Report R-196

PROGRAMMING FOR WHIRLWIND I

June 11, 1951

Sponsored by

OFFICE OF NAVAL RESEARCH

Report by
Hrand Saxenian

ELECTRONIC COMPUTER DIVISION
SERVOMECHANISMS LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
CAMBRIDGE 39, MASSACHUSETTS

ABSTRACT

The Whirlwind I computer, WWI, at MIT is a high-speed electronic digital computer. It will be used in a wide variety of applications involving computations for mathematical and engineering problems, accounting, statistical analysis, and simulation and control processes. The preparation of such problems for WWI is essentially a matter of expressing the processes to be performed in terms of a list of the computer's basic operations. Such a list is called a program, and there are a maximum of 32 basic operations, each of which WWI performs automatically upon receiving a single instruction from a program. WWI also automatically sequences the operations listed in a program so that they are performed one after the other.

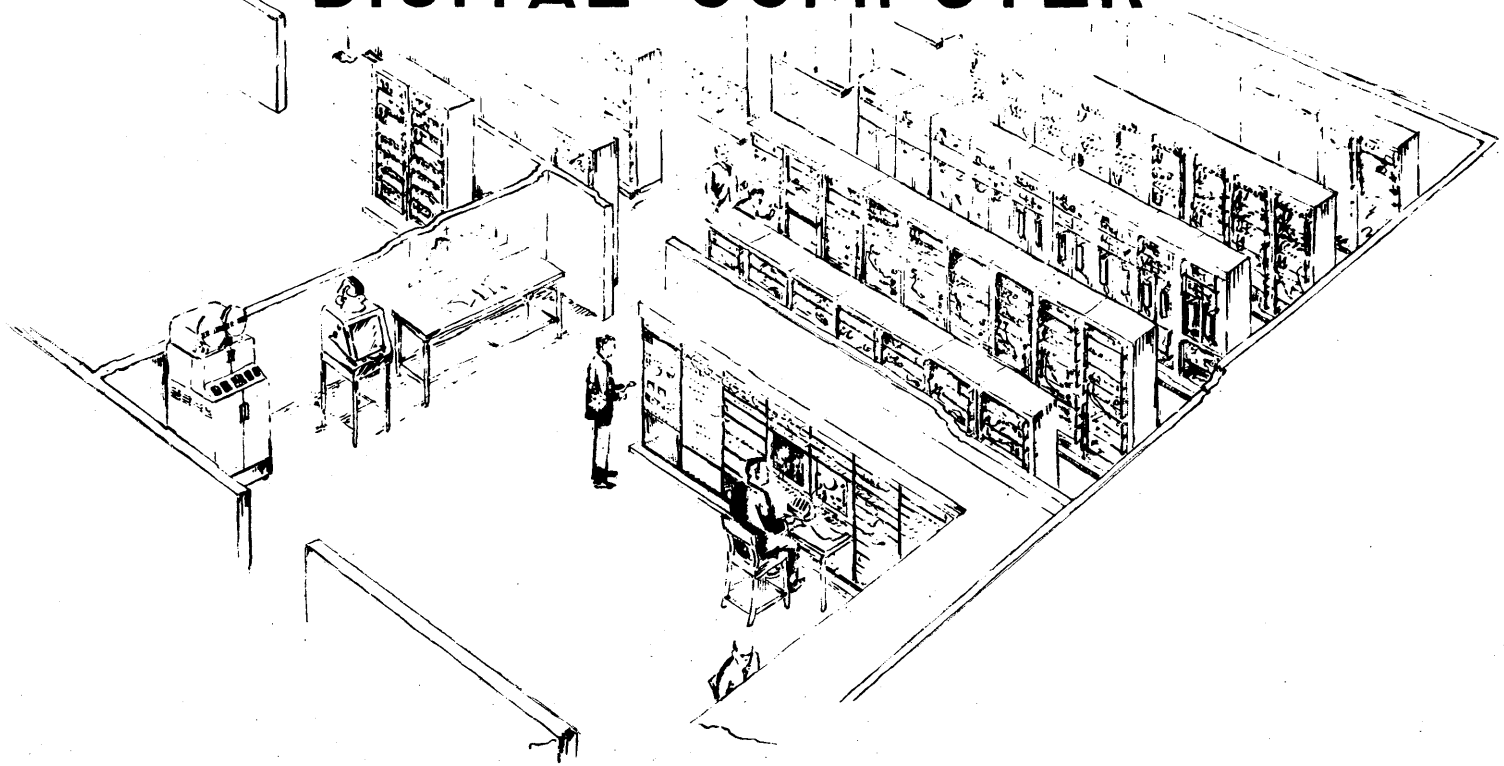
The terminology and concepts of programming for WWI and of WWI itself, essential to an understanding of programming for the computer, are presented here in the section on Essentials of Programming and in the Appendix. A progressively developed set of examples of programs is included to indicate how programs are developed, to illustrate the nature of various ways of handling different processes, and to suggest the flexibility of programming techniques. Use is made of flow diagrams and of subroutine techniques, both in the analysis of problems and in the preparation of programs.

Once the programmer understands the basic WWI operations and how programs are developed from them, the programming of a problem reduces to an analysis of the original problem. This analysis is the same, of course, as if the actual computations were to be made by less automatic operations.

WHIRLWIND I

is a high-speed electronic

DIGITAL COMPUTER



CONTENTS

PAGE

ABSTRACT 1

FOREWORD 3

INTRODUCTION 4

WHAT WWI IS 5

WHAT WWI DOES 6

I. ESSENTIALS OF PROGRAMMING FOR WWI 9

A. WHAT A PROGRAM IS 10

B. ELEMENTS OF PROGRAMS 11

C. FORMS OF PROGRAMS AND WORDS 12

D. BASIC COMPUTER ELEMENTS 15

E. WHAT WWI DOES WITH PROGRAMS 16

F. SUBROUTINES 18

G. FLOW DIAGRAMS 19

II. DEVELOPMENT OF PROGRAMS FOR WWI 20

LIST OF EXAMPLES 21

III. APPENDIX 61

A. BINARY NUMBERS IN WWI ..... A i

B. DECIMAL-TO-BINARY CONVERSION ..... A iv

C. SHIFTING NUMBERS IN WWI ..... A vi

D. PROGRAMMING IN OCTAL (BASE 8) FORM ..... A vii

E. SHORT GUIDE TO CODING ..... A ix

FOREWORD

This report is an introduction to programming for the Whirlwind Computer, WWI, at MIT.

It begins with a general statement of what WWI is and what WWI can do. This statement is intended to suggest both the possible areas of application and the limitations of a high-speed digital computer such as WWI.

Part I presents essentials of WWI programming.

Part II presents examples of WWI programs. The examples are chosen to illustrate how programs are developed and to suggest various programming techniques.

Part III is an appendix which includes a discussion of numbers in WWI and the WWI operation code.

INTRODUCTION

WHAT WWI IS

WHAT WWI DOES

WHAT WWI IS

WWI is a high-speed electronic digital computer. It is a rapid and versatile tool for processing information. Built into it are electronic circuits for automatically sequencing and performing several basic operations. Complicated processes may be developed from sequences of these basic operations. Appropriate instructions for the performance of these operations are prepared in the form of programs. Programs are read into and stored in WWI internal storage. WWI then carries out these instructions.

The high speed of operation of WWI is made possible by the use of electronic circuits instead of moving parts used in slower devices. Operation time is measured in millionths of a second. An average of ten thousand operations such as addition, multiplication, and division are performed in one second. At this rate, in 15 minutes WWI performs error-free calculations which would require 15 years of continuous hand calculation.

All WWI's circuits work on the basis of their ability to perform the simple task of distinguishing between the presence and absence of current flow. The digits 1 and 0 are assigned to the two states: current on and current off. The computer does all its calculating by counting with these two digits alone, in the binary system, just as we are accustomed to calculating with ten digits in the decimal system.

Since WWI counts with discrete digits, it is called a digital computer. This distinguishes it from analog devices which measure continuously changing quantities and give analogous physical, rather than numerical, indications of magnitude.

WHAT WWI DOES

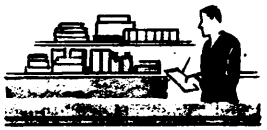
Upon receiving appropriate instructions from a previously prepared program WWI automatically performs sequences of its basic operations very rapidly, so that:

1) It can save many hours of labor on routine computational tasks.
2) It makes practical the performance of many mathematical and statistical manipulations which would be otherwise too costly or time consuming.
3) It makes possible the automatic control of many processes.

The following drawing suggests a few of the possible applications of high-speed digital computers.

