program being run in parallel. A technique has been developed to have only one Segmenter for all FACT programs in operation. The FACT programs could have their segments intermingled in memory instead of having distinct areas as is now required. The most difficult part of this design would be the setting of restart points. The Segment Control Table would be employed to identify the areas occupied by the segments of the program for which a restart was being set.

To further conserve memory, the operations of the Segmenter would themselves be segmented.

The Segment Control Table is the source of information to the Segmenter as to the size of each segment. This table could have its entries changed by the program and enable specified segments to grow in size according to the dynamic needs of the routine.

The Segmenter makes no attempt to save the settings of a segment it is about to destroy. Such a segment could be dumped into secondary storage prior to its destruction. A choice could be made when the segment was again called as to having it loaded in its initial state or in the state it was in the last time it appeared in memory.

Acknowledgements

Acknowledgement should be given to the staffs of Computer Science Corporation, Inglewood, California, Philip Hankins and Company, Inc., Arlington, Massachusetts and the Honeywell FACT Group for their work in the design and implementation of that described in this paper.

SEGMENT CONTROL TABLE

SEG. NO.	STATUS*	LENGTH	FOLLOWER**	LOCATION
0	A	500	1	0,0500
1	I	500	6	0,1000
2	0	400		
3	A	200	5	1,0000
4	0	600		
5	A	600	8	1,0200
6	I	300	L	0,1500
7	0	900		
8	I	200	L	1,0800
9	0	300		
10	0	300		

BANK CONTROL TABLE

BANK NO.	FIRST SEG.	LAST SEG.	UNUSED STORAGE	LOWEST LOC.	HIGHEST LOC	LOWEST LOC UNLOADED
0	0	6	1000	500	1999	1800
1	3	8	1000	000	1799	1000

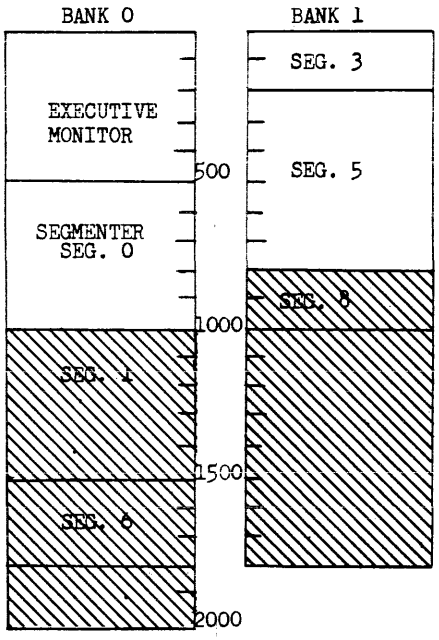


Fig. 1 STARTING CONDITION OF EXAMPLE

SEGMENT CONTROL TABLE (PARTIAL)

SEG. NO.	STATUS	LENGTH	FOLLOWER	LOCATION
0	A	500	1	0,0500
1	A	500	6	0,1000
2	0	400		
etc.				

BANK CONTROL TABLE

BANK NO.	FIRST SEG.	LAST SEG.	UNUSED STORAGE	LOWEST LOC.	HIGHEST LOC.	LOWEST LOC UNLOADED
0	0	6	500	500	1999	1800
1	3	8	1000	000	1799	1000

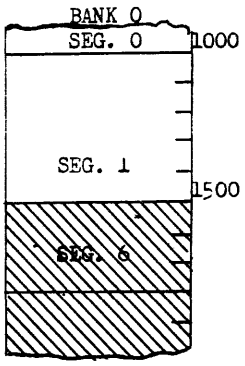


Fig. 2 ACTIVATE INACTIVE SEGMENT 1

* A = ACTIVE; I = INACTIVE, in HSM; 0 = OUT OF HSM
** L = LAST SEGMENT IN BANK

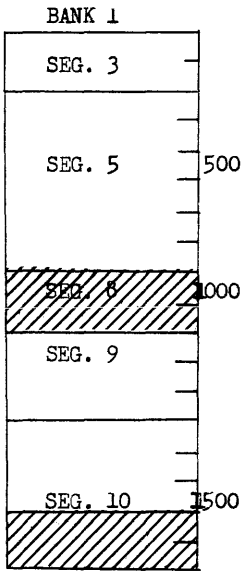
SEGMENT CONTROL TABLE

SEG. NO.	STATUS	LENGTH	FOLLOWER	LOCATION
.				
.				
8	I	200	9	1,0800
9	A	300	10	1,1000
10	A	300	L	1,1300

BANK CONTROL TABLE

BANK NO.	FIRST SEG.	LAST SEG.	UNUSED STORAGE	LOWEST LOC.	HIGHEST LOC.	LOWEST LOC. UNLOADED
0	0	6	500	500	1999	1800
1	3	10	400	000	1799	1600

Fig. 3 LOAD SEGMENTS 9 and 10 INTO END OF STORAGE



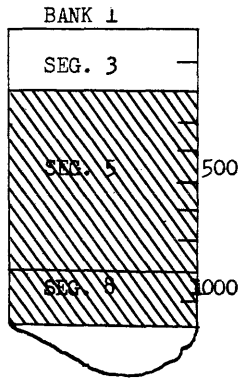
SEGMENT CONTROL TABLE

SEG. NO.	STATUS	LENGTH	FOLLOWER	LOCATION
.				
.				
4	0	600		
5	I	600	8	1,0200
6	A	300	L	0,1500

BANK CONTROL TABLE

BANK NO.	FIRST SEG.	LAST SEG.	UNUSED STORAGE	LOWEST LOC.	HIGHEST LOC.	LOWEST LOC. UNLOADED
0	0	6	500	0500	1999	1800
1	3	10	1000	0000	1799	1600

Fig. 4 RELEASE SEGMENT 5



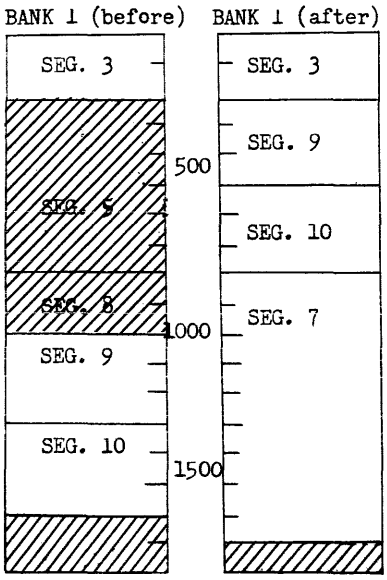
SEGMENT CONTROL TABLE

SEG. NO.	STATUS	LENGTH	FOLLOWER	LOCATION
0	A	500	1	0,0500
1	A	500	6	0,1000
2	0	400		
3	A	200	9	1,0000
4	0	600		
5	0	600		
6	I	300	L	0,1500
7	A	900	L	1,0800
8	0	200		
9	A	300	10	1,0200
10	0	200	7	1,0500

BANK CONTROL TABLE

BANK NO.	FIRST SEG.	LAST SEG.	UNUSED STORAGE	LOWEST LOC.	HIGHEST LOC.	LOWEST LOC. UNLOADED
0	0	6	500	500	1999	1800
1	3	7	100	000	1799	1700

FIG. 5 LOAD SEGMENT 7 INTO AN OPENING



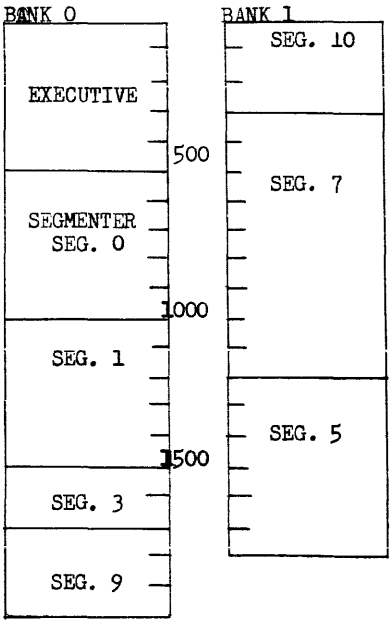
SEGMENT CONTROL TABLE

SEG. NO.	STATUS	LENGTH	FOLLOWER	LOCATION
0	A	500	1	0,0500
1	A	500	3	0,1000
2	0	400		
3	A	200	9	0,1500
4	0	600		
5	A	600	L	1,1100
6	0	300		
7	A	900	5	1,0200
8	0	200		
9	A	300	L	0,1700
10	A	200	7	1,0000

BANK CONTROL TABLE

BANK NO.	FIRST SEG.	LAST SEG.	UNUSED STORAGE	LOWEST LOC.	HIGHEST LOC.	LOWEST LOC. UNLOADED
0	0	9	000	500	1999	2000
1	10	5	000	000	1799	1800

FIG. 6 LOAD SEGMENT 5 AFTER SQUISHING



SEGMENT 3
REFERENCE REGION

TAG	REFERENCED SEGMENT	LOCATION	BAD PARITY
R1	SEG. 1	0,0210	x
R2	SEG. 5	1,0500	
R3	SEG. 9	0,0181	x
STARTING CONDITION			
R1	SEG. 1	0,1210	
R2	SEG. 5	1,0500	
R3	SEG. 9	0,0181	x
SEGMENT 1 ACTIVATED			
R1	SEG. 1	0,1210	
R2	SEG 5	1,0500	
R3	SEG. 9	1,1181	
SEGMENTS 9 and 10 LOADED			
R1	SEG. 1	0,1210	
R2	SEG. 5	0,0300	x
R3	SEG. 9	1,1181	
RELEASE SEGMENT 5			
R1	SEG. 1	0,1210	
R2	SEG. 5	0,0300	x
R3	SEG. 9	1,0381	
LOAD SEGMENT 7 (MOVE SEGMENT 9)			
R1	SEG. 1	0,1210	
R2	SEG. 5	1,1500	
R3	SEG. 9	0,1881	
LOAD SEGMENT 5 (MOVE SEGMENT 9)			

FIG. 7 CHANGES TO SEGMENT 3 REFERENCE REGION WORDS

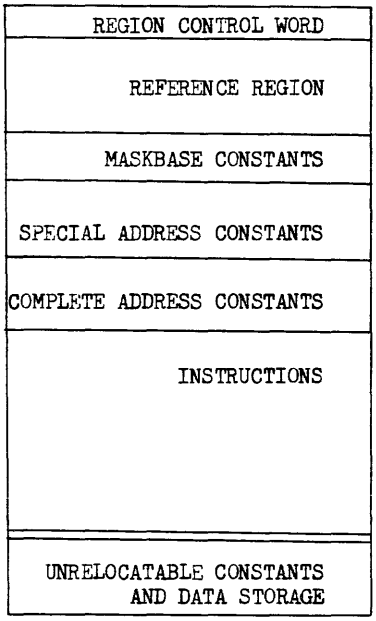


FIG. 8 TYPICAL FACT SEGMENT STRUCTURE

A GENERAL TEST DATA GENERATOR FOR COBOL

Lt. Richard L. Sauder

Automation Techniques Branch

Headquarters, Air Force Logistics Command

Wright-Patterson Air Force Base, Ohio

This article discusses the effort being made by the Air Force Logistics Command in developing a method of generating effective program test data. This "Test Data Generator" is designed to operate in conjunction with the COBOL compiler implemented by AFLC. As such, the system not only builds data conforming to descriptions given in the Data Division of a COBOL program but also places in these items necessary data relationships to test the logic of the COBOL program. Both the utilization and the method of operation of the system are discussed in this paper.

Introduction

One of the major underdeveloped areas that still exists in the development of programming techniques is that of insuring adequate checkout of programs before release for use. Often the logical paths within a program are highly complex. The effort to insure that each of these is functionally correct can be prohibitive - so much so that only the most obvious and most frequently employed segments of the program are tested thoroughly.

Compilers that have been employed in the past decade have aided in reducing this problem. By eliminating many careless coding errors, they have not only insured a greater degree of operation in a shorter time but have also released more time for more elaborate testing of the compiled program. The fact still remains, however, that unless adequate test data is available the debugging effort is hampered. Recognizing this deficiency, the Air Force Logistics Command last year began developing a method of producing this data. The immediate result is a Test Data Generator designed to operate in conjunction with the COBOL compiler implemented by AFLC. As such it performs two main functions:

1. Builds data of the format and description specified in the COBOL source program and

2. Inserts in these elements data specified by the user to meet inter (and intra) element requirements and relationships.

These functions basically fulfill the needs for producing effective test data. Obviously it is necessary to produce elements conforming to their given formats. But since there are few business oriented runs which do not require dependency among various elements, a more pertinent criterion is that of controlling the content of these elements.

The method of operation and the information necessary to fulfill these functions will now be described in detail.

Utilization of the COBOL Data Division

Determining the format of data fields results from a thorough interpretation of the Data Division of the COBOL source program being tested. Within this Division, options exist to describe in detail both the structure of every element within a record and the relationships of these elements to one another. Those options which affect the generation of data are defined as follows:

1. Name
2. Level number to establish the relationship of one unit of data to others.
3. Size in number of characters.
4. Class indicating the type of data, i.e., alphabetic, numeric, alphanumeric.
5. Usage to establish the number system in which the element is represented.
6. Occurs to define repeated occurrences of the same element.
7. Range to establish limits for the value of the element.
8. Sign specifying the operational sign of the element.
9. Synchronize positioning the element within or across computer words.
10. Redefines to allow the same area to have more than one description.
11. Picture - a graphical representation of the element.
12. Value to define a stated value for the element.

In addition to these element descriptions, options are present which describe the files to which the records belong, e.g., tape and file labels, the number of records within a file and the number of elements within these records.

Construction of Formatted Data

Since in the generation of data we are concerned only with those records in a program designated for input, the interpretation of the source program Data Division produces only sets of parameters dealing with elements of input records.

In all cases this set includes the size of the element of data and its relative location within the record. Other information produced depends on the type of element. Type in this context is defined as one of four classifications:

1. Literal indicating that a single value has been defined for the element.
2. Conditional indicating that a restricted set of variables only may appear in the field. (Conditional values in COBOL data descriptions are designated by a level number of 88.)
3. Random indicating that no values were assigned to the field so that the content of the element is determined only by the options: class, range, and base.
4. Sequential indicating that the original value of the element is to be incremented by a fixed or a random amount.

All information concerning these types can be obtained from the data descriptions with the exception of those items to be sequenced, and the amount, if known, by which they are to be incremented. Therefore, disregarding these items, it is now possible to generate "garbage" test records with only the COBOL source tape for the program serving as input. The method of generation if this were desired would be as follows:

1. Interpretation of the program's Data Division, construction of sets of parameters and grouping of these parameters by record designation.
2. Isolation of all parameters for a particular record and employment of each set of parameters, in turn, to build a single test record with the following criteria:
   a. If the element is literal or conditional, insert the value or chosen value into its relative position in the record.
   b. If the item is random, generate random characters (determined by the class of the item) filling the element.
3. Repetition of step two until the desired number of test records has been generated. Repetition of the entire process for the next record description, and so forth.

In testing applications, it is often true that some information which would be generated in this manner is insignificant or unnecessary. For instance, a thirty character random field reserved for a manufacturer's name could possibly be deleted or replaced by a two character code representing the name if this element is not involved in any logical decision in the program. This action would not only reduce the time needed for generation, but also would considerably improve the appearance of the data. In addition, elements may not be described in the Data Division to a sufficient degree for testing purposes. For instance, a stock-number may be described as alphanumeric when in reality the first half of the number is alphabetic and the remainder numeric.

For these reasons the Data Generator includes the option to modify data descriptions by the deletion or replacement of these descriptions or by the insertion of new descriptions. The method by which these modifications are expressed will be discussed later.

Specification of Data Relationships

The data generated in the manner described falls far short of being adequate for production testing. Though some control on the content of the records is obtained by stating literal and conditional values for elements, the great majority of the output is unrelated "garbage" adequate only for testing error logic. For this reason the ability to specify data relationships and requirements is included with the generator. The type of relationships desired may be of the following nature:

> If field-1 of record-A equals field-1 of record-B then field-2 of record-A must be in the range 7-15. If field-2 of record-A is less than 10 and field-3 of record-A is not less than 24 then field-7 of record-B must equal "A"; but if field-2 of record-A equals 10, then field-7 of record-B equals "C". If field-7 of record-B equals "C" then field-8 of record-B equals 0 or 2 . . . . . .

Via this option the formatted records produced can be edited to contain combinations of these relationships, which in turn can test the logic of the program concerned. To allow a flexibility of this magnitude, a procedural type system similar in objective to the COBOL Procedure Division is provided. Just as the Procedure Division contains a series of statements arranged to specify the logic of a computer program, this system (named the Relation Section) is a series of statements arranged to specify the logic of data relationships.

The Relation Section is formed by a series of conditional and declarative statements consisting of a restricted set of operations and the operands involved. The operands are those element names which appear in the Data Division of the program or those defined by the method of modification explained before, and any quantity appearing as a literal value. The same conventions apply to expressing these operands as in the COBOL Procedure Division.

Example:

> If FIELD-1 equals SUBFIELD-X of FIELD-A of RECORD-1 . . .
>
> If CLASS-CODE-1 equals "ABC" then IN-CODE of FIELD-C equals 375 . . .

The set of operations allowed consists of three types: relational, arithmetic and control.

Relational Operations

The relational elements are the same as those defined in the COBOL language for expressing conditional relationships. In the Relation Section, however, these elements serve a dual purpose. They first express conditional relationships as in COBOL. In addition, however, they are employed in imperative statements to prescribe an action between given operands. Used this way, the convention is established that given

oprnd-m (relation) oprnd-n,

oprnd-m is adjusted, if necessary, to satisfy the given relation. For instance, given: FLD-A equals FLD-B. If inequality exists, the content of FLD-A is made equal to that of FLD-B.

The relations (and their associated mnemonics) are:

EQ	equals
NEQ	is not equal to
GR	is greater than
NGR	is not greater than
LS	is less than
NLS	is not less than
ZER	is zero
NZR	is not zero
POS	is positive
NPS	is not positive
NEG	is negative
NNG	is not negative
RNG	range of (element-j) is (value-a) - (value-b). (value-c), (value-d),...(value-m).

Examples:

1. If FLD-B EQ FLD-C, ITEM-NO-1 NLS 200017.
2. FLD-3B of RECD-1 NNG; FLD-2A of FLD-2 of RECD-B NEQ FLD-2B.
3. If ELEMENT-C GR 7 or if ELEMENT-P EQ "AB", RNG FLD-3A is 7-19.
4. RNG ITEM-PQ is "A", "B", "D", "X", 9, 13.

Arithmetic Operations

\+ element-j PLUS element-k
\- element-l MINUS element-m
* element-n TIMES element-p
/ element-q DIVIDED BY element-r

Examples:

1. If FLD-E EQ FLD-C + FLD-D, then ITEM-1 EQ ELEMENT-1 * 3.
2. If ELEMENT-B - 35 GR 7, RNG FLD-F is 0, 1.

Control Operations

At present three control operations are permitted.

1. END denotes completion of data specifications.
2. GO to path-i.

Example: If FLD-R EQ 7 and FLD-P NLS 100, GO TO AB-1; otherwise GO TO AB-2.
AB-1. RNG FLD-H is . . . . . .

3. BLD BUILD record-name-j.
This operation signals the construction of a particular record described in the Data Division. If references are made to elements within a record for which the BLD option has not been stated, these items are ignored.

The option simultaneously releases a record which has been constructed and initiates the construction of a new one.

Example: 1. BLD RECORD-1, RECORD-2.
2. If ITEM-CODE is POS, BLD RECORD-3.

Additional Relation Section Options

Several storage areas of varying sizes are set aside for use with the Relation Section. Each of these areas has a fixed name and may be referenced as an operand by any appropriate operation stated. The major reason for providing these areas is to allow trailer type items to be specified. For instance, if a stock-number of an item must appear in fifteen successive trailers, this stock-number can be generated and placed in a storage area and then be moved later to the appropriate element when the trailer items are built.

These areas must be used when an element of a table is referenced by subscripting.

Examples.

1. If TRALR-CODE-1 EQ 1, BILD-19 EQ STOCK-NO-1. . . . .
. . . . BLD TRLR-RECORD-1A. STOCK-NO OF TRLR-RECORD-1A EQ BILD-19 . . . .
2. BILD-6 EQ 7. If TABLE-1 (BILD-6) EQ . . . .

NOTE: BILD-6 and BILD-19 are names of assigned storage areas.

Preparation of Relation Section for Input

The examples that have been given for the use of the Relation Section have indicated a

moderately free format. On the contrary, a tabular form of input has been chosen because the task of specifying the operands and the desired operation (or operations) is generally simplified by employing tables. Two tables - one containing operands, the other containing operations - are required to specify a set of relations and actions. Figure 1 describes the formats of these tables. In this illustration also is the tabular representation of the following series of statements.

A1. If IN-1 EQ FLD-2, then STOCK of RECORD-A EQ STOCK of RECORD-2, BILD-6 EQ 5, TBL-1 (BILD-6) GR TBL-2 (BILD-6).
A2. If START-CODE NLS 4, then RNG FLD-18 is "A", "B", "X", go to A5; otherwise go to A4.
A4. BLD RECORD-C. Go to A5.
A5. BLD RECORD-A, RECORD-B. Go to A3.

In this tabular form, series of disjunctive statements are expressed horizontally across successive columns; conjunctive statements are expressed vertically in one column. For example, given the series of expressions:

If (A EQ B and C LS D) or J NLS K or (E NQ "X" and F POS), THEN BLD Q.

This series is expressed tabularly as follows:

NAME-TABLE

10	A	B
20	C	D
30	E	"X"
40	F	
50	J	K
60	Q	

OPERATION-TABLE

	P1	P2	P3	P4
Condition-Group				
10	EQ			
20	LS			
30			NQ	
40			POS	
50		NLS		
Result-Group				
60				BLD
GoTo	P4	P4	P4	
Else	P2	P3	P5	

Additional Necessary Information

Two additional sections are provided in the input format for expressing those options needed in constructing formatted data.

Control Section. Within this section information is given as to the number of test items to be constructed and those elements to be sequenced. The COUNT option specifies this total number for each record within one file. The format is as follows:

```
COUNT  file-name
       record-1   amount
       record-2   amount
          .          .
          .          .
          .          .
       record-m   amount
```

Only those input files for which test records are to be constructed must be listed.

The SEQUENCE option is specified in the following manner:

SEQUENCE (Name) (Orig. value) (Increment).

Examples:

```
CONTROL SECTION.
COUNT    FILE-1
         RECD-A    5000
         RECD-B    1500
COUNT    FILE-2
         RECD-C    7000
SEQUENCE FLD-ZC    100000    100
         OF RECD-B
SEQUENCE ELEM-P1   ABCDE
```

Overlay Section. This section contains those elements of the Data Division which are to be modified. Any of three options may be employed depending upon whether an element is to be deleted, replaced, or inserted. (The "line-number" referred to in the following explanations is that found in the COBOL Data Division.)

1. Deletion:

line-no-1 DELETE (thru line-no-2).

Example: 100024 DELETE THRU 100027.
100105 DELETE.

2. Insertion:

line-no-1 INSERT (n) lines.
The lines to be inserted must directly follow the INSERT statement.

Example: 100046 INSERT 4 LINES.
02 FIELD-A size is 6. Value is "ABC123".
02 FIELD-B size is 9. Class is numeric synchronized.

3. Replacement.

line-no-1 (contents of new line).

Example: 100049 03 IN-CODE size is 30.
100083 -LUE IS "X-CO".

The Control, Overlay, and Relation Sections provide sufficient information to generate test items fulfilling both functions of the Test Data Generator. The combined system is titled the "Requirements Division." A complete example of this division is given in figure 2.

Construction of Edited Data

At present work is proceeding on a network analysis routine which will serve the following purpose. After the parameters describing record formats have been constructed, these parameters

NAME-TABLE-j

101	IN-1	FLD-2
102	START-CODE	4
(REF #)	(O P E R	A N D S)
174	RECORD-C	
175	RECORD-A	RECORD-B
184	STOCK OF RECORD-A	STOCK OF RECORD-2
190	BILD-6	5
195	TBL-1 (BILD-6)	TBL-2 (BILD-6)
201	FLD-18	"A", "B", "X"
(REF #)	(O P E R	A N D S)

OPERATION-TABLE-j

Path	A1	A2	A3	A4	A5
Condition-Group					
101	EQ				
102		NLS			
(REF #)	(O P E R A T I O N S)				
Result-Group					
174				BLD	
175					BLD
184	EQ				
190	EQ				
195	GR				
201		RNG			
(REF #)	(O P E R A T I O N S)				
GoTo	A2	A5		A5	A3
Else	A2	A4			

Figure 1. Relation Section Input Format

```
D00000  REQUIREMENTS DIVISION.

D00010  CONTROL SECTION.
D00020  COUNT       FILE-AA
D00030              RECD-AA              7000
D00040              RECD-INVENTORY       3500
D00050  COUNT       FILE-BA
D00060              REPLACEMENTS         1500
D00070              ADDITIONS            2000
D00080  COUNT       FILE-BB
D00090              ALTERNATES           750
D00100  SEQUENCE    STOCK-NO-A2          ABCD1234567
D00110  SEQUENCE    ITEM-CODE OF         1000000        100
D00120              RECD-INVENTORY

D10000  OVERLAY SECTION.
D10010  100035  03  REQ-CODE SIZE IS 6 NUMERIC.
D10020  100076  DELETE THRU 100079.
D10030  100080  02  CO-NAME CLASS ALPHA SIZE 30 VALUE "AB".
D10040  100157  INSERT 4 LINES.
D10050          05  SECND-PART.
D10060          88  ACCEPT-CON  SIZE 1 VALUE "A".
D10070          88  REJECT-CON  SIZE 1 VALUE "B".
D10080          88  END-CON     SIZE 1 VALUE "E".

D20000  RELATION SECTION.
D20010  NAME-TABLE-1
```

D20020			
D20030			
D20040	250	REQ-CODE	LEAD-NO OF
D20050			RECD-AA
D20060	260	SECND-PART	"A"
D20070	270	SECND-PART	"B"
D20100	300	SECND-PART	"E"
D20110	310	ACTIVE-CNT	370000
D20120	320	COST-CODE	7, 8, 9
D20130			
D20140	510	RECD-AA	
D20150	520	REPLACEMENTS	
D20160	530	COST-CODE	7 - 12
D20170	540	ALTERNATES	
D20180	550	BILD-6	
D20190	560	TRLR-CODE	1
D20200	570	ADDITIONS	
D20210			
D20220			

OPERATION-TABLE-1

Path	PA1	PA2	PA3	PA4	P5	P6
Condition-Group						
250			NLS	GR		
260					EQ	
270				EQ		
300		EQ				
310		GR				
320				RNG		
Result-Group						
510	BLD					
520	BLD					
530			RNG			
540						BLD
550					EQ	
560					+	
570					BLD	
GOTO	PA2	END	PA4	A6	A6	PA1
ELSE	PA2	PA3	PA1	A5	PA1	A7

Figure 2. Sample Problem Employing the Requirements Division

will be modified before each test record is generated so that relationships specified may appear in the record. In this manner, different branches in the networks of relations given can be traced until the maximum number of records requested has been generated.

This scheme introduces a significant problem. Two branches must emanate from every branch point in a logic network; therefore, for n branchpoints, there exist 2^n paths. In terms of the problem at hand, if only fifteen conditional statements (stemming from a common origin) were expressed, there would be over 30,000 possible methods of combining the data relations stated. With a limited number of test items to be generated, then, it is likely that several important combinations would be omitted.

To overcome this problem, it is possible to designate priority branches in the network. All path names in the Relation Section which are prefixed by the letter "P" will be considered priority paths and will be investigated first.

Generation of a Test COBOL Source Program

A temporary method of generating edited test items has been devised to operate until an acceptable network analysis routine is completed. Via this method, the information given in the Data Requirements Division and portions of the Data Division of the program to be tested are edited into a COBOL Source Program. This program when compiled accepts as input the unedited data files generated as a result of the interpretation of the Data Division. Each of the records defined in a file is assigned an input area in Working Storage of the edited source program. When a BUILD operation is encountered, an edited item, if present, is released from this area and a new "garbage" item is read in. The source coding generated to test relational operations merely involves a test comparing the operands involved. If the test fails, the first operand is adjusted to satisfy the relation.

To test error logic of the program, the test COBOL program is designed to generate improper relations at random. An error indicator is inserted in these items if space is available. This method of editing data relationships is admittedly less efficient than an analysis type sytem. However, the extensive use of this approach will help in indicating necessary revisions to the analysis routine being developed.

Conclusion

The use of the Test Data Generator requires work on the part of the programmer or systems analyst. This has a definite advantage, however, in that it forces him to review the logic of the data structure of a program. If this can be accomplished without reference to the program, the chances that the data generated will locate logical discrepancies in the program are increased. For this same reason, the concept of determining necessary data relationships by analyzing the Procedure Division of the program has been discarded. Although this method would require little work for the programmer or analyst, it would likewise eliminate all possibilities of locating missing branches in the program. Some thought has been given, however, to an analyzer of this type which would cross-check the statements of the Requirements Division and thereby possibly indicate logical discrepancies before any test data is generated. This method would also help to locate logical errors in the Requirements Division, should they exist.

Any implications that the Data Generator described is the final solution to program debugging are hereby denied! The system in its present state is merely an intermediate stage of development. Extensive applications of the generator will indicate the advisability of whether to continue with this line of thought or to revise the approach to any expedient degree. It is felt that this development will help to further understanding of how best to express data in a manner which may be of value to COBOL and problem-oriented languages in general.

DATA STRUCTURES THAT GENERALIZE RECTANGULAR ARRAYS

Samuel A. Hoffman*

Burroughs Corporation, Burroughs Laboratories

Paoli, Pennsylvania

Summary

Three problems associated with structured data sets -- description, allocation, and retrieval -- are briefly considered. The first is a problem in describing data to a compiler in a formal manner that will allow the compiler program to allocate storage and retrieve data when the data is referenced. The second is the problem of providing contiguous storage for the data set in such a way that when suitably described pieces of the data are required at run time, the positions of these pieces of data relative to some base will be determinable. The third problem is that of referencing or describing subsets of the data set.

A class of data structures is defined. This class is shown to be a generalization of the class of structures which are representable as n-dimensional rectangular arrays. These structures are termed generalized structures; a formal method of describing these structures (by descriptors) is defined. The formal entities, called reference expressions, that describe the data to be retrieved from storage, are also defined for this class of structures. Finally, the appropriate form of the storage mapping function is derived. The storage mapping function is the mathematical expression which relates the description of an item of data to its position in memory. The manner in which this function is derived from the descriptor is shown. In every case, the work performed for generalized structures is shown to be a direct generalization of the corresponding considerations for rectangular arrays.

Finally, an ALGOL program for the Burroughs 220 computer is briefly described. The program simulates the actions that a compiler would take upon receiving a descriptor in forming the storage mapping function, and the actions that would be carried out at run time when a reference expression is presented.

Description, Allocation, and Retrieval

When a program is being written for a digital computer, the programmer or programming system must see to it that memory space is suitably allocated for the data to be processed by the program. This is true regardless of the language in which the program is being written. When the data is structured, the allocation should be made so that calls for the data can reflect this structure. Because computer memories are sequentially organized -- by words, characters, bits, etc. -- the allocation and retrieval problem has usually involved linearly mapping the data into memory.

When a compiler is being used by the programmer, there are three facets to the structured data problem.

1. The structure of the data must be supplied to the compiler program in a suitable symbolic form.
2. The compiler program must have a routine that allocates a contiguous portion of memory for each instance of the structured data set.
3. A means must be available for the programmer to refer to individual (or groups of) items of the stored data.

These three facets may be characterized as description method, storage mapping function, reference expressions; the three facets are bound, respectively, to the more general problems of description, allocation, and retrieval.

The classical employment of structured data in computer programs has been that of n-dimensional rectangular arrays; that is, the data is considered to be indexed by n coordinates, where the allowable values of the ith index are 1, 2, ... , a_i. The description method for such a structure is simply given by the n-tuple of a_i's (a_1, a_2, ... , a_n) and, for identification purposes, a preceding name. It should be recognized at this point that the n-tuple itself defines the structure in question. The preceding identifier can then be interpreted as a free variable for which substitution will be made when a particular data set (an instance of the structure) is being declared as having this structure. However, in this paper, structures are assumed to be defined only in the context of particular instances of those structures.

In any case, then, a name or identifier followed by an n-tuple of integers provides a complete description of the data set so named as an n-dimensional rectangular array. To refer to a particular element of a data set so structured, it is necessary only to prescribe another n-tuple of, say, x_i's, where x_i is (an integer) less than or equal to a_i. Thus, a name (the same as in the descriptor) followed by an n-tuple is the general form of the reference expression for such structures. The typical storage mapping function in these cases is a program which computes

$$(x_n - 1) + \sum_{i=1}^{n-1} \left(\prod_{j=i+1}^{n} a_j \right) (x_i - 1)$$

*Now with Kettelle & Wagner, Consultants, Paoli, Pennsylvania

The range of values for this expression is from 0 to $(\pi_{k=1}^{n} a_k) - 1$. These values are typically added to some base address to determine the position of the referenced item of data. In general, the quantities

$$p_i = \prod_{j=i+1}^{n} a_j, \quad i = 0, 1, 2, \ldots, n - 1$$

are computed at compile time. In addition, p_0 units of memory are reserved for the overall data set, and the quantities $p_1, p_2, \ldots, p_{n-1}$ are stored to be used as coefficients in determining specific positions when x_i's are specified at run time.