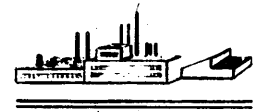
INFORMATION-PROCESSING SYSTEMS
using
DIGITAL COMPUTERS
will have many
APPLICATIONS

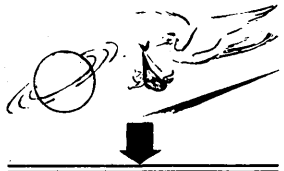

gun control


inventory

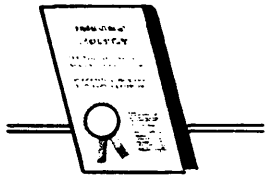
$$\frac{d^2\theta}{dt^2}+a\frac{d\theta}{dt}+b\theta=f(t)$$
mathematics

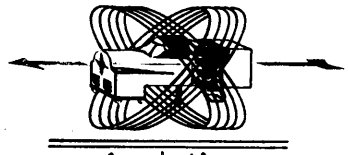

air traffic control

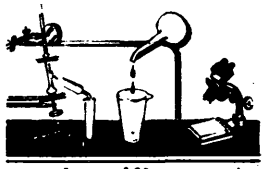

industrial process control


census


economic analysis


insurance handling


simulation


scientific and engineering computations

As with all tools, the effective usage of WWI requires a full appreciation of what it is designed to do. Basically it can in single operations:

1) Store and transfer information within itself.
2) Perform the arithmetic operations of addition, subtraction, multiplication, and division.
3) Distinguish between positive and negative numbers and take one of two alternate courses in a program on the basis of the distinction.
4) Send electrical pulses to various external devices and control their action accordingly.

It can automatically perform these single operations in sequence. Sequences of these simple operations may be combined by the programmer into very lengthy and complicated processes.

Because there is a considerable similarity between operations in such machines as WWI and those in a simple nervous system, and because these operations are performed so rapidly, it has become popular to think of such machines as giant brains. This analogy is not very helpful to one who is preparing a program for the computer. His job is to specify every operation in the process which the computer is to carry out. This means that he must first completely analyze the process, using general terms instead of specific values. Then he must prepare a program in coded form (which describes the procedure step by step), to which the computer is built to respond. When the computer is given this program it proceeds to carry

it out with specific numerical values, also supplied either by the programmer or by external devices.

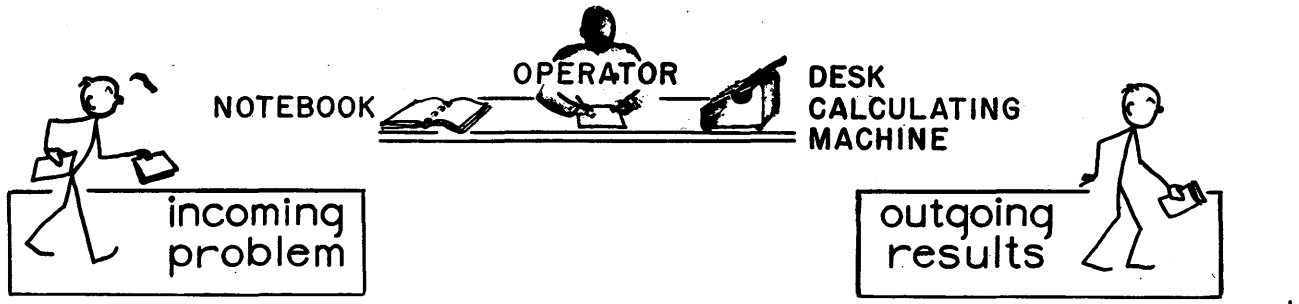
This relationship of programmer to WWI deserves graphic illustration. While studying the following drawing, imagine the operator of the desk calculator as the programmer for Whirlwind I. Imagine him to the left of the heavy arrow labelled "problem to be done". In preparing the program of the problem, he must go through the very same analysis which he goes through for the same problem when using the desk calculator.

All calculations handled by WWI can also be handled on a desk calculator. WWI's usefulness lies in its speed in carrying out processes involving computation. This makes the performance of many processes economically justifiable for the first time; further it makes possible automatic control of certain processes which, because of the speed required, could be handled in no other way.

Of course program preparation time must be included in any estimate of WWI's cost in solving a particular problem. For this reason many problems which can be solved manually in a reasonable time, and which need not be solved often, may not be suitable for solution on a large computer. On the other hand, many very simple routine tasks can be economically handled by WWI because they can be programmed once and for all, thus eliminating the need for separate analysis each time the task is performed.

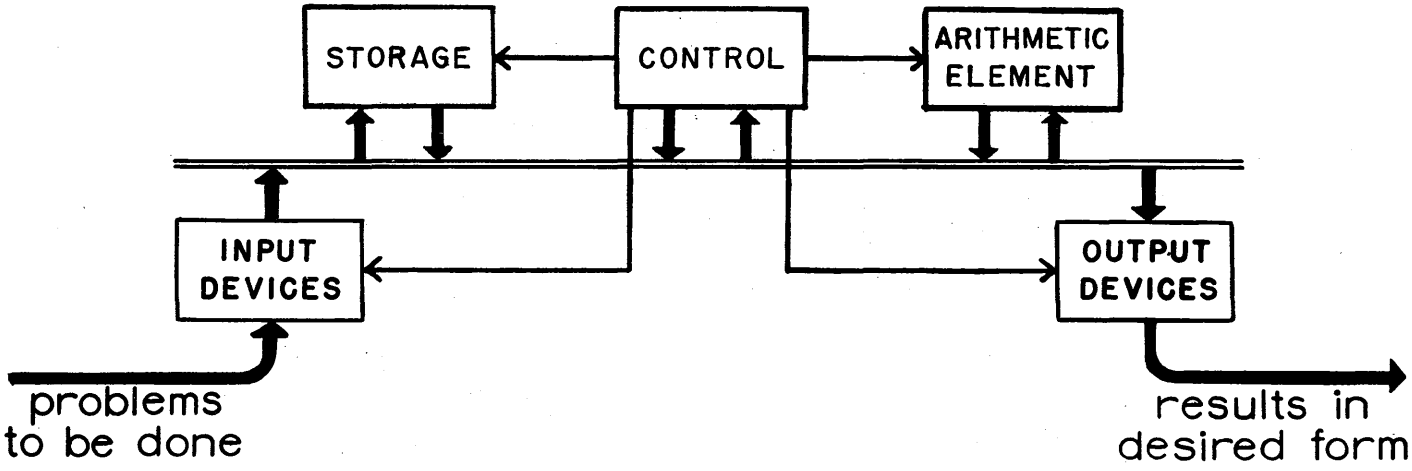
BASIC COMPUTER ELEMENTS

comparison between manual computation



and

WHIRLWIND I computation



PART I

ESSENTIALS OF PROGRAMMING

FOR NWI

A. WHAT A PROGRAM IS

A program is a sequence of instructions and numerical values, in coded form. It instructs the computer to perform a particular process, one operation at a time.

Before a program is performed, it is stored, in appropriate form, in the storage unit. It is then executed by the combined operation of the control and arithmetic units.

WWI can distinguish between and perform 32 basic kinds of operations. These operations are described in "The Order Code" which is included in the Appendix (D).

B. ELEMENTS OF PROGRAMS

The instructions and numbers in a program are both called words. Words are stored in separate registers of the storage unit. Each register is numbered, and the number designating a register is called its address.

A programmed number specifies the sign and magnitude of the number.

An instruction is a combination of one of the 32 operations plus the address of the storage register which contains the word to be operated on.

As an example of the exact meaning of an instruction, suppose that at some time in a program it is necessary to subtract + 1/2. The programmer would include two words in the program for this purpose. One would be the number + 1/2. The other would be the instruction to subtract the contents of the register which contains the word + 1/2. Thus, if + 1/2 were stored in the register whose address is 249 (or simply register # 249), the programmed instruction would be su 249, where su is the coded form of subtract.

Specifically the instruction su 249 means, subtract the contents of register # 249. Thus instructions may be likened to algebraic notation where the contents of a register may be changed just as may the values assigned to algebraic symbols.

People have a natural but incorrect tendency when first beginning to program to write an instruction as an order plus the word to be operated on rather than as an order plus the address of this word; e.g., the correct form of the instruction above is su 249, not su + 1/2.

C. FORMS OF PROGRAMS AND WORDS

Programs are originally written in standard form, that is, with the alphabetical abbreviations of the order code and ordinary decimal numbers, as in the preceding example.

Before WWI can respond to the program, the standard form must somehow be converted to binary form. The task of conversion is assigned to the computer, instructed by a conversion program which has been written once and for all.

The programmer normally does all programming in standard form, and need not concern himself with the details of conversion. However, he will more fully understand programming if he is familiar with the form in which numbers and instructions are represented in WWI.

A WWI storage register consists of 16 binary digit positions: each digit position may contain a 1 or a 0; see the drawing at the end of this section. When a word within a register represents an instruction, the first 5 digit positions are taken up by the binary coded representation of the operation and the last 11 by the address of the word to be operated on.

When a word represents a number, the first digit position, called the sign digit, indicates whether the number is positive (+) or negative (-). The last 15 indicate its magnitude. For a + number the sign digit is 0. For a - number the sign digit is 1, and the - number's magnitude is represented by changing all 0's and 1's of the positive magnitude of that number to 1's and 0's respectively.

Thus, $+\frac{3}{8}$ is represented in a register as:

and the negative number $-\frac{3}{8}$ as:

This representation is more fully explained in Appendix A.

The negative form of the number is called the complement of the positive form, and vice versa.

Thus at any one time a register contains a word which may be interpreted both as a binary number and as a coded instruction. Whenever a word is sent to the control element, it becomes an instruction, while whenever it is sent to the arithmetic element it is treated as a number. This will become clearer when examples are considered later on. The fact that any register may contain either an instruction or a number interchangeably makes for a more flexible machine.

WWI is so designed that arithmetic operations are normally handled as if the binary point of a number (corresponding to the decimal point in the decimal number system) is considered to be fixed at the left of the 15 digits which represent its magnitude. Thus only numbers of magnitude less than 1 are carried in a register. In other words, a register carries numbers in the range between -1 and +1.

Of course WWI must be able to handle problems involving numbers greater than 1. A number equal to or greater than 1 in magnitude is pre-multiplied by some other number, normally called a scale factor, (usually a power of 2) so chosen that the result is less than 1. Both the scale-factored number and the scale factor itself are then needed to represent the original number. The scale factor may be assigned permanently and remembered by the programmer, or it may be stored in a separate register (as the exponent of the power of 2). The program may be written to deal properly with the scale factor whenever it deals with the scale-factored number.

Effective scale-factoring of numbers in the original program and of the numbers arising during its execution is one of the programmer's more tedious tasks. If the magnitude of a number resulting from one of the arithmetic operations equals or exceeds 1 (called overflow), the computer automatically stops, gives an alarm, and indicates the point in the program where the overflow occurred.

The binary numbers representing addresses in instructions are considered to be positive integers with the binary point at the right-hand end of the word.

When numbers are represented in digital form, their accuracy is limited by the number of digits used in the representation. The accuracy of 15 binary digits (one part in 2^{15} or 32, 768) is equivalent to that of about 4.5 decimal digits. Greater accuracy than this may be obtained by using more than one register to contain a number.

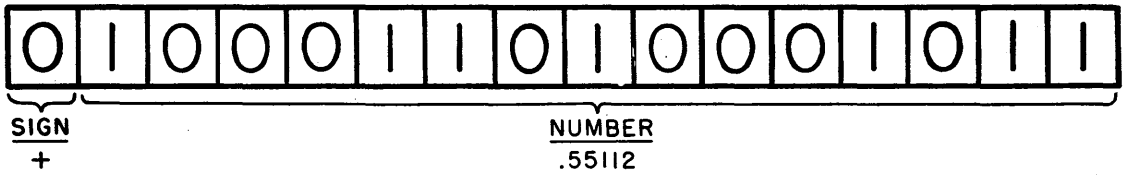
An explanation of binary numbers sufficient for general programming purposes is included in the Appendix.

WHIRLWIND I

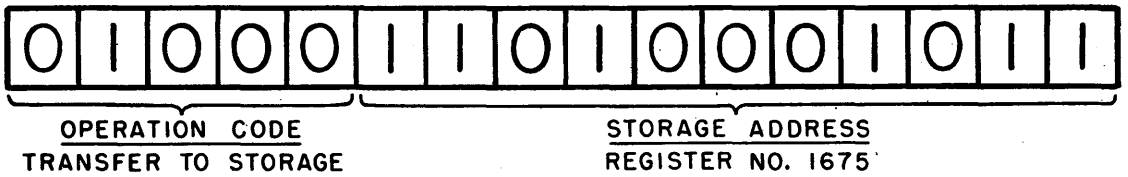
WORDS

are held in storage registers
and represent either
NUMBERS or *INSTRUCTIONS*

number



instruction



CONTROL DETERMINES WHETHER A GIVEN WORD REPRESENTS A NUMBER OR INSTRUCTION

D. BASIC COMPUTER ELEMENTS

The four basic elements of the Whirlwind computer are:

1. Input-output, to introduce programs and data into the computer proper and to extract results of calculations.
2. Storage, for a program and data. (Storage is now primarily composed of 304 registers; this number is being increased to a few thousand in the near future.)
3. Arithmetic element, which adds, subtracts, multiplies, divides when so instructed.
4. Control, which coordinates the over-all performance of WWI.

These elements are inter-connected by a main bus or communication link. The bus provides for the interchange of information necessary for WWI's automatic operation.

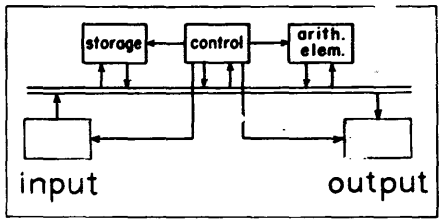
Whenever WWI is operating, the control element is generating a continuous and rapid succession of electrical pulses. These pulses are fed to the input of an electronic distribution system through which they are sent to the circuits required to carry out each operation as called for by the program.

The next four drawings are illustrations of the basic computer elements.

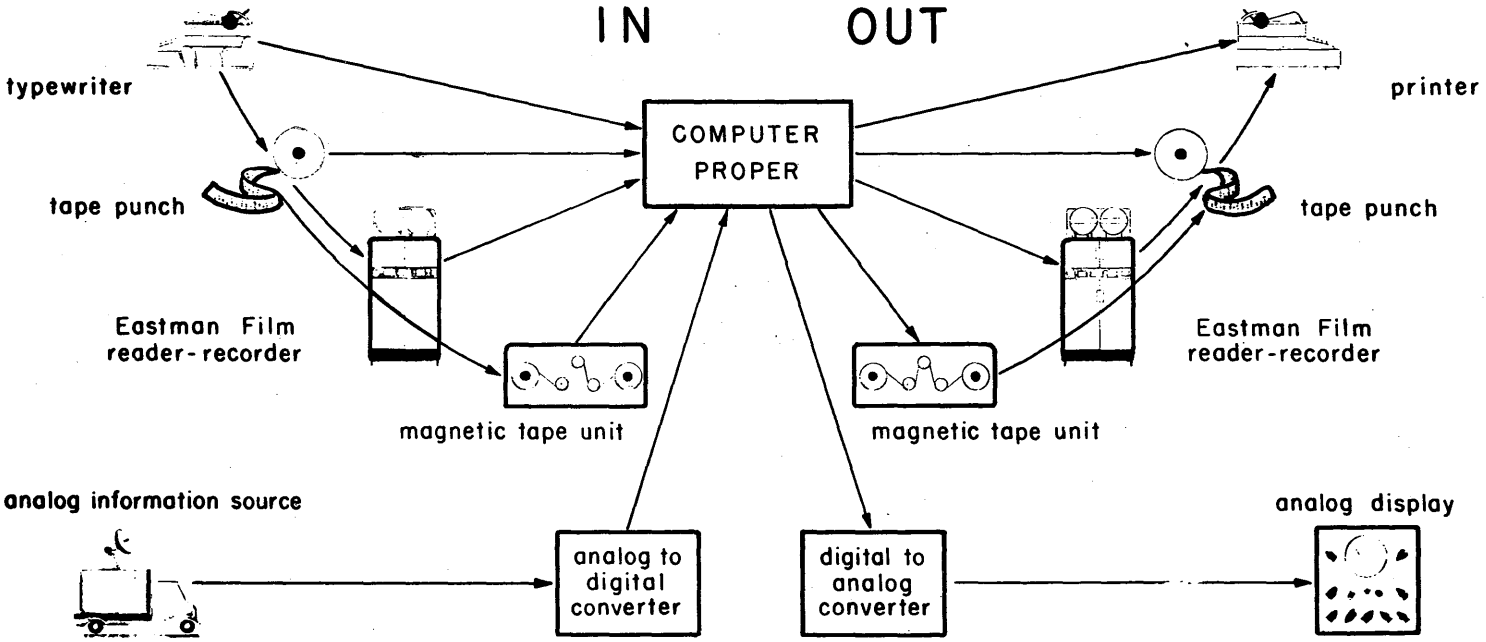
basic computer elements

INPUT AND OUTPUT

alternative ways of getting information into and out of WHIRLWIND I



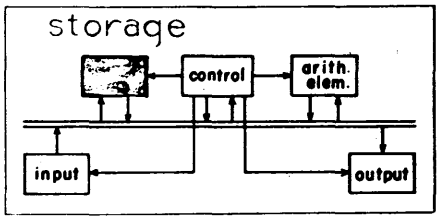
SELECTION OF DEVICE USED IS DONE BY CONTROL