The following discussion defines a class of data structures that is more general than the n-dimensional rectangular array, as well as a description method and a form for the reference expressions which are entirely suitable for inclusion in high-level symbolic programs. The discussion also derives the form of the storage mapping function for structures so described.

Generalized Structures

This section describes and defines a class of data structures; the class includes n-dimensional rectangular arrays as special cases. This generalization is not an idle one, but is motivated by the fact that, although any such structure can, by suitable manipulation, be eventually incorporated into n-dimensional arrays, it is very often neither convenient nor efficient to do so. The result of such incorporation is often a "sparse" matrix which is wasteful of memory. That any such structure can be eventually described by an n-dimensional array follows from the fact that any such structure is mappable into a linear (one-dimensional) (n = 1) array. Furthermore, in this day of procedure-oriented languages, the description technique which is most natural is the preferred one, and the contention here is that the structures and description technique to be defined are the natural ones for many data sets.

An English language description of the set of structures under consideration might help in understanding the formal definition to follow. The structures under consideration are, broadly, those that can be presented in an "outline" or "list" or "level" form, with the additional property that multiple instances of any "level" can occur. For example, a personnel file might be made up of 40 records, each record being composed of, say, three distinct items -- name, salary history, and work evaluation list. The item "salary history" may allow for the ten most recent raises and dates; the item "work evaluation list" may provide for the ten most recent ratings. In outline or list form, the items might be stated as

Personnel Record (40 instances)
- Name
- Salary History (10 instances)
  - Date
  - New Salary
- Evaluation List (10 instances)
  - Rating

For each item in the structure that is not a heading for a list of subitems -- that is, for each datum -- the description must assign the number of data "particles" of which the item is composed. The particles may be bits, characters, bytes, etc., but it is assumed that, whatever the basic particle is, it is the same throughout. Thus, for each of the data items (name, date, new salary, rating) in the above example, a number of characters, say, would be assigned.

It should be understood that whenever multiple instances are indicated, the structures below each instance are all identical.

Generalized Structure Descriptors

The set of descriptions of these structures are now defined in a formal manner, using the now well-known Backus definition form.[1] In effect, a set of strings of symbols are defined, which, when given the proper interpretation, are, in each case, a description of the kind of data structure discussed above.

Remarks similar to those above, concerning the difference between a skeletal structure with free variable names, and particular instances of such structures where these variables are bound by context, must also be made here. The difference will little affect this discussion, and, for convenience, the identifiers will be assumed to be contextually meaningful. The metalinguistic variables ::=, <, >, and | will be employed, and the classes <INTEGER> and <IDENTIFIER> are assumed to be already defined. Let the class of formal representations of these descriptions be called FIELD, where this is defined by

<FIELD> ::= <IDENTIFIER> (<INTEGER>) | <IDENTIFIER> (<INTEGER>, <FIELD SEQUENCE>);
<FIELD SEQUENCE> ::= <FIELD> | <FIELD SEQUENCE> <FIELD>.

Using single Roman capitals as IDENTIFIERs, the following examples of FIELDs can be given:

A (4)
A (4, B(5))
A (4, B(5) C(6))
A (4, B(5, D(3) E(2)) C(6)).

The personnel file above would have the form:

P (40, U(20) S(10, D(6) T(5)) E(10, R(2))).

Those structures definable by FIELDs will be referred to as generalized structures, and an interpreted FIELD will be referred to as a gen-

eralized structure descriptor, or, simply, a descriptor.

If FIELDs defined as follows:

<FIELD> ::= <IDENTIFIER> (<INTEGER>) |
<IDENTIFIER> (<INTEGER>, <FIELD>)

are considered, it is clear that these FIELDs represent a subset of the set of all FIELDs, and that this subset is equivalent to the set of n-dimensional arrays. The association proceeds as follows:

$A\ (a_1, a_2, \ldots, a_n)$ corresponds to

$A_1\ (a_1, A_2(a_2, \ldots, A_n(a_n)) \ldots)$.

There is, to be sure, a redundancy in the description when the generalized structure is, in fact, an n-dimensional array. This follows from the fact that there is descriptive information in the position of the elements of the n-tuple. It will be shown, when discussing the reference expressions, that an important feature of the generalized structure method is that there are no restrictions engendered by position.

Storage Mapping Functions and Reference Expressions

The class of structures under consideration, and the method of describing the structures in a symbolic program, have been defined by FIELDs; this section discusses the form of the reference expressions and the storage mapping function, as well as the processing of the FIELDs that is carried out at compile time. For the discussion, it is convenient to consider a class of pictorial representations known as L-trees -- trees that have legs, as well as branches.

It is well known how "lists" or "outlines" can be uniquely associated with trees. The structures discussed here, which can exist as multiple copies of identical substructures, require another type of branching symbolism -- thus, the legs.

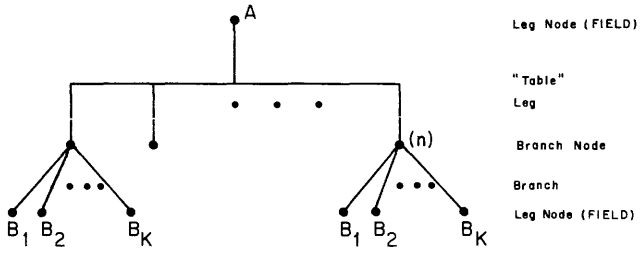


Fig. 1. L-Tree Representation of the FIELD $A(n, B_1(\ ) \ldots B_k(\ ))$

In general, the FIELD $A(n, B_1(\ )\ B_2(\ ) \ldots B_K(\ ))$ has the representation indicated in Fig. 1. As shown, the number of legs emanating from the table below leg node A is n. Each leg terminates in a branch node from which emanates the K branches of the FIELD SEQUENCE. These branches, along with the integer n, define FIELD A. Further, if the FIELD is a special one of the form A(n), it is represented as indicated by Fig. 2. In this figure, the n endpoints have no further emanations and represent particles of data. In short, to

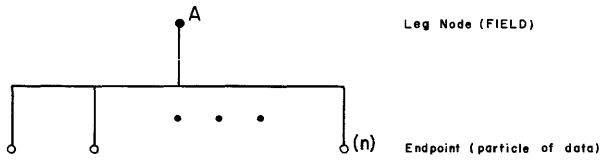


Fig. 2. L-Tree Representation of the FIELD A(n)

each FIELD corresponds a leg node, and to each FIELD SEQUENCE correspond n branch nodes, below each of which the structure is identical. The branch nodes are not named, but they do have a natural order, and, in a picture, an integer indicating the number of branch nodes present will follow the last of a set of identically structured branch nodes, if they are not all drawn.

This pictorial representation clarifies the manner of referring to a specific particle of data. Since leg nodes are referenced by identifiers, and branch nodes by integers, reference to a specific particle of data is accomplished by specifying a sequence of identifier-integer couples, rather than a sequence of integers (with one identifier) as for rectangular arrays. The identifier portion of the couple refers to a leg node, and the integer part to one of the n branch nodes immediately following the referenced leg node -- or, in the special case, the integer part of the couple refers to a particular particle.

More formally, then, a reference expression is defined as a set of couples $\{(A_i, x_i)\}$ for i = 1, 2, ... , k, such that A_i is an identifier and x_i is an integer. Thus, if there is, in the descriptor, $A_i(a_i, B_1(\ ) \ldots B_j(\ ))$, then $1 \le x_i \le a_i$, and A_{i+1} is one of the B's, or, if the form $A_j(a_j)$ appears in the descriptor, then k = j.

Thus, the example given above, namely:

P(40, U(20) S(10, D(6) T(5)) E(10, R(2))),

has the L-tree indicated in Fig. 3. The arrows in the figure point to the particles referred to by the following reference expressions:

{(P, 2), (U, 2)}
{(P, 2), (S, 3), (T, 4)}
{(P, 2), (E, 2), (R, 1)}.

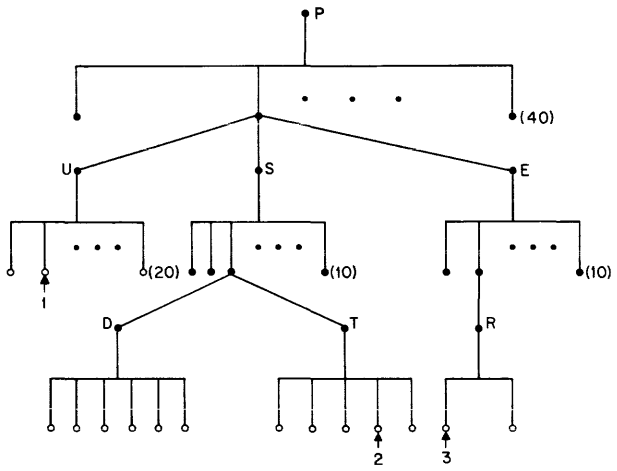


Fig. 3. L-Tree for P(40, U(20) S(10, D(6) T(5)) E(10, R(2)))

It is now clear that the uniqueness of reference is not destroyed if the couples in a reference expression are rearranged from the natural ordering. This characteristic is at least partial repayment for the redundancy which is present when the structure is an n-dimensional rectangular array.

For rectangular arrays, there is a natural way to map the set onto a linear array. The effect of this mapping procedure is that $(v_1, v_2, \ldots, v_n)$ precedes $(w_1, w_2, \ldots, w_n)$ in the linear array if and only if

1. $v_1 < w_1$, or
2. There exists a t such that $v_i = w_i$ for $i < t$ and $v_t < w_t$.

Thus,

(1, 1, ... , 1) is in the 0th position
(1, 1, ... , 2) is in the 1st position
.
.

$(1, 1, \ldots, 1, a_n)$ is in the $(a_n - 1)$st position
$(1, 1, \ldots, 2, 1)$ is in the a_nth position
etc.

It is apparent that this ordering is precisely that provided by the storage mapping function previously given for n-dimensional arrays.

With the aid of L-trees, the natural mapping of the data particles onto a linear array may be discussed, as well as the storage mapping function that is implied, for the case of generalized structures.

At each node of an L-tree, a natural ordering on the legs or branches -- a left-to-right ordering -- can be defined. This, in turn, imparts an ordering on all of the data particles. To achieve this ordering, the tree is traversed according to the regime of moving always down the leftmost leg or branch not already traversed; when such downward movement cannot continue, the traversing moves to the next leg to the right, or, if there are none, back to the branch node at the next level up, then to the next branch to the right, if possible, or else up to the next level, and so forth. In this way, each particle is encountered once and only once; the order in which the particles are encountered is the order which will be generated by the storage mapping function to be defined for generalized structures.

This ordering can be more explicitly defined as follows. Let $\{(A_i, X_i)\}$ be the reference expression for data particle P_1, and $\{(C_j, Y_j)\}$ be the reference expression for P_2, where $i = 1, 2, \ldots, k_1$, $j = 1, 2, \ldots, k_2$, and $A_1 = C_1$. Then P_2 will precede P_1 in the ordering if

1. $Y_1 < X_1$, or
2. There exists a $t > 1$ such that

   $C_i = A_i$ and $Y_i = X_i$, for $i < t$, and

   C_t precedes A_t in the FIELD SEQUENCE that defines A_{t-1}, or
3. There exists a $t > 1$ such that

   $C_i = A_i$ and $Y_i = X_i$, for $i < t$, and

   $C_t = A_t$ and $Y_t < X_t$.

The position in the linear array, then, of a particle, P_1, referenced by $\{(A_i, X_i)\}$, $i = 1, \ldots, k$, is equal to the total number of particles that precede P_1, by virtue of 1, 2, or 3, above. It is clear that these conditions are mutually exclusive; that is, particle P_2 can precede P_1 by virtue of only one of these conditions, and if the conditions 2 or 3 are pertinent, then a specific t is involved. Thus, β_1 is defined to be the total number of particles that satisfy condition 1, α_t to be the total number of particles that satisfy condition 2 for t, and β_t to be the total number of particles that satisfy condition 3 for t, where $2 \leq t \leq k$. It follows from the above that the sets comprising these totals are disjoint sets and the number of particles in their union is the total number of particles preceding P_1. Thus, the position of P_1 is given by

$$\sum_{t=1}^{k} (\alpha_t + \beta_t),$$

where $\alpha_1 = 0$. What this means in terms of the L-tree is demonstrated next.

For each leg node (or FIELD), F, let M(F) be the total number of particles that lie below each of the n legs of F. If the legs of F are particles, M(F) = 1. Let N(F) = n · M(F). Note that, if $B_1, B_2, \ldots, B_j$ are the FIELDs in the FIELD SEQUENCE for F (that is, successive branches of a branch node), then $M(F) = \sum_{l=1}^{j} N(B_l)$. Let us define $Q(B_1) = 0$, and $Q(B_i) = \sum_{l=1}^{i-1} N(B_l)$ for $i = 2, \ldots, j$. Note that $Q(B_i)$ is the total number of particles below $B_1, B_2, \ldots, B_{i-1}$.

It should be clear that, for P_1 given by $\{(A_i, x_i)\}$, $i = 1, 2, \ldots, k$, $\beta_1 = M(A_1)(x_1 - 1)$, $\beta_t = M(A_t)(x_t - 1)$, and $\alpha_t = Q(A_t)$.

Thus, if for each FIELD (or IDENTIFIER), F, the quantities Q(F) and M(F) are computed, the particle referenced by $\{(A_i, x_i)\}$ is to be found (or placed) in the position of the linear storage numbered:

$$\sum_{i=1}^{k} [Q(A_i) + M(A_i)(x_i - 1)],$$

where $Q(A_1)$ is set equal to 0.

The Q's and M's can be computed at compile time quite readily, given the generalized structure descriptor. The amount of storage reserved for such a structure is $N(A_1) = a_1 M(A_1)$ particle positions. At run time, when the x_i's and A_i's are supplied, the relative address is easily computed. Note that the A_i's must be supplied in this general case, in addition to the x_i's -- for n-dimensional arrays only the x_i's need be given -- since not all particles are derived from the same set of A's, although all of the particles have A_1 in common.

It is interesting to note the relationship of these quantities to the quantities in the storage mapping function of an n-dimensional rectangular array. Let the generalized structure descriptor be $A_1(a_1, A_2(a_2, \ldots, A_k(a_k)) \ldots)$. Since each FIELD SEQUENCE has only one FIELD, $Q(A_i) = 0$ for each i. Now, $M(A_k) = 1$, $N(A_k) = a_k \cdot 1 = a_k$, and $M(A_{k-1}) = N(A_k) = a_k$. In general, $M(A_{j-1}) = N(A_j) = a_j \cdot M(A_j)$. Thus, by induction,

$$M(A_j) = \prod_{i=j+1}^{k} a_i, \quad \text{for } j = 1, \ldots, k - 1.$$

This is precisely the expression for the p's derived above for rectangular arrays.

This completes the assertion that the description method, reference expressions, and storage mapping functions for generalized structures are direct generalizations of those for n-dimensional rectangular arrays.

Storage Mapping Function Computation

Appendix A is a Burroughs ALGOL program[2] for the Burroughs 220 computer which simulates, first, the computations carried out by a computer upon receiving the generalized structure descriptor at compile time, then the computations that would be carried out at run time when presented with reference expressions.

The first task of the program is to place the input FIELD into a vector, one word for each symbol. A table is then constructed which has one row for each identifier or FIELD of the input. This table is, in some respects, similar to a Perlis[3] threaded list; the same names for the columns have, in fact, been adopted.

The f column of the table is defined, for the FIELD (or IDENTIFIER or ROW) named A, as follows. If A is a FIELD of the form A(n), then f(A) is 2 or 0, depending upon whether or not A is the last FIELD in a FIELD SEQUENCE. If A is a FIELD of the form A(n, B_1( ) ...), then f(A) is 3 or 1, depending, again, upon whether or not A is the last FIELD in a FIELD SEQUENCE.

The ℓ column is defined as follows. If A is of the form A(n, B_1( ) B_2( ) ...), then ℓ(A) is B_1; if A is of the form A(n), ℓ(A) is undefined.

For the r column, if A is the overall FIELD, r is undefined. If f(A) is 0 or 1, then r(A) is the succeeding FIELD in the FIELD SEQUENCE in which A occurs. Otherwise (that is, f(A) is 2 or 3), r is the FIELD which is defined by the FIELD SEQUENCE of which A is the last FIELD in the sequence.

Finally, a c column is defined; this column has no counterpart in threaded lists. For each FIELD, A, the c column contains the associated INTEGER in that descriptor; that is, whether A is of the form A(n) or A(n, B_1( ) ...), c for that A is n. No information appears in this table that is not in the original descriptor, although the table is now in a more usable form for computing the N's, M's and Q's.

If f(A) is 0 or 2, M(A) is set equal to 1, and N(A) is set equal to c(A). From this point on, the task is reduced to finding a FIELD (defined by a FIELD SEQUENCE) all of whose FIELD SEQUENCE components have their N's already defined. Thus, if A is defined by A(n, B_1( ) ... B_k( )), then

$$M(A) = \sum_{i=1}^{k} N(B_i),$$

and N(A) = c(A) · M(A), where, of course, c(A) = n, in this case. At this time $Q(B_i)$ can also be computed for these i's by

$$Q(B_1) = 0$$

$$Q(B_j) = \sum_{i=1}^{j-1} N(B_i) = Q(B_{j-1}) + N(B_{j-1}).$$

This process is iterated until N(A) is found for the A which is the overall FIELD being processed.

This processing is carried out by constructing an incidence matrix, G, by using the f, ℓ, and r columns, such that the matrix has as many rows and columns as there are FIELDs, and has a 1 in row A, column B, if B is in the FIELD SEQUENCE defining A. The A's for which M's are ready to be computed are then determined by comparing the rows of G to a vector which has 1's for those FIELDS for which N is already determined.

The G matrix is constructed by finding, for each FIELD, A, for which f(A) is 1 or 3, the constituents of its FIELD SEQUENCE, by looking at ℓ(A), r(ℓ(A), r(r(ℓ(A))), etc., until r(. . . r(ℓ(A)) . . .) = A. All of the FIELDs so encountered are constituents. The order in which the FIELDs are encountered is later used (when the N's for the FIELDs are known) in computing the Q's of these constitutents of the FIELD SEQUENCE, A.

These computations are now summarized for the example above. The input, as before, is

P(40, U(20) S(10, D(6) T(5)) E(10, R(2))).

The following table is constructed (where * means undefined):

	f	ℓ	r	c
P	1	U	*	40
U	0	*	S	20
S	1	D	E	10
D	0	*	T	6
T	2	*	S	5
E	3	R	P	10
R	2	*	E	2

The following are then computed in order:

M(U) = 1
N(U) = 20
M(D) = 1
N(D) = 6
M(T) = 1
N(T) = 5
M(R) = 1
N(R) = 2
Q(R) = 0
M(S) = N(D) + N(T) = 11
N(S) = 10·11 = 110
Q(D) = 0
Q(T) = 6
M(E) = N(R) = 2
N(E) = 10·2 = 20
M(P) = N(U) + N(S) + N(E) = 150
N(P) = 40·150 = 6000
Q(U) = 0
Q(S) = N(U) = 20
Q(E) = Q(S) + N(S) = 130
Q(P) = 0.

Thus, in tabular form:

	M	N	Q
P	150	6000	0
U	1	20	0
S	11	110	20
D	1	6	0
T	1	5	6
E	2	20	130
R	1	2	0

The 6000 in this table indicates the total storage required for this FIELD; beyond this point, only the M and Q columns need be retained for processing the reference expressions at run time.

Next to be considered is the manner in which this data would be used to determine the position indicated by the reference expression ⌈(P, 2), (S, 3), (T, 4)⌉. Using the storage mapping function,

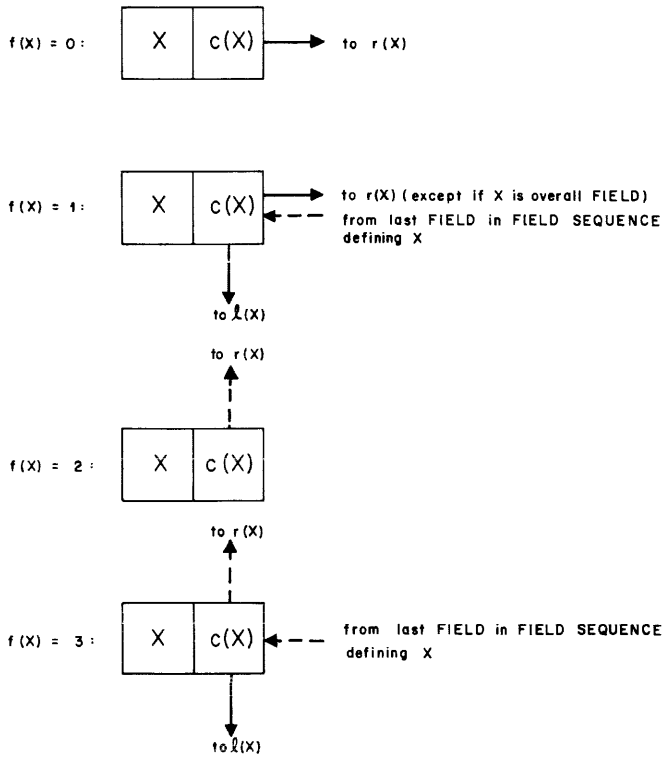


Fig. 4. Definition of "Boxes" in Terms of the Quantities f(x), ℓ(x), r(x), and c(x)

$$\Sigma[Q(A_i) + (x_i - 1)M(A_i)] = [0 + (2 - 1)\cdot 150] + [20 + (3 - 1)\cdot 11] + [6 + (4 - 1)\cdot 1]$$
$$= 150 + 42 + 9 = 201$$

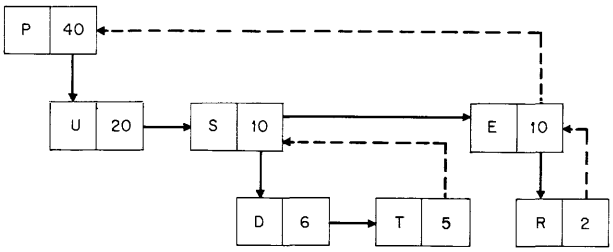


Fig. 5. "Box" Diagram for P(40, U(20) S(10, D(6) T(5)) E(10, R(2)))

There is another useful pictorial method for representing the structures under consideration. This method is easily derived directly from the descriptor. Each FIELD has a "box." Two sections are associated with each box. The first section contains the IDENTIFIER, X, of the FIELD; the second section contains c(X). There are arrows emanating from and entering the boxes, the nature of which depends upon f(X). This dependence is given in Fig. 4. Using these definitions, the descriptor

P(40, U(20) S(10, D(6) T(5)) E(10, R(2)))

can be indicated as in Fig. 5.

Further Considerations

There are several questions concerning generalized structures which are still under consideration, but for which results cannot be presented at this time.

The first of these questions involves a more general definition of FIELD which allows a (previously defined) FIELD to be referred to by FIELD name, alone, without indicating the associated FIELD SEQUENCE, or the number of data particles. In addition, the generalization from INTEGER to ARITHMETIC EXPRESSION is highly desirable, but will, of course, delay the M and Q computations until run time. The same delay is incurred in computing the coefficients for the storage mapping function for n-dimensional rectangular arrays, when arithmetic expressions are used in the descriptor.

A second area of present and future interest is that of nonhomogeneous memory and/or data. For example, the particles of data may not be the same throughout. If single bits are used in some places, and six-bit characters in others, for example, then clearly the descriptor, reference expressions, and mapping function must reflect this. The problems here seem reasonable to handle, but, of course, the descriptor definition must be modified.

In keeping with the notion, alluded to before, that structure definition is distinct from instances of that structure, the IDENTIFIERs could be removed from a descriptor, and the remains considered as a structure definition. In terms of the L-tree, the effect would be merely to remove the IDENTIFIERs -- the tree does not change. In declaring data which has such a structure, it would be necessary only to supply a sequence of IDENTIFIERs which could be interleaved with the skeletal representation, placing one before each left parenthesis. Using the example once again, the structure could be defined as

(40, (20) (10, (6) (5)) (10, (2))).

The instance noted above could now be declared by naming this skeletal structure and providing the sequence of IDENTIFIERs P, U, S, D, T, E, R. The M's and Q's can be determined from the skeletal structure alone, and are usable for any subsequent instance. All of this is, of course, a notational simplification for writing the structure descriptor in terms of free-variable IDENTIFIERs, and then binding these IDENTIFIERs by a sequence of IDENTIFIERs.

It is interesting to consider the set of such skeletal structures as being in reality a generalization of all Euclidean spaces; that is,

<GENERALIZED POINT> ::= (<INTEGER>) | (<INTEGER>, <GENERALIZED-SEQUENCE>);
<GENERALIZED SEQUENCE> ::= <GENERALIZED POINT> | <GENERALIZED SEQUENCE> <GENERALIZED POINT>

defines a set of formal objects, <GENERALIZED POINT>, which is a direct generalization of the collection of all n-tuples of integers for all n.

Acknowledgment

The author wishes to thank Mr. K. Speierman, who suggested the value of a study in this area, and who made many useful contributions to the completion of the work.

References

1. J. W. Backus, "The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference," presented at the First International Conference on Information Processing (ICIP), June 13-23, 1959, in Paris, France (IBM Corporation, New York).

2. (Burroughs Corporation), "Burroughs Algebraic Compiler - A Reference Manual," Bulletin 220-21011-P, January 1961.

3. Alan J. Perlis and Charles Thornton, "Symbol Manipulation by Threaded Lists," Communications of the ACM, April 1960, Vol. 3, No. 4, pp. 195-204.

Appendix A - Burroughs ALGOL Program

The following is taken directly from the printer output of the Burroughs 220 computer, as employed for the Burroughs ALGOL program as described above.

```
COMMENT
            THIS PROGRAM TAKES A GENERALIZED DATA STRUCTURE FIELD
            AS INPUT AND GENERATES A TABLE UPON WHICH THE GENERALIZED
            LINEAR STORAGE MAPPING FUNCTION OPERATES.THE INPUT
            REQUIRES EACHSYMBOL IN THE FIELD TO BE ON A SEPARATE CARD
           CODED IN THE 8220 ALPHANUMERIC CODE. EACH IDENTIFIER AND
            INTEGER MUST BE ONE  CHARACTER LONG. LET THE NUMBER OF
            CARDS BE KM.$
COMMENT     GIVEN THE INPUT FIELD. THE OUTPUT WILL BE IN THE  FORM
            OF A TABLE  WITH ONE ROW FOR EACH IDENNTIFIER. THESE
           ARE LISTED UNDER  THE HEADING B(J). THE OTHER COLUMNS
           ARE J. F(J). L(J). R(J). C(J). N(J). M(J). AND G(J).
             WHEN THIS IS COMPLETED. THE PROGRAM IS READY TO ACCEPT
           REFERENCE EXPRESSIONS$
INTEGER     OTHERWISE$
ARRAY       IN(100),B(25),P(25),C(25),L(25),S(25),K(25),KP(25),M(25),
  F(25),R(25),RN(25),LN(25),SIG(25),TAU(25),N(25),G(25,25),
            T(25),TN(25),Q(25)$
 INPUT     FIELD(KM,FOR I =(1,1,KM)$IN(I))$
FORMAT       HEAD (B9,*J*,B6,*B(J)*,B6,*F(J)*,B6,*L(J)*,B6,*R(H)*,B6,
            *C(J)*,B6,*N(J))*,B6,*M(J)*,B6,*G(J)*,W2)$
FORMAT    FORM(I10,B9,A1,I10,B9,A1,B9,A1,I10,I10,I10,I10,W0)$
 OUTPUT   TABLE(FOR J=(1,1,JM)$(J,B(J),F(J),L(J),R(J),C(J),N(J),
            M(J),Q(J)))$
TAKEIN..    READ($$FIELD)$
COMMENT     THIS ASSIGNS SUCCESSIVE VALUES OF J TO INPUT
            FIELD IDENTIFIERS. IDENTIFIERS ARE STORED IN B(J).
            THE POSITION IN THE INPUT STRING OF B(J) IS IN P(J).
            JM IS THE LIMIT OF J$
COMMENT     NOTE THAT LEFT PARENTHENSES ARE STORED AS
            0400000000. RIGHTS AS 2400000000. AND COMMAS AS
            2300000000.  INTEGERS 0 TO 9 ARE STORED AS 8000000000.
            8100000000. ETC.$
            J=1$
            FOR KA=(1,1,KM)$
               BEGIN
 IF (IN(KA) NEQ 0400000000) AND (IN(KA) NEQ 2400000000)
  AND (IN(KA) NEQ 2300000000) AND ((IN(KA)/10000000000) NEQ 8)$
                  BEGIN
                  B(J)=IN(KA)$
                    P(J)=KA$
                    J=J+1
                  END$
               END$
            JM=J-1$
COMMENT     THIS ASSIGNS NUMBER OF COPIES OF JTH FIELD IDENTIFIER
            TO C(J) AND ASSIGNS L(J) WHENEVER B(J) IS A LIST HEAD$
            FOR J=(1,1,JM)$
               BEGIN
                  C(J)=(IN(P(J)+2)-8000000000)/100000000$
                  IF IN(P(J)+3) EQL 2300000000$
                    L(J)=IN(P(J)+4)$
               END$
COMMENT     THIS FINDS THE POSITION, S(J), IN TH INPUT STRING OF
            THE RIGHT PARENTHESIS FOR THE JTH FIELD DENTIFIERS$
            FOR J=(1,1,JM)$
            BEGIN
                  CTR=1$
               KB=P(J)+2$
DON..
                  IF CTR NEQ 0$
                    BEGIN
    IF IN(KB) EQL 0400000000$
                              CTR=CTR+1$
     IF IN(KB) EQL 2400000000$
                              CTR=CTR-1$
                           KB=KB+1$
            GO TO DON$
                    END$
                  S(J)=KB-1$
               END$
COMMENT     THIS GENERATES F(J),R(J),LN(J),RN(J). THE LATTER TWO
            ARE RESPECTIVELY B INVERSE OF L(J) AND B INVERSE OF R(J).
            THAT IS, B(LN(J))=L(J), B(RN(J))=R(J).  IN CASE F(J)=0,
            OR 2, THIS SETS N(J)=C(J), M(J)=1, AND K(J)=1.  IN
            GENERAL K(J) WILL EQUAL 1 ONCE N(J) HAS BEEN
            DETERMINED$
            F(1)=1$
            R(1)=0$
            RN(1)=0$
            FOR KA=(1,1,JM)$
              IF B(KA) EQL L(1)$
              LN(1)=KA$
            FOR J=(2,1,JM)$
                  BEGIN
                        SJ=S(J)+1$
                     PJ=P(J)+3$
 EITHER IF (IN(SJ) NEQ 2400000000) AND (IN(PJ) NEQ 2300000000)$
                        BEGIN
                           KP(J)=1$
                           K(J)=1$
                           M(J)=1$
                           G(J,J)=1$
                           N(J)=C(J)$
                           F(J)=0$
                        R(J)=IN(SJ)$
                           FOR KA=(1,1,JM)$
                              BEGIN
                                 IF B(KA) EQL R(J)$
                                 RN(J)=KA$
                              END$
                        END$
  OR IF (IN(SJ) NEQ 2400000000) AND (IN(PJ) EQL 2300000000)$
                        BEGIN
                           F(J)=1$
                           R(J)=IN(SJ)$
                           FOR KA=(1,1,JM)$
                              BEGIN
                                 IF B(KA) EQL R(J)$
                                 RN(J)=KA$
                                 IF B(KA) EQL L(J)$
                                    LN(J) = KA$
                              END$
                        END$
    OR IF (IN(SJ) EQL 2400000000) AND (IN(PJ) NEQ 2300000000)$
                        BEGIN
                           KP(J)=1$
                           K(J)=1$

                           M(J)=1$
                           N(J)=C(J)$
                           F(J)=2$
                           KA=1$
 KAPPA..       IF S(KA) NEQ SJ$
                              BEGIN
                                 KA=KA+1$
                                 GO TO KAPPA$
                              END$
                           R(J)=B(KA)$
                           RN(J)=KA$
                        END$
                     OTHERWISE$
                        BEGIN
                           F(J)=3$
                           KA=1$
LAMBDA..                   IF S(KA) NEQ SJ$
                              BEGIN
                                 KA=KA+1$
                                 GO TO LAMBDA$
                              END$
                           R(J)=B(KA)$
                        RN(J)=KA$
                           FOR KA=(1,1,JM)$
                              BEGIN
                                 IF B(KA) EQL L(J)$
                                 LN(J)=KA$
                              END$
                        END$
               END$
COMMENT     THIS FILLS THE ROWS OF G FOR F(J)=1 OR 3$
 FOR J=(1,1,JM)$
 BEGIN  IF (F(J) EQL 1) OR (F(J) EQL 3)$
 BEGIN  X=LN(J)$
 BETA..  G(J,X)=1$
 IF RN(X) NEQ J$
 BEGIN  X=RN(X)$
 GO TO BETA$
 END$ END$ END$
COMMENT     THIS FINDS THE SIG(J) VECTORS BY SUMMING THE G(J,,ROWS$
MU..        FOR J = (1,1,JM)$
               BEGIN
                  SIG(J) = 0$
                  FOR I = (1,1,JM)$
                     SIG(J) = SIG(J) + G(J,I)$
               END$
COMMENT     THIS FINDS SMALLEST J FOR WHICH SUM G(J,I)K(I) = SIG(J) ,
            AND FOR WHICH K EQUALS 0, AND COMPUTES
            M(J) AND N(J).  REPEATS UNTIL NO J,S REMAIN$
DELTA..     FOR J=(1,1,JM)$
               BEGIN
                  TAU(J)=0$
                  FOR I = (1,1,JM)$
                 TAU(J)=TAU(J)+G(J,I),K(I)$
               END$
            IY=0$
             FOR I = (1,1,JM)$
               BEGIN
         IF (TAU(I) EQL SIG(I)) AND (K(I) EQL 0)$
                     BEGIN
                        IY = 1$
 M(I)=0$
                        FOR IX = (1,1,JM)$
 M(I)=M(I)+G(I,IX),N(IX)$
 N(I)=C(I),M(I)$
 K(I)=1$
                     END$
               END$
             IF IY EQL 1$
               GO TO DELTA$
COMMENT     THIS COMPUTES Q VECTOR $
            Q(1) = 0$
            I=1$
EPSILON..   IF (F(I) EQL 0) OR (F(I) EQL 2)$
```

```
                BEGIN
                   IF I EQL JM$
                  GO TO PUTOUT$
                   I = I+1 $
                   GO TO EPSILON$
                END$
 Q(LN(I))=0$
 IV=LN(I)$
 ZETA..  IF RN(IV) EQL I$
                BEGIN
                   IF I EQL JM$
             GO TO PUTOUT$
                   I = I+1$
                   GO TO EPSILON$
                END $
             Q(RN(IV)) = N(IV) + Q(IV)$
             IV = RN(IV)$
             GO TO ZETA$
PUTOUT..     WRITE($$ HEAD)$
             WRITE ($$ TABLE,FORM)$
COMMENT      AT THIS POINT THE PROGRAM WILL ACCEPT  REFERENCE
            EXPRESSIONS IN THE FORM IDENTIFIER, INTEGER, IDENTIFIER,
            INTEGER ETC. EACH IS ON  A SEPARATE CARD AND ALPHANU-
            MERICALLY CODED. THEY MUST BE PRECEDED BY A CARD
             CONTAINING TZ, THE NUMBER OF CARDS IN THE
            REFERENCE EXPRESSION$
 INPUT    TEST(TZ,FOR I=(1,1,TZ)$T(I))$
    OUTPUT  POSITION(SMF)$
    FORMAT  POS(I10,W0)$
 TESTSTART..
    READ     ($$TEST)$
             FOR I=(1,1,TZ/2)$
                BEGIN
                  T(2I)=(T(2I)-8000000000)/100000000$
                   FOR J=(1,1,JM)$
                      BEGIN
  IF B(J) EQL T(2I-1)$
                                TN(I)=J$
                      END$
                END$
COMMENT      THIS IS THE STORAGE MAPPING FUNCTION COMPUTATIONS$
             SMF=0$
             FOR I=(1,1,TZ/2)$
          SMF=SMF+M(TN(I)),(T(2I)-1)+Q(TN(I))$
            WRITE  ($$POSITION,POS)$
             GO TO TESTSTART$
THEEND..     FINISH$
```

AN EXPERIMENTAL TIME-SHARING SYSTEM

Fernando J. Corbató, Marjorie Merwin-Daggett, Robert C. Daley

Computer Center, Massachusetts Institute of Technology

Cambridge, Massachusetts

Summary

It is the purpose of this paper to discuss briefly the need for time-sharing, some of the implementation problems, an experimental time-sharing system which has been developed for the contemporary IBM 7090, and finally a scheduling algorithm of one of us (FJC) that illustrates some of the techniques which may be employed to enhance and be analyzed for the performance limits of such a time-sharing system.