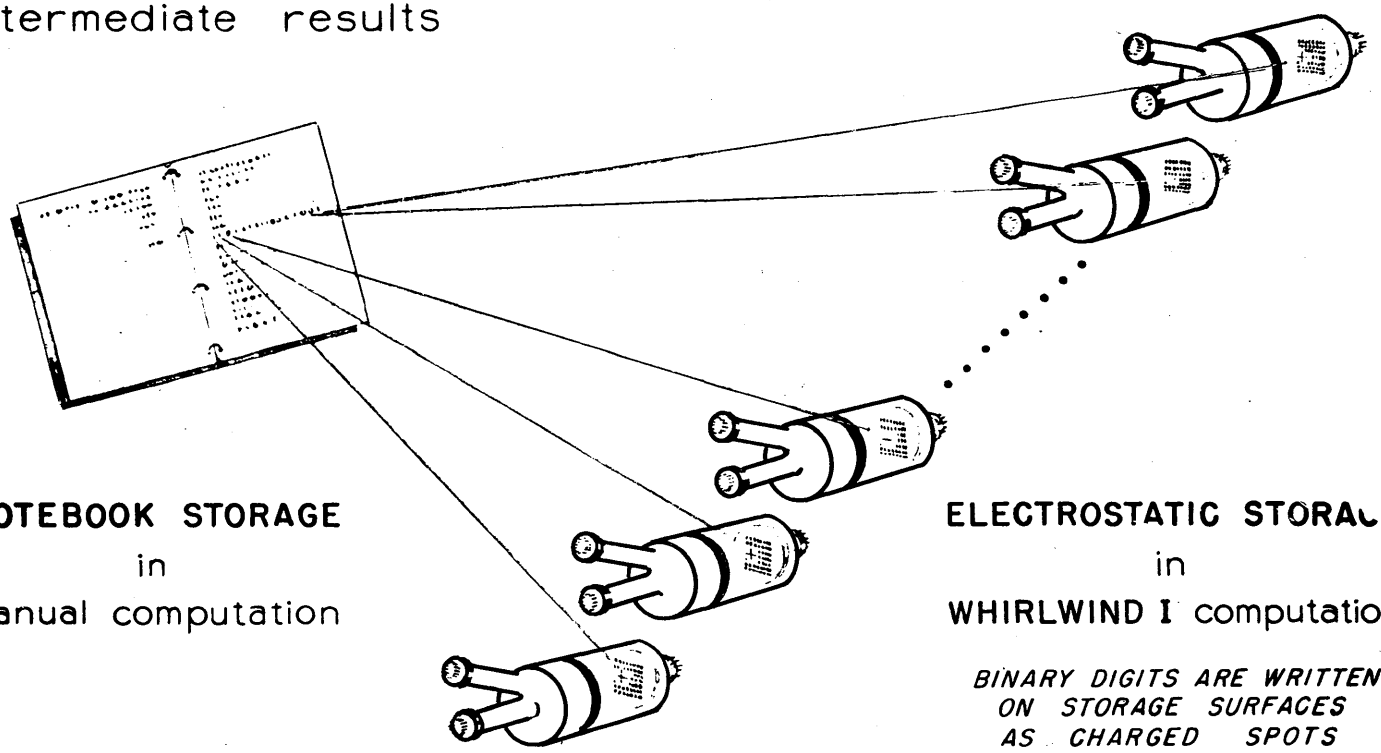


basic computer elements

STORAGE

the memory for initial data, instructions, and intermediate results





NOTEBOOK STORAGE
in
manual computation

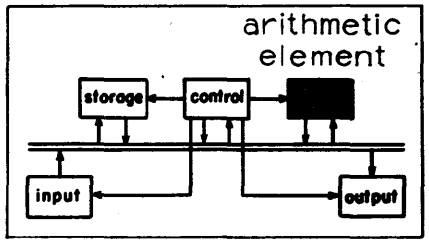
ELECTROSTATIC STORAGE
in
WHIRLWIND I computation

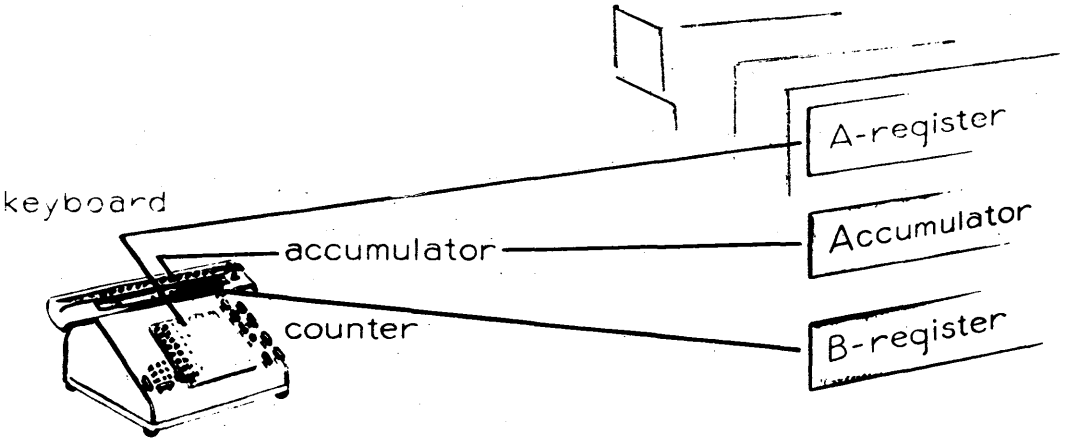
BINARY DIGITS ARE WRITTEN ON STORAGE SURFACES AS CHARGED SPOTS

basic computer elements

ARITHMETIC ELEMENT

of WHIRLWIND I compared
with a desk calculator





Receives number from the main bus-Holds addend, subtrahend, multiplicand or divisor

The adding unit-Holds the result of an addition, multiplication or subtraction and the dividend in division

Auxiliary register-Holds the multiplier or quotient

DESK CALCULATOR

ARITHMETIC ELEMENT

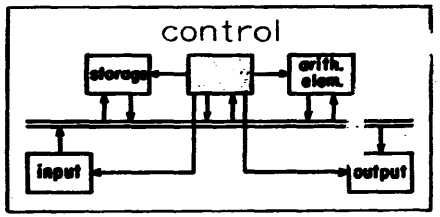
PERFORMS ACTUAL ARITHMETIC OPERATIONS SUCH AS —

add • subtract • multiply • divide

basic computer elements

CONTROL

Just as in manual computation the operator controls the steps to be performed - so in WHIRLWIND I computation



CONTROL

takes next instruction in sequence from storage - examines it and sends pulses at the proper times to the various parts of the computer to perform the necessary *PROCESSES*

instructs **STORAGE** to

SELECT A STORAGE REGISTER AND READ IN OR READ OUT EITHER NUMBERS OR INSTRUCTIONS

instructs **INPUT** to

SELECT EXTERNAL DEVICE START — READ — STOP

instructs **OUTPUT** to

SELECT EXTERNAL DEVICE START — RECORD — STOP

instructs **ARITHMETIC ELEMENT** to

CLEAR REGISTERS · ADD · SUBTRACT · MULTIPLY DIVIDE · COMPARE

E. WHAT WWI DOES WITH PROGRAMS

WWI normally has an input program stored in its first 32 registers. This input program reads the program to be performed into any other desired registers of the storage unit. At the present time, programs are read into WWI from punched paper tape. In addition, magnetic tape input and photographic film input are being developed.

After the program has been read into and stored in WWI storage, it is ready to be performed. Upon a start signal, the various instructions in the program are performed in the sequence in which the program lists them.

External data needed during performance of a program can be fed to storage from various types of equipment such as radar sets, teletype lines, pressure guages, etc.

The results of a program can be read out of WWI in various forms. At present they can be typed, punched on paper tape, and displayed on oscilloscopes.

The Order Code in Appendix E gives the functions of the various orders. The orders in the first group send controlling pulses to input-output equipment. Those in the second group provide for the transfer of information within the computer. The two orders in the third group modify the sequential operation of control to permit transfer of operations from one point in a program to any other point. All the orders in the last two groups are known as the arithmetic orders.

The following abbreviations for three special-purpose arithmetic registers are used in the explanations given with the order code. Each of these

three registers has a 16 digit word capacity, just as do those in storage.

AC = accumulator. The adding and shifting register in which sums and products are accumulated and retained.

BR = B-register. The shifting register extending to the right of the accumulator, used in the formation of products and quotients.

AR = A-register. Primarily a buffer storage for words to be added or subtracted into AC. There is only one order, ta, with which it is necessary to consider the contents of the A-register.

WWI was originally designed as an experimental model. As it has been developing, more permanent plans have been made for it. Nevertheless, it is still in an experimental stage. This is particularly true of the input-output equipment and techniques for its use. For this reason the temporary orders listed in Appendix E, are being used at present, instead of the first block of input-output orders listed in the Short Guide to Coding. The programs for particular problems will not be changed substantially, if at all, by these temporary conditions.

F. SUBROUTINES

Every time a particular job is programmed for WWI the program becomes available to future programmers, thereby eliminating much repetition of effort.

For instance, routine programs called subroutines, have been written for computing polynomials, for computing most of the common trigonometric functions, for computing the square root of a number, for arranging a set of numbers in ascending order of magnitude, for interpolating in a table of values, and for other tasks.

If a programmer wants to have one of these tasks performed as a part of a longer program, he need merely copy the subroutine program already available for the desired task, make appropriate changes of address, and include it in the longer program.

An even more efficient way of handling subroutines is being developed for WWI. It is planned to compile a library of subroutines to be stored in a form accessible to WWI. Any program requiring the performance of one of these subroutines will include one instruction directing it to the desired subroutine. The subroutine will then be performed with the values supplied to it by the main program. Upon completion of the subroutine, control will automatically be returned to the point at which it left the main program.

Each subroutine available to WWI in this form can be considered an extension of WWI's basic order code, since the performance of the subroutine requires but one instruction in the main program.

G. FLOW DIAGRAMS

The structure of a program may be clearly charted on a flow diagram. A flow diagram consists of a series of statements about what the program does. The statements are enclosed in boxes which are connected by arrows which indicate the course of the program. The statements included in the boxes of a flow diagram may represent one, a few, or many instructions, depending on the purpose at hand.

Flow diagrams are used primarily for clarification purposes. Whereas programs are written in the language of the computer, flow diagrams need not be. They are helpful to the programmer in analyzing a problem and planning a program. They are helpful to anyone interested in what a program does.

PART II

DEVELOPMENT OF PROGRAMS FOR WWI

The following examples have been selected to illustrate how programs are developed for WWI and to suggest other applications and techniques.