Introduction

The last dozen years of computer usage have seen great strides. In the early 1950's, the problems solved were largely in the construction and maintenance of hardware; in the mid-1950's, the usage languages were greatly improved with the advent of compilers; now in the early 1960's, we are in the midst of a third major modification to computer usage: the improvement of man-machine interaction by a process called time-sharing.

Much of the time-sharing philosophy, expressed in this paper, has been developed in conjunction with the work of an MIT preliminary study committee, chaired by H. Teager, which examined the long range computational needs of the Institute, and a subsequent MIT computer working committee, chaired by J. McCarthy. However, the views and conclusions expressed in this paper should be taken as solely those of the present authors.

Before proceeding further, it is best to give a more precise interpretation to time-sharing. One can mean using different parts of the hardware at the same time for different tasks, or one can mean several persons making use of the computer at the same time. The first meaning, often called multiprogramming, is oriented towards hardware efficiency in the sense of attempting to attain complete utilization of all components[5,6,7,8]. The second meaning of time-sharing, which is meant here, is primarily concerned with the efficiency of persons trying to use a computer[1,2,3,4]. Computer efficiency should still be considered but only in the perspective of the total system utility.

The motivation for time-shared computer usage arises out of the slow man-computer interaction rate presently possible with the bigger, more advanced computers. This rate has changed little (and has become worse in some cases) in the last decade of widespread computer use.[10]

In part, this effect has been due to the fact that as elementary problems become mastered on the computer, more complex problems immediately become of interest. As a result, larger and more complicated programs are written to take advantage of larger and faster computers. This process inevitably leads to more programming errors and a longer period of time required for debugging. Using current batch monitor techniques, as is done on most large computers, each program bug usually requires several hours to eliminate, if not a complete day. The only alternative presently available is for the programmer to attempt to debug directly at the computer, a process which is grossly wasteful of computer time and hampered seriously by the poor console communication usually available. Even if a typewriter is the console, there are usually lacking the sophisticated query and response programs which are vitally necessary to allow effective interaction. Thus, what is desired is to drastically increase the rate of interaction between the programmer and the computer without large economic loss and also to make each interaction more meaningful by extensive and complex system programming to assist in the man-computer communication.

To solve these interaction problems we would like to have a computer made simultaneously available to many users in a manner somewhat like a telephone exchange. Each user would be able to use a console at his own pace and without concern for the activity of others using the system. This console could as a minimum be merely a typewriter but more ideally would contain an incrementally modifiable self-sustaining display. In any case, data transmission requirements should be such that it would be no major obstacle to have remote installation from the computer proper.

The basic technique for a time-sharing system is to have many persons simultaneously using the computer through typewriter consoles with a time-sharing supervisor program sequentially running each user program in a short burst or quantum of computation. This sequence, which in the most straightforward case is a simple round-robin, should occur often enough so that each user program which is kept in the high-speed memory is run for a quantum at least once during each approximate human reaction time ($\sim$.2 seconds). In this way, each user sees a computer fully responsive to even single key strokes each of which may require only trivial computation; in the non-trivial cases, the user sees a gradual reduction of the response time which is proportional to the complexity of the response calculation, the slowness of the computer, and the total number of active users. It should be clear, however, that if there are n users actively requesting service at one time, each user will only see on the average 1/n of the effective computer speed. During the period of high interaction rates while debugging programs, this should not be a hindrance since ordinarily the required amount of computation needed for each debugging computer response is small compared to the ultimate production need.

Not only would such a time-sharing system improve the ability to program in the conventional manner by one or two orders of magnitude, but there would be opened up several new forms of computer usage. There would be a gradual reformulation of many scientific and engineering applications so that programs containing decision trees which currently must be specified in advance would be eliminated and instead the particular decision branches would be specified only as needed. Another important area is that of teaching machines which, although frequently trivial computationally, could naturally exploit the consoles of a time-sharing system with the additional bonus that more elaborate and adaptive teaching programs could be used. Finally, as attested by the many small business computers, there are numerous applications in business and in industry where it would be advantageous to have powerful computing facilities available at isolated locations with only the incremental capital investment of each console. But it is important to realize that even without the above and other new applications, the major advance in programming intimacy available from time-sharing would be of immediate value to computer installations in universities, research laboratories, and engineering firms where program debugging is a major problem.

Implementation Problems

As indicated, a straightforward plan for time-sharing is to execute user programs for small quantums of computation without priority in a simple round-robin; the strategy of time-sharing can be more complex as will be shown later, but the above simple scheme is an adequate solution. There are still many problems, however, some best solved by hardware, others affecting the programming conventions and practices. A few of the more obvious problems are summarized:

Hardware Problems:

1. Different user programs if simultaneously in core memory may interfere with each other or the supervisor program so some form of memory protection mode should be available when operating user programs.

2. The time-sharing supervisor may need at different times to run a particular program from several locations. (Loading relocation bits are no help since the supervisor does not know how to relocate the accumulator, etc.) Dynamic relocation of all memory accesses that pick up instructions or data words is one effective solution.

3. Input-output equipment may be initiated by a user and read words in on another user program. A way to avoid this is to trap all input-output instructions issued by a user's program when operated in the memory protection mode.

4. A large random-access back-up storage is desirable for general program storage files for all users. Present large capacity disc units appear to be adequate.

5. The time-sharing supervisor must be able to interrupt a user's program after a quantum of computation. A program-initiated one-shot multivibrator which generates an interrupt a fixed time later is adequate.

6. Large core memories (e.g. a million words) would ease the system programming complications immensely since the different active user programs as well as the frequently used system programs such as compilers, query programs, etc. could remain in core memory at all times.

Programming Problems:

1. The supervisor program must do automatic user usage charge accounting. In general,

the user should be charged on the basis of a system usage formula or algorithm which should include such factors as computation time, amount of high-speed memory required, rent of secondary memory storage, etc.

2. The supervisor program should coordinate all user input-output since it is not desirable to require a user program to remain constantly in memory during input-output limited operations. In addition, the supervisor must coordinate all usage of the central, shared high-speed input-output units serving all users as well as the clocks, disc units, etc.

3. The system programs available must be potent enough so that the user can think about his problem and not be hampered by coding details or typographical mistakes. Thus, compilers, query programs, post-mortem programs, loaders, and good editing programs are essential.

4. As much as possible, the users should be allowed the maximum programming flexibility both in choices of language and in the absence of restrictions.

Usage Problems

1. Too large a computation or excessive typewriter output may be inadvertently requested so that a special termination signal should be available to the user.

2. Since real-time is not computer usage-time, the supervisor must keep each user informed so that he can use his judgment regarding loops, etc.

3. Computer processor, memory and tape malfunctions must be expected. Basic operational questions such as "Which program is running?" must be answerable and recovery procedures fully anticipated.

An Experimental Time-Sharing System for the IBM 7090

Having briefly stated a desirable time-sharing performance, it is pertinent to ask what level of performance can be achieved with existant equipment. To begin to answer this question and to explore all the programming and operational aspects, an experimental time-sharing system has been developed. This system was originally written for the IBM 709 but has since been converted for use with the 7090 computer.

The 7090 of the MIT Computation Center has, in addition to three channels with 19 tape units, a fourth channel with the standard Direct Data Connection. Attached to the Direct Data Connection is a real-time equipment buffer and control rack designed and built under the direction of H. Teager and his group.* This rack has a variety of devices attached but the only ones required by the present systems are three flexowriter typewriters. Also installed on the 7090 are two special modifications (i.e. RPQ's): a standard 60 cycle accounting and interrupt clock, and a special mode which allows memory protection, dynamic relocation and trapping of all user attempts to initiate input-output instructions.

In the present system the time-sharing occurs between four users, three of whom are on-line each at a typewriter in a foreground system, and a fourth passive user of the background Fap-Mad-Madtran-BSS Monitor System similar to the Fortran-Fap-BSS Monitor System (FMS) used by most of the Center programmers and by many other 7090 installations.

Significant design features of the foreground system are:

1. It allows the user to develop programs in languages compatible with the background system,

2. Develop a private file of programs,

3. Start debugging sessions at the state of the previous session, and

4. Set his own pace with little waste of computer time.

Core storage is allocated such that all users operate in the upper 27,000 words with the time-sharing supervisor (TSS) permanently in the lower 5,000 words. To avoid memory allocation clashes, protect users from one another, and simplify the initial 709 system organization, only one user was kept in core memory at a time. However, with the special memory protection and relocation feature of the 7090, more sophisticated storage allocation procedures are being implemented. In any case, user swaps are minimized by using 2-channel overlapped magnetic tape reading and writing of the pertinent locations in the two user programs.

The foreground system is organized around commands that each user can give on his typewriter and the user's private program files which presently (for want of a disc unit) are kept on a separate magnetic tape for each user.

* This group is presently using another approach[9] in developing a time-sharing system for the MIT 7090.

For convenience the format of the private tape files is such that they are card images, have title cards with name and class designators and can be written or punched using the off-line equipment. (The latter feature also offers a crude form of large-scale input-output.) The magnetic tape requirements of the system are the seven tapes required for the normal functions of the background system, a system tape for the time-sharing supervisor that contains most of the command programs, and a private file tape and dump tape for each of the three foreground users.

The commands are typed by the user to the time-sharing supervisor (not to his own program) and thus can be initiated at any time regardless of the particular user program in memory. For similar coordination reasons, the supervisor handles all input-output of the foreground system typewriters. Commands are composed of segments separated by vertical strokes; the first segment is the command name and the remaining segments are parameters pertinent to the command. Each segment consists of the last 6 characters typed (starting with an implicit 6 blanks) so that spacing is an easy way to correct a typing mistake. A carriage return is the signal which initiates action on the command. Whenever a command is received by the supervisor, "WAIT", is typed back followed by "READY." when the command is completed. (The computer responses are always in the opposite color from the user's typing.) While typing, an incomplete command line may be ignored by the "quit" sequence of a code delete signal followed by a carriage return. Similarly after a command is initiated, it may be abandoned if a "quit" sequence is given. In addition, during unwanted command typeouts, the command and output may be terminated by pushing a special "stop output" button.

The use of the foreground system is initiated whenever a typewriter user completes a command line and is placed in a waiting command queue. Upon completion of each quantum, the time-sharing supervisor gives top priority to initiating any waiting commands. The system programs corresponding to most of the commands are kept on the special supervisor command system tape so that to avoid waste of computer time, the supervisor continues to operate the last user program until the desired command program on tape is positioned for reading. At this point, the last user is read out on his dump tape, the command program read in, placed in a working status and initiated as a new user program. However, before starting the new user for a quantum of computation, the supervisor again checks for any waiting command of another user and if necessary begins the look-ahead positioning of the command system tape while operating the new user.

Whenever the waiting command queue is empty, the supervisor proceeds to execute a simple round-robin of those foreground user programs in the working status queue. Finally, if both these queues are empty, the background user program is brought in and run a quantum at a time until further foreground system actively develops.

Foreground user programs leave the working status queue by two means. If the program proceeds to completion, it can reenter the supervisor in a way which eliminates itself and places the user in dead status; alternatively, by a different entry the program can be placed in a dormant status (or be manually placed by the user executing a quit sequence). The dormant status differs from the dead status in that the user may still restart or examine his program.

User input-output is through each typewriter, and even though the supervisor has a few lines of buffer space available, it is possible to become input-output limited. Consequently, there is an additional input-output wait status, similar to the dormant, which the user is automatically placed in by the supervisor program whenever input-output delays develop. When buffers become near empty on output or near full on input, the user program is automatically returned to the working status; thus waste of computer time is avoided.

Commands

To clarify the scope of the foreground system and to indicate the basic tools available to the user, a list of the important commands follows along with brief summaries of their operations:

1. | α

   α = arbitrary text treated as a comment.

2. login | α | β

   α = user problem number
   β = user programmer number

   Should be given at beginning of each user's session. Rewinds user's private file tape; clears time accounting records.

3. logout

   Should be given at end of each user's session. Rewinds user's private file tape; punches on-line time accounting cards.

4. input

   Sets user in input mode and initiates automatic generation of line numbers. The user

types a card image per line according to a format appropriate for the programming language. (The supervisor collects these card images at the end of the user's private file tape.) When in the automatic input mode, the manual mode may be entered by giving an initial carriage return and typing the appropriate line number followed by | and line for as many lines as desired. To reenter the automatic mode, an initial carriage return is given.

The manual mode allows the user to overwrite previous lines and to insert lines. (cf. File Command.)

5. edit | α | β

α = title of file

β = class of file

The user is set in the automatic input mode with the designated file treated as initial input lines. The same conventions apply as to the input command.

6. file | α | β

α = title to be given to file

β = class of language used during input

The created file will consist of the numbered input lines (i.e. those at the end of the user's private file tape) in sequence; in the case of duplicate line numbers, the last version will be used. The line numbers will be written as sequence numbers in the corresponding card images of the file.

For convenience the following editing conventions apply to input lines:

a. an underline signifies the deletion of the previous characters of the line.

b. a backspace signifies the deletion of the previous character in the field.

The following formats apply:

a. FAP: symbol, tab, operation, tab, variable field and comment.

b. MAD, MADTRAN, FORTRAN: statement label, tab, statement. To place a character in the continuation column: statement label, tab, backspace, character, statement.

c. DATA: cols. 1-72.

7. fap | α

Causes the file designated as α, fap to be translated by the FAP translator (assembler). Files α, symtb and α,bss are added to the user's private file tape giving the symbol table and the relocatable binary BSS form of the file.

8. mad | α

Causes file α,mad to be translated by the MADtranslator (compiler). File α,bss is created.

9. madtrn | α

Causes file α,madtrn (i.e. a pseudo-Fortran language file) to be edited into an equivalent file α,mad (added to the user's file) and translation occurs as if the command mad|α had been given.

10. load | α_1 | α_2|...|α_n

Causes the consecutive loading of files α_i,bss (i=1,2,...,n). An exception occurs if α_i=(libe), in which case file α_{i+1},bss is searched as a library file for all subprograms still missing. (There can be further library files.)

11. use | α_1 | α_2 |...| α_n

This command is used whenever a load or previous use command notifies the user of an incomplete set of subprograms. Same α_i conventions as for load.

12. start | α | β

Starts the program setup by the load and use commands (or a dormant program) after first positioning the user private file tape in front of the title card for file α,β. (If β is not given, a class of data is assumed; if both α and β are not given, no tape movement occurs and the program is started.)

13. pm | α

α = "lights", "stomap", or the usual format of the standard Center post-mortem (F2PM) request: subprogram name | loc_1 | loc_2 | mode | direction where mode and direction are optional.

Produces post-mortem of user's dormant program according to request specified by α. (E.g. matrix | 5 | 209 | flo | rev will cause to be printed on the user's typewriter the contents of subprogram "matrix" from relative locations 5 to 209 in floating point form and in reverse sequence.)

14. skippm

Used if a pm command is "quit" during output and the previous program interruption is to be restarted.

15. listf

Types out list of all file titles on user's private file tape.

16. printf | α | β | γ

Types out file α,β starting at line number γ. If γ is omitted, the initial line is assumed. Whenever the user's output buffer fills, the command program goes into an I/O wait status allowing other users to time-share until the buffer needs refilling.

17. xdump | α | β

Creates file α,β (if β omitted, xdump assumed) on user's private file tape consisting of the complete state of the user's last dormant program.

18. xdump | α | β

Inverse of xdump command in that it resets file α,β as the user's program, starting it where it last left off.

Although experience with the system to date is quite limited, first indications are that programmers would readily use such a system if it were generally available It is useful to ask, now that there is some operating experience with the 7090 system, what observations can be made. An immediate comment is that once a user gets accustomed to computer response, delays of even a fraction of a minute are exasperatingly long, an effect analogous to conversing with a slow-speaking person. Similarly, the requirement that a complete typewritten line rather than each character be the minimum unit of man-computer communication is an inhibiting factor in the sense that a press-to-talk radio-telephone conversation is more stilted than that of an ordinary telephone. Since maintaining a rapid computer response on a character by character basis requires at least a vestigial response program in core memory at all times, the straightforward solution within the present system is to have more core memory available. At the very least, an extra bank of memory for the time-sharing supervisor would ease compatibility problems with programs already written for 32,000 word 7090's.

For reasons of expediency, the weakest portions of the present system are the conventions for input, editing of user files, and the degree of rapid interaction and intimacy possible while debugging. Since to a large extent these areas involve the taste, habits, and psychology of the users, it is felt that proper solutions will require considerable experimentation and pragmatic evaluation; it is also clear that these areas cannot be treated in the abstract for the programming languages used will influence greatly the appropriate techniques. A greater use of symbolic referencing for locations, program names and variables is certainly desired; symbolic post-mortem programs, trace programs, and before-and-after differential dump programs should play useful roles in the debugging procedures.

In the design of the present system, great care went into making each user independent of the other users. However, it would be a useful extension of the system if this were not always the case. In particular, when several consoles are used in a computer controlled group such as in management or war games, in group behavior studies, or possibly in teaching machines, it would be desirable to have all the consoles communicating with a single program.

Another area for further improvement within the present system is that of file maintenance, since the presently used tape units are a hindrance to the easy deletion of user program files. Disc units will be of help in this area as well as with the problem of consolidating and scheduling large-scale central input-output generated by the many console users.

Finally, it is felt that it would be desirable to have the distinction between the foreground and background systems eliminated. The present-day computer operator would assume the role of a stand-in for the background users, using an operator console much like the other user consoles in the system, mounting and demounting magnetic tapes as requested by the supervisor, receiving instructions to read card decks into the central disc unit, etc. Similarly the foreground user, when satisfied with his program, would by means of his console and the supervisor program enter his program into the queue of production background work to be performed. With these procedures implemented the distinction of whether one is time-sharing or not would vanish and the computer user would be free to choose in an interchangable way that mode of operation which he found more suitable at a particular time.

A Multi-Level Scheduling Algorithm

Regardless of whether one has a million word core memory or a 32,000 word memory as currently exists on the 7090, one is inevitably faced with the problem of system saturation where the total size of active user programs exceeds that of the high-speed memory or there are too many active user programs to maintain an adequate response at each user console. These conditions can easily arise with even a few users if some of the user programs are excessive in size or in time requirements. The predicament can be alleviated if it is assumed that a good design for the system is to have a saturation procedure which gives graceful degradation of the response time and effective real-time computation speed of the large and long-running users.

To show the general problem, Figure 1 qualitatively gives the user service as a function of n, the number of active users. This service parameter might be either of the two key factors: computer response time or n times the real-time computation speed. In either case there is some critical number of active users, N, representing the effective user capacity, which causes saturation. If the strategy near saturation is to execute the simple round-robin of all users, then there is an abrupt collapse of service due to the sudden onset of the large amount of time required to swap programs in-and-out of the secondary memory such as a disc or drum unit. Of course, Figure 1 is quite qualitative since it depends critically on the spectrum of user program sizes as well as the spectrum of user operating times.

To illustrate the strategy that can be employed to improve the saturation performance of a time-sharing system, a multi-level scheduling algorithm is presented. This algorithm also can be analyzed to give broad bounds on the system performance.

The basis of the multi-level scheduling algorithm is to assign each user program as it enters the system to be run (or completes a response to a user) to an ℓth level priority queue. Programs are initially entered into a level ℓ_o, corresponding to their size such that

$$\ell_o = \left\lceil \log_2 \left(\left\lfloor \frac{w_p}{w_q} \right\rfloor + 1 \right) \right\rceil \qquad (1)$$

where w_p is the number of words in the program, w_q is the number of words which can be transmitted in and out of the high-speed memory from the secondary memory in the time of one quantum, q, and the bracket indicates "the integral part of". Ordinarily the time of a quantum, being the basic time unit, should be as small as possible without excessive overhead losses when the supervisor switches from one program in high-speed memory to another. The process starts with the time-sharing supervisor operating the program at the head of the lowest level occupied queue, ℓ, for up to 2^ℓ quanta of time and then if the program is not completed (i.e. has not made a response to the user) placing it at the end of the ℓ+1 level queue. If there are no programs entering the system at levels lower than ℓ, this process proceeds until the queue at level ℓ is exhausted; the process is then iteratively begun again at level ℓ+1, where now each program is run for $2^{\ell+1}$ quanta of time. If during the execution of the 2^ℓ quanta of a program at level ℓ, a lower level, ℓ', becomes occupied, the current user is replaced at the head of the ℓth queue and the process is reinitiated at level ℓ'.

Similarly, if a program of size w_p at level ℓ, during operation requests a change in memory size from the time-sharing supervisor, then the enlarged (or reduced) version of the program should be placed at the end of the ℓ'' queue where

$$\ell'' = \ell + \left\lceil \log_2 \left(\left\lfloor \frac{w_p''}{w_p} \right\rfloor + 1 \right) \right\rceil . \qquad (2)$$

Again the process is re-initiated with the head-of-the-queue user at the lowest occupied level of ℓ'.

Several important conclusions can be drawn from the above algorithm which allow the performance of the system to be bounded.

Computational Efficiency

1. Because a program is always operated for a time greater than or equal to the swap time (i.e. the time required to move the program in and out of secondary memory), it follows that the computational efficiency never falls below one-half. (Clearly, this fraction is adjustable in the formula for the initial level, ℓ_o.) An alternative way of viewing this bound is to say that the real-time computing speed available to one out of n active users is no worse than if there were 2n active users all of whose programs were in the high-speed memory.

Response Time

2. If the maximum number of active users is N, then an individual user of a given program size can be guaranteed a response time,

$$t_r \leq 2Nq\left(\left\lfloor \frac{w_p}{w_q} \right\rfloor + 1 \right) \qquad (3)$$

since the worst case occurs when all competing user programs are at the same level. Conversely, if t_r is a guaranteed response of arbitrary value and the largest size of program is assumed, then the maximum permissible number of active users is bounded.

Long Runs

3. The relative swap time on long runs can be made vanishingly small. This conclusion follows since the longer a program is run, the higher the level number it cascades to with a correspondingly smaller relative swap time. It is an important feature of the algorithm that long runs must in effect prove they are long so that programs which have an unexpected demise are detected quickly. In order that there be a finite number of levels, a maximum level number, L, can be established such that the asymptotic swap overhead is some arbitrarily small percentage, p:

$$L = \left[\log_2 \left(\left[\frac{w_{pmax}}{pw_q} \right] + 1 \right) \right] \quad (4)$$

where w_{pmax} is the size of the largest possible program.

Multi-level vs. Single-level Response Times

4. The response time for programs of equal size, entering the system at the same time, and being run for multiple quanta, is no worse than approximately twice the response-time occurring in a single quanta round-robin procedure. If there are n equal sized programs started in a queue at level ℓ, then the worst case is that of the end-of-the-queue program which is ready to respond at the very first quantum run at the $\ell+j$ level. Using the multi-level algorithm, the total delay for the end-of-the-queue program is by virtue of the geometric series of quanta:

$$T_m \sim q2^{\ell} \left\{ n(2^j-1) + (n-1)\, 2^j . \right\} \quad (5)$$

Since the end-of-the-queue user has computed for a time of $2^{\ell}(2^j-1)$ quanta, the equivalent single-level round-robin delay before a response is:

$$T_s \sim q2^{\ell} \left\{ n(2^j-1) \right\} . \quad (6)$$

Hence

$$\frac{T_m}{T_s} \sim 1 + \left(\frac{n-1}{n} \right) \left(\frac{2^j}{2^j-1} \right) \sim 2 \quad (7)$$

and the assertion is shown. It should be noted that the above conditions, where program swap times are omitted, which are pertinent when all programs remain in high-speed memory, are the least favorable for the multi-level algorithm; if swap times are included in the above analysis, the ratio of T_m / T_s can only become smaller and may become much less than unity. By a similar analysis it is easy to show that even in the unfavorable case where there are no program swaps, head-of-the-queue programs that terminate just as the $2^{\ell+j}$ quanta are completed receive under the multi-level algorithm a response which is twice as fast as that under the single-level round-robin (i.e. $T_m / T_s = 1/2$).

Highest Serviced Level

5. In the multi-level algorithm the level classification procedure for programs is entirely automatic, depending on performance and program size rather than on the declarations (or hopes) of each user. As a user taxes the system, the degradation of service occurs progressively starting with the higher level users of either large or long-running programs; however, at some level no user programs may be run because of too many active users at lower levels. To determine a bound on this cut-off point we consider N active users at level ℓ each running 2^{ℓ} quanta, terminating, and reentering the system again at level ℓ at a user response time, t_u, later. If there is to be no service at level $\ell+1$, then the computing time, $Nq2^{\ell}$, must be greater than or equal to t_u. Thus the guaranteed active levels, ℓ_a, are given by the relation:

$$\ell_a \leq \left[\log_2 \left(\frac{t_u}{Nq} \right) \right] \quad (8)$$

In the limit, t_u could be as small as a minimum user reaction time ($\sim$.2 sec.), but the expected value would be several orders of magnitude greater as a result of the statistics of a large number of users.

The multi-level algorithm as formulated above makes no explicit consideration of the seek or latency time required before transmission of programs to and from disc or drum units when they are used as the secondary memory, (although formally the factor w_q could contain an average figure for these times). One simple modification to the algorithm which usually avoids wasting the seek or latency time is to continue to operate the last user program for as many quanta as are required to ready the swap of the new user with the least priority user; since ordinarily only the higher level number programs would be forced out into the secondary memory, the extended quanta of operation of the old user while seeking the new user should be but a minor distortion of the basic algorithm.

Further complexities are possible when the hardware is appropriate. In computers with input-output channels and low transmission rates to and from secondary memory, it is possible to overlap the reading and writing of the new and old users in and out of high-speed memory while operating the current user. The effect is equivalent to using a drum giving 100 % multiplexor usage but there are two liabilities, namely, no individual user can utilize all the available user memory space and the look-ahead procedure breaks down whenever an unanticipated scheduling change occurs (e.g. a program terminates or a higher-priority user program is initiated).

Complexity is also possible in storage allocation but certainly an elementary procedure and a desirable one with a low-transmission rate secondary memory is to consolidate in a single block all high-priority user programs whenever sufficient fragmentary unused memory space is available to read in a new user program. Such a procedure is indicated in the flow diagram of the multi-level scheduling algorithm which is given as Figure 2.

It should also be noted that Figure 2 only accounts for the scheduling of programs in a working status and still does not take into account the storage allocation of programs which are in a dormant (or input-output wait status). One systematic method of handling this case is to modify the scheduling algorithm so that programs which become dormant at level ℓ are entered into the queue at level $\ell+1$. The scheduling algorithm proceeds as before with the dormant programs continuing to cascade but not operating when they reached the head of a queue. Whenever a program must be removed from high-speed memory, a program is selected from the end-of-the-queue of the highest occupied level number.

Finally, it is illuminating to apply the multi-level scheduling algorithm bounds to the contemporary IBM 7090. The following approximate values are obtained:

q = 16 m.s. (based on 1% switching overhead)

w_q = 120 words (based on one IBM 1301 model 2 disc unit without seek or latency times included)

$t_r \leq 8Nf_{sec.}$ (based on programs of (32k)f words)

$\ell_a \leq \log_2 (1000/N)$ (based on t_u = 16 sec.)

$\ell_o \leq 8$ (based on a maximum program size of 32K words)

Using the arbitrary criteria that programs up to the maximum size of 32,000 words should always get some service, which is to say that $\max \ell_a = \max \ell_o$, we deduce as a conservative estimate that N can be 4 and that at worst the response time for a trivial reply will be 32 seconds.

The small value of N arrived at is a direct consequence of the small value of w_q that results from the slow disc word transmission rate. This rate is only 3.3% of the maximum core memory multiplexor rate. It is of interest that using high-capacity high-speed drums of current design such as in the Sage System of in the IBM Sabre System it would be possible to attain nearly 100% multiplexor utilization and thus multiply w_q by a factor of 30. It immediately follows that user response times equivalent to those given above with the disc unit would be given to 30 times as many persons or to 120 users; the total computational capacity, however, would not change.

In any case, considerable caution should be used with capacity and computer response time estimates since they are critically dependent upon the distribution functions for the user response time, t_u, and the user program size, w_p, and the computational capacity requested by each user. Past experience using conventional programming systems is of little assistance because these distribution functions will depend very strongly upon the programming systems made available to the time-sharing users as well as upon the user habit patterns which will gradually evolve.

Conclusions

In conclusion, it is clear that contemporary computers and hardware are sufficient to allow moderate performance time-sharing for a limited number of users. There are several problems which can be solved by careful hardware design, but there are also a large number of intricate system programs which must be written before one has an adequate time-sharing system. An important aspect of any future time-shared computer is that until the system programming is completed, especially the critical time-sharing supervisor, the computer is completely worthless. Thus, it is essential for future system design and implementation that all aspects of time-sharing system problems be explored and understood in prototype form on present computers so that major advances in computer organization and usage can be made.

Acknowledgements

The authors wish to thank Bernard Galler, Robert Graham and Bruce Arden, of the University of Michigan, for making the MAD compiler available and for their advice with regard to its adaption into the present time-sharing system. The version of the Madtran Fortran-to-Mad editor program was generously supplied by Robert Rosin of the University of Michigan. Of the MIT Computation Center staff, Robert Creasy was of assistance in the evaluation of time-sharing performance, Lynda Korn is to be credited for her contributions to the pm and madtran commands, and Evelyn Dow for her work on the fap command.

References

1. Strachey, C., "Time Sharing in Large Fast Computers," Proceedings of the International Conference on Information Processing, UNESCO (June, 1959), Paper B.2.19.

2. Licklider, J. C. R., "Man-Computer Symbiosis," IRE Transactions on Human Factors in Electronics, HFE-1, No. 1 (March, 1960), 4-11.

3. Brown, G., Licklider, J. C. R., McCarthy, J., and Perlis, A., lectures given spring, 1961, Management and the Computer of the Future, (to be published by the M.I.T. Press, March, 1962).

4. Corbato, F. J., "An Experimental Time-Sharing System," Proceedings of the IBM University Director's Conference, July, 1961 (to be published).

5. Schmitt, W. F., Tonik, A. B., "Sympathetically Programmed Computers," Proceedings of the International Conference on Information Processing, UNESCO, (June, 1959) Paper B.2.18.