For convenience of illustration, instructions for programs will be numbered as though they were to be stored consecutively in registers, starting with the address 1. Stored numbers will begin at the address 200 (an arbitrary designation). Numbers and instructions can be assigned interchangeably to any address. However, instructions must always follow each other consecutively in the order in which they are to be performed, except when a special instruction within the program itself orders a deviation from this rule.

When an order appears for the first time in the following examples the reader should refer to the Order Code in Appendix E for its exact description.

LIST OF EXAMPLES

PAGE

A. ADDITION OF A SERIES OF NUMBERS 23

1. WITHOUT SCALE FACTORING

2. WITH SCALE FACTORING

B. EVALUATION OF A POLYNOMIAL 25

1. WITHOUT SCALE FACTORING

2. WITH SCALE FACTORING

C. USE OF SUBROUTINES 28

1. TO FIND TAN Θ, GIVEN SUBROUTINES FOR SIN Θ AND COS Θ, $\Theta < 45$

D. CONDITIONAL PROGRAMMING. FLOW DIAGRAMS 31

1. USE OF CP ORDER TO DIRECT CONTROL TO THE TAN Θ PROGRAM ONLY IF Θ IS < 45

2. FLOW DIAGRAM FOR TAN Θ PROGRAM

3. GENERAL REMARKS ON USE OF CP

4. SELECTION OF LARGEST OF A SET OF NUMBERS BY DIRECT METHOD

5. + 0 AND - 0 AND EFFECT ON CP OPERATIONS

6. TWO KINDS OF COUNTERS; 0 TO n, -n TO 0

7. CYCLICAL PROGRAM (USING A COUNTER) FOR D.4. CHANGING INSTRUCTIONS DURING OPERATION OF A CYCLICAL PROGRAM

8. FLOW DIAGRAM FOR ARRANGING A SET OF NUMBERS IN ORDER OF MAGNITUDE

E. ITERATIVE PROGRAMS 44

1. TO FIND $\sqrt{x}$ BY UNIFORMLY INCREASING SUCCESSIVE TRIALS

2. $\sqrt{x}$ BY NEWTON'S METHOD

PAGE

F. GENERAL NATURE OF PROGRAMS FOR: 48

1. SIMULTANEOUS ALGEBRAIC EQUATIONS

2. INTEGRATION

3. DIFFERENTIATION

G. SIMULATION AND DISPLAY 53

1. SOLUTION OF DIFFERENTIAL EQUATIONS DESCRIBING MOTION, AND OSCILLOSCOPE DISPLAY OF THE PATH OF A BOUNDING BALL

H. USE OF TABLES IN WWI 56

1. GENERAL PROCEDURE

2. LINEAR INTERPOLATION

A. ADDITION OF A SERIES OF NUMBERS

The first group of examples will illustrate the programming of arithmetic procedures.

A.1.1. WITHOUT SCALE FACTORING

A program to add three numbers together is given here. This program adds the numbers a, b, and c, and stores the result in a register. It then sends control to the first instruction of the next job. It assumes that each of the numbers is less than 1 in magnitude, and also that at no time in the process will a number arise of value greater than 1 (i.e., that no overflow will occur).

Instruction		Effect
1.	ca 200	Clears AC; leaves a in AC
2.	ad 201	" (a+b) " "
3.	ad 202	" (a+b+c)" "
4.	ts 203	" (a+b+c)" " and in Register #203
5.	sp (address of register holding the first instruction of the next job to be done)	Send control to proper address

Data		
200.	a	
201.	b	
202.	c	
203.	---	Contains (a+b+c) after instruction in Register #4 is executed.

A.2. WITH SCALE FACTORING

Usually, the exact values of a, b, and c to be used with a program such as this are not known, but their probable range usually is known for most practical applications. Such a situation could be handled by scale factoring to accommodate the maximum possible values that could arise during the program. Scale factoring is done by multiplying a number by 1/2 enough times that the resultant (scale-factored) value is less than 1 in magnitude.

For example, if it were known that a, b, and c each lie wihhin the range -5 to +5, since $5 \times 2^{-3} = 5/8$, they could be stored in the form of $a \times 2^{-3}$. But the maximum possible value of the sum of a + b + c is 15. The largest scale-factored form of 15 which is less than 1 is 15×2^{-4}. Therefore, the appropriate adaptation of the program to handle values in this range would be:

1-5.		(Instructions would be unaltered.)
	⋮	
200.	$a \times 2^{-4}$	
201.	$b \times 2^{-4}$	
202.	$c \times 2^{-4}$	
203.	---	(Contains $(a+b+c) \times 2^{-4}$ after instruction in register #4 is executed.)

Correspondingly, if the maximum value that could arise in this program were 2000, an appropriate scale factor would be 2^{-11}, because $2000 \times 2^{-11} = \frac{2000}{2048}$ is less than 1.

Whenever scale factoring is resorted to, the scale factors must be remembered either by the programmer or in the program itself, so that final results can be reconverted to their actual values.

A simple program like the above may be encountered as a part of a larger program, where different sections of the whole, or program calculate and store a, b, and c, before execution of the instructions which perform the addition.

B. EVALUATION OF A POLYNOMIAL

B.1. WITHOUT SCALE FACTORING

The evaluation of the polynomial $ax^2 + bx + c$ for a particular value of x may be programmed as follows:

Instruction		Effect
1.	ca 202	Clears AC, a in AC
2.	mr 201	ax in AC
3.	ad 203	ax + b in AC
4.	mr 201	ax^2 + bx in AC
5.	ad 204	ax^2 + bx + c in AC
6.	ts 205	Transfer ax^2 + bx + c to register #205
7.	sp (address of next job)	
	!	
	!	
	!	
Data		
201.	x	
202.	a	
203.	b	
204.	c	
205.	---	Receives ax^2 + bx + c

This evaluation could have been accomplished by a different sequence of operations; for example, by first computing and storing ax^2, then forming bx + c, and finally adding together these two values. However, this procedure requires several additional instructions and storage registers.

B.2. WITH SCALE FACTORING

With the method just described for evaluating a polynomial, an appropriately scale-factored form of the first few terms of the Maclaurin series for sin Θ, can be used to compute sin Θ. The first three terms are:

$$\Theta - \frac{\Theta^3}{3!} + \frac{\Theta^5}{5!} \approx \sin \Theta ,$$

where the angle Θ is expressed in radians. Θ is considered to lie in the range 0 to $\frac{\pi}{2}$ radians (0 to 90°), but $\frac{\pi}{2}$ is equal to 1.57. Θ is scale-factored to $\Theta/2$, whose value is always less than 1 in the range considered. Rewriting the polynomial in Θ as one in $\Theta/2$, there results:

$$2(\Theta/2) - \frac{8(\Theta/2)^3}{3!} + \frac{32(\Theta/2)^5}{5!} \approx \sin \Theta$$

For the limiting case of $\Theta = \frac{\pi}{2}$, the value of sin Θ is equal to 1. Therefore, sin Θ must be scale-factored to $\frac{1}{2}$ sin Θ, and the final resulting polynomial is:

$$(\Theta/2) - \frac{4(\Theta/2)^3}{3!} + \frac{16(\Theta/2)^5}{5!} \approx 1/2 \sin \Theta$$

or

$$(\Theta/2) - \frac{2(\Theta/2)^3}{3} + \frac{2(\Theta/2)^5}{15} \approx 1/2 \sin \Theta$$

The corresponding program to evaluate 1/2 sin Θ is:

Instructions		Effect
1.	ca 200	$\theta/2$
2.	mr 200	$(\theta/2)^2$
3.	mr 202	$\frac{2}{15}(\theta/2)^2$
4.	su 201	$-\frac{2}{3}+\frac{2}{15}(\theta/2)^2$
5.	mr 200	$-\frac{2}{3}(\theta/2)+\frac{2}{15}(\theta/2)^3$
6.	mr 200	$-\frac{2}{3}(\theta/2)^2+\frac{2}{15}(\theta/2)^4$
7.	mr 200	$-\frac{2}{3}(\theta/2)^3+\frac{2}{15}(\theta/2)^5$
8.	ad 200	$(\theta/2)-\frac{2}{3}(\theta/2)^3+\frac{2}{15}(\theta/2)^5 = 1/2 \sin\theta$
9.	ts 203	
10.	sp next job	
	⋮	
	Data	
200.	$\theta \times 2^{-1}$	
201.	2/3	
202.	2/15	
203.	---	Receives 1/2 sin θ

The maximum accuracy lost by neglecting the fourth and higher terms of the sin θ series is about .005 at 90°. For angles less than 43° the accuracy lost is less than .00003 (or 2^{-15}), less than can be discerned in a 15-digit binary number. The inclusion of the fourth term of the series would decrease the error of this sin θ approximation to less than 2^{-15} for all angles up to 75°.

Any desired degree of additional accuracy could be obtained by using more terms of the series (making the equation inherently more accurate), and more than one register to contain each number (making it

possible to represent this greater accuracy in WWI).

The range of values computed by this program could be extended beyond 90° by the addition of a few orders to sense what quadrant the angle was in and accordingly to adjust the sign of the result.

There are several other ways of finding polynomials to approximate a function over a particular range (such as the least squares method or Lagrange's method). The particular method selected usually depends on the amount of storage required for the desired accuracy in the approximation.

Another way of finding values of functions is to have the program look up the desired value in a stored table of values, but this normally requires considerably more storage space than does direct computation. The techniques for the use of stored tables is given in section H.

C. USE OF SUBROUTINES

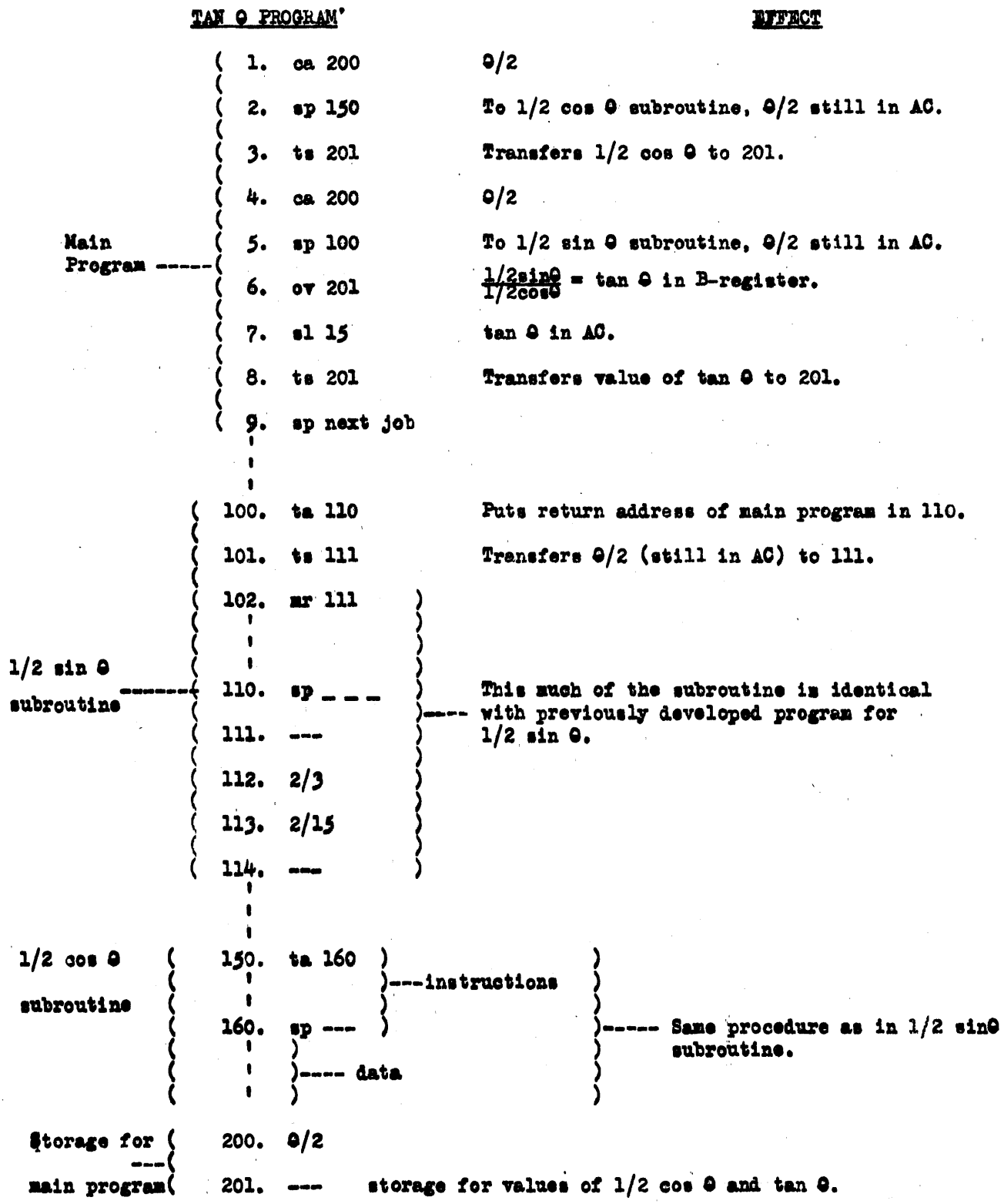
C.1. TO FIND TAN θ

If subroutines for finding 1/2 sin θ and 1/2 cos θ were already available to a main program, tan θ could be found easily by using the relationship $\tan \theta = \sin \theta / \cos \theta$.

The program given here is valid for angles up to but not including 45°; for, without additional scale factoring, overflow would occur. A method for testing whether the angle θ is actually less than 45° and for using the program given here only if θ is less than 45° is given in sections D.1-2.

Only the main program should be read at first. It indicates the overall procedure, with the sp orders directing the computer to proceed to the two subroutines and having the effect of leaving 1/2 cos θ and 1/2 sin θ in the accumulator. The mechanics of the subroutine procedure is indicated following the program.

Report R-156


TAN θ PROGRAM'
EFFECT
Main Program
1. ca 200
θ/2
2. sp 150
To 1/2 cos θ subroutine, θ/2 still in AC.
3. ts 201
Transfers 1/2 cos θ to 201.
4. ca 200
θ/2
5. sp 100
To 1/2 sin θ subroutine, θ/2 still in AC.
6. ov 201
1/2sinθ / 1/2cosθ = tan θ in B-register.
7. sl 15
tan θ in AC.
8. ts 201
Transfers value of tan θ to 201.
9. sp next job
1/2 sin θ subroutine
100. ta 110
Puts return address of main program in 110.
101. ts 111
Transfers θ/2 (still in AC) to 111.
102. mr 111
110. sp ---
111. ---
112. 2/3
113. 2/15
114. ---
This much of the subroutine is identical with previously developed program for 1/2 sin θ.
1/2 cos θ subroutine
150. ta 160
160. sp ---
instructions
data
Same procedure as in 1/2 sinθ subroutine.
Storage for main program
200. θ/2
201. ---
storage for values of 1/2 cos θ and tan θ.

The dv and sl orders are introduced in the main program. Whenever the dv (divide) order is used, the quotient of the operation is left in the B-register. dv is usually followed by sl 15 (shift everything in the B-register and in the AC, 15 digit positions to the left), which places the quotient in the AC.

The sp 150 in register #2 interrupts the sequential performance of instructions in the main program, sending control to 150. During the performance of the sp 150 in register #2, the next address 3, following the sp order in the main program, is placed in the A-register. The ts 160 order in 150 (next to be performed) takes this address 3 from the A-register and places it in the address section of the instruction 160, making it sp 3. Thus, upon completion of the 1/2 cos θ subroutine, control is returned to 3 in the main program with the value of 1/2 cos θ in the AC.

Similarly sp 100 in 5 sends control to the 1/2 sin θ subroutine; and the combined action of ts 110 in 100 and sp ... in 110 returns control to 6, with the value of 1/2 sin θ in the AC.

When appropriate subroutines are available, as was supposed in this example, the programmer's job is greatly simplified. His attention is confined primarily to the main program in which he determines the proper use of the subroutines.

A given subroutine may be used as many times as needed in a main program. Each time, an sp instruction in the main program to the ts instruction beginning the subroutine is all that is required.

D. CONDITIONAL PROGRAMMING, FLOW DIAGRAMS

The subprogramming order sp makes possible versatility in programming by interrupting the sequential performance of instructions. The conditional program order cp provides additional versatility by letting the computer decide whether or not to interrupt the sequential performance of instructions.

If the number in the AC when a cp instruction is being performed is negative, the cp operates exactly as an sp. If the number in the AC is positive, the cp is ignored and the sequential performance of instructions is continued.

Thus, WWI chooses between two courses in a program each time a cp instruction is given. It is in this sense, and only in this sense, that WWI makes decisions.

D.1. USE OF CP ORDER FOR DETERMINING IF θ IS $< 45^\circ$

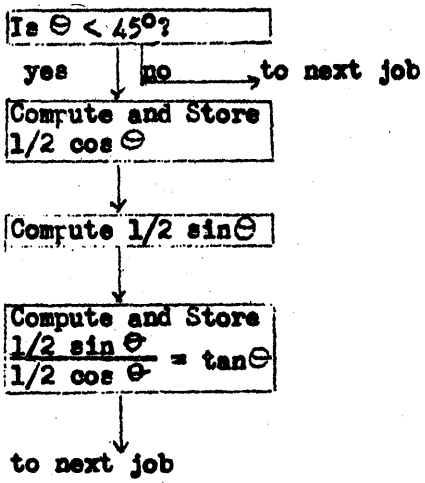
As a typical example of the use of the cp, consider the preceding program which computes tan θ for values of θ less than $\pi/4$ radians (45°). Assume that values of θ greater than 45° are also sometimes available in register 200 (in the scale-factored form $\theta/2$) before the operation of the program. The following three instructions inserted ahead of the main program would have the following effect. For θ less than $\pi/4$ the program would continue with the evaluation of tan θ. For values of θ equal to or greater than $\pi/4$, control would be sent to some other program.

	Instructions	Effect
1.	ca RC* ($\pi/4$) x 1/2	$\pi/4$ x 1/2
2.	su 200	1/2 ($\pi/4 - \theta$)
3.	cp (address of next job)	Goes to next job only if ($\pi/4 - \theta$) is negative, otherwise continues with tan θ program.

* The abbreviation RC should be read "Address of Register Containing"

D.2. FLOW DIAGRAM FOR TAN θ PROGRAM

With this addition, the tan θ program could be represented on a flow diagram as:



Each time arrows marked "yes" and "no" leave a box of a flow diagram, the use of a cp in the program is indicated.