6. Codd, E. F., "Multiprogram Scheduling," Communications of the ACM, 3, 6 (June, 1960), 347-350.

7. Heller, J., "Sequencing Aspects of Multiprogramming," Journal of the ACM, 8, 3 (July, 1961), 426-439.

8. Leeds, H. D., Weinberg, G. M., "Multiprogramming," Computer Programming Fundamentals, 356-359, McGraw-Hill (1961).

9. Teager, H. M., "Real-Time Time-Shared Computer Project," Communications of the ACM, 5, 1 (January, 1962) Research Summaries, 62.

10. Teager, H. M., McCarthy, J., "Time-Shared Program Testing," paper delivered at the 14th National Meeting of the ACM (not published).

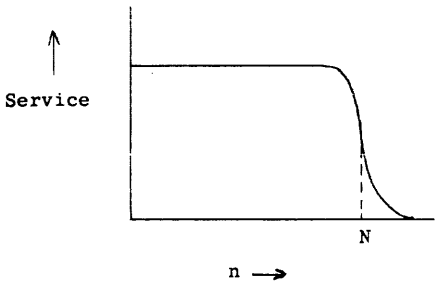


Figure 1. Service vs. Number of Active Users

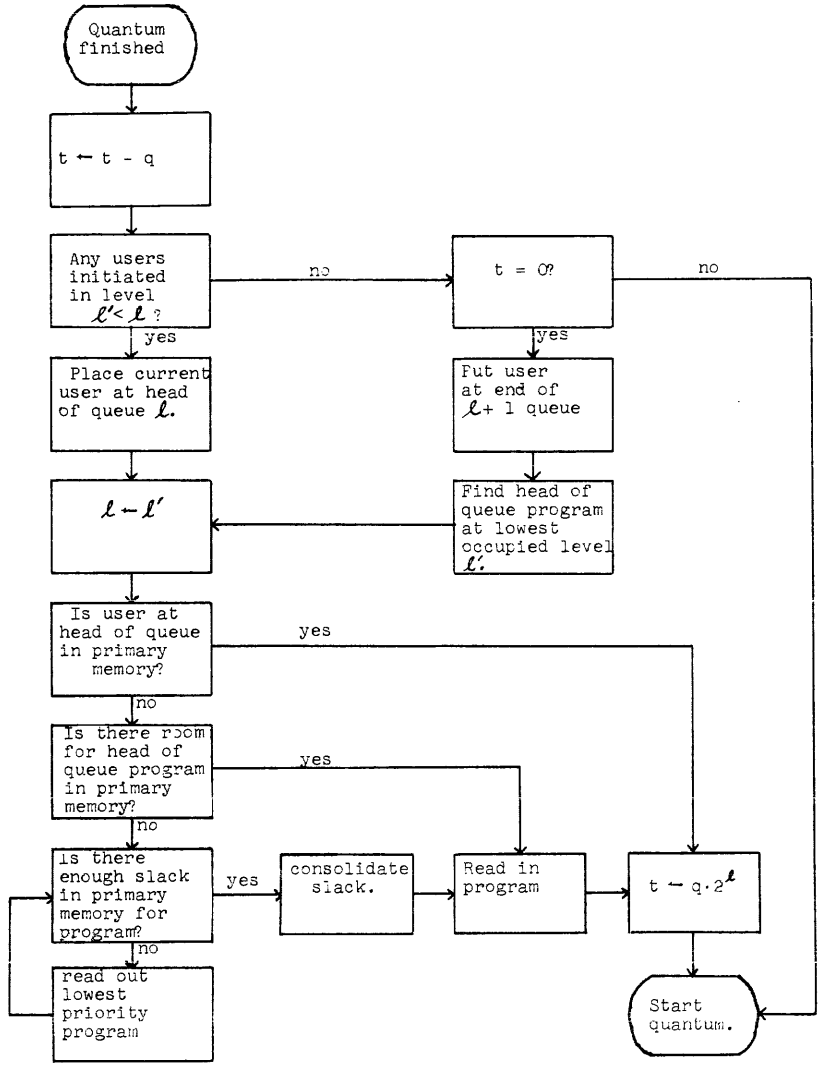


Figure 2. Flow Chart of Multi-Level Scheduling Algorithm

A PROGRAMMING LANGUAGE

Kenneth E. Iverson

Research Division, IBM Corporation

Yorktown Heights, New York

The paper describes a succinct problem-oriented programming language. The language is broad in scope, having been developed for, and applied effectively in, such diverse areas as microprogramming, switching theory, operations research, information retrieval, sorting theory, structure of compilers, search procedures, and language translation. The language permits a high degree of useful formalism. It relies heavily on a systematic extension of a small set of basic operations to vectors, matrices, and trees, and on a family of flexible selection operations controlled by logical vectors. Illustrations are drawn from a variety of applications.

The programming language outlined[1] here has been designed for the succinct description of algorithms. The intended range of algorithms is broad; effective applications include microprogramming, switching theory, operations research, information retrieval, sorting theory, structure of compilers, search procedures, and language translation. The symbols used have been chosen so as to be concise and mnemonic, and their choice has not been restricted to the character sets provided in contemporary printers and computers. A high degree of formalism is provided.

Basic Operations

The language is based on a consistent unification and extension of existing mathematical notations, and upon a systematic extension of a small set of basic arithmetic and logical operations to vectors, matrices, and trees. The arithmetic operations include the four familiar arithmetic operations and the absolute value (denoted by the usual symbols) as well as the floor, ceiling, and residue functions denoted and defined as follows:

Name	Symbol	Definition
floor	$\lfloor x \rfloor$	$\lfloor x \rfloor \leq x < \lfloor x \rfloor + 1$
ceiling	$\lceil x \rceil$	$\lceil x \rceil \geq x > \lceil x \rceil - 1$
residue mod m	$m \mid n$	$n = mq + m \mid n; 0 \leq m \mid n < m,$

where $\lfloor x \rfloor$, $\lceil x \rceil$, m, n, and q are integers.

The logical operations and, or, and not are defined upon the logical variables 0 and 1 and are denoted by $\wedge$, $\vee$, and an overbar. They are augmented by the relational statement (proposition) $(x\,R\,y)$ defined as follows. If x and y are any entities (e.g., numerals, alphabetic literals, or logical variables) and R is any relation defined upon them, then $(x\,R\,y)$ is equal to 1 or 0 according to whether the relation R does or does not hold between x and y. Thus $(i=j)$ is the familiar Kronecker delta δ_{ij}, $(u \neq v)$ is the exclusive or function of the logical variables u and v, and

$$\text{sgn } x = (x > 0) - (x < 0)$$

is the familiar sign function, defined as +1, 0, or -1 according as x is strictly positive, zero, or strictly negative.

Vectors and Matrices

A vector is denoted by an underlined lowercase letter (e.g., $\underline{x}$), its i-th component by $\underline{x}_i$, and its dimension by $\nu(\underline{x})$. A matrix is denoted by an underlined uppercase letter (e.g., $\underline{X}$), its i-th row by $\underline{X}^i$, its j-th column by $\underline{X}_j$, its ij-th element by $\underline{X}^i_j$, its row dimension (i.e., the common dimension of its row vectors) by $\nu(\underline{X})$, and its column dimension by $\mu(\underline{X})$.

All of the basic operations are extended component-by-component to vectors and matrices. Thus,

$$\underline{z} = \underline{x} + \underline{y} \iff \underline{z}_i = \underline{x}_i + \underline{y}_i,$$

$$\underline{Z} = \underline{X} \times \underline{Y} \iff \underline{Z}^i_j = \underline{X}^i_j \times \underline{Y}^i_j$$

$$\underline{w} = \underline{u} \wedge \underline{v} \iff \underline{w}_i = \underline{u}_i \wedge \underline{v}_i,$$

$$\underline{w} = (\underline{x} < \underline{y}) \iff \underline{w}_i = (\underline{x}_i < \underline{y}_i)$$

The symbol $\underline{\epsilon}(n)$ will denote a logical vector of dimension n whose components are all unity, and $\underline{\epsilon}(n)$ therefore denotes the zero vector. The

dimension n will be elided whenever it is clear from context.

The cyclic left shift of a vector $\underline{x}$ is called left rotation. It is denoted by $k\uparrow \underline{x}$ and defined by the relation:

$$\underline{y} = (k\uparrow \underline{x}) \Longleftrightarrow \underline{y}_i = \underline{x}_j ,$$

where $j = 1 + \nu(\underline{x}) \mid (i + k - 1)$. Right rotation is denoted by $k\downarrow \underline{x}$ and is defined analogously.

Reduction

For each basic binary operator $\odot$, the $\odot$-reduction of a vector $\underline{x}$ is denoted by $\odot/\underline{x}$ and defined by

$$\odot/\underline{x} = (\cdots((\underline{x}_1 \odot \underline{x}_2) \odot \underline{x}_3) \odot \cdots \odot \underline{x}_{\nu(\underline{x})}).$$

Thus $+/\underline{x}$ is the sum, and $\times/\underline{x}$ is the product of all components of $\underline{x}$. Moreover, if $\underline{u} = (1, 0, 1)$, then $+/\underline{u} = 2$, $+/\overline{\underline{u}} = 1$, $\wedge/\underline{u} = 0$, and $\vee/\underline{u} = 1$.

Reduction is extended to matrices row-by-row:

$$\underline{z} = \odot/\underline{X} \Longleftrightarrow \underline{z}_i = \odot/\underline{X}^i ,$$

and column-by-column:

$$\underline{z} = \odot//\underline{X} \Longleftrightarrow \underline{z}_i = \odot/\underline{X}_i ,$$

the column reduction being distinguished by the double slash.

For example, if $\underline{u} = (\underline{u}_1, \underline{u}_2)$ is a logical vector of dimension two, then De Morgan's law may be expressed as:

$$\wedge/\underline{u} = \overline{\vee/\overline{\underline{u}}} .$$

Moreover, this is the valid generalization of De Morgan's law to a vector $\underline{u}$ of arbitrary dimension.

The reduction operation can be extended to any relation R by substituting R for $\odot$ in the formal definition above. The parentheses in the definition now signify relational statements as well as grouping, and thus the expression $\neq/\underline{u}$ denotes the application of the exclusive-or over all components of the logical vector $\underline{u}$. For example, $\neq/(1, 0, 1) = ((1 \neq 0) \neq 1) = (1 \neq 1) = 0$. Induction can be used to show that

$$\neq/\underline{u} = 2 \mid +/\underline{u}, \text{ and } =/\underline{u} = \overline{2 \mid +/\overline{\underline{u}}} .$$

Hence, $\neq/\underline{u} = \overline{=/\overline{\underline{u}}}$, a useful companion to De Morgan's law.

Matrix Product

The conventional matrix product $\underline{A}\ \underline{B}$ may be defined as:

$$(\underline{A}\ \underline{B})^i_j = +/(\underline{A}^i \times \underline{B}_j).$$

Adopting the notation $\underline{A} \overset{+}{\times} \underline{B}$ for this product makes explicit the roles of the basic operations + and $\times$, and suggests the following useful generalization:

$$\underline{A} \overset{\odot_1}{\odot_2} \underline{B} = \odot_1/(\underline{A}^i \odot_2 \underline{B}_j) ,$$

where $\odot_1$ and $\odot_2$ are any operators or relations with suitable domains.

Thus $\underline{C} = \underline{M} \overset{\wedge}{=} \underline{X}$ is an "incidence matrix" such that $C^i_j = 1$ if and only if $\underline{M}^i$ and $\underline{X}_j$ agree in all components. If $\underline{M}$ is the memory of a binary computer (i.e., $\underline{M}$ is a logical matrix and row $\underline{M}^i$ is the i-th word of memory), and if $\underline{x}$ is an "argument register", then (treating $\underline{x}$ as a one-column matrix)

$$\underline{s} = \underline{M} \overset{\wedge}{=} \underline{x}$$

defines the sense vector $\underline{s}$ of an associative memory such that $\underline{s}_i = 1$ if and only if word $\underline{M}^i$ agrees with $\underline{x}$.

As further examples, $\underline{R} = \underline{P} \overset{+}{>} \underline{Q}$ gives the number of places in which a component of $\underline{P}^i$ exceeds the corresponding component of $\underline{Q}_j$, and $\underline{K} = \underline{P} \overset{\wedge}{>} \underline{Q}$ is a "covering matrix" which indicates which rows of $\underline{P}$ cover (exceed in every component) which rows of $\underline{Q}$.

De Morgan's law and the identity

$$\neq/\underline{u} = \overline{=/\overline{\underline{u}}}$$

establish a duality with respect to negation between $\wedge$ and $\vee$, and between $\neq$ and $=$. This duality is easily extended to matrices. For example,

$$\underline{A} \overset{\wedge}{=} \underline{B} = \overline{\overline{\underline{A}} \overset{\vee}{\neq} \overline{\underline{B}}} .$$

Selection

The formation of a vector $\underline{y}$ from a vector $\underline{x}$ by deleting certain components of $\underline{x}$ will be denoted by

$$\underline{y} = \underline{u}/\underline{x} ,$$

where $\underline{u}$ is a logical vector (of dimension $\nu(\underline{x})$ and $\underline{x}_i$ is deleted if and only if $\underline{u}_i = 0$. Thus $\underline{u}/\underline{x}$ and $\overline{\underline{u}}/\underline{x}$ provide a disjoint decomposition of $\underline{x}$, and $(\underline{x} > \epsilon)/\underline{x}$ is the vector of all strictly positive components of $\underline{x}$.

The operation $\underline{u}/\underline{x}$ is called compression and is extended to matrices by row and by column as follows:

$$\underline{Z} = \underline{u}/\underline{X} \Longleftrightarrow \underline{Z}^i = \underline{u}/\underline{X}^i ,$$
$$\underline{Z} = \underline{u}//\underline{X} \Longleftrightarrow \underline{Z}_i = \underline{u}/\underline{X}_i .$$

The familiar identity concerning partitioned matrices can now be generalized as follows:

$$\underline{X} \overset{+}{\times} \underline{Y} = (u/\underline{X}) \overset{+}{\times} (u//\underline{Y}) + (u/\underline{X}) \overset{+}{\times} (u//\underline{Y}).$$

Since the identity depends only on the associativity and commutativity of the operators + and $\times$, it holds also for all operators (and relations) possessing these properties. Moreover,

$$u/(\underline{X} \overset{+}{\times} \underline{Y}) = \underline{X} \overset{+}{\times} (u/\underline{Y}),$$

and

$$u//(\underline{X} \overset{+}{\times} \underline{Y}) = (u//\underline{X}) \overset{+}{\times} \underline{Y}.$$

To illustrate other uses of compression, consider a bank ledger $\underline{L}$ so arranged that $\underline{L}^i$ represents the i-th account, and $\underline{L}_1$, $\underline{L}_2$, and $\underline{L}_3$ represent the vector of names, of account numbers, and of balances, respectively. Then the preparation of a list $\underline{P}$ (in the same format) of all accounts having a balance less than two dollars is described by the statement:

$$\underline{P} \leftarrow (\underline{L}_3 < 2\epsilon)//\underline{L},$$

where the arrow denotes specification (in the sense of the symbol = in Fortran and the symbol := in Algol).

Three useful operations converse to compression are defined as follows:

Mesh: $\underline{z} = \backslash \underline{a}, \underline{u}, \underline{b} \backslash \Leftrightarrow \overline{u}/\underline{z} = \underline{a}; \quad u/\underline{z} = \underline{b}$

Mask: $\underline{z} = /\underline{a}, \underline{u}, \underline{b}/ \Leftrightarrow \overline{u}/\underline{z} = \overline{u}/\underline{a}; \; u/\underline{z} = u/\underline{b}$

Expansion: $\underline{z} = u \backslash \underline{b} \Leftrightarrow \underline{z} = \backslash \overline{\epsilon}, \; u, \underline{b} \backslash$.

They may be extended to matrices in the established manner and are related by the obvious identities:

$$\backslash \underline{a}, \underline{u}, \underline{b} \backslash = /\overline{u} \backslash \underline{a}, \; u, \; u \backslash \underline{b} /,$$

$$/\underline{a}, \underline{u}, \underline{b}/ = \backslash \overline{u} / \underline{a}, \; u, \; u / \underline{b} \backslash .$$

Special Vectors

In addition to the full vector $\epsilon(n)$ already defined, it is convenient to define the prefix vector $a^j(n)$ as a logical vector of dimension n whose first j components are unity. The suffix vector $\omega^j(n)$ is defined analogously. An infix vector having i leading 0's followed by j 1's can be denoted by $i \downarrow a^j$. When used in compression operations, these special vectors are very useful in specifying fixed formats.

Mixed Base Value

If the components of the vector $\underline{y}$ specify the radices of a mixed base number system and if $\underline{x}$ is any numerical vector of the same dimension, then the value of $\underline{x}$ in that number system is called the base y value of x, is denoted by $\underline{y} \perp \underline{x}$, and is defined formally by

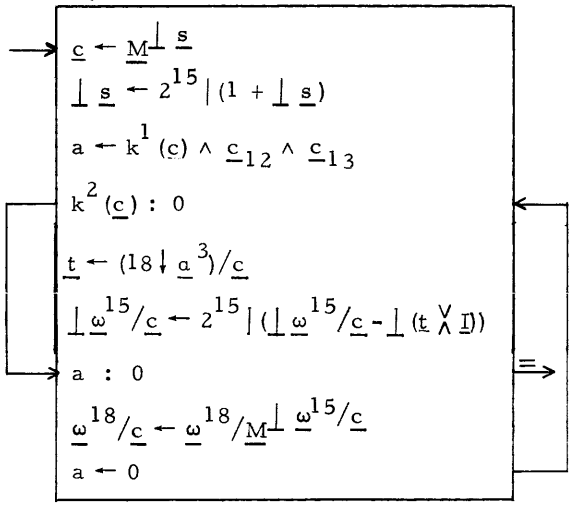
$$\underline{y} \perp \underline{x} = \underline{w} \overset{+}{\times} \underline{x},$$

where $\underline{w}$ is the weighting vector defined by $\underline{w}_{\nu(\underline{w})} = 1$ and $\underline{w}_{i-1} = \underline{w}_i \times \underline{y}_i$. For example, if $\underline{y} = (24, 60, 60)$ and if $\underline{x} = (1, 2, 5)$ is the elapsed time in hours, minutes, and seconds, then $\underline{y} \perp \underline{x} = 3725$ is the elapsed time in seconds.

The value of $\underline{x}$ in a decimal system is denoted by $(1\,0\,\epsilon) \perp \underline{x}$, and in a binary system by either $(2\,\epsilon) \perp \underline{x}$, or $\perp \underline{x}$. Moreover, if y is any real number, then $(y\,\epsilon) \perp \underline{x}$ denotes the polynomial in y with coefficients $\underline{x}$.

Application to Microprogramming

To illustrate the use of the notation in describing the operation of a computer, consider the IBM 7090 with a memory $\underline{M}$ of dimension $2^{15} \times 36$, a command vector $\underline{c}$ of dimension 36 representing the instruction next to be executed, a sequence vector $\underline{s}$ of dimension 15 representing the instruction counter, and a 3×15 index matrix $\underline{I}$ representing the three index registers. The instruction fetch phase of operation (excluding the channel trap) can then be described as in Figure 1. 0-origin indexing will be used for all vectors and matrices.



Instruction Fetch of IBM 7090

Figure 1

Step 1 shows the selection of the next instruction from the memory word specified by the instruction counter $\underline{s}$ and its transfer to the command register $\underline{c}$. Step 2 shows the incrementation (reduced modulo 2^{15}) of the counter $\underline{s}$. The logical function $k^1(\underline{c})$ of step 3 determines whether the instruction just fetched belongs to the class of instructions which are subject to indirect addressing, the bits $\underline{c}_{12}$ and $\underline{c}_{13}$ determine whether this particular instruction is to be indexed, and a is therefore set to unity only if indirect addressing is to be performed.

The function $k^2(\underline{c})$ determines whether the instruction is indexable. If $k^2(\underline{c}) = 0$, the branch from step 4 to step 7 skips the potential indexing of steps 5 and 6. As shown by step 6, indexing proceeds by oring together the index registers (i.e., rows of $\underline{I}$) selected by the three tag bits $\underline{t} = (18 \downarrow \underline{a}^3)/\underline{c}$, subtracting the base two value of the resulting vector from the address portion of the instruction $\underline{c}$, reducing the result modulo 2^{15}, and respecifying the address portion of $\underline{c}$ accordingly.

The fetch terminates immediately from step 7 if no indirect addressing is indicated; otherwise step 8 performs the indirect addressing and the indexing phase is repeated. Step 9 limits the indirect addressing to a single level. It may be noted that all format information is presented directly in the microprogram.

The description of the execution phase of computer operation will be illustrated by the family of instructions for logical functions in the 7090. Representing the 38-bit accumulator by the logical vector $\underline{u}$ (with $\nu(\underline{u}) = 38$, and with $\underline{u}_0$, $\underline{u}_1$, and $\underline{u}_2$ representing the sign, q and p bits, respectively), these instructions may be described as in Figure 2.

Or to storage	ORS	$\underline{M}\perp\underline{\omega}^{15}/\underline{c} \leftarrow (\underline{\alpha}^2/\underline{u}) \vee \underline{M}\perp\underline{\omega}^{15}/\underline{c}$
Or to accumulator	ORA	$\underline{u} \leftarrow \underline{u} \vee (\underline{\alpha}^2 \backslash \underline{M}\perp\underline{\omega}^{15}/\underline{c})$
And to storage	ANS	$\underline{M}\perp\underline{\omega}^{15}/\underline{c} \leftarrow (\underline{\alpha}^2/\underline{u}) \wedge \underline{M}\perp\underline{\omega}^{15}/\underline{c}$
And to accumulator	ANA	$\underline{u} \leftarrow \underline{u} \wedge (\underline{\alpha}^2 \backslash \underline{M}\perp\underline{\omega}^{15}/\underline{c})$
Exclusive or to accumulator	ERA	$\underline{u} \leftarrow (\underline{u} \neq (\underline{\alpha}^2 \backslash \underline{M}\perp\underline{\omega}^{15}/\underline{c}))$
		$\underline{\alpha}^2/\underline{u} \leftarrow \underline{\epsilon}$
Complement magnitude	COM	$\underline{\alpha}^1/\underline{u} \leftarrow \underline{\alpha}^1/\underline{\bar{u}}$
Clear magnitude	CLM	$\underline{\alpha}^1/\underline{u} \leftarrow \underline{\epsilon}$
Change sign	CHS	$\underline{u}_0 \leftarrow \underline{\bar{u}}_0$
Set sign plus	SSP	$\underline{u}_0 \leftarrow 0$
Set sign minus	SSM	$\underline{u}_0 \leftarrow 1$
Store logical word	SLW	$\underline{M}\perp\underline{\omega}^{15}/\underline{c} \leftarrow \underline{\alpha}^2/\underline{u}$
Clear and add logical	CAL	$\underline{u} \leftarrow \underline{\alpha}^2 \backslash \underline{M}\perp\underline{\omega}^{15}/\underline{c}$
Add and carry logical	ACL	$y \leftarrow \perp(\underline{\alpha}^2/\underline{u}) + \perp \underline{M}\perp\underline{\omega}^{15}/\underline{c}$
		$\perp\underline{\alpha}^2/\underline{u} \leftarrow (2^{36} \mid y) + (y > 2^{36})$

Instructions in IBM 7090

Figure 2

Ordered Trees

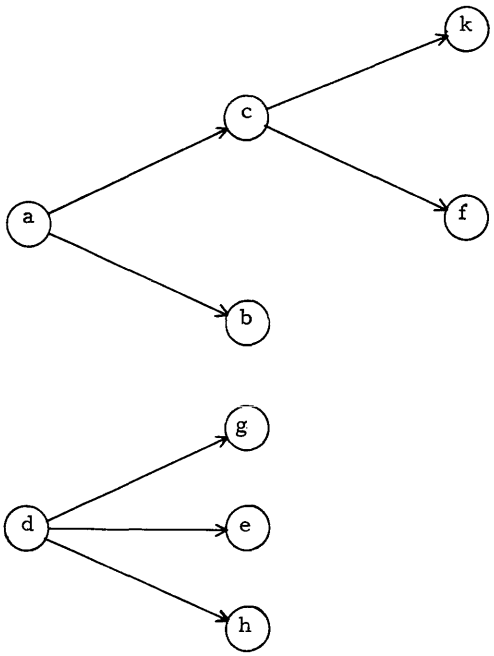
A tree can be represented graphically as in Figure 3(a) and, since it is a special case of a directed graph, it can also be represented by a node vector $\underline{n}$ and connection matrix $\underline{C}$ as in Figure 3(b).

Defining an ordered tree as a tree in which the set of branches emanating from each node has a specified ordering, a simple indexing system for ordered trees can be defined in the manner indicated by Figure 4(a). The dimension of the vector index $\underline{i}$ assigned to each node is equal to the number of the level on which it occurs. If T is an ordered tree, then $T_{\underline{i}}$ will denote the subtree rooted in the node with index $\underline{i}$. Thus $T_{(1,1)}$ is the subtree enclosed in broken lines in Figure 4(a). Moreover, $T^{\underline{i}}$ will denote the (unique) path vector comprising all nodes from the root to node $\underline{i}$. Thus $T^{(1,2)} = (a, b)$. The maximum dimension occurring among the index vectors is called the height of the tree and is denoted by $\nu(T)$.

The index vector notation proves very convenient in describing algorithms involving tree operands. Moreover, it provides a simple formalization of the two important representations of a tree, the Lukasiewicz representation and the level-by-level list.

If each index vector is augmented by sufficient null components (denoted by 0) to bring all to a common dimension equal to the height of the tree, then they may all be arrayed in an index matrix $\underline{I}$ as in Figure 4(b). If, as in Figure 4(b), the node vector $\underline{n}$ is defined such that $\underline{n}_j$ is the value of the node indicated by (the significant part of) the index vector $\underline{I}^j$, then the matrix obtained by appending $\underline{n}$ to $\underline{I}$ provides an unambiguous representation of the tree T. It is convenient to also append the degree vector $\underline{d}$ such that $\underline{d}_j$ is the degree of (i.e., the number of branches emanating from) node $\underline{I}^j$. The resulting matrix of $\nu(T) + 2$ columns is called a full list matrix of T.

Any matrix obtained by interchanging rows of a full list matrix is also a full list matrix and is again an unequivocal representation of the tree. If, as in Figure 4(c), the index vectors are right justified and arranged in increasing order on their value as integers (in a positional number system), the resulting list matrix is called a full right list matrix and is denoted by $\rceil T$. If, as in Figure 4(b), the index vectors are left justified and arranged in increasing order as fractions, the list matrix is called a full left list matrix and is denoted by $\lceil T$.



(a)

$\underline{n}$ = (a, c, f, k, e, d, h, g, b,)

$$\underline{C} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(b)

FIGURE 3

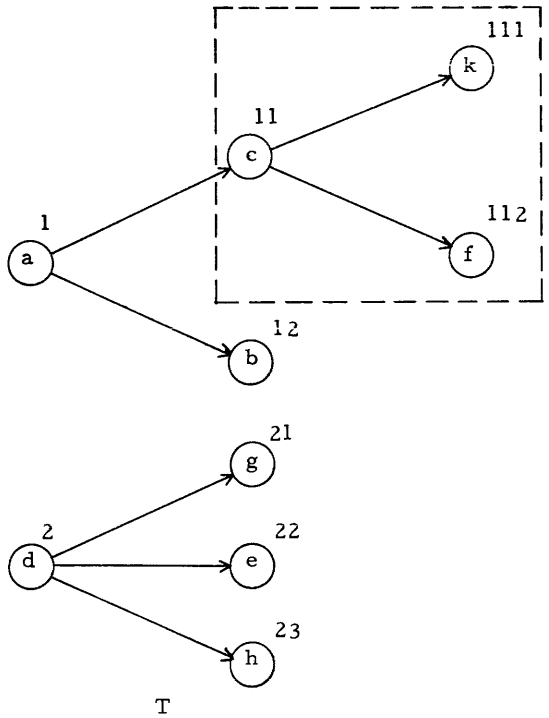
The right list groups nodes by levels, and the left list by subtrees. Moreover, in both cases the degree and node vector together (that is, $\underline{a}^2/(\ \rceil T)$ or $\underline{a}^2/(\ \lceil T)$) can be shown to provide an unequivocal representation of the tree without the index matrix. If, as in the case of a tree representing a compound statement involving operators of known degree, the degree vector $\underline{d}$ is a known function of the node vector $\underline{n}$, then the tree may be represented by the node vector $\underline{n}$ alone. In the case of the left list, this leads to the familiar Lukasiewicz* notation for compound statements.

Applications of Tree Notation

The tree notation and the right and left list matrices are useful in many areas**. These include sorting algorithms such as the repeated selection sort[4], the analysis of compound statements and their transformation to optimal form as required in a compiler, and the construction of an optimal variable-length code of the Huffman[5] prefix type. The sorting algorithm proceeds level by level and the right list representation of the tree is therefore most suitable. The analysis of a compound statement proceeds by subtrees and the left list (i.e., Lukasiewicz) form is therefore appropriate. The index vectors of the leaves of any tree clearly form a legitimate Huffman prefix code and the construction of such a code proceeds by combining subtrees in a manner dictated by the frequencies of the characters to be encoded, and therefore employs a left list. However, the characters are finally assigned in order of decreasing frequency to the leaves in right list order, and the index matrix produced must therefore be brought to right list order.

* First introduced by J. Lukasiewicz[2], and first analyzed by Burks et al[3].

** For detailed treatments of the applications mentioned, see Reference 1.



(a)

d	n	I		
2	a	1	0	0
2	c	1	1	0
0	k	1	1	1
0	f	1	1	2
0	b	1	2	0
3	d	2	0	0
0	g	2	1	0
0	e	2	2	0
0	h	2	3	0

$\lceil T$

(b)

d'	n'	I'		
2	a	0	0	1
3	d	0	0	2
2	c	0	1	1
0	b	0	1	2
0	g	0	2	1
0	e	0	2	2
0	h	0	2	3
0	k	1	1	1
0	f	1	1	2

$\rceil T$

(c)

FIGURE 4

REFERENCES

1. Iverson, K. E., "A Programming Language", Wiley, 1962.

2. Lukasiewicz, Jan, Aristotle's Syllogistic From the Standpoint of Modern Formal Logic, Clarendon Press, Oxford, 1951, p. 78

3. Burks, A. W., D. W. Warren, and J. B. Wright, An Analysis of a Logical Machine Using Parenthesis-free Notation", Mathematical Tables and Other Aids to Computation, Vol. VIII (1954), pp 53 - 57

4. Friend, E. H., "Sorting on Electronic Computer Systems", J. Assoc. Comp. Mach. 3, pp 134 - 168 (March 1956)

5. Huffman, D. A., "A Method for the Construction of Minimum Redundancy Codes", Proc. IRE, Vol. 40 (1952) pp 1098 - 1101

DESIGN OF A ONE-MEGACYCLE ITERATION RATE DDA

R. E. Bradley and J. F. Genna

Hazeltine Technical Development Center, Inc.

Indianapolis, Indiana

Summary

This paper describes the design of a parallel digital differential analyzer which operates at a rate of one million iterations per second. SPEDAC (Solid-State Parallel Expandable Differential Analyzer Computer) features parallel organization of the integrators, serial-parallel arithmetic within the integration cycle, 26-bit word length, and the integral inclusion of a digital function generator. The computer is programmed in analog computer fashion by means of plugboard interconnection of the integrators.

To achieve the one megacycle iteration rate, the arithmetic circuits operate at a six megacycle clock rate performing trapezoidal integration. The use of a parallel magnetic core memory permits direct parallel communication and hybrid operation with external large scale general purpose digital computers.

History

The first scientifically useful differential analyzer computer was designed by V. Bush and A. H. Caldwell.[1] It was an electro-mechanical computer using 18 ball and disk integrators, decimal constant multiplier gear boxes, and servo-controlled relay switching for interconnecting the integrators. It provided a maximum of .01% accuracy in the solution of differential equations, and was used extensively by scientists on a computing service basis during World War II.

During World War II the development of the voltage operational amplifier allowed the construction of an amplifier with extremely high gain and input impedance and near zero output impedance. This made possible the construction of an accurate electronic analog computer in 1947.[2] Since then the general purpose electronic analog computer has been refined to such an extent that it provides easily set up, precise, accurate (maximum of .01%), real time solutions to elaborate mathematical problems requiring hundreds of operational amplifiers. It also is used in simulation studies to simulate portions of physical systems. Within its area of application, its economy is unmatched by other types of computers.

The desire for greater accuracy, more flexibility, and simpler equipment caused Northrup Aircraft, Inc. in 1950 to design the first digital differential analyzer (DDA), designated MADDIDA.[3] It was an all binary, magnetic drum type machine using rectangular incremental integration and serial operation. In comparison with analog computers, the digital computation permitted greater accuracy, the programming procedure was more difficult, and the speed of operation was much slower.

A number of drum type, serial memory, serial operation DDA's were designed in the next few years.[4,5] These included the CRC-105, the Bendix D-12, the Bendix DA-1, the Litton 20, and several other special purpose computers such as the Autonetics' VERDAN. Various features such as decimal data entry, trapezoidal integration, and solid state circuitry were offered by various computers. However, they all shared the one common drawback to real time operation of being slow in operating speed due to serial operation.

The first great leap forward in operating speed was taken by Packard Bell Corporation in the design of TRICE.[6] It is a DDA using parallel organization of the integrators (all required integrators coexist physically) and serial arithmetic within the integrators. Since all integrators operate simultaneously, one iteration cycle (in this case ten microseconds) is just the time required for one integrator to complete its updating process. Since it is three orders of magnitude faster than previous DDA's, it is capable of performing real time computation for many types of problems. The parallel organization of integrators permits the simplified programming technique of analog computers through the interconnection of integrators on an analog

computer type patchboard.