D.3. GENERAL REMARKS ON USE OF CP ORDER

The cp order used alone chooses between two alternative courses on the basis of the sign of the number in the accumulator. The previous example used the cp to make this choice on the basis of the sign of the difference between two numbers. Thus, the cp, used with other orders, made the choice on the basis of which of two numbers was larger. With other procedures using the cp it is possible to make various selections such as the largest of a set of numbers. With the cp order checking the contents of a counter it is possible to program the cyclical repetition of a set of orders a predetermined number of times.

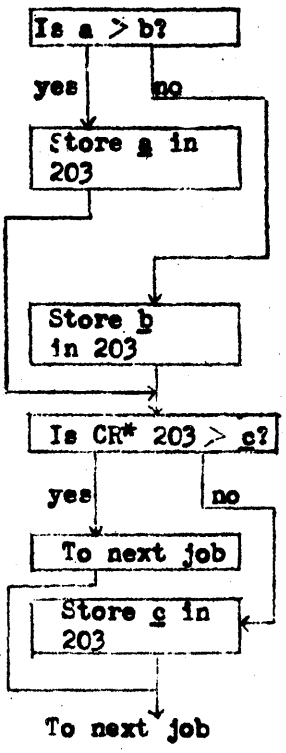
The following examples demonstrate such techniques.

The cp order may be used in a variety of other ways to distinguish between different situations or things which have numerical values assigned to them. Such things as problems of logic and various economic games may be developed by having the cp distinguish between the signs associated with true and false statements and different conditions.

D.4. SELECTION OF THE LARGEST OF A SET OF NUMBERS - DIRECT METHOD

This program finds the largest of three numbers, a, b, and c and stores it in register 203.

Instruction		Effect if the largest number is: a	b	c	i-j
1.	ca 200	a	a		
2.	su 201	(a-b)	(a-b)		1-3
3.	cp 7	pos.	neg.	pos. or neg.	
4.	ca 200	a			4-6
5.	ts 203	a			
6.	sp 9				
7.	ca 201		b		7-8
8.	ts 203		b		
9.	su 202	(a-c)	(b-c)	(a-c) or (b-c)	9-10
10.	cp 12	pos.	pos.	neg.	
11.	sp next job				11
12.	ca 202			c	12-13
13.	ts 203			c	
14.	sp next job				



*The abbreviation CR should be read "contents of register _ _ _."

Data

200. a

201. b

202. c

203. _ _ _ Containe a, b, or c after instruction 14

This program could be extended to find the largest of any number of values by the addition of a series of comparison orders for each additional value. The necessary repetition of identical orders (with different address sections) in such a program is avoided in programs using counters--to be described in sections D.6.-7.

D.5. + 0 AND - 0 AND EFFECT ON CP OPERATIONS

The number 0 is represented in WWI as either + 0 or as - 0. Normally a 0 in a program refers to + 0. However, when 0 is the result of an addition or a subtraction order, the result is left in the accumulator in the form - 0. The cp order is actuated by - 0 as by any other negative number. This point must be considered whenever the cp order is used in checking relative magnitudes of numbers, as in the preceding example, and in counters, used in examples to follow.

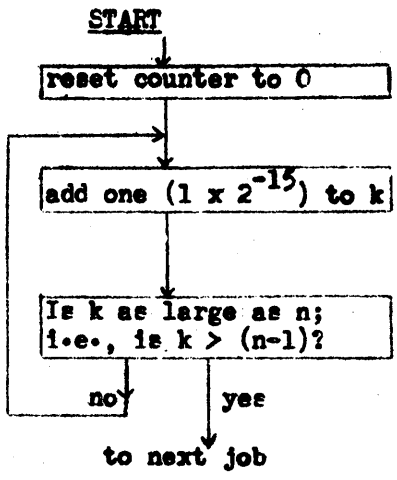
The exceptions to this rule are that (+ 0) plus (+ 0) leaves (+ 0) in the accumulator, and (+ 0) minus (- 0) also leaves (+ 0) in the accumulator.

In the preceding example, if a and b were equal in value and 1 larger than c, the path followed in the program would be the one indicated for b. This is because a - b would equal - 0, which would actuate the cp instruction in register #3.

D.6. COUNTERS: 0 TO n, AND -n to 0

M.I can be programmed to count up to a given number n. These two programs are called counters. It is necessary to store (n-1) in register #201 to make the cycle be performed n times, because of -0.

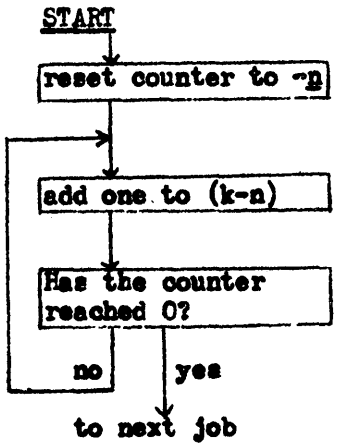
0 to n counter		in AC on k'th cycle	
1.	ca 200	0	1-2
2.	ts 202		
3.	ao 202	k	3
4.	su 201	k - (n-1)	
5.	cp 3		4-5
6.	sp next job		
!			
!			
200.	0		
201.	(n-1)(x 2^{-15})		
202.	$k(2^{-15})$, counter		



k = number of times the cycle has been repeated

Essentially this program counts from 0 to n. The following program counts from -n to 0.

-n to 0 counter		in AC on k'th cycle	
1.	cs 200	-(n-1)	1-2
2.	ts 201		
3.	ao 201	-(n-1) + k	3
4.	cp 3	neg. until k = n	
5.	sp next job		4
	⋮		
200.	$(n-1)(2^{-15})$		
201.	$-(n-1) + k\ (2^{-15})$, counter		

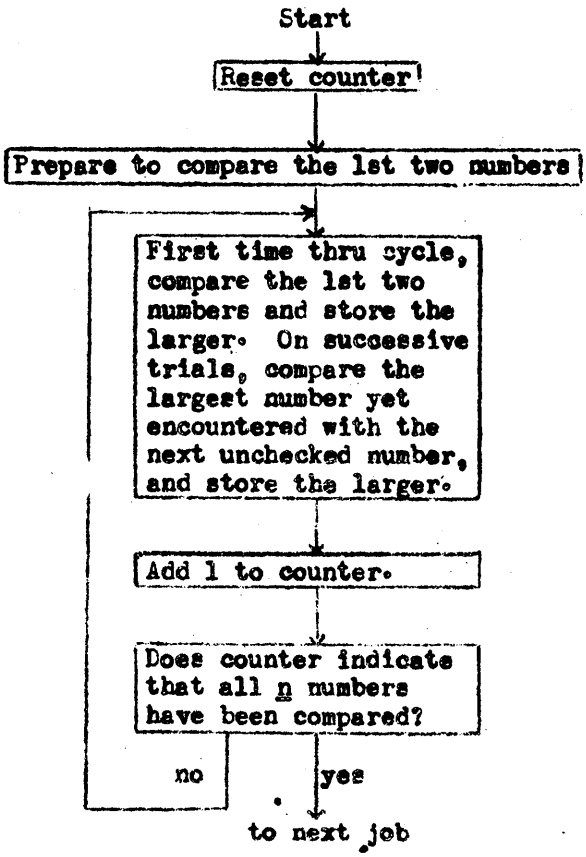


The second of these two counters requires two fewer registers than the first. Therefore it would be used in preference to the first except when it may be desirable to keep track of the value of k specifically.

These programs can be adapted to repeat a certain process n times by having the counter cycle include the desired process.

D.7. CYCLICAL PROGRAM FOR FINDING LARGEST OF A SET OF NUMBERS; CHANGING INSTRUCTIONS DURING OPERATION OF A CYCLICAL PROGRAM

This program uses a counter in finding the largest of n numbers stored in WWI and shows how WWI can change its own instructions. The counter provides for the repetition of a series of comparison orders until all n numbers have been checked. The general form of the program may be planned on a flow diagram.



This procedure suggests itself because it is similar to that followed by a person selecting the largest of a series of numbers.

The td order is introduced here. Both the td order and the ao order are used to change the address section of an instruction so that the same order may be used to operate on the contents of different registers on different runs through the cycle.

The possibility of programming WWI to change its own instructions is, as is the use of the cp, one of its most important features.

Cyclical Program For Finding Largest of n Numbers

Instructions		Effect
*1.	cs 200	Reset counter
*2.	ts 201	
3.	ca 204	x_1
4.	ts 202	Store x_1 as the largest number yet found.
5.	ca 203	Address of x_2
6.	td 8	Transfer address of x_2 into the ca order in reg. 8
7.	td 11	" " " " " " " " " " 11
8.	ca _ _ _	Comparison orders
9.	su 202	
10.	cp 13	
11.	ca _ _ _	Store newly found maximum
12.	ts 202	
13.	ao 8	Prepare to compare next number
14.	ao 11	
*15.	ao 201	Increase counter by 1
*16.	cp 8	Have all n numbers been checked
*17.	sp next job _ _ _	
	⋮	
	⋮	
	⋮	
Data		
*200.	$(n-2) \times 2^{-15}$	
*201.	counter _ _ _	
202.	x max. _ _ _	Storage for largest number yet encountered
203.	ri 205	Address of x_2. The binary coded form of the order ri is 0C000.
204.	x_1	n numbers, x_1 through x_n
205.	x_2	
	⋮	
	⋮	
	⋮	
203 + n.	x_n	

*Registers used for the counter in this program.