The Hazeltine high speed DDA designated SPEDAC (Solid-state Parallel, Expandable, Differential Analyzer Computer) provides a further increase in speed of one order of magnitude with its one megacycle iteration rate, while retaining simple analog patchboard programming. This increase in speed is achieved by means of serial-parallel arithmetic computation within the integrators in addition to the parallel organization of the integrators. An additional feature of interest to analog simulation laboratory personnel is the inclusion of a digital function generator as an integral part of its design. Its design concepts are described in the following paragraphs.

Design Concepts

General

The principal concept of the design philosophy of SPEDAC is the realization of a very high speed digital machine which is operationally analagous to a general purpose analog computer with the increased accuracy and reliability of a digital computer. SPEDAC is an all solid state digital differential analyzer which is parallel in logical organization and serial-parallel in arithmetic operation.

The system concept will now be described with reference to the simplified block diagram given in Figure 1.

The inputs and outputs of every computing element are wired to a plugboard. The computer is programmed by patching the plugboard to interconnect the computing elements in the desired manner. The plugboard assembly is centrally located in the rack assembly in order to minimize and equalize the signal transmission time between computing elements. The maximum allowable transmission time between computing elements is 40 nanoseconds (e.g. 40 feet of high propogation coefficient coaxial cable).

Since the digital integrators require six operational timing pulses per iteration period, the one megacycle model computer contains a master timer which generates the six megacycle clock pulses and distributes them throughout the computer.

The master timer, illustrated in Figure 2, consists primarily of a multiply tapped delay line and sufficient gating to control the recirculating start pulse. Since only six pulses are required to synchronize the operation in one iteration cycle, a start pulse generated either by the start button or from an external control is gated down the multiply tapped 1 μsec delay line. For a one microsecond iteration period the start pulse emerging from the end of the delay line is recirculated back down the delay line with no external delay. However, the whole system may be time scaled downward by inhibiting the output gates a number of 1 μsec periods. The length of this inhibit may be preselected by a binary counter and a matrix enabled from a set of scaling switches. Thus the iteration rate may be scaled downward from one megacycle by any integral value. Occasionally a problem will arise where the programming and scaling is greatly simplified if an iteration rate equal to a binary multiple is available. For this reason an additional .907 μsec delay line is provided and may be switched in series with the recirculating start pulse. This results in a basic period of 1.907 μsec and yields an iteration rate equal to 524,288 cycles per second or 2^{19}. This rate may then be scaled down with the counter and matrix to yield submultiple binary iteration rates.

Computer Inputs

Data Entry. Data may be entered into the computer either manually or automatically. The manual input is by means of a high speed keyboard, and the automatic input is via paper tape, magnetic tape, Flexowriter, or directly from a digital computer. This information is decoded into a standard 4-bit binary coded decimal form and shifted into the Message and Address registers. The Address Selection Matrix receives the address data and advances the memory counters to the indicated memory address. The Message Register presents the encoded data to the Decimal/Binary Converter. This converter sequentially translates the binary coded decimal digits to pure binary numbers and simultaneously adds the binary numbers to obtain a pure binary number equal to the multi-digit binary coded decimal number in the Message Register. This binary number is also present at this time in the Memory Register and is loaded as the initial

condition into the register of the integrator determined by the Address Selection Matrix. The procedure is continued until all the initial conditions, scaling factors, and integration limits have been entered into the designated registers. Since the information is recirculated back into memory as it is read out, all of the input parameters are retained in memory primarily for use in a repetitive type of operational mode. Additional planes of memory are utilized as the Function Generator Storage. Discrete breakpoints and associated slope values of a function may be stored in these planes by the operator through the various means of input or directly by a general purpose digital computer.

Computer Outputs

Data Readout. Data may be read out of the computer in a continuous fashion during a compute mode, or in a manual or automatic fashion during a read out mode. The continuous read out is accomplished by interconnecting on the pre-patch panel one or more digital-to-analog converters. The contents of the R and Y registers and the initial conditions stored in memory are read out in much the same fashion as the data is read in. Read out selection is accomplished either through the external enabling of the Address Selection Matrix, (manually or automatically) or by sequentially advancing through the memory addresses. The data is then presented to the Output Register where it is stored during the binary-to-binary coded decimal-to-decimal conversion process.

Computing Modules

Digital Integrator. The high speed of computation is accomplished by performing the arithmetic operations in serial-parallel, that is, each register is broken up into small sections which add in an independent serial fashion, and each section's carry is sensed simultaneously in parallel. Thus, the speed of operation (i.e. the iteration rate) for practical purposes is independent of the word length and the complexity of the carry gates is directly proportional to the word length rather than factorially as in an all-parallel arithmetic.

The Integrator illustrated in Figure 3 consists primarily of two of these binary registers. One is called the R Register and the other is the Y Register. The initial conditions are read into the Y Register in parallel fashion from the initial condition register in the memory unit. Signals representing increments of one or two dependent variables and their associated signs may be plugged into each integrator via the pre-patch panel. In the "compute" mode, the Integrator adds these inputs to the contents of the Y Register and adds one-half of this increment to the contents of the R Register for the purpose of trapezoidal correction. When an increment of the independent variable is received, the Integrator adds the Y register contents to the R Register contents; thus, the R Register contains

$$\left[(n-1)Y_0 + \sum_{n=0}^{K} \frac{3}{2}\Delta Y_{n-1} + \frac{1}{2}\Delta Y_n\right]\Delta X$$

which is the continuous integral of the dependent variable function provided the independent variable increment approaches a relative zero and the iteration rate approaches a practical infinity. As the R Register fills and overflows, the overflow is detected and produces a signal which approximates the equation

$$dz = (Y_1 + Y_2)dx$$

If the dz's are accumulated in a second Y Register, the second Y Register will contain an approximation to the total integral,

$$z = \int (Y_1 + Y_2)dx$$

The inputs and outputs of the integrator are:

Inputs -
(1) Dyl, Dy2
(2) ± sign dyl, ± sign dy2
(3) dx
(4) ± sign dx

Outputs -
(1) dz
(2) ± sign dz

During a typical iteration cycle, the following operations are performed:

(1) Load the sum of dyl, dy2 into the Y Register; and at the same time, feed a trapezoidal correction to the R Register.

(2) If a dx is present, add or subtract the contents of the Y Register to or from the R Register according to the sign of dx.

(3) Determine if an overflow of the R Register has been produced and

present on output dz.

The basic building block of the Integrator is the counter-gate combination shown in Figure 4.

Counters #1 and #2 both contain N flip flop stages. These counters count up or down depending upon the condition of the input ± line. Bits can be loaded into any of the N stages of either counter.

The carry storage logic receives three inputs. These inputs are: (1) Overflow from last stage of the counter, (2) Counter Full enable from the counter, (3) Carry enable from an external source. The carry storage logic will present an output to the output gate when input 1 or when input 2 and 3 are present.

N connecting gates connect the outputs of counter #1 to the inputs of the corresponding stages of counter #2. To add or subtract the contents of counter #1 to or from counter #2, these gates are strobed by sequential timing pulses. The number of timing pulses required to combine the contents of counters #1 and #2 and detect an output is N for the connecting gates plus 1 for the carry (N+1) plus 1 for the output gate, a total of (N+2).

The output gate is strobed by a timing pulse after the above arithmetic operation has been completed.

In the Integrators these basic counter-gate blocks are combined into a series network as shown in Figure 5. The #1 counters now form a counter register of NxZ stages. The #2 counters form an identical register.

To add or subtract in register #1, the bit is gated into the proper stage, as determined by a scaling matrix, and then the output gates of the counters are simultaneously strobed to complete the operation.

To add or subtract the contents of register #1 to or from #2, each basic counter-gate block described in Figure 4 simultaneously goes through the process of gating #1 counter into #2 counter. Each output gate of register #2 is then simultaneously strobed to complete the operation. The number of timing pulses used to complete this operation is still (N+2) regardless of the number of basic blocks used in the registers.

This method of implementing an integrator by the use of counters has the desirable characteristics of:

(a) The complexity of each basic counter stage is independent of the register size.

(b) The time required for the addition or subtraction operation is independent of the register size for all practical register size requirements. This enables the utilization of 6 megacycle circuits as opposed to 30 megacycle circuits required in a 26-bit serial integrator.

(c) Fewer parts are used to accomplish these operations than would be needed for the conventional parallel adder approach.

Block 1 of Figure 6 is the dy summer and timer. The summer sums the scaled values of dy1 and dy2, then strobes this sum into the Y and R Registers. This sum is fed to the R Register such that the rectangular increments of integration are corrected to the more exact trapezoidal increments. (The correction is accomplished by feeding +1/2 dy to the R Register as dy is loaded into the Y Register.) Although an inspection of the trapezoidal correction factor, with respect to an individual integrator, results in an apparent error in the order of dydx, this is not the case. Since the source of the incremental input is a preceding computing element, the information is delayed one iteration period and would result in a second order truncation error if a -1/2 dy trapezoidal correction factor were used.

As these bits are being strobed in, the sign of the bit is fed to the Y Register ± enable and to the R Register ± logic, Block 2. This sign information thus determines the direction of count of the two registers.

The loading of dy bits to the Y Register may cause the register to pass through zero or to overflow its capacity. The overflow logic of Block 4 determines either of these conditions by examining the final carry storage logic of the last stage in the Y Register, the sign of the Y Register, and the sign of dy.

Should the Y Register pass through zero, the overflow logic changes the sign of the Y Register. Should the capacity of the Y Register be exceeded,

then an alarm signal is initiated to stop the problem.

If the trapezoidal correction causes the R Register to pass through zero or exceed the capacity of the register, then R overflow logic, Block 5, will change the sign of R in the case of passing through zero, or store the excess bit in the case of exceeding the register capacity.

The next operation in the cycle is the loading of the Y Register into the R Register. This is triggered by the presence of a dx.

Dx is fed into the dx storage and timing logic, Block 3. This logic then strobes (in sequence) the gates connecting the output of the Y Register to the inputs of the R Register. This logic is timed and controlled by the synchronizer and the limits of integration set on this integrator.

During this portion of the cycle, the ± enable for the R Register is controlled by the sign of dx through the R Register ± logic, Block 2.

The R overflow logic now determines if an output has been generated by the operations of this cycle, or if a sign change is necessary for the R Register. This decision is based upon the conditions of the last carry logic stage in the R Register, the sign of the R Register, the sign of the Y Register, the sign of dx, and the results of the trapezoidal correction. This completes the cycle.

The Y and R Registers have read-out gates which allow the contents of the registers to be read out of the integrator. The Y Registers have read-in gates which allow initial conditions to be loaded into the registers. These read-in and read-out gates are used in setting up the problem in the integrator and examining the results of computations.

Constant Multiplier. The constant multipliers are similar to the digital integrator. In the constant multiplier the Y register need not be a counter as there are no dY inputs, and also the trapezoidal correction logic is not necessary.

The constant multiplier utilizes one parallel accumulator register (the R register) and two parallel static registers, the C or Y register and the I_0 (initial condition) register.

During operation, the constant multiplier accepts one input, dX, and produces an incremental output, CdX, in which C is a constant. The I_0 register is used to enter the constant C into the C register at the start of the problem. In repetitive computations, the I_0 register, which is located in the memory, may be used to enter new values of the constant C into the C register.

When operating with a function generator, successive values of C represent the successive values of the function slopes as the function passes from one breakpoint to successive breakpoints. The output of the constant multiplier then represents an incremental rate, proportional to the slope of the function.

Variable Multiplier. The products of differential variables are formed using the mathematical expression for the differential of the product of X and Y.

$$d(XY) = YdX + XdY + dXdY$$

This product is formed in the SPEDAC variable multiplier with two special integrators and a summing network connected as shown in Figure 7.

The integrators in the variable multipliers differ from the standard SPEDAC integrators in several ways. Since a dY increment to the variable multiplier is used as the dependent variable increment for integrator #1 and as the independent variable increment for integrator #2, the timing of the utilization of the dY increment must be correspondingly synchronized. Therefore the timing of the operational procedure of integrator #1 is the same as that of the standard SPEDAC integrator and the timing of integrator #2 is inverted.

For rectangular integrators this results in an output of the form:

$$Y_{(n-1)}\ X_n + X_{(n-1)}\ Y_n + X_n\ Y_n$$

This represents the exact differential product of the two variables X and Y. Thus, the use of a variable multiplier to accomplish variable multiplication is more desirable than the use of two trapezoidal integrators, because integrator truncation error is avoided.

Other Computing Elements. Several other minor computing elements are available in the computer system, described briefly as follows:

1. Digital Servo - One of these minor elements is the digital servo which is used as a decision gate or a nulling device.

2. Summer - The summer accepts the outputs and associated signs of any other six computing elements patched into its input and presents a binary output plus sign equal to the algebraic sum of the inputs which may be patched into a following computing element.

3. Integration Limits - An Integration Limit module is available which is merely a counter and gating to accept an incremental variable input and either gate out or inhibit this increment depending upon a binary number initially read into the counter. These inputs and outputs are also patchable.

Function Generator

An unique feature of this computer is the inclusion of a digital function generator as an integral part of the system. A simplified block diagram of this component is illustrated in Figure 8.

A portion of the rapid access memory is used to store twenty-two bit words (21 bits plus sign), representing the breakpoints and associated slopes of desired functions. Data may be entered into the function generator storage via the DDA's automatic input.

The first three characters of a function generator message contain an address for one of the function generators. The remaining characters of the message specify the breakpoint and the slope. The breakpoints and slopes follow alternately in the sequence of breakpoint occurrence with respect to the independent variable increments.

After all the data have been read into the memory unit and a "start compute" signal is received, the independent variable increments are gated into their respective dX counters, and the instantaneous value of each counter is compared with the value in the breakpoint register. When the counter and the breakpoint register contain the same value, a compare signal assigns a read-out sequence for use by the read out control unit.

The read out control unit generates signals to update the address register and to enable the address gates for advancing the memory to the adjacent location which contains the corresponding slope value. The read out control also provides an enable for the Y register gates to allow the slope value to be read in parallel fashion into the selected constant multiplier.

An independent variable counter and a breakpoint comparator are used for each function to determine the occurrence of the function breakpoints. When a new breakpoint is reached, the new slope is read (in parallel) out of memory and into the C register of a constant multiplier. Between breakpoints the same slope is retained in the C register. The constant multiplier uses the independent variable of the function as its incremental input, and has an output whose rate is proportional to the slope in the C register.

The function may be generated either forward or backward with direction reversals accomplished within five microseconds. The maximum reversal rate is one per ten microseconds. If more than one comparator senses a compare in the same cycle, each of the indicated slope values is loaded into its corresponding constant multiplier before any of the breakpoint registers are updated. In the event of a "worst case" condition of five simultaneous compares, all five slope values are updated in twenty microseconds. In this case, another twenty microseconds is required to update all five breakpoint registers, and a minimum spacing of only forty microseconds between the breakpoints of any one function is required.

The number of breakpoints and slopes per function is variable in multiples of 32. As the number of breakpoints per function is increased, however, the number of functions capable of being stored in the standard memory decreases. For example, twelve 32 breakpoint functions, six 64 breakpoint functions, etc., may be utilized.

Circuits

The high computational speed of SPEDAC is due primarily to its parallel organization and series-parallel arithmetic. Since only six pulse times per iteration cycle are required, the

logical devices are operating well within the state of the art at six megacycles. Thus, conventional transistor flip flops and cascaded diode gates rather than unreliable or exotic electronic configurations are employed. Furthermore, the series-parallel logic provides a relatively easy transition to higher computational rates as the state of the art of semiconductors improves. Investigation of this expansion is under constant evaluation.

Packaging

All components are mounted on 4" x 9" printed circuit cards which are assembled in removable drawers. These drawers are in turn mounted on sliding tracks in the cabinet assembly in a manner such that the drawer is supported by the track even when fully extended.

The majority of the one megacycle enables throughout the system are carried via teflon hook-up wire which is loosely distributed in plastic wiring troughs in order to eliminate the cross talk of a tight harness. The six megacycle clock pulses and signals are distributed via teflon coaxial cable terminated with buffer amplifiers.

The mid-section of the cabinet includes the writing desk, the control panel and the patch bay. When seated at the desk, the operator may easily reach the control panel and the patch bay. Four access doors at the rear of the console provide access to the cabling, power supply units and patch panel.

Applications

Quite often, when solving sets of differential equations on an analog computer, there exists an extreme contrast in the dynamic ranges of two or more of the physical variables contained in the equations. When the problem is scaled such that the variables of major interest lie within the dynamic range of the computer, the accuracy of the remaining variables may be degraded beyond useful limits. In many cases this results in very unrealistic values of the points of secondary interest.

Due to the very high speed and large range of SPEDAC, mating this computer and an analog computer in hybrid operation will resolve many of these difficulties by proper apportionment of the equation variables. It is evident that this computer is quite useful in solving sets of equations requiring differential computation with respect to variables which are arbitrary functions of time, partial differential equations, and extreme dynamic range problems such as missile and satellite tracking problems requiring a uniform degree of accuracy over the entire range of values of the variables.

SPEDAC provides a very high speed complementary computing aid for general-purpose digital computers. The master timer is so designed to enable complete enslavement externally. Thus the input-output and computing modes of operation may be in complete synchronization with and under the complete control of a stored program on a general-purpose digital computer. The input-output data are channeled through the High Speed Data Transfer Register to and from the fully addressable magnetic core memory at data transfer rates up to 200,000 words per second. The data transfer rate is limited only by the input-output data transfer capability of the general-purpose digital computer.

General-purpose digital computers are relatively slow in solving sets of simultaneous differential equations because the solution is performed by means of numerical approximation methods using iterative procedures. Equations involving many trigonometric functions also reduce the computing speed of a general-purpose digital computer, due to the necessity of approximating trigonometric functions by expansions in power series or by means of table lookups.

SPEDAC is ideally suited to performing both of the above types of computations at very high speeds, but with digital computer accuracy. Thus, its use in conjunction with a general-purpose digital computer increases the computing speed of the general-purpose digital computer and consequently increases the work load potential of the computing facility.

References

1. Bush, V., and A. H. Caldwell: "A New Type of Differential Analyzer", J. Franklin Inst., Vol. 240, No. 4, pp. 255-326, Oct. 1945.

2. Jackson, A. S., "Analog Computation", McGraw-Hill, pp. 3-5, 1960.

3. Anon., "Computing Machines", Mechanical Engineering, V. 73, April 1951, pp. 325-327.

4. Donan, J. F., "The Serial Memory Digital Differential Analyzer", Mathematical Tables and Other Aids to Computation, Vol. 6, No. 38 pp. 102-112, Apr. 1952.

5. Mendelson, M. J., "The Decimal Digital Differential Analyzer", Aeronautical Engineering Review, Vol. 13, pp. 42-54, Feb. 1954.

6. Mitchell, J. M. and S. Ruhman, "The TRICE---A High Speed Incremental Computer", IRE National Convention Record, 1958. Part 4. (IRE National Convention, New York, March 24-27, 1958).

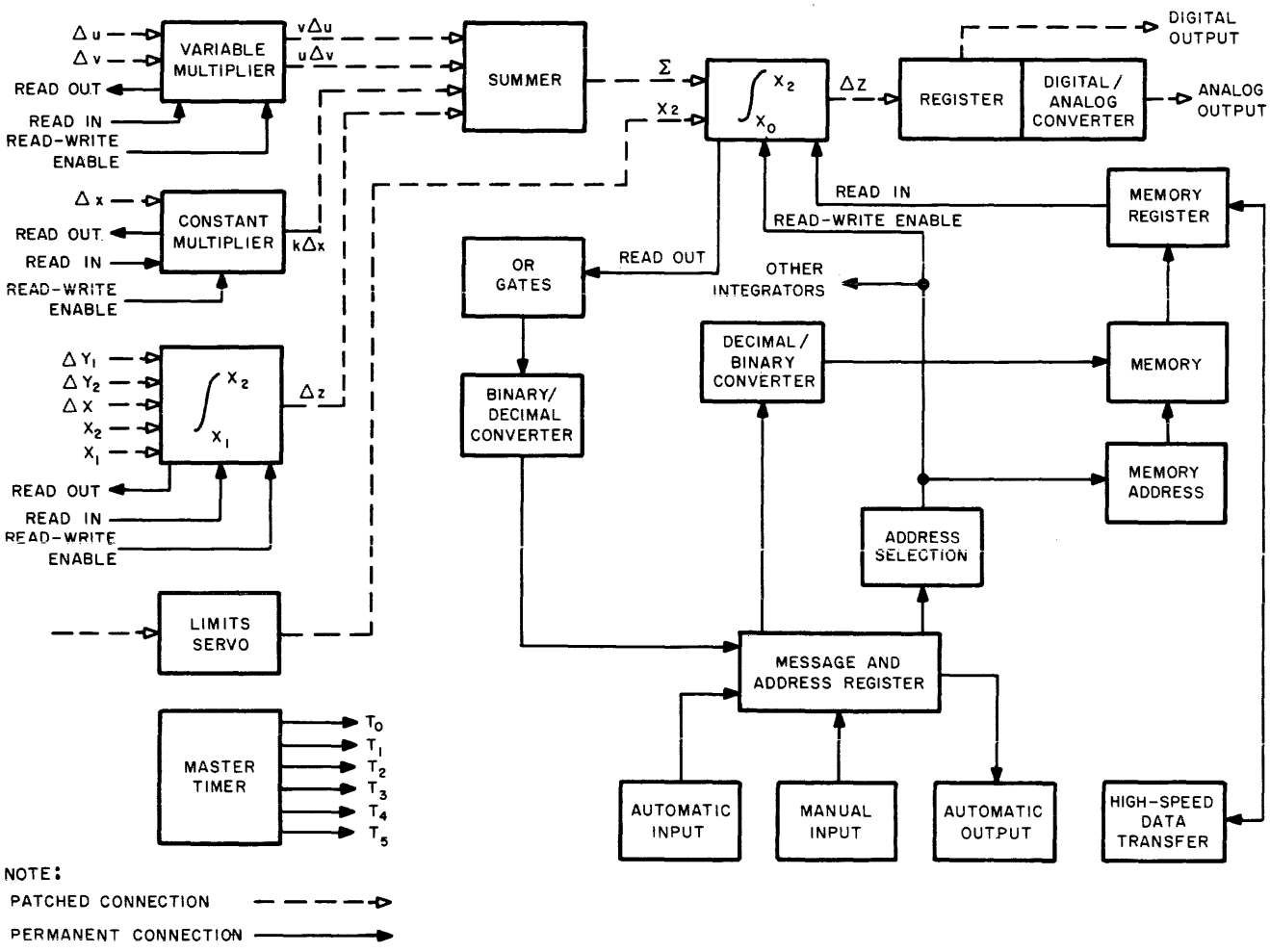


Figure 1. System Block Diagram

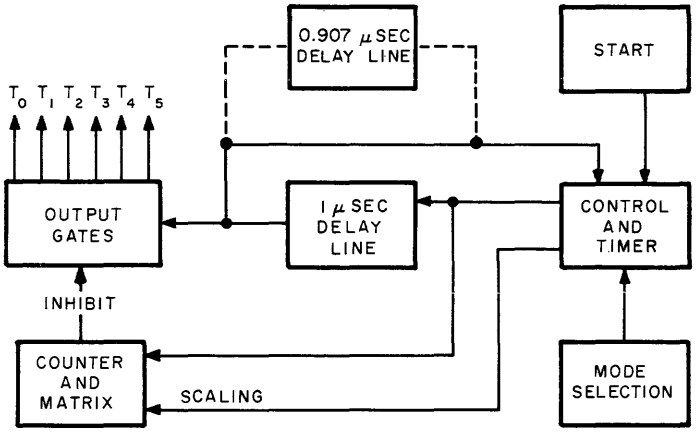
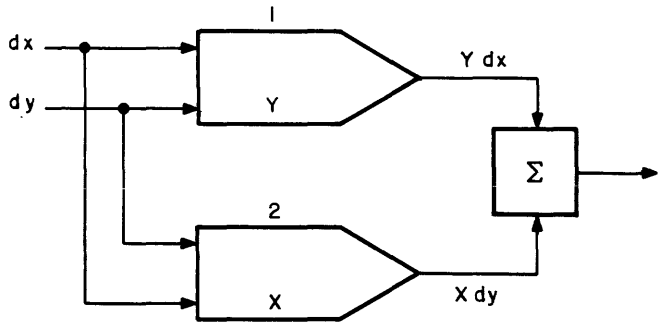
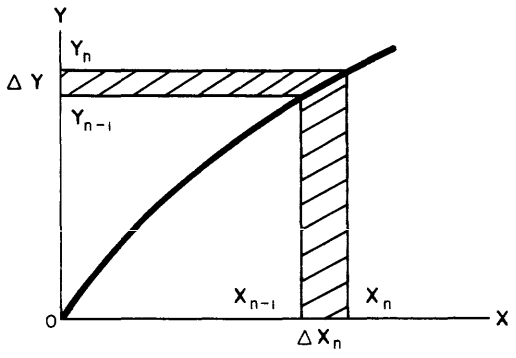


Figure 2. Master Timer





$$\Delta (XY) = Y_n \Delta X_n + X_{n-1} \Delta Y_n$$

$$Y_n \Delta X_n = Y_{n-1} \Delta X_n + \Delta X_n \Delta Y_n$$

$$\therefore \Delta (XY) = Y_{n-1} \Delta X_n + X_{n-1} \Delta Y_n + \Delta X_n \Delta Y_n$$

Figure 3. Digital Integrator

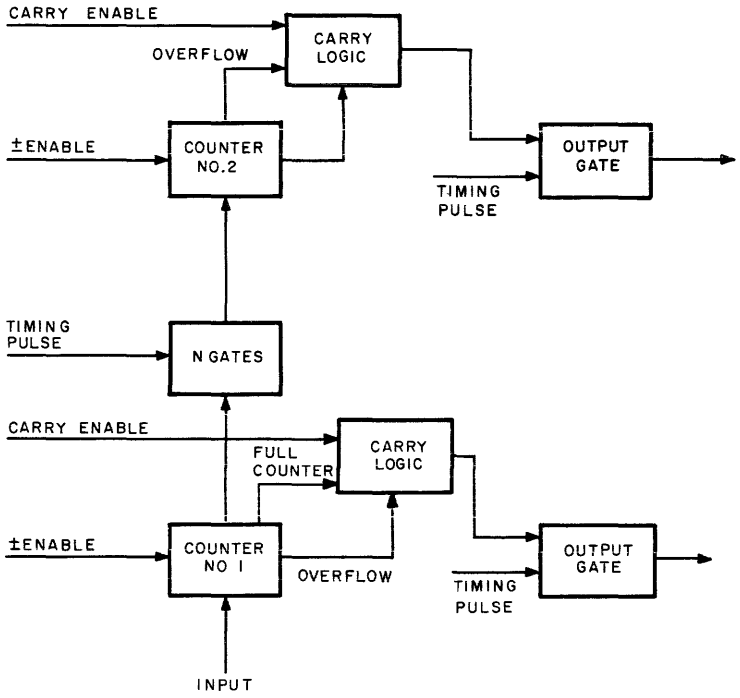


Figure 4. SPEDAC Integrator Counter-Gate Block

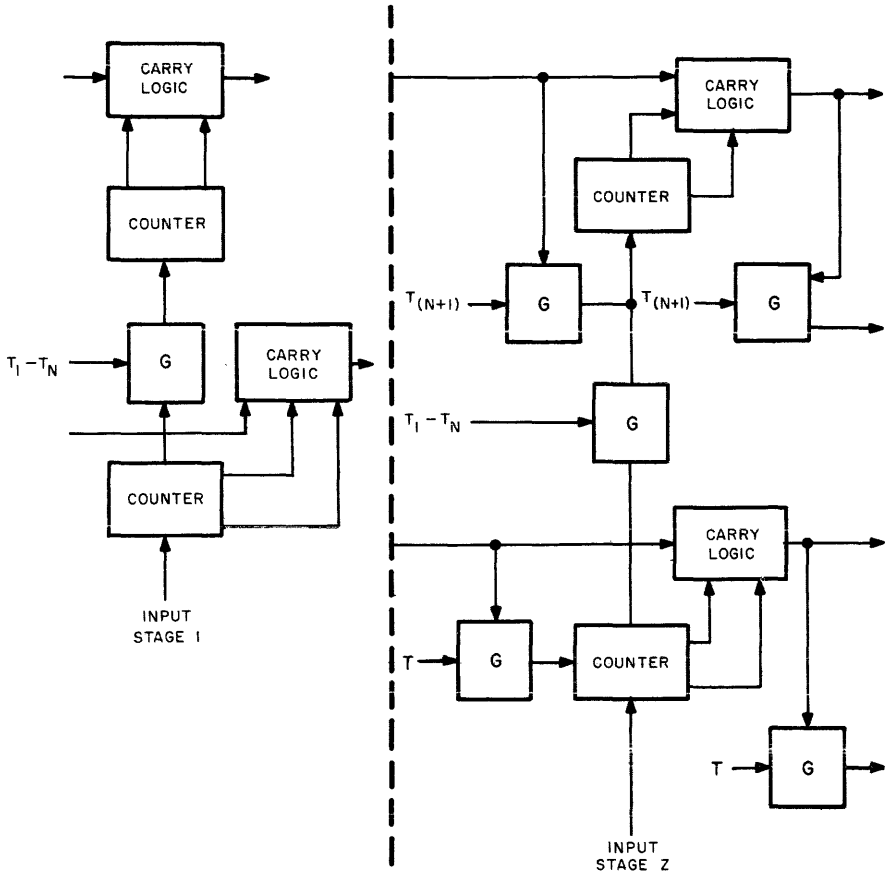


Figure 5. SPEDAC Basic Integrator Registers

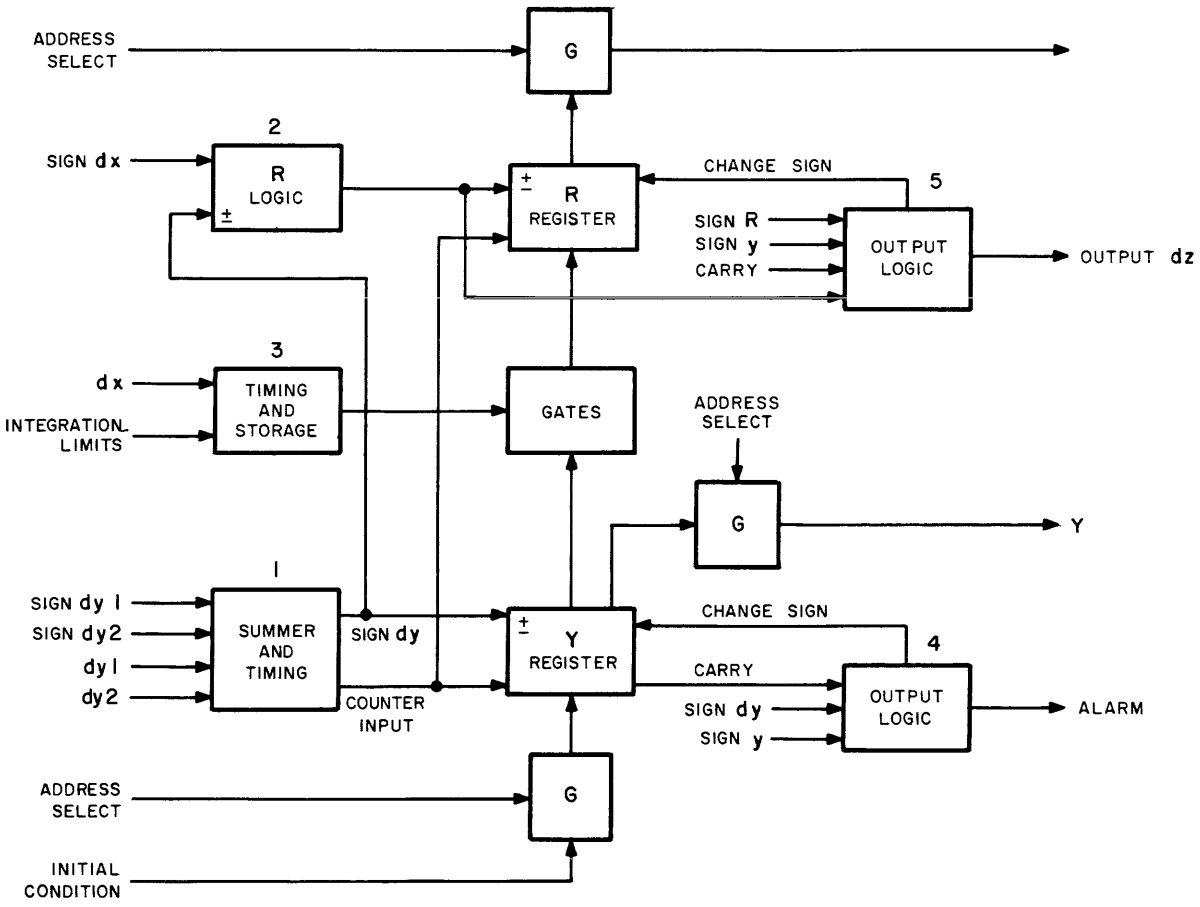
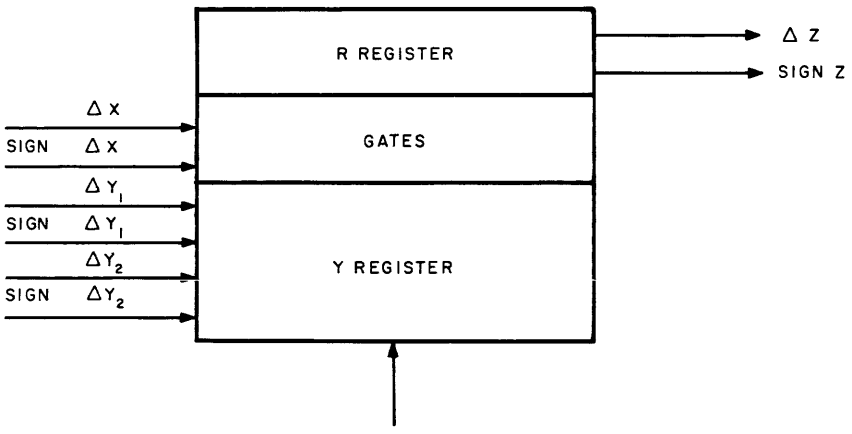


Figure 6. Integrator Logic Diagram



$$\left[(n-1)\ Y_0 + \sum_{n=0}^{n=k} \frac{3}{2}\,\Delta Y_{n-1} + \frac{1}{2}\,\Delta Y_n\right]\Delta x$$

Figure 7. SPEDAC'S Variable Multiplication Method

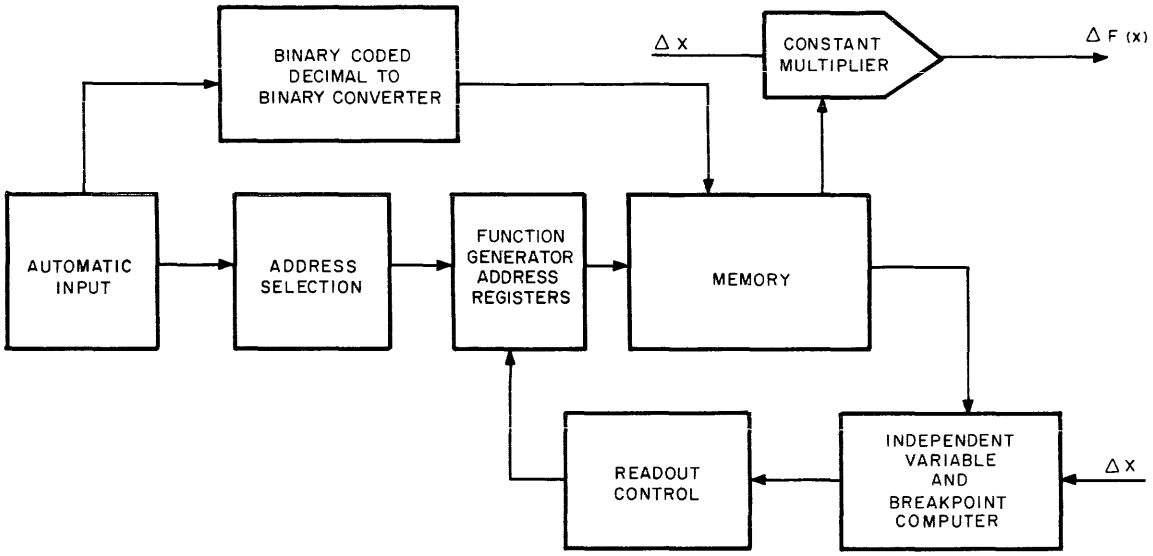


Figure 8. Function Generator Block Diagram

DDA ERROR ANALYSIS USING SAMPLED DATA TECHNIQUES

Don J. Nelson
University of Nebraska
Lincoln, Nebraska

Summary

Sampled data techniques were first applied to digital operations by Linvill and Saltzer[1,2] in order to study the errors resulting from the use of numerical integration techniques. The purpose of this paper is to develop an understanding for the mechanics of errors in the digital differential analyzer and to evolve a conceptually simple error theory. This is accomplished by establishing some of the basic comcepts regarding the applications of sampled data techniques to the integration process; developing the matrix model for the solution of a system of linear differential equations with constant coefficients on a digital differential analyzer and then using the W-transform to finalize the error theory. It is then easily shown that simple adjustments to the coefficient matrix of linear differential equations with constant coefficients will allow one to obtain solutions to these equations the accuracy of which is limited only by round-off errors.

Mechanics of Error in the DDA

Fundamentals

Before progressing directly into the analysis of DDA systems, it is desirable to examine briefly some of the tools to be used. The discrete nature of the variables in digital systems allows the use of sampled data techniques[3] which are already widely used. However, some of the concepts involved in DDA systems are sufficiently different to warrant special attention. Of particular interest is the mechanics of integration.

If $y(t)$ is a continuous function, its sampled counterpart can be represented by $y^*(t)$ where

$$y^*(t) = y(t) \cdot \sum_{n=1}^{\infty} \delta(t-nT) \qquad (1)$$

It is to be noticed that given $y(t)$, $y^*(t)$ is uniquely determined. However, the inverse relation is not unique. That is, given $y^*(t)$, there are an infinite number of functions $y(t)$ that will yield the same sampled function.

Quite frequently it is desirable to convert (by some practical technique) the samled function back into a continuous function. If a Fourier analysis of the unsampled signal shows that it contains frequencies $f < f_o/2$ where f_o is the sampling frequency, the unsampled function can theoretically be recovered identically by passing the sampled function through an ideal low-pass filter that passes all frequencies below f_o. (This is simply a paraphrase of the sampling theorem) As a matter of fact, in order to theoretically recover the original signal, the sampling frequency is required only to be at least twice the bandwidth of the unsampled signal.

In the following work, recovery by practical techniques is not the point in question. The concern here is with the sampled function that has resulted and the sampled function that is desired. This will be done by sampling the original continuous function, operating on this sampled function with approximate operators and then comparing the result to the sampled function derived from performing the true operation on the original function and then sampling this true function. Hence the non-uniqueness of the derived sampled function is not a matter of concern nor are any of the previously mentioned recovery restrictions.

The Laplace transform of the sampled $y(t)$ is given by the convolution of $Y(s)$ and the Laplace transform of $\delta(t-nT)$. This can be written in two different ways:

$$Y^*(s) = \frac{1}{2\pi j} \int_c Y(\omega) \cdot \frac{1}{1 - e^{-(s-\omega)T}} \cdot d\omega \qquad (2)$$

$$Y^*(s) = \frac{1}{2\pi j} \int_c \frac{1}{1 - e^{-\omega T}} \cdot Y(s-\omega) \cdot d\omega \qquad (3)$$

where residues are considered only at the poles of the first function inside the integral sign.

The two forms shown yield considerably different appearing results. Equation (3) yields residues at the poles of $\frac{1}{1 - e^{-\omega T}}$ while (2) yields residues at the poles of $Y(\omega)$. Both equations shed some light on the theory of sampled data systems.

Because of the prevalence of the function e^{-sT} in the equations that follow, this work will use $z=e^{-sT}$ as a substitution where it will simplify the equations. Another substitution that will be used is $a=e^{-\alpha T}$. If these substitutions are used, the following transforms may be obtained.

$y(t)$	$Y(s)$	$Y^*(s)$	
c	$\frac{c}{s}$	$\frac{c}{1-z}$	(4)
ca	$\frac{c}{s+\alpha}$	$\frac{c}{1-az}$	(5)
cta	$\frac{c}{(s+\alpha)^2}$	$\frac{Taz}{(1-az)^2}$	(6)
$\frac{ct^n a}{n!}$	$\frac{c}{(s+\alpha)^{n+1}}$	$\frac{1}{n!}\frac{d^n}{d\omega^n}\left[\frac{1}{1-az\,e^{\omega T}}\right]_{\omega=-\alpha}$	(7)

If concern is maintained for the solution of linear or piece-wise linear systems, these pairs are the only ones needed.

Let $y(t)$ be a continuous function and let

$$x(t) = \int_0^t y(t)\,dt. \tag{8}$$

Let $Y(s)$ and $X(s)$ be the Laplace transforms of $y(t)$ and $x(t)$, respectively, and $Y^*(s)$ and $X^*(s)$ be the Laplace transforms (with z substituted for e^{-sT}) of their sampled counterparts.

Then using the convolution integral, (2),

$$Y^*(s) = \frac{1}{2\pi j}\int_c Y(\omega)\cdot\frac{1}{1-ze^{\omega T}}\cdot d\omega \tag{9}$$

$$X^*(s) = \frac{1}{2\pi j}\int_c X(\omega)\cdot\frac{1}{1-ze^{\omega T}}\cdot d\omega$$

$$= \frac{1}{2\pi j}\int_c \frac{Y(\omega)}{\omega}\cdot\frac{1}{1-ze^{\omega T}}\cdot d\omega. \tag{10}$$

Since sampling and summation are commutative, $y(t)$ can be broken into its exponential components. This is most easily accomplished in the Laplace transform domain by partial fraction expansion.

In general, then $Y(s)$ will be of the form

$$Y(s) = \sum_{r=1}^{m}\frac{A_{0,r}}{s^r} + \sum_{i=1}^{n}\sum_{r=1}^{k_i}\frac{A_{i,r}}{(s+\alpha_i)^r}. \tag{11}$$

The reason for making the pole at zero a special case is that integration creates a special case for poles at the origin.

In order to understand the mechanics of integration, consider a typical term from each of the sums of (11).

First, let

$$Y(s) = \frac{A_{0,n}}{s^n} \tag{12}$$

Then

$$Y^*(s) = \frac{1}{2\pi j}\int_c \frac{A_{0,n}}{\omega^n}\left(\frac{1}{1-ze^{\omega T}}\right)d\omega$$

$$= \left[\frac{A_{0,n}}{(n-1)!}\cdot\frac{d^{n-1}}{d\omega^{n-1}}\left(\frac{1}{1-ze^{\omega T}}\right)\right]_{\omega=0}. \tag{13}$$

and

$$X^*(s) = \frac{1}{2\pi j}\int_c \frac{A_{0,n}}{\omega^{n+1}}\left(\frac{1}{1-ze^{\omega T}}\right)d\omega$$

$$= \left[\frac{A_{0,n}}{n!}\cdot\frac{d^n}{d\omega^n}\left(\frac{1}{1-ze^{\omega T}}\right)\right]_{\omega=0}. \tag{14}$$

Second, let

$$Y(s) = \frac{A_{1,n}}{(s+\alpha)^n}. \tag{15}$$

Then

$$Y^*(s) = \left[\frac{A_{1,n}}{(n-1)!}\cdot\frac{d^{n-1}}{d\omega^{n-1}}\left(\frac{1}{1-ze^{\omega T}}\right)\right]_{\omega=-\alpha} \tag{16}$$

and

$$X^*(s) = \left[\frac{A_{1,n}}{(n-1)!}\cdot\frac{d^{n-1}}{d\omega^{n-1}}\left(\frac{1}{\omega(1-ze^{\omega T})}\right)\right]_{\omega=-\alpha}$$

$$+ \frac{A_{1,n}}{\alpha^n}\cdot\frac{1}{1-z}. \tag{17}$$

Corresponding to the general term for Y(s) in (11), the general terms for Y*(s) and X*(s) are

$$Y^*(s) = \sum_{r=1}^{m} \left[\frac{A_{o,r}}{(r-1)!} \cdot \frac{d^{r-1}}{d\omega^{r-1}} \left(\frac{1}{1-ze^{\omega T}} \right) \right]_{\omega=0} + \sum_{i=1}^{n} \sum_{r=1}^{k_i} \left[\frac{A_{1,r}}{(r-1)!} \cdot \frac{d^{r-1}}{d\omega^{r-1}} \left(\frac{1}{1-ze^{\omega T}} \right) \right]_{\omega=-\alpha} \quad (18)$$

$$X^*(s) = \sum_{r=1}^{m} \frac{A_{o,r}}{r!} \cdot \frac{d^r}{d\omega^r} \left(\frac{1}{1-ze^{\omega T}} \right)_{\omega=0} + \sum_{i=1}^{n} \sum_{r=1}^{k_i} \frac{A_{i,r}}{\alpha_1^r} \cdot \frac{1}{1-z} + \sum_{i=1}^{n} \sum_{r=1}^{k_i} \left[\frac{A_{i,r}}{(r-1)!} \cdot \frac{d^{r-1}}{d\omega^{r-1}} \left(\frac{1}{\omega(1-ze^{\omega T})} \right) \right]_{\omega=-\alpha} \quad (19)$$

Note that $\frac{d^n}{d\omega^n}\left[A(\omega)B(\omega)\right]$ yields

$$C_o^n A_n(\omega)B_o(\omega) + C_1^n A_{n-1}(\omega)B_1(\omega) + C_2^n A_{n-2}(\omega)B_2(\omega) \text{ ------ } . \quad (20)$$

One of the terms of (19) when expanded as in (20) will yield

$$-\frac{1}{\alpha_i} Y^*_{i,r}(s),$$

but the rest of the terms of the expansion are not expressable in terms of Y*(s). This is indicative of the difficulty of finding some difference operator to operate on Y*(s) to achieve, effectively, true integration. In particular note that the Laplace integral operator, 1/s, will definitely not be useable as an operator upon the sampled function in order to yield the sampled counterpart of the integrated continuous function. This is obvious anyway since 1/s operating on an impulse function must yield a continuous function and not a sampled function.

The relatively involved computation indicated in (19) can be expressed much more efficiently by use of the W-transform[4,5]. Those familiar with sampled data technique will recognize that the expressions previously derived are essentially in Z-transform form. (Generally, however, Z is e^{sT} rather than e^{-sT}.) The W-transform can be defined by saying that if a function has a Z-transform equal to F(Z), then the W-transform becomes G(W) = WF(W). The inverse transform becomes

$$f(t) = \frac{1}{2\pi j} \int_c G(W) W^{t/T} dW \quad (21)$$

where quite amazingly f(t) is the unsampled (continuous) function satisfying the sampling theorem. This becomes an extremely powerful and important relationship. Since f(t) is continuous, it is possible to defferentiate both sides with respect to t, getting

$$f'(t) = \frac{1}{2\pi j} \int_c \frac{\ln W}{T} G(W) W^{t/T} dW \quad (22)$$

Hence in the W-plane $\frac{\ln W}{T}$ becomes a differential operator--something which has not been found for the function in the Z-plane. The result of this discussion on integration can be expressed diagramatically as in Figure 1.

In the DDA, the purpose of the integrator is to accept two inputs, Δx and Δy and to form $\Delta z = y\Delta x$. Δx is then formed into z in the "y" register of another integrator. In the connection of integrators, it is assumed that the z formed is equal to the integral of ydx. It is therefore necessary to examine the methods used for this operation and then compare the results with (19).

There are many methods of approximating the integral of y(t) based on samples[2,6]. However, the primary two upon which operational DDA's have been based are rectangular (Euler) integration and trapezoidal integration. These two methods will be compared with (19). If any other techniques should be implemented, the method of comparison would be the same.

Rectangular Integration (Open Loop Integration)

In "closed type" rectangular integration, the value of x at the time n is assumed to be

$$x_n = x_{n-1} + y_n \Delta t . \quad (23)$$

In a sampled data representation, Δt becomes the sampling interval T, and (23) becomes

$$\left[X^*(s)\right]' = zX^*(s) + TY^*(s)$$

$$\text{or} \quad \left[X^*(s)\right]' = \frac{T}{(1-z)} \cdot Y^*(s) \ .$$

However, an expansion of this last function shows

$$x(0) + x(T)z + x(2T)z^2 + \text{----}$$

$$= T\left[1 + z + z^2 + \text{---}\right]\left[y(0) + y(T) + y(2T) + \text{---}\right]$$

$$= Ty(0) + T\left[y(0) + y(T)\right]z$$

$$+ T\left[y(0) + y(T) + y(2T)\right]z^2 + \text{--}$$

which is in error from the desired equation by an amount

$$Ty(0) + Ty(0)z + Ty(0)z^2 + \text{----} \ .$$

Hence the machine equation must necessarily be

$$\left[X^*(s)\right]' = \frac{T}{(1-z)} \cdot Y^*(s) - \frac{Ty(0)}{(1-z)} \ . \qquad (24)$$

The last term of (24) is essentially taken care of in the machine by setting the initial conditions. In the following work y(0) will quite frequently be zero, thereby eliminating the last term from consideration.

Establishing the general term for the output of the machine integrator requires having (24) operate on (18). This yields for the rectangular integration case

$$\left[X^*(s)\right]' = \sum_{r=1}^{m}\left[\frac{A_{o,r}}{(r-1)!}\left(\frac{T}{1-z}\right)\frac{d^{r-1}}{d\omega^{r-1}}\left(\frac{1}{1-ze^{\omega T}}\right)\right]_{\omega=0}$$

$$- \frac{Ty(0)}{(1-z)} + \sum_{i=1}^{n}\sum_{r=1}^{k_i}$$

$$\left[\frac{A_{i,r}}{(r-1)!}\left(\frac{T}{1-z}\right)\frac{d^{r-1}}{d\omega^{r-1}}\left(\frac{1}{1-ze^{\omega T}}\right)\right]_{\omega=-\alpha} \qquad (25)$$

In the interest of compactness, let $d^n/d\omega^n$ be represented by D^n and let $e^{\omega T}$ be represented by Ω. Also let the error function be

$$U^*(s) = X^*(s) - \left[X^*(s)\right]' \ .$$

Then, subtracting (25) from (19) yields

$$U^*(s) = \sum_{r=1}^{m}\left[\frac{A_{o,r}}{r!}\left(D^r - \frac{rT}{1-z}D^{r-1}\right)\left(\frac{1}{1-z\Omega}\right)\right]_{\omega=0}$$

$$- \frac{Ty(0)}{(1-z)} + \sum_{i=1}^{n}\sum_{r=1}^{k_i}\left[\frac{A_{i,r}}{(r-1)!} \cdot \right.$$

$$\left.\left(D^{r-1}\frac{1}{\omega} - \frac{T}{1-z}D^{r-1}\right)\left(\frac{1}{1-z\Omega}\right)\right]_{\omega=-\alpha} \qquad (27)$$

This function is fairly formidable, but leads to rather interesting results.[7] The errors associated with the integration of some rather simple functions are shown in Table 1. It should be pointed out here, that the process used in finding the error function does not limit y(t) to functions satisfying the sampling theorem.

Trapezoidal Integration (Open Loop Integration)

Trapezoidal integration follows the rule

$$x(nT) = x\left[(n-1)T\right] + T\left[\frac{y\left[(n-1)T\right] + y(nT)}{2}\right] \qquad (28)$$

which becomes

$$\left[X^*(s)\right]' = X^*(s)z + \frac{T}{2}\left[Y^*(s)z + Y^*(s)\right]$$

$$\text{or} \left[X^*(s)\right]' = Y^*(s)\left[\frac{T(1+z)}{2(1-z)}\right] - \frac{Ty(0)}{2(1-z)} \qquad (29)$$

where the second term of (29) is required to correct for the fact that y(0) may not be zero.

Using the general term (18) for Y(s) yields for the general case

$$\left[X^*(s)\right]' = \sum_{r=1}^{m}\left[\frac{A_{o,r}}{r!}\left(\frac{T(1+z)}{2(1-z)}\right)D^{r-1}\left(\frac{1}{1-z\Omega}\right)\right]_{\omega=0}$$

$$- \frac{Ty(0)}{2(1-z)} + \sum_{i=1}^{n}\sum_{r=1}^{k_i}\left[\frac{A_{i,r}}{(r-1)!}\left(\frac{T(1+z)}{2(1-z)}\right)\right.$$

$$\left.D^{r-1}\left(\frac{1}{1-z\Omega}\right)\right]_{\omega=-\alpha} \ . \qquad (30)$$

Subtracting (30) from (19) yields the error function

$$U^*(s) = \sum_{r=1}^{m} \frac{A_{o,r}}{r!}\left[\left(D^r - \frac{rT(1+z)}{2(1-z)}\,D^{r-1}\right)\left(\frac{1}{1-z\Omega}\right)\right]_{\omega=0} - \frac{Ty(0)}{2(1-z)} + \sum_{i=1}^{n}\sum_{r=1}^{k_i}\frac{A_{i,r}}{(r-1)!}\left[D^{r-1}\left(\frac{1}{\omega} - \frac{T(1+z)}{2(1-z)}\right)\left(\frac{1}{1-z\Omega}\right)\right]_{\omega=-\alpha} . \quad (31)$$

Application of this formula to simple functions yields the errors shown in Table 1 under Trapezoidal Integration.

Comments Regarding Open Loop Integration

By straightforward, but rather lengthy computations, it can be shown[7] that the maximum per unit error (magnitude) in the integration of a function with a pole at $s=-\alpha$ (regardless of the order of the pole) is approximated by $\alpha T/2$ for rectangular and $\alpha^2T^2/2\cdot3!$ for trapezoidal integration. (Absolute errors are approximately $T/2$ and $T^2/2\cdot3!$ respectively.) It is interesting to notice that there is no phase shift in the integration of a cosine wave when trapezoidal integration is used. This is a valuable asset in closed loop DDA operation.

Mechanization of the DDA

The digital differential analyzer is sufficiently different, conceptually, in its operation that, before proceeding, a few words regarding its mechanization are in order. The parallel type of operation will be considered first.

Consider the set of simultaneous differential equations

$$\dot{X} = AX + F(t). \quad (32)$$

The machine representation is of the form

$$\Delta X = \left[AX + F(t)\right]\Delta t . \quad (33)$$

To mechanize this equation, the machine utilizes $x_i(nT)$ to produce the values of $\Delta X(nT)$, then uses both of these to produce $x_i\,(n+1)T$. At first glance, this would seem to preclude the use of closed form integration techniques. [If $x(nT) = \int_0^{nT} y(t)dt$, open form integration techniques are those that attempt to produce the true integral from knowledge of $x(mT)$ for m from zero to n-1. Closed form integration techniques are those that attempt to produce the true integral from knowledge of $x(mT)$ for m from zero to n.] However, the machine variable is discrete in nature. Since $\Delta x_i(nT)$ is added to $x_i(nT)$ to produce $x_i(n+1)T$, the next value of x_i is actually known and so it is possible to produce the "integral increment" in closed form. Hence one operation utilizes $x_i(nT)$ to form Δx_i and then forms

$$X\left[(n+1)T\right] = X(nT) + \Delta X(nT).$$

The formation of $\Delta X(nT)$ establishes the rule of integration used by the machine.

Before continuing, consider the following simple example. A system of equations

$$\dot{x}_1 = a_{11}x_1 + a_{12}x_2$$

$$\dot{x}_2 = a_{21}x_1 + a_{22}x_2$$

is represented on the machine by

$$\Delta x_1 = a_{11}x_1\Delta t + a_{12}x_2\Delta t$$

$$\Delta x_2 = a_{21}x_1\Delta t + a_{22}x_2\Delta t .$$

The schematic for the machine set-up is shown in Figure 2.

The solution is initiated by establishing initial conditions for $x_i(0)$. The $x_i(0)$ are operated on to produce the $\Delta x_i(0)$. When the first Δt occurs, x_i is updated to produce $x_2(T)\Delta t$ which in turn forms the new $\Delta x_2(T)$. This completes one full cycle. The computer solution evolves as this cycle is continuously repeated.

As previously mentioned, integration in the DDA occurs in the formulation of the function $\Delta X(nT)$. The mechanized scheme may use either open or closed form numerical techniques. Consider, for example, trapezoidal integration. The quantity $\Delta X(nT)$ is developed by

$$\Delta X(nT) = A\left[X(nT) + \frac{\Delta X(nT)}{2}\right]\Delta t + \left[\frac{f(nT) + \Delta f(nT)}{2}\right]\Delta t.$$

This is the equivalent of the mathematical expression

$$\int_{nT}^{(n+1)T} y\,dt \cong \Delta x(nT) = \left[\frac{y[(n+1)T] + y(nT)}{2}\right]\Delta t$$

$$= \left[\frac{y(nT) + y(nT) + \Delta y(nT)}{2}\right]\Delta t$$

$$= \left[y(nT) + \frac{\Delta y(nT)}{2}\right]\Delta t .$$

Hence

$$\int_{nT}^{(n+1)T.} X\,dt = \int_{nT}^{(n+1)T} \left[AX + F\right]dt$$

becomes

$$\Delta X(nT) = A\left[\frac{X[(n+1)T] + X(nT)}{2}\right]\Delta T + \left[\frac{F(nT) + F(n+1)T}{2}\right]\Delta t .$$

This in turn becomes

$$\Delta X(nT) = A\left[X(nT) + \frac{\Delta X(nT)}{2}\right]\Delta t + \left[\frac{F(nT) + F[(n+1)T]}{2}\right]\Delta t$$

$$= \left[1 - A\frac{\Delta t}{2}\right]^{-1}\left[AX(nT) + \left(\frac{F(nT) + F[(n+1)T]}{2}\right)\right]\Delta t .$$

In the machine, Δt is represented by T, so that the last equation can be put in the form

$$X\left[\frac{2(1-z)}{T(1+z)}\right] = (AX + F).$$

Therefore, the machine's differential operator becomes

$$D^*(z) = \frac{2(1-z)}{T(1+z)} .$$

In the serial type DDA, only one integrator is updated at one time. This means that the mechanization takes on the following form (for rectangular integration without a driving function).

$$\Delta x_1(nT) = \left[a_{11}x_1(nT) + a_{12}x_2(nT) + a_{13}x_3(nT) + \text{--}\right]\Delta t$$

$$\Delta x_2(nT) = \left[a_{21}x_1\left[(n+1)T\right] + a_{22}x_2(nT) + a_{23}x_3(nT) + \text{--}\right]\Delta t$$

$$\Delta x_3(nT) = \left[a_{31}x_1\left[(n+1)T\right] + a_{32}x_2\left[(n+1)T\right] + a_{33}x_3(nT) + \text{--}\right]\Delta t , \text{ etc.}$$

This complicates the error theory somewhat but the analysis follows the same pattern as that established in the following work.

Analysis of DDA Techniques

In analyzing solutions to differential equations on the DDA, it is desireable to use the W-transform because one may use the differential operator in the W-plane.

If the W-transform is applied to the set of equations

$$\dot{X} = AX \tag{34}$$

the result in the W-domain is

$$\frac{\ln w}{T} X(w) = AX(w) \tag{35}$$

or

$$\left[\frac{\ln w}{T} I - A\right] X(w) = 0. \tag{36}$$

The determinant of the square matrix in (36) yields a polynomial in $(\ln w)/T$ which when solved for yields the poles in the W-plane.

For example, if $sX-A$ yields a pole at $s=-\alpha$, then a root of the polynomial would be $(\ln w)/T = -\alpha$ which yields a pole at $w=e^{-\alpha T}$. This is in accordance with the theory of differential equations and W-transform theory.

The problem, then, is what sort of error occurs in the location of these poles if one uses an approximate integration operator rather than the true operator.

Substitution of the W-transform for the Z form in (24) and (29) shows that the W-transform of the rectangular and trapezoidal integration techniques are,

for rectangular:

$$\left[X(w)\right]' = \left[\frac{T}{w-1}\right]\left[Y(w)\right]' - \frac{TY(0)}{w-1} \qquad (37)$$

for trapezoidal:

$$\left[X(w)\right]' = \left[\frac{T(w+1)}{2(w-1)}\right]\left[Y(w)\right]' - \frac{TY(0)}{2(w-1)} . \qquad (38)$$

The differential operators, therefore, are for the rectangular case :

$$\left[Y(w)\right]' = \left[\frac{w-1}{T}\right]\left[X(w)\right]' + y(0) \qquad (39)$$

for the trapezoidal case:

$$\left[Y(w)\right]' = \left[\frac{2(w-1)}{T(w+1)}\right]\left[X(w)\right]' + \frac{Y(0)}{w+1} . \qquad (40)$$

1. The Rectangular Case. Substitution of (39) into (34) yields

$$\left(\frac{w-1}{T}\right)\left[X(w)\right]' = A\left[X(w)\right]' - X(0)$$

or

$$\left[\left(\frac{w-1}{T}\right)I - A\right]\left[X(w)\right]' = -X(0)$$

or

$$\left[X(w)\right]' = \left[\left(\frac{w-1}{T}\right)I - A\right]^{-1} X(0). \qquad (41)$$

The poles of $\left[X(w)\right]'$ are hence the roots of the determinant

$$\left|\left(\frac{w-1}{T}\right)I - A\right| .$$

If the true roots of the original equation lie at $w_i = e^{-\alpha_i T}$, then the resulting polynomial in (w-1)/T will yield roots at

$$\left(\frac{w-1}{T}\right) = -\alpha, \text{ so that } w_i = 1 - \alpha_i T.$$

Comparison with the expansion of $e^{-\alpha T}$ shows the error in the location of the roots to be

$$\frac{(\alpha_i T)^2}{2!} - \frac{(\alpha_i T)^3}{3!} + - - - \approx \frac{a_i T^2}{2!} . \qquad (42)$$

Therefore the product αT must be kept small if the solution is to closely resemble the true solution. This requirement is similar to the one discovered for small error in open loop integration in the first part of this paper.

An interesting observation may be made regarding the solution of the equation $x'' + \omega^2 x = 0$. Here the poles in the W-plane should lie at $w = e^{\pm j\omega T}$. (Poles on the unit circle yield undamped oscillations.) However, if one uses rectangular integration, the poles will lie at $1 \pm \omega T$ (see Figure 3). These poles lie outside the unit circle and hence the solution will yield exponentially increasing oscillations. The further they are from the origin, the greater will be the negative damping and the greater the error in the resulting frequency.

2. The Trapezoidal Case. Substitution of (40) into (34) yields

$$\left(\frac{2(w-1)}{T(w+1)}\right)\left[X(w)\right]' = A\left[X(w)\right]' + \frac{X(0)}{w+1}$$

$$\text{or } \left[\left(\frac{2(w-1)}{T(w+1)}\right)I - A\right]\left[X(w)\right]' = \frac{X(0)}{w+1}$$

$$\text{and } \left[X(w)\right]' = \left[\left(\frac{2(w-1)}{T(w+1)}\right)I - A\right]^{-1} \frac{X(0)}{w+1} \qquad (43)$$

In this case the poles occur for

$$\left(\frac{2(w-1)}{T(w+1)}\right) = -\alpha , \text{ or for}$$

$$w = \frac{1-\alpha T/2}{1+\alpha T/2} = 1-\alpha T + \frac{(\alpha T)^2}{2^1} - \frac{(\alpha T)^3}{2^2} + - - - .$$

This shows that the position of the pole will be in error by an amount

$$\frac{(3!-2^2)(\alpha T)^3}{2^2 \cdot 3!} - \frac{(4!-2^3)(\alpha T)^4}{2^3 \cdot 4!}$$

$$+ - - - \approx \frac{(\alpha T)^3}{12} . \qquad (44)$$

Again it is seen that trapezoidal integration produces less error than rectangular integration. There appears to be an introduction of a pole at w= -1, see (43). However, this pole will be cancelled when the inverse of the matrix is taken. This will always be the case regardless of the integration operator. Hence the only frequencies present will be the negative roots of the matrix (except for those introduced by round-off errors).

Considering again the solution of the equation $x'' + \omega^2 = 0$, where the poles should lie at $w = e^{\pm j\omega T}$, it is seen that the poles lie instead at

$$w = \frac{1 \pm j\omega T/2}{1 \pm j\omega T/2} .$$

Since the magnitude of the numerator and denominator are the same, the poles will lie on the unit circle. Hence the solution will be undamped and is at least without error in this respect. The actual location of the poles, however, is at

$$e^{\pm 2\tan^{-1}\omega T/2},$$

which means the location of the poles will be in error by approximately

$$e^{\pm j\frac{(\omega T)^2}{12}}$$

This means the angle per unit error in W-plane $\approx \frac{(\omega T)^2}{12}$. (Comparison with Table 1 shows that this is the same as the per unit error in the magnitude of an integrated cos function.) For an error of about 1 percent the product ωT should be about 0.33. This is rather interesting since it indicates that approximately ten samples per cycle will yield 1 percent accuracy in the resultant frequency. For an error of 0.1 percent, thirty samples should be taken; for 0.01, one hundred samples; etc. In the closed loop system, establishment of proper initial conditions will preclude any error in magnitude since poles lie on the unit circle. (This of course neglects errors caused by rounding.)

It can be said, then, that theory regarding sensitivity[7] and errors due to parameter perturbation[8] can be applied to DDA systems in the W-plane. The perturbation of roots, however, is more distinct since the differential operators are of a precise nature.

Function Generation

The ability to determine errors precisely immediately suggests the possibility of modifying the original differential equation to accurately produce the desired function--or the desired solution to the original differential equation. This indeed is possible for those functions that can be described by linear differential equations with constant coefficients. Here the roots in the w-Plane are given precisely by $w_i = e^{s_i t}$, where the s_i are the s-plane poles of the desired function. If the poles of the modified differential equation are γ_i, then

for rectangular integration:

$$\gamma_i = \frac{w_i - 1}{T} = \frac{e^{s_i t} - 1}{T}$$

and for the trapezoidal integration:

$$\gamma_i = \frac{2(w_i - 1)}{T(w_i + 1)} = \frac{2(e^{s_i t} - 1)}{T(e^{s_i t} + 1)}.$$

The modified differential equation may then be derived from $\prod_{i=1}^{n}(s-\gamma_i)$.

The resulting differential equation is then put in matrix form and solved using the desired operator. The matrix will generally consist of the constants of the type tangent φ or $e^{\alpha T}$. The accuracy is limited only by truncation of these terms and round-off in the computer. Hence, this technique could be invaluable for accurate generation of simple functions.

Round-Off Errors

Round-off errors are particularly difficult to resolve. Since these errors are unpredictable without prior knowledge of the solution, it would seem that deterministic methods would fail to produce any useful results. However, a few concepts are presented here which may be useful in some considerations.

Consider the system of linear differential equations

$$\dot{X} = AX.$$

In the W-transform domain, this becomes

$$\frac{\ln w}{T} Z(w) = AX(w).$$

The machine equation is actually set up to perform

$$\left[X(w)\right]' = O(w)\left[A\right]\left[X(w)\right]' - \left(\frac{k}{w-1}\right)AX(0) + \left(\frac{1}{w-1}\right)X(0) \qquad (45)$$

where O(w) is the mechanized integration operator, and the k is a constant necessary to achieve the proper initial conditions, mathematically. The round-off process can be considered as the addition of some function, $\varepsilon(w)$ to the function after the "integration" operation has been performed.

Equation (45) then becomes

$$\left[X(w)\right]' = O(w)A\left[X(w)\right]' - \left(\frac{k}{w-1}\right)AX(0) + \left(\frac{1}{w-1}\right)X(0) + E(w), \qquad (46)$$

$$\text{So } \left[X(w)\right]' = \left[I-O(w)A\right]^{-1}\left\{\left[1-kA\right]\left(\frac{1}{w-1}\right)X(0) + E(w)\right\}. \quad (47)$$

Supposedly the poles of E(w) will be more or less randomly distributed in the normal case. However, it is observed from (47) that should poles of E(w) coincide with poles of the "unrounded" $\left[X(w)\right]'$, it would be possible to obtain almost any kind of error from the round-off procedure. However, E(w) cannot have poles of higher order than one by nature of its origin. This means that the worst effect that could occur would be to increase the order of some "natural" pole by one. The worst type of pole would be one lying on the unit circle in the W-plane since this would give a response of tsin ωt in the equivalent unsampled domain. If one should know the location of the poles in the "unrounded" $\left[X(w)\right]'$, it would be possible to set a bounds on the error by assuming the worst case, namely one pole in E(w) lying on each pole of $\left[X(w)\right]'$.

It is believed by the author to be not possible to extend the theory of error caused by round-off any further than has been done in the previous discussion if deterministic methods are to be used. However, it is possible that some statistical studies might show that this type of error has some predictable distribution.

References

1. Linvill, W. K. and Salzer, J. M., "Analysis of Control Systems Involving Digital Computers", Proc. IRE, Vol. 41, No. 7, July 1953, pp 901-906.

2. Salzer, J. M., "Frequency Analyses of Digital Computers Operating in Real Time", Proc. IRE, Vol. 42, No. 2, February 1954, pp 457-466.

3. Suskind, A. K., Editor, "Notes on Analog-Digital Conversion Techniques", John Wiley and Sons, Inc., New York, 1957.

4. Gilliland, M. C., "The W-Transform", a paper presented at the Northwest Joint Computer Conference, October 1960

5. Gilliland, M. C., "The Spectral Evaluation of Iterative Differential Analyzer Integration Techniques", Proc. West. Joint Comp. Conf., Vol. 18, May 1961, pp 507-518.

6. Hildebrand, F. B., "Introduction to Numerical Analysis", McGraw-Hill Book Company, Inc., New York, 1956.

7. Miller, K. S. and Murray, F. J., "A Mathematical Basis For An Error Analysis of Differential Analyzers", MIT Journal of Mathematics and Physics, Vol. 32, July-October 1953, pp 136-163.

8. Nelson, Don J., "A Foundation For the Analysis of Analog Oriented Combined Computer Systems", Ph.D. dissertation, Stanford University, 1961.

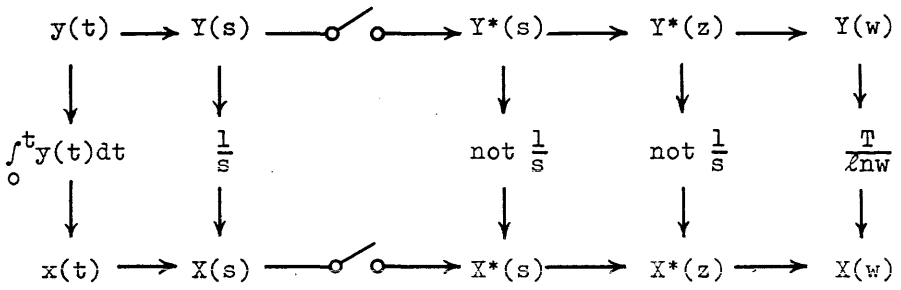


Figure 1. Integration Operations.

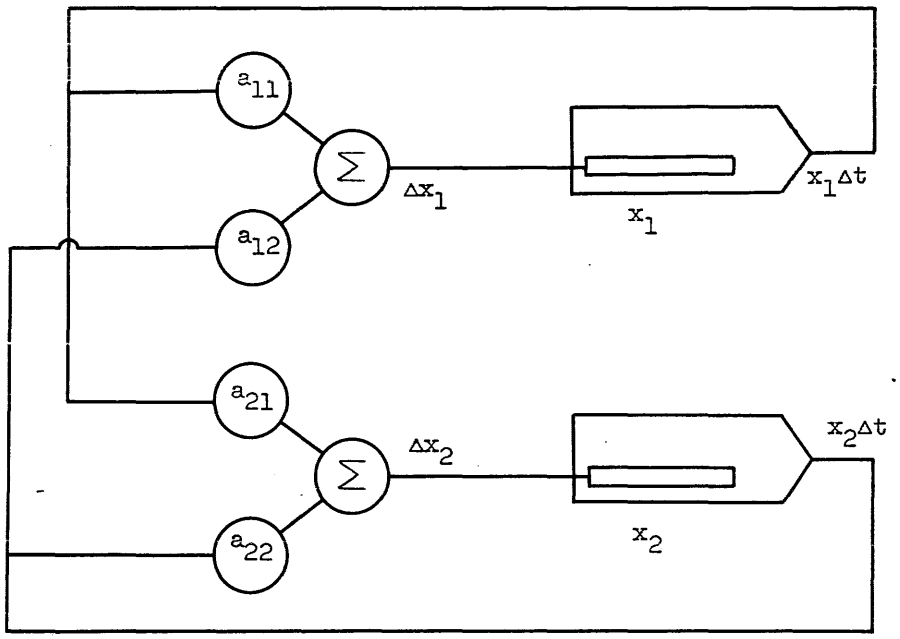


Figure 2. Machine Representation of X = AX.

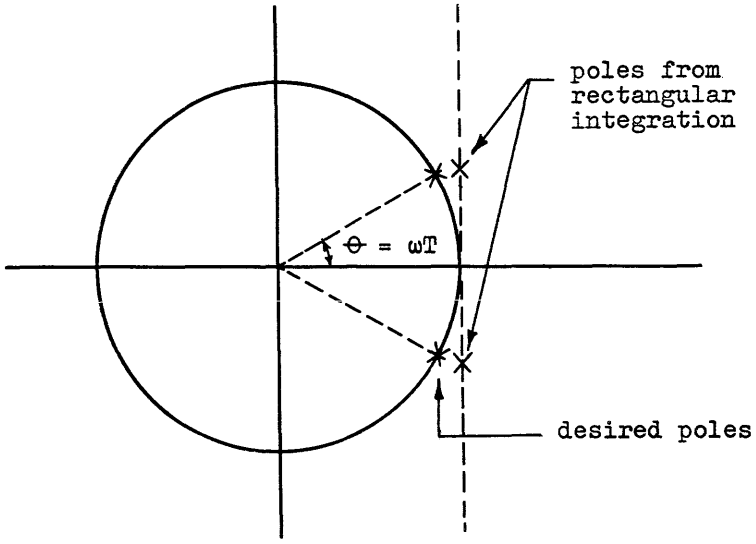


Figure 3. Location of Poles $x'' + \omega^2 x = 0$.

y(t)	Absolute Error (Equivalent Continuous Function)		True Integral
	Rectangular	Trapezoidal	
1	0	0	t
t	$-\frac{Tt}{2}$	0	$\frac{t^2}{2!}$
$\frac{t^2}{2!}$	$-\frac{Tt^2}{4} - \frac{T^2t}{12}$	$-\frac{T^2t}{6}$	$\frac{t^3}{3!}$
$\frac{t^3}{3!}$	$-\frac{Tt^3}{12} - \frac{T^2t^2}{24}$	$-\frac{T^2t^2}{12}$	$\frac{t^4}{4!}$
$\frac{t^4}{4!}$	$-\frac{Tt^4}{48} - \frac{T^2t^3}{12} - \frac{T^4t}{6!}$	$-\frac{T^2t^3}{72} - \frac{T^4t}{6!}$	$\frac{t^5}{5!}$
$e^{-\alpha t}$	$-\frac{T}{2}(1 - e^{-\alpha t})$	$-\frac{\alpha T^2}{12}(1 - e^{-\alpha t})$	$-\frac{1}{\alpha}(1 - e^{-\alpha t})$
$te^{-\alpha t}$	$+\frac{T}{2}te^{-\alpha t} - \frac{T^2}{12}(1 - e^{-\alpha t})$	$+\frac{\alpha T^2}{12}te^{-\alpha t} - \frac{T^2}{12}(1 - e^{-\alpha t})$	$-\frac{1}{\alpha}te^{-\alpha t} + \frac{1}{\alpha^2}(1 - e^{-\alpha t})$
$\cos \omega t$	$+\frac{T}{2} + \frac{\omega T}{12}\sin \omega t - \frac{T}{2}\cos \omega t$	$+\frac{\omega T^2}{12}\sin \omega t$	$+\frac{1}{\omega}\sin \omega t$
	Actual Function Developed		
$\cos \omega t$	$-\frac{T}{2} + \frac{T}{2 \sin \omega T/2}\sin(\omega t + \omega T/2)$	$+\frac{T \cos \omega T/2}{2 \sin \omega T/2}\sin \omega t$	

Table I. Absolute Errors.

HYBRID TECHNIQUES APPLIED TO OPTIMIZATION PROBLEMS

Hans S. Witsenhausen

Electronic Associates, Inc.

Princeton, New Jersey

Summary

A function dependent on the solution of a set of differential equations containing adjustable parameters, can be minimized by systematic search procedures in parameter space. Such procedures can be implemented by a hybrid system consisting of a general purpose analog computer and a digital expansion providing parallel logic building blocks. Programming of such a system is illustrated for a simple search procedure in n parameters.

Introduction

The solutions of many problems benefit from the use of a combination of both analog and digital computing devices. The required hybrid techniques of computation can be carried out by the linkage of an analog computer with a general purpose digital computer. For iterative solution of problems, in particular system optimization, a more desirable approach is the addition to the analog computer of a digital expansion system designed to provide primarily:

(a) Flexible automatic logic control of an analog computation or a long sequence of analog computations

(b) Large capacity, fast-access storage for problem solutions requiring a sequence of computations with variable function storage.

The assembly of many analog and digital computing components in an integral system gives the programmer complete flexibility in his organization of a problem solution. To demonstrate what is possible with such a system this paper considers a problem solution which makes good use of the flexible automatic logic provided by the proposed digital expansion system. For simplicity in explanation, this more basic feature of the hybrid computer is applied to the solution of a straightforward problem in optimization.

The digital expansion system provides "logic building blocks" which can be connected to form in this case the automatic control of the optimization program. These digital modules are electronic (solid-state) devices providing high computing speed. Any logical function could be implemented by relays and stepping switches, as has been done in many analog computer applications in the past.[6,7,8] However, higher speeds, flexibility and programming convenience are required in order to take full advantage of the speed of analog computers. This is not done in the repetitive operation mode where successive computations are simply repetitious or differ only by small perturbations. The full possibilities are realized by speeding up the usual real-time operation where the operator makes intelligent decisions between computations, on the basis of the fresh information just obtained. Successive computations can then be distinctly different.

The digital modules are interconnected on a removable patch panel. Programming is conveniently similar in principle to the approach familiar to analog computer users.

Digital Expansion System

General Features

The digital expansion system building blocks use two voltage levels to represent 0 and 1. A connection carrying a 1 level is said to be energized. Logical functions can be mechanized by interconnection of gates operating on these levels. Sequential logic requires flip-flops. Changes in the state of a flip-flop are clocked by a central system clock. Frequency of the clock pulses is adjustable from several megacycles down to manual control, pulse by pulse, by a pushbutton. This enables the operator to check parts of the set-up at leisure.

Counters provided with the system can be used to generate signals at binary or decimal submultiples of the central clock frequency.

Logic Building Blocks

For the present purpose, only four types of building blocks are required: (See Fig. 1)

(a) AND gates (2 inputs), the negated output is provided.

(b) OR gates (up to 6 inputs), the negated output is provided. (AND and OR gates, although logically interchangeable through negation, are distinguished for ease of intuitive programming)

(c) General purpose flip-flops (GPFF)

These are clocked RST flip-flops. The S (set) input commands the 1 state; the R (reset) input the 0 state; the T (trigger) input commands state reversal. Only one of these inputs may be energized at a time. The input levels are sampled by the clock pulse. Changes in state take place accordingly after disappearance of the pulse and the new state is established before the following pulse. Assembled in groups of four, these flip-flops have two group-input controls: a"clear" input which resets all four, and an "enable" input which can be used to inhibit the

RST inputs.

(d) Quadruple flip-flops in shift register connection. (QFF) These are sets of four flip-flops with internal connection into a shift register. Each set has eight outputs, the 0 and 1 states of each flip-flop.

The inputs are as follows:

Serial Inputs: The serial set (SS) and serial reset (SR) inputs are applied only to the first flip-flop and can be patched, in particular, from either the output of the last flip-flop of another QFF package for construction of long registers or the output of the last flip-flop in the register for closure of a loop.

Shift Input (SH): Transfers the state of each flip-flop to the next in order on one clock pulse and enables the serial inputs to control the first flip-flop.

Clear Input: (CLR) Resets all four flip-flops.

Set and Load Inputs: One set input(S) is provided for each flip-flop. The load input (L) is common to the building block. When the load input is energized, flip-flops for which the set input is energized will go into the 1 state.

Programming of the Digital Expansion System

The techniques for programming a digital expansion system are presently unfamiliar to most analog and digital computer users. The program is not introduced as a list of successive instructions but as a set of patching connections. This implies a task of organization similar to that needed in the design of a digital computer. However, there is considerable difference in the emphasis placed on component economy. Economy is essential for a computer design but of little concern in the programming of the digital expansion system as long as enough components are available. Thus, an empirical approach to digital expansion system programming, by successive simplifications and rearrangements is entirely acceptable.

The logic building blocks provide entirely parallel operation though it is possible to perform digital operations serially when desired. The digital and analog programs are introduced by their respective patch panels and interconnected by trunk lines forming one single set of parallel analog-digital components.

Several signals from analog comparators can be accepted simultaneously and combined to form several logical functions, while simultaneously several commands to analog switches (electronic relays, analog memory, mode control) can be generated.

This paper shows some aspects of this programming technique, for consideration by analog computer users. To this effect, it considers the general optimization problem which is both significant and representative of the class of applications requiring logic operations rather than large capacity storage.

For clarity, no attempt is made to reduce the number of building blocks to a minimum.

Use of Subroutines

Operations in the analog computer have to follow a definite time sequence. Whenever a sequence of operations has to be repeated many times, it is defined as a subroutine. One way to implement a subroutine is to use a ring counter, acting effectively as an electronic stepping switch, where the sequence of states determines the sequence of operations. A ring counter is obtained by assembling QFF units into a shift register of the desired length and then connecting the outputs of the last stage to the serial inputs of the first QFF. Initially the first flip-flop in this loop is set to the 1 state and the others are in the zero state. When the shift input is energized each clock pulse will advance the sequence by moving the 1 state to the next flip-flop resetting the previous one. After all flip-flops have been energized successively the next shift returns the ring to the initial state. Each flip-flop can command in parallel a set of operations, including the initiation of other subroutines.

The effective use of subroutines depends on two features:

(a) Appropriate control by other routines.

(b) Provision for delays necessary for operation of analog components.

Control of Interlocking Subroutines

When a routine calls, at one stage, for the execution of a subroutine it is often necessary to halt the advance of this routine until the subroutine has gone through its entire cycle. This interlock can be obtained by the formation for a given subroutine A of 3 signals.

sA: this signal is the call for subroutine A, it is formed by an OR gate which receives signals from all the sources of calls, generally different positions in one or more other routines.

eA: this signal is generated at the time routine A returns to its initial state. It is produced by an AND gate supplied from the last state of A and the shift input of A. The eA signal is energized for one clock time and is called the end signal of subroutine A. If a subroutine should be repeated n times, a separate counter (analogous to index registers) can be used to produce the end pulse.

eA^x: this signal is produced by an OR gate supplied by eA and by the negation $\overline{sA}$ of sA. Called the interlock signal, eA^x is fanned out to all routines which call subroutine A.

The required interlock is obtained by supplying the shift input of a routine by the AND combination of its own call signal and the interlock signals of all the routines for which it calls.

Delays

Some routines control electronic relays, analog memory and mode of integrators in the analog computer. When exercising such control the routine must incorporate delays to allow sufficient time for changes in computation to take place so that solutions do not include the transient response of amplifiers and the charging of capacitors necessary between consecutive operations.

These delays can be obtained by an AND gate at the shift input to the routine; this gate is fed by the shift command and timing signals of slow rate derived from the carries of the central clock counter. Different rates can be used in different routines.

If delays of different sizes are required in the same routine, the routine itself can switch control from timing signals of one rate to those of another. Alternatively general purpose delay units (one-shot multi-vibrators) can be used to inhibit shifting during the required time span.

The delay required while the analog computer is going through an operate cycle, is obtained by inhibiting shifts of the controlling subroutine in the operate mode; this takes care of the cases where the run time is variable, depending on some condition in the problem.

Signals From Digital To Analog Equipment

Signals computed by digital components and transferred to the analog computer are required for the following purposes:

(a) Control of the analog computer mode.

(b) Control of the mode of individual integrators or groups of integrators.

(c) Control of analog storage units.

(d) Control of electronic gates used for switching operations.

For the control of the analog computer modes (OPERATE, HOLD, INITIAL CONDITION) input terminations are provided on the digital patchboard. A pulse applied to any input termination will trigger the corresponding mode.

The other control functions are effected by electronic gates which accept digital signals. The opening or closing of the gate is a function of the applied digital level. In many cases, a GPFF will buffer these commands so that the condition of the gate is maintained until new signals are fed to the GPFF. One GPFF can control several gates in synchronism.

Signals From Analog To Digital Equipment

Particular signals produced within the analog computer are transferred to the digital equipment. These signals provide information of:

(a) The analog computer mode, and

(b) The state of comparators

The computer mode signals are ICL, HL, and OPL which are energized whenever the analog computer is in the initial condition, hold, or operate mode, respectively.

Comparators will supply one of two levels depending on the sign of the algebraic sum of their analog voltage inputs. These levels are normalized for direct utilization in the digital system.

The "Optimization Problem"

A set of algebraic and differential equations, mechanized on the analog computer, uniquely determines in one computation (the run) a value E, the loss-function or error-function to be minimized by manipulation of n parameters $\lambda_1 \ldots\ldots, \lambda_n$.

This problem occurs in many situations;

--solution of a set of algebraic equations
--matching of boundary conditions
--search for eigenvalues
--process optimization
--model building
--curve fitting etc.....

While special techniques exist for each class of problem, the general problem is considered here.

Analytical methods[1,2,5] for solving optimization problems, such as that of steepest descent, make use of partial derivatives determined either from the analytical problem statement or by parameter influence techniques[3]. The experimental approach considered is quite different conceptually, for the search for an optimum is entirely based on repeated tests with different parameter combinations. Experimental methods[4,6] can use a varied degree of sophistication to accelerate convergence to an optimum and to overcome possible difficulties due to the features of the unknown hypersurface. $E(\lambda_1, \ldots\ldots, \lambda_n)$.

The price of refined search procedures increases very rapidly with the number n of parameters. For the purpose of illustrating the methods of hybrid programming of any search tech-

nique the following simple approach is selected.

Search Procedure

Assume normalization of all parameters so that a unique "exploration distance", h (say 1 volt on a ± 100 volt computer), is acceptable.

Given λ_{io} and E_o at an arbitrary starting point, the first partial derivatives are approximated by

$$\frac{\partial E}{\partial \lambda_i} \cong \frac{\delta E^+_i - \delta E^-_i}{2h}$$

where

$$\delta E^+_i = E(\lambda_{1o},\ldots,\lambda_{io}+h,\ldots,\lambda_{no}) - E_o$$

$$\delta E^-_i = E(\lambda_{1o},\ldots,\lambda_{io}-h,\ldots,\lambda_{no}) - E_o$$

All n partial derivatives are thus approximated by means of 2n calculations of E (runs). In order to place most of the load in the digital part of the hybrid system, the values of the partial derivatives are stored in quantized form. Instead of storing $\partial E/\partial \lambda_i$, a quantity p_i, susceptible of only 3 values, is defined by comparison with a limit L selected by the computer operator. Extension to a higher number of values poses no special problem.

$$p_i = +1 \quad \text{if} \quad \delta E^+ - \delta E^- < -2hL$$

$$p_i = 0 \quad \text{if} \quad -2hL < \delta E^+ - \delta E^- < 2hL$$

$$p_i = -1 \quad \text{if} \quad \delta E^+ - \delta E^- > 2hL$$

The vector with components p_i defines a direction in n-space in which a decrease in E is probable. It is close to the steepest descent direction for which the components are $\partial E/\partial \lambda_i$. The program stops if all $p_i = 0$. With one of the 3^n-1 possible directions determined, the parameters are changed to $\lambda_{io} + p_i\Delta_1$, where Δ_1 is a fixed quantity (say 2 volts). If an improvement, indicated by a decrease in E, is obtained the values of all parameters λ_{io} are "updated" to give a new starting point. Additional changes by steps, Δ_1, in the same direction are carried out until no more improvement occurs. Then a smaller step-size Δ_2 (say .2 volt) is selected and when this also fails, the partial derivatives are redetermined and the process repeated. If, after a new determination of the quantities p_i, even a single small step leads to no improvement, the program stops. If the analog computer does not overload, the program will ultimately reach one of the stop conditions and this should occur in the vicinity of the optimum.

An advantage of this technique is that the equipment requirement, as demonstrated later, has a slow linear increase with n. It is recognized that many possible features of the E function (high curvatures, local minima, narrow winding "canyons", etc.) will defeat the search procedure; on the other hand, the very same features can also defeat more elaborate and equipment-consuming alternatives.

If a Euclidean metric based on voltage is used, the step length $\Delta\sqrt{\Sigma p_i^2}$ will vary between Δ and $\Delta\sqrt{n}$ depending on the direction selected. This is acceptable because there is no a priori reason for a voltage metric to have special significance.

Program Refinements:

The procedure outlined above can be improved at a small expense in equipment by addition of the following features:

(1) If, after determination of direction, at least one large step Δ_1 is successful, then no small steps are used before a new determination of direction. However, when a single large step does not decrease E, a smaller step Δ_2 is used. This program arrangement is a time-saving feature.

(2) For non-convex E functions it is necessary to redetermine direction from time to time even if improvement is constantly obtained. To this end an upper limit N (say 8) is put on the number of steps in a direction before redetermination of direction.

(3) The limit L for quantization of the slopes should be carefully chosen. If L is too large, the program will stop in any "flat" region of the E-function and for non-convex functions this final value of E may be far from a minimum. If L is too small, all quantities p_i will have values ±1 most of the time. This has the disadvantage of reducing the effective set of possible directions from 3^n-1 to 2^n (for n=6 from 728 to 64) and therefore the efficiency of the search procedure.

Many refinements are possible to avoid this difficulty. The following is selected for its economy of equipment.

Each slope is compared with two limit values L_1 and L_2 ($L_1 < L_2$). If no slope exceeds L_2 the quantities p_i are based on L_1. If any slopes exceed L_2 the quantities p_i are based on L_2. This procedure is poor whenever many slopes are clustered in a narrow range around either L_1 or L_2, but this is unlikely to happen frequently in a long optimization. The procedure requires just one more comparator in the analog computer.

(4) Whenever desired, half the exploration runs can be saved by replacing the slope evaluation $\frac{\delta E^+ - \delta E^-}{2h}$ by $\frac{\delta E^+}{h}$, that is use forward instead of central differences.

Analog Program

From the statement of the optimization procedure the analog circuit (Fig.2) can be derived immediately as follows.

The problem equations are mechanized by circuits which accept n input signals λ_i. The last updated values of the parameters λ_{io} are held on n analog memories. The size h, Δ_1, or Δ_2 of the increments $\delta\lambda$ is selected by 2 switches and distributed with both signs to n pairs of switches; for each parameter one switch determines the sign of $\delta\lambda_i$ the other determines the addition of $\delta\lambda_i$ to λ_{io}. This addition is performed by an analog memory so that the new value $\lambda_i = \lambda_{io} + \delta\lambda_i$ can be transferred to the λ_{io} memories when updating of the parameters is required.

The duration of the operate period (the run) is determined in general by the problem itself and can be dependent on the parameters. A comparator produces the hold signal when the appropriate condition is satisfied. At that time the value E becomes available.

The previous updated value E_o is held on a memory amplifier. The difference $E - E_o$ is fed to two memory amplifiers so that δE^+ and δE^- can be successively computed, stored and then compared. The value E itself is stored to permit updating of E_o when required. If the problem integrators can conveniently be used as storage for the value E the number of analog memories required at the output can be reduced from four to two.

For determination of the direction parameters p_i the sign of $\delta E^+ - \delta E^-$ is sensed by a comparator, and the absolute value of this difference is compared with two constants by two more comparators.

After a step has been carried out the change in the E function is stored in one of the δE memories, say δE^+, and the sign of this quantity is sensed by a comparator to determine whether improvement has been obtained. Table 1 shows the conditions for digital control of the circuit switches.

The comparators deliver information about the E-function to the digital equipment according to Table 2.

Digital Expansion System Program

The digital expansion system program must insure that, once begun, the computation progresses automatically in its search for an optimum. This part of the program has inputs only from the IMP, N, L_1, L_2 comparator signals and the analog computer mode signals.

Program Structure

The first step in producing such a program is to specify a flow diagram of the required sequence of operations. (Fig. 3) This flow diagram organizes the operations into the following subroutines:

Master Routine: selects the step size h, Δ_1 or Δ_2 tests the two stop conditions and calls alternately for the subroutines Explore, for determination of a direction, and Proceed, for carrying out steps in that direction.

Explore Routine: calls for a computation of E with a single parameter displaced by +h then by -h. It steps through all parameters storing the p_i as they are obtained. When all p_i have been obtained it sets the $\delta\lambda_i$ switches accordingly.

Run Routine: executes the computation of E for the prevailing λ_i values.

Proceed Routine: A step size (Δ_1 or Δ_2) being selected by the Master routine and a direction (a set of p_i) having been set by the Explore routine, the Proceed routine displaces the λ_i accordingly, calls for computation of E and tests for improvement. The routine repeats as long as improvement is obtained, unless a preset number

TABLE 1

UNITS	DIGITAL LEVEL 0	1
Δ_1/Δ_2 relay	Δ_1	Δ_2
h/Δ relay	h	Δ
$\delta\lambda_i$ Off/On relay $i = 1, 2, \ldots n$	Off	On
$\delta\lambda_i$ + / - relay	+	-
Stores for E_o and λ_{io}	store	track
Stores for -E and $-\lambda_i$	track	store
δE^+ store	track	store
δE^- store	track	store

of steps is exceeded, and calls for the updating routine after each improvement.

Update Routine: transfers the present values of λ_i and E into the λ_{io}, E_o memory units.

Program Mechanization

The subroutines are instrumented in detail as follows:

Subroutine C (Fig. 4) (RUN) will start with the analog computer in its IC mode. The call signal sC is obtained by an OR gate (not shown) which combines all sources of calls from the EXPLORE and PROCEED routines. The routine will be started after new commands have been given for the $\delta\lambda$; a delay is necessary for acquisition of the new λ_i by the analog computer (especially if some of them are initial conditions). Next the OP mode is triggered and the subroutine inhibited by the OP level. Upon automatic hold, the routine resumes with a delay to allow for acquisition of the final δE values. Then the δE^+ store and, if permitted by a signal ENHM (enable hold minus) generated in the EXPLORE routine, the δE^- store are switched to hold by setting the flip-flops controlling the memories. After another delay, the subroutine returns to its initial state, generating the end signal eC. The interlock signal eC^x is used to inhibit the calling routines.

Subroutine D (Fig. 5) (UPDATE) will update the λ_{io}, E_o values to the last λ_i, E values. Here delays are provided to enable the tracking stores to acquire steady-state values. Since only two states are required the ring counter is replaced by a GPFF with the trigger input acting as the shift control. Signals eD and eD^x are generated, eD^x inhibits the PROCEED routine.

The delays for subroutines C and D are obtained by the use of a slow clock rate (SLCL) derived from a carry of a clock counter.

Subroutine A (Fig. 6) is the program for determination of approximate partial derivatives (EXPLORE). It is associated with shift registers δ^x, δ, N^x, N.

δ^x and N^x control the analog value of $\delta\lambda$: The data obtained during exploration is stored in registers δ and N.

The subrgutine begins with the δ^x, N^x cleared. First δ_1^x is loaded and subroutine C called then N_1^x is loaded (implementing $\delta\lambda$=-h) and subroutine C called with ENHM turned on. At this stage the correct value of N, L_1, L_2 appear on the comparators. A test is carried out to determine which comparison level should be used. The first occurence of a slope $> L_2$ will clear all previous δ's, set flip-flop L, which controls determination of future δ's on the basis of L_2 alone. If L_2 is not exceeded, δ's are selected according to L_1. The routine shifts the new data into the δ, N stores, sends the δE memories to the tracking mode and repeats, shifting to the next parameter. When the last parameter is reached the δ^x, N^x are cleared and the δ, N are transferred into δ^x, N^x registers in a parallel loading operation. The end signal and the interlock signal eA^x are generated.

Subroutine B (Fig. 7) will carry out steps in the chosen direction (PROCEED). It calls upon subroutine C for an analog run, then senses the IMP comparator level. If no improvement is obtained the end pulse $eB^{(1)}$ is generated. If improvement is obtained, subroutine D is requested for updating and the S flip-flop associated with the master routine is set, to record success.

TABLE 2

Analog Condition		Digital Level	Name	Meaning
sgn δE^+	+	0	IMP	Improvement
	-	1		
sgn $(\delta E^- - \delta E^+)$	+	0	N	Best direction
	-	1		is $\delta\lambda_i < 0$
sgn $\left(\left\| \delta E^- - \delta E^+ \right\| \cdot 2hL_j \right)$	+	0	L_j	Change in λ_i is
	-	1		advisable

The end signal eD of the UPDATE routine increments a counter (r) and subroutine B repeats unless r=N where the routine end pulse $eB^{(2)}$ is generated. The two types of end signals are combined to generate the interlock signal eB^{x}.

Master Routine M (Fig. 8) first calls upon A for exploration, selecting step size h. Next it tests the stop condition $\delta_1 = .. = \delta_n = 0$. It then calls upon B (PROCEED) with the large step-size Δ_1. If, and only if, at the end of B, S = 0 (no improvement) B is repeated with the smaller step-size Δ_2. Next the stop condition S = 0 is tested. If S = 1 it is reset to 0, the δ^{x}, N^{x} are cleared and routine M repeats.

Initialization (not illustrated): It is convenient to initialize the digital system by clearing all its flip-flops from a central control. At this time the operator can introduce starting values for λ_{io}. The hybrid program will start as soon as the system clock is turned on. A non-repeating initialization routine is patched to perform the following operations: (a) set all flip-flops which should be energized initially; (b) call on the RUN subroutine to calculate and store the value E_o corresponding to the initial parameter values; (c) start the master routine of the problem.

Equipment Requirements

If no further rearrangements are carried out, the following numbers of components are required, where n is the number of parameters.

On the analog side:
- 2n + 2 switches
- 2n + 4 analog memory units
- 5 comparators

On the digital side:
- 11 GPFF
- $6 + 4 \left[\frac{n}{4}\right]$ QFF
- 50 AND/OR gates
- 1 counter

Conclusions

The addition of the digital expansion system providing logic capability to the general purpose analog computer will enlarge its capabilities as an automatic computing device.

The less flexible relay and stepping-switch arrangements used in the past are eliminated and full use is made of the computing speed of the analog computer. This alone brings within reach problems, such as optimization, where computing times were previously excessive in most cases. The further addition of point storage in digital form on delay lines will permit the use of serial techniques for solution of integral and partial differential equations. Problems of a statistical nature can be handled with a maximum of automatic operation. Many simpler auxiliary functions for GPAC's can be easily carried out (automatic scale changes, editing of graphical plots which are not continuous in time, Fourier analysis).

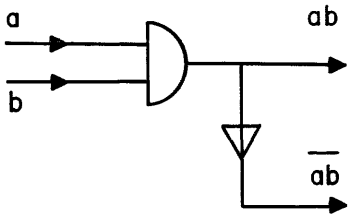
Acknowledgment

The author is indebted to Mr. J. Paul Landauer of Electronic Associates, Incorporated, for information and suggestions.

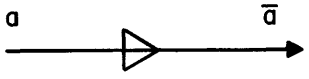
References

1. Brooks, S. H.,"A Comparison of Maximum-Seeking Methods," Operations Research, 7, No. 4, 1959.
2. Kinney, R., "A Model Adaptation Technique for Computer Controlled Batch Chemical Process," M.S.Thesis, Case Institute of Technology,1960.
3. Brunner, W., "An Iteration Procedure for Parametric Model Building and Boundary Value Problems," Western Joint Computer Conference,1961.
4. Box, G. E. P. and Wilson, K. B., Journal Royal Statistical Society, B13, 1, 1951.
5. Lapidus, L., Shapiro, E., Shapiro, S. and Stillman, R. E., "Optimization of Process Performance", A.I.Ch.E. Journal, 7, 288, 1961.
6. Munson, J. and Rubin, A. I., "Optimization by Random Search on the Analog Computer," I.R.E. Trans. EC-8, No. 2, 200, 1959.
7. Follin, J. W., Emch, G. F. and Walters, F. M. "Modification and Additions to the REAC," Project Cyclone Symposium II, p. 173, 1952.
8. Hansen, G. R., "The Time-Sequence Controller for Automatic Operation of the Electronic Differential Analyzer," Project Typhoon Symposium III, p. 65, 1953.

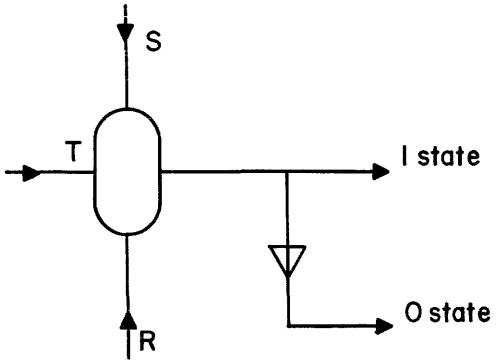
SYMBOLS



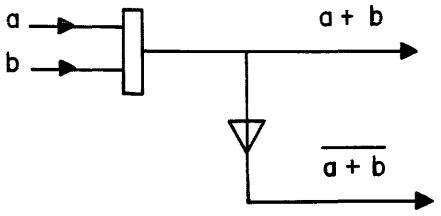
AND gate



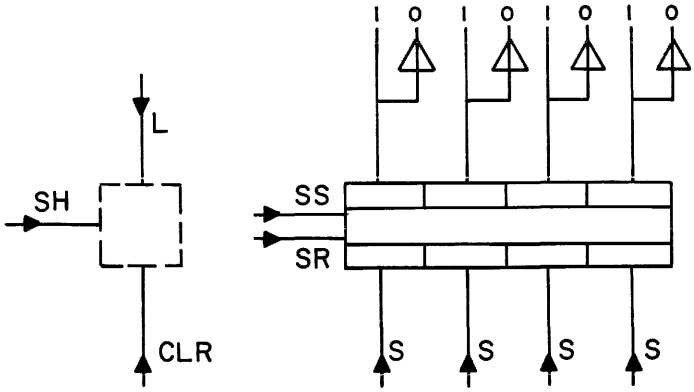
negation



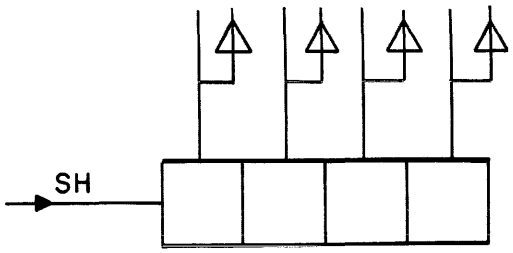
GPFF general purpose flip-flop



OR gate



QFF quadruple flip-flop



Ring Counter

Figure 1: Digital System Symbols

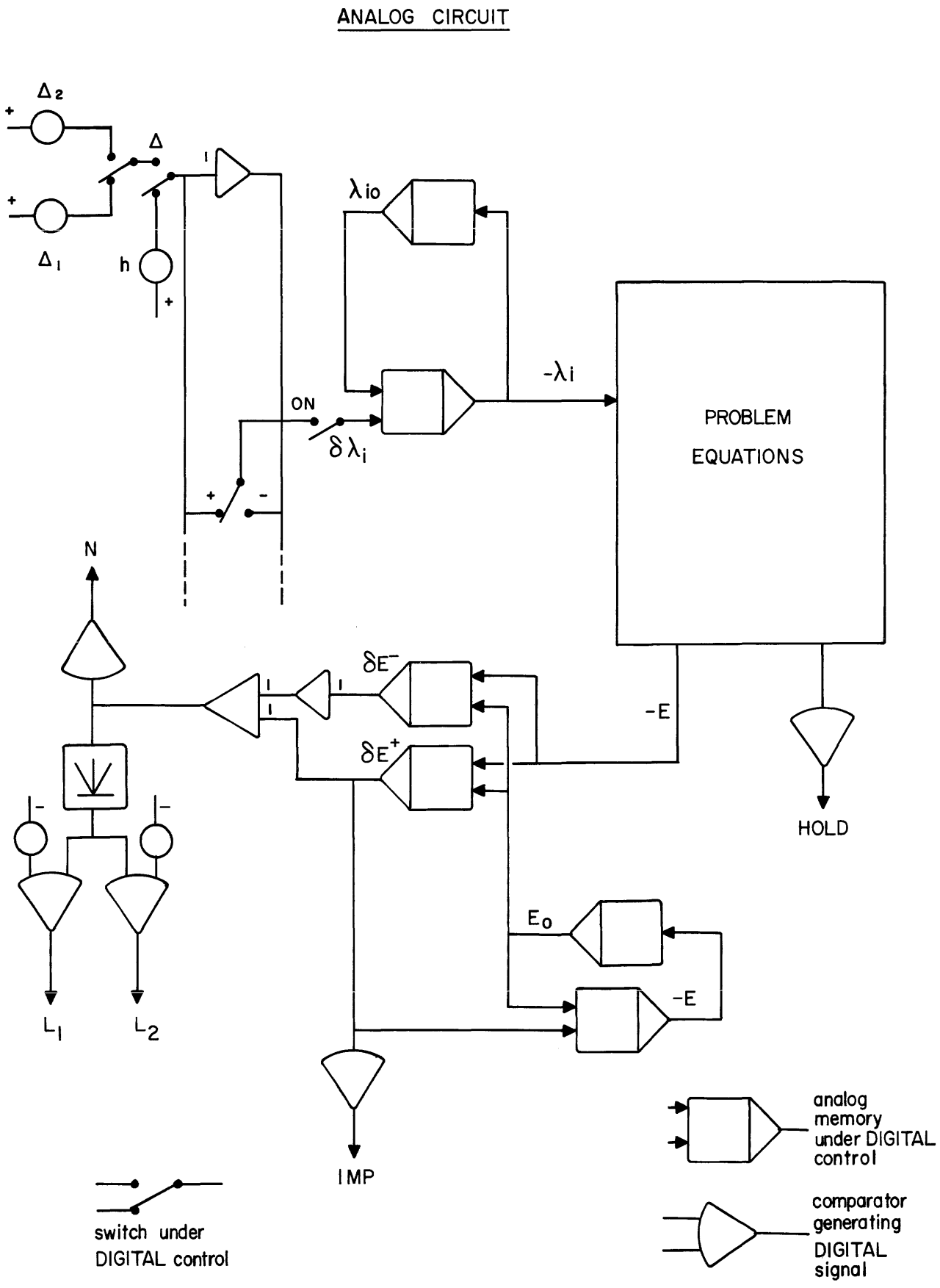


Figure 2: Analog Computer Circuit

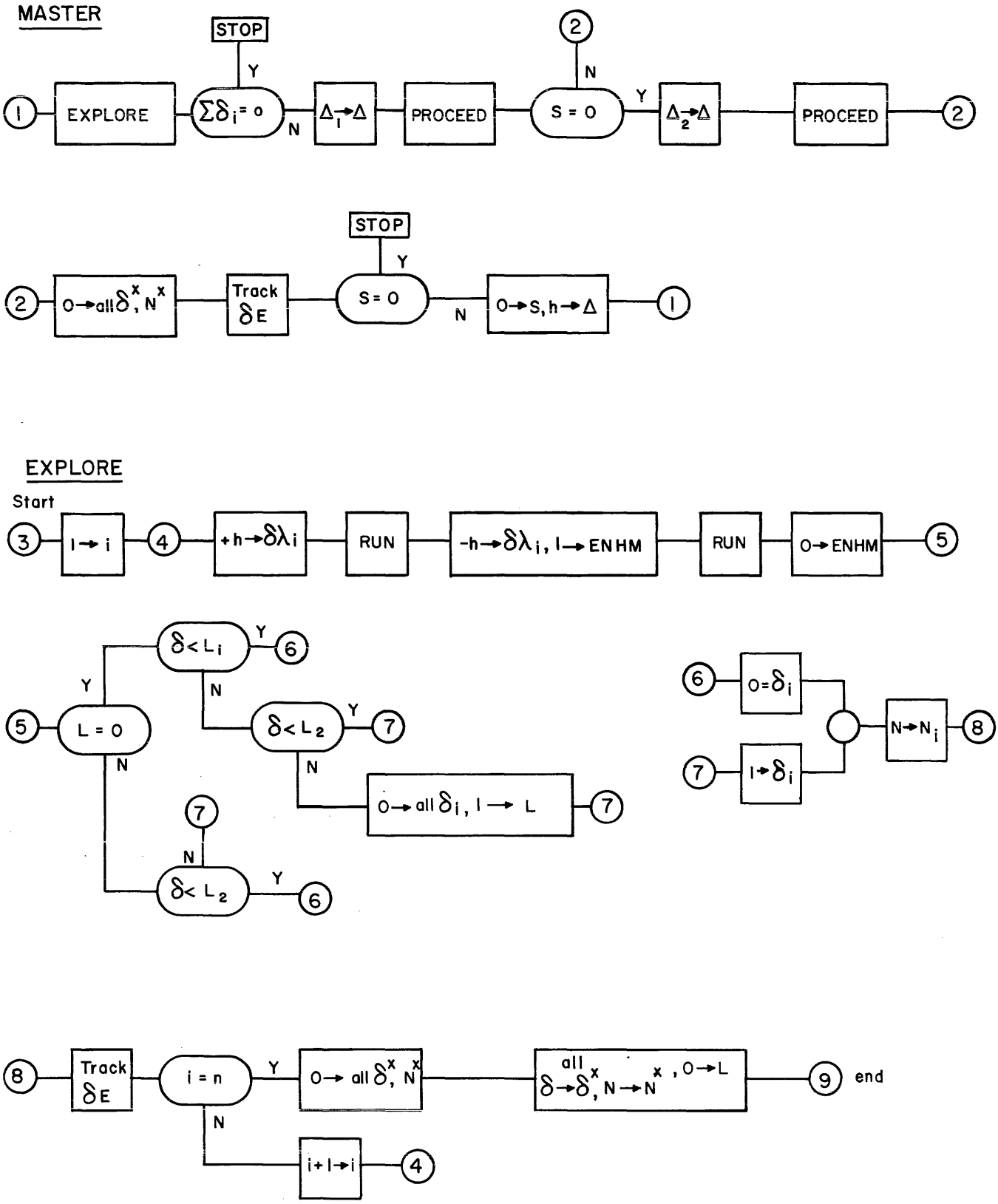
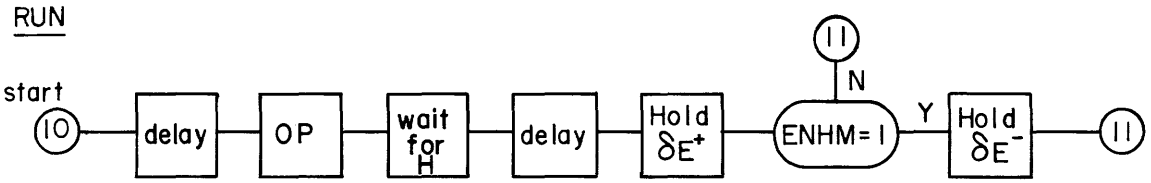
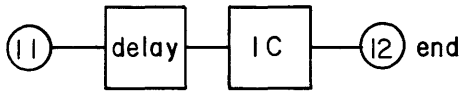
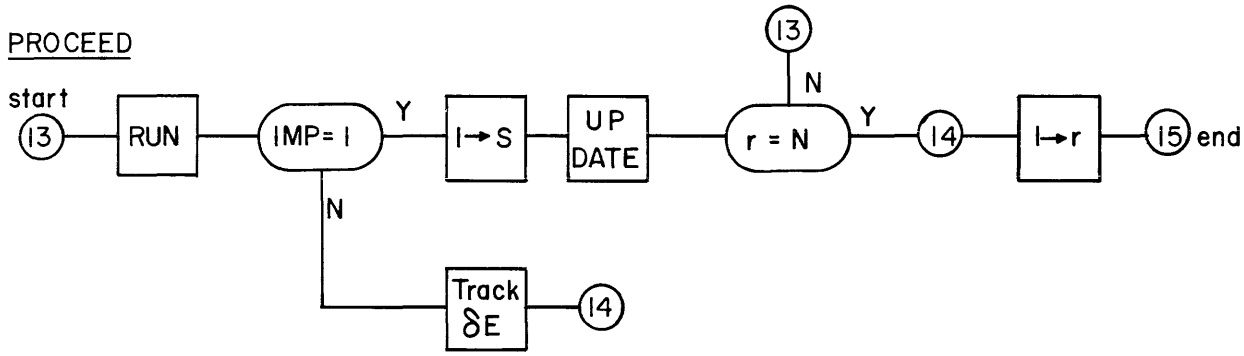
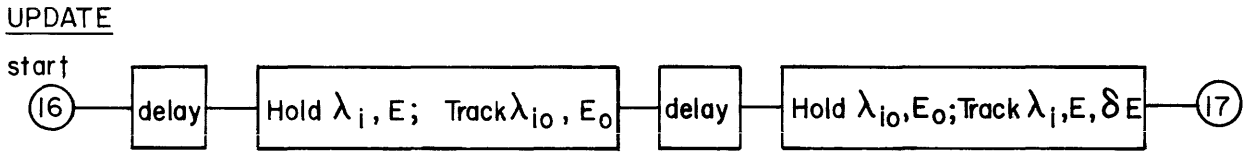


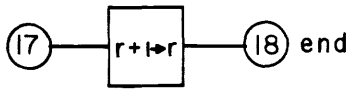
Figure 3: Program Flow Diagram


RUN
start
10
delay
OP
wait for H
delay
Hold δE+
11
N
ENHM=1
Y
Hold δE-
11


11
delay
IC
12
end


PROCEED
start
13
RUN
IMP=1
Y
I→S
UP DATE
13
N
r = N
Y
14
I→r
15
end
N
Track δE
14


UPDATE
start
16
delay
Hold λi, E; Track λio, Eo
delay
Hold λio, Eo; Track λi, E, δE
17


17
r+1→r
18
end

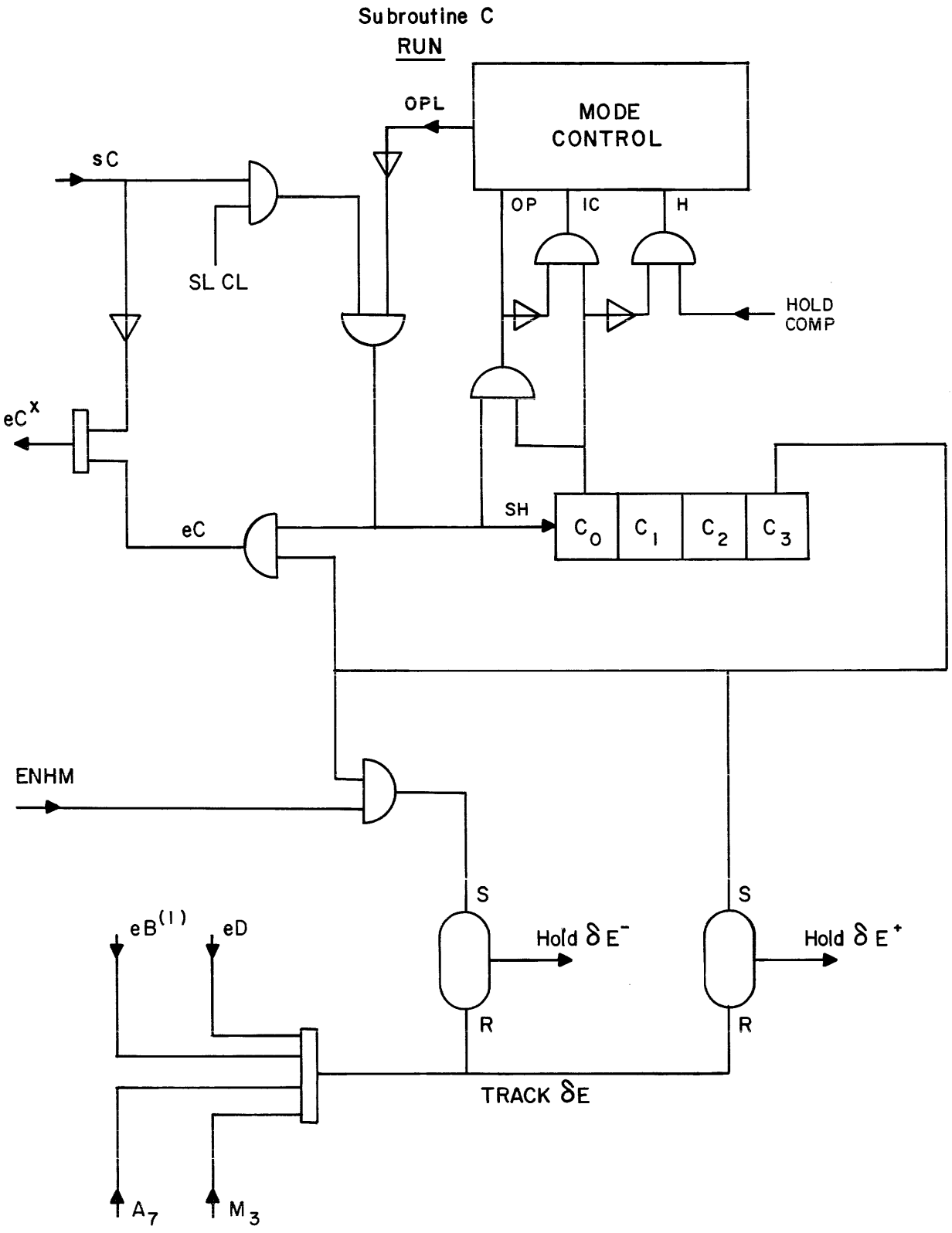


Figure 4: Subroutine C. RUN

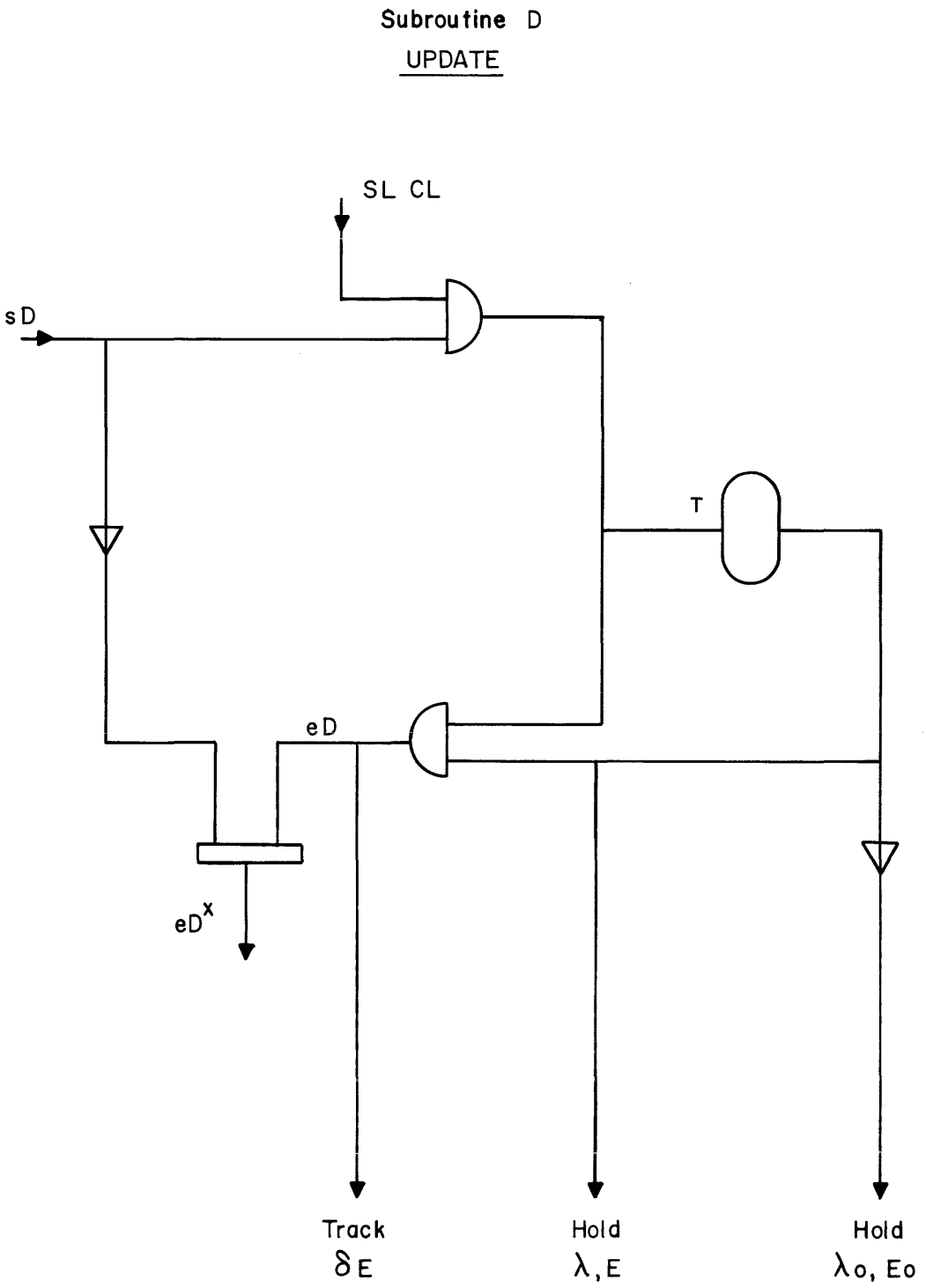


Figure 5: Subroutine D. UPDATE

Subroutine A
EXPLORE AND STORE

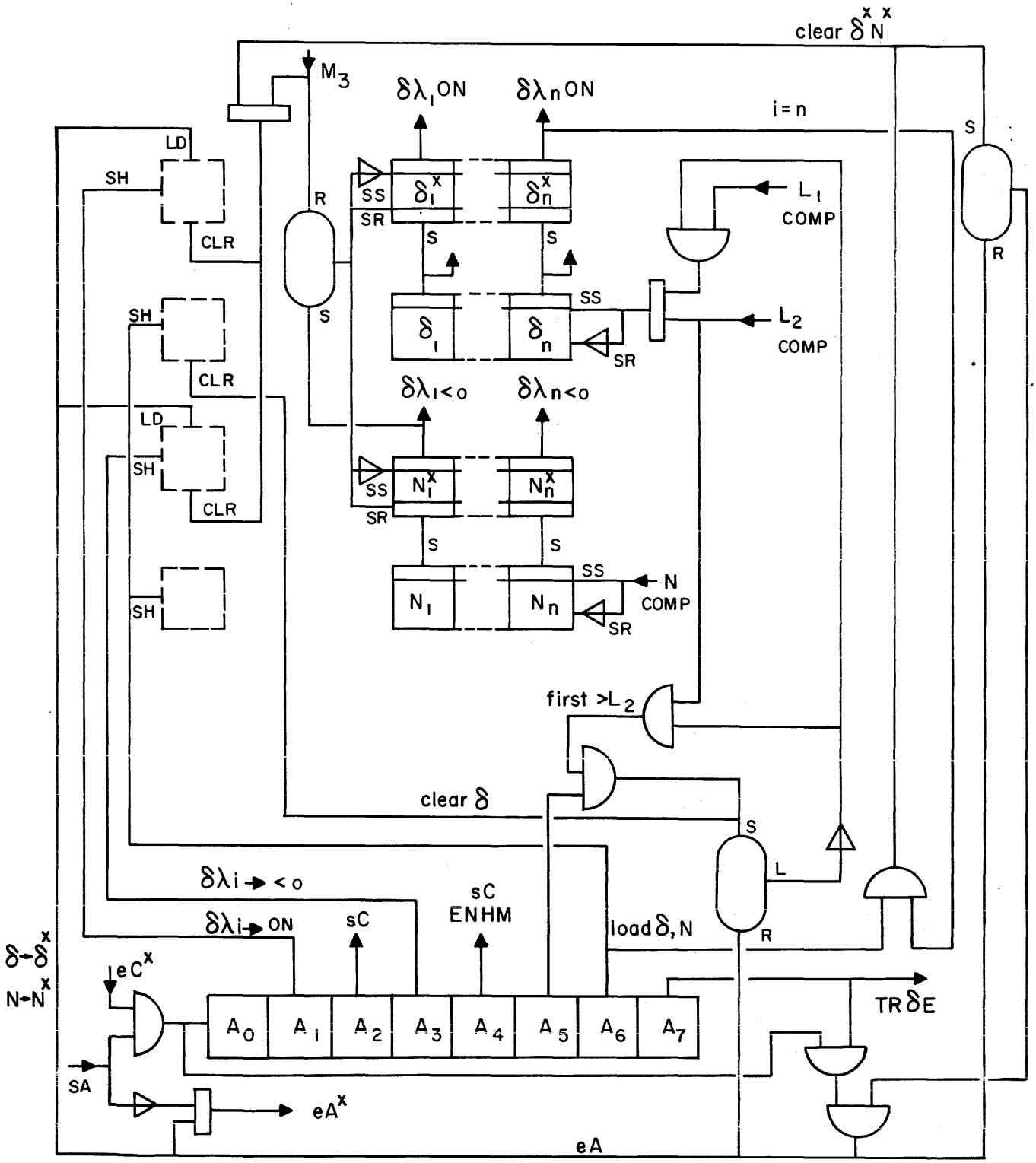


Figure 6: Subroutine A. EXPLORE

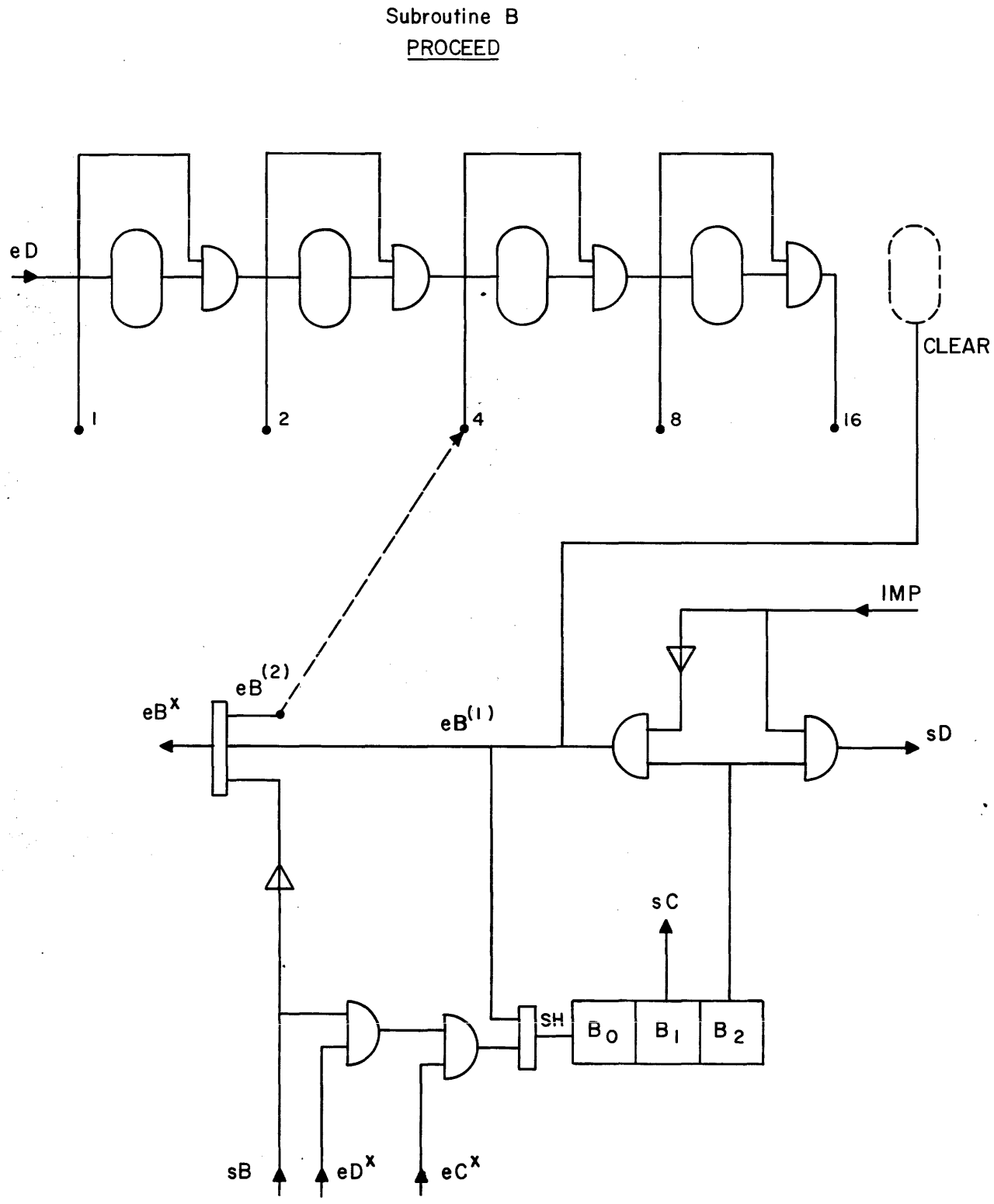


Figure 7: Subroutine B. PROCEED

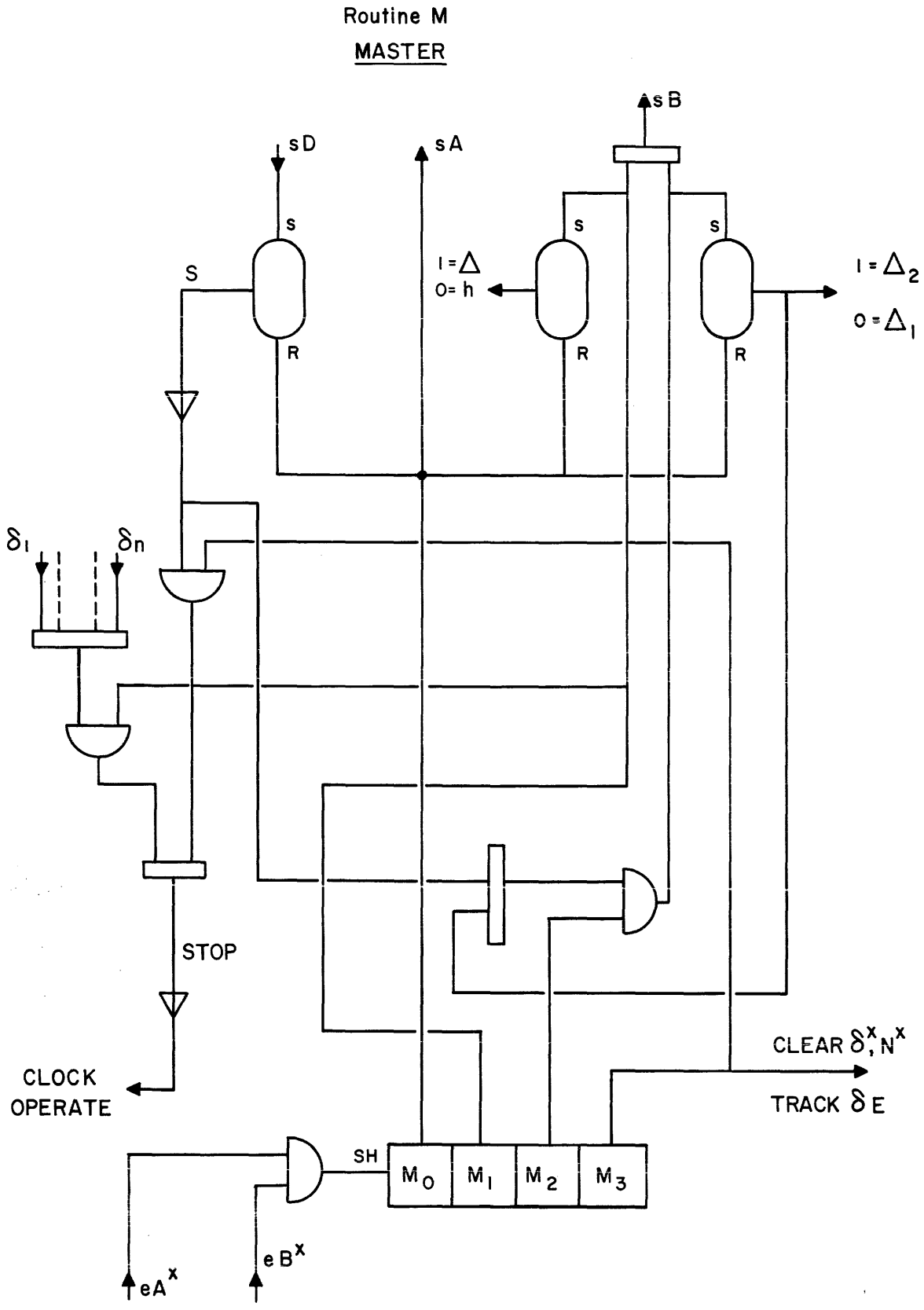


Figure 8: Routine M. MASTER