Instructions 1, 2, 15, 16, and 17 are the same as the instructions

of the -n to 0 counter program given earlier. The counter reset instructions must be performed outside the main cycle, while the ao, cp, and sp instructions must be performed within the main cycle if they are to control the number of times the main cycle is performed.

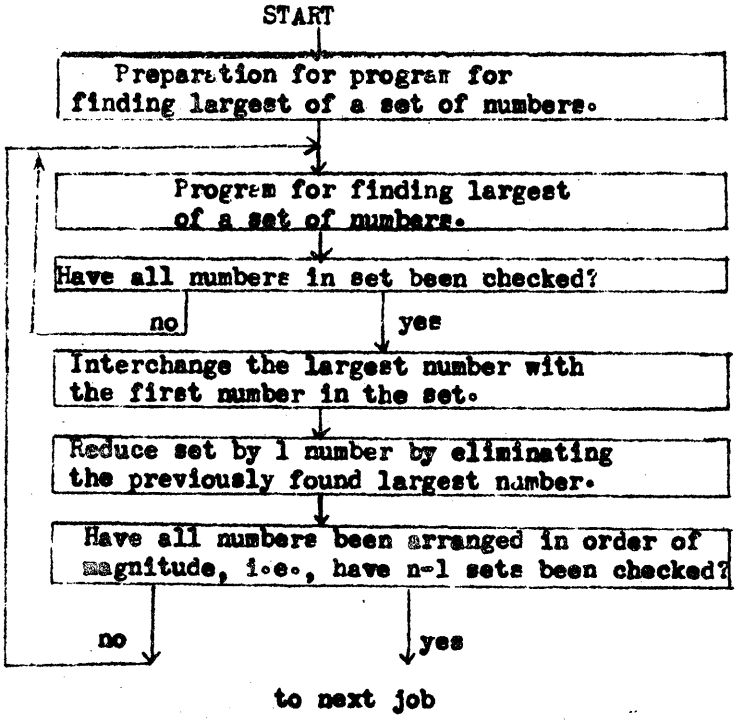
Whenever a counter is used it is important to consider the exact number of times the controlled cycle is to be performed. To compare n numbers as in this program, the comparison must be performed $(n-1)$ times. This requires that the counter here be reset to $-(n-2)$. If the counter instructions within the main cycle had preceded, instead of followed, the comparison instructions, the counter would have been reset to $-(n-1)$ for $(n-1)$ performances of the comparison instructions.

D.8. FLOW DIAGRAM FOR ARRANGING A SET OF NUMBERS IN ORDER OF MAGNITUDE

The preceding program may be extended to rearrange a set of n numbers in descending order. One way of doing this is to find the largest number in the set and exchange it with the first number in the set. Next find the largest of the set of n-1 remaining numbers and exchange it with the first number in the new set. Continue the process until all numbers have been arranged.

A program following such a procedure would consist of two main cycles, one within another. The inner cycle would be the same as the main cycle of the previous program for finding the largest number in a set. The outer cycle would in addition arrange for the interchange of numbers and the elimination of numbers already arranged in order of magnitude from the next set to be checked.

A flow diagram for this process may look as follows:



The interchanging process requires ca and td and ts orders. The ca takes the instruction which contains the address of the register of the largest number, the td transfers this address to a ts _ _ _ instruction. This ts follows a ca RC (first number in the set). Thus the first number in the set is transferred to the register that originally contained the largest number in the set. Similarly the largest number is transferred to the register which originally contained the first number.

The following sets of instructions added to the previous program would provide for this interchange:

First number to original register of largest number		Largest number to original register of first number	
n.	ca RC address of largest number	n^1.	ca 3
n + 1.	td n + 3	n^1 + 1.	td n^1 + 3
n + 2.	ca 204	n^1 + 2.	ca 202
n + 3.	ts _ _ _ _	n^1 + 3.	ts _ _ _ _

To eliminate the first number from successive sets after the interchange, it is necessary to give an ao order to instruction 3, and to 203, which set up the first register of the new set to be checked.

The addition of one extra counter will check to see if all n - 1 sets have been checked.

ITERATIVE PROGRAMS

It is frequently necessary or convenient to solve numerical problems by iterative methods--that is, by methods which make successive trial attempts at a solution until a satisfactory solution is reached. WWI's rapid operation makes such methods practical.

L.1. TO FIND $\sqrt{x}$ BY LINEARLY INCREASING SUCCESSIVE TRIALS

A simple iterative program for finding the square root of a positive number x less than 1 is given here. Because as written this procedure requires several thousand trials to find the square root which can be evaluated readily in other ways, it is of little practical importance, but serves as an illustration of what can be done by iterative methods. Let $\sqrt{x'}$ indicate successive trial values of $\sqrt{x}$.

Instructions		Effect	
1.	ca 202		
2.	ts 200		
3.	ao 200	$\sqrt{x'}$	
4.	mr 200	x'	
5.	su 201	$x' - x$	negative until x' becomes larger than x
6.	cp 3		
7.	sp next job		

Data

200. $\sqrt{x'}$ initially 0, finally $\sqrt{x}$ to within 2^{-15}

201. x

202. 0

The ao 200 increases $\sqrt{x'}$ by 2^{-15} on each successive cycle until x' just exceeds x. The first value of $\sqrt{x'}$ tried is 2^{-15}, and x' here is 2^{-30}, which would appear as + 0 in the accumulator. In fact, x' would appear as + 0 in the accumulator until $\sqrt{x'} = 2^{-7.5}$ was tried, giving

$x' = 2^{-15}$. Thus, if the above program were used, $2^{-7.5}$, instead of 0, would be stored in register #202 in order to conserve computer time.

This program may be used for values of x greater than 1 by scale factoring x by 2^{-n}. This would leave $x(2)^{-n/2}$ in register #200.

If 2^{-15} were smaller than the accuracy required for a particular problem, the ao 200 could be replaced by the three orders ca 20C, ad RC-(desired accuracy), and ts 200. Alternatively, an sl n and a ts 200 following the ao 200 would give a difference of $2^{-15} \times 2^{n}$ between successive values of $\sqrt{x'}$.

With iterative programs it is important to consider the amount of computer operating time required to execute the program. An average operating time of 10^{-4} seconds should be allowed for the execution of each instruction. This is equivalent to the performance of 10,000 instructions per second.

The $\sqrt{x}$ program given here would require up to about 12 seconds for values of x near 1. This estimate is made on the basis of about 2^{15} or about 32,000 iterations of the 4 instructions # 3-6. If between successive trials steps larger than 2^{-15} were used, this time would be cut down proportionally, at the expense of accuracy.

The next program for finding the $\sqrt{x}$ by Newton's iterative method converges on the solution very rapidly. On the other hand it requires about twice as much storage space. In many problems where alternative methods of solution are available the choice will depend largely on operating time versus storage space.

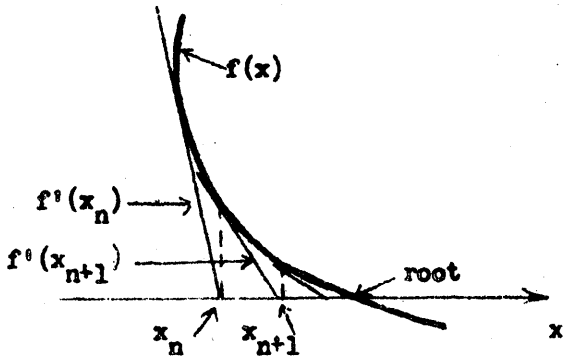
These programs of iterative processes are given as examples of the general procedure used with such processes. The addition of (n-2) instructions of the form mr RC $\sqrt[n]{x'}$ to the program just given would give a program to find the nth root of x. Similarly, Newton's method, used to find a square root in the following program, may be developed to find a root of any nth degree f(x).

E.2. $\sqrt{x}$ BY NEWTON'S METHOD

Newton's method for finding a root of an equation $f(x) = 0$ uses the formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

where x_n is the nth approximation to the root, and $f'(x_n)$ is the slope of the curve at x_n. Successive values of x_n are found at the intersection of the slope $f'(x_n)$ with the x axis.



The sketch indicates that successive approximations converge rapidly on the value of the root. Therefore a high degree of accuracy can be obtained with a few iterations.

To find the square root of a number a let $f(x) = x^2 - a = 0$. The formula for successive approximations to the positive root of this equation becomes:

$$x_{n+1} = x_n + 1/2 \left(\frac{a}{x_n} - x_n\right).$$

This program assumes that $2^{-14} < a < 1 - 2^{-14}$. Starting with an initial $x_1 = 1 - 2^{-15}$, it finds successive values of x_n until $|x_n - x_{n+1}| \leq 2^{-14}$. The last value found for x_n will be an approximation to $\sqrt{a}$, accurate to within 2^{-14}.

Thus the program must have stored initially three values--the number a, the first approximation $x_1 = 1 - 2^{-15}$, and the desired accuracy 2^{-14}. There should be reserved a register to contain x_n. Another register is necessary to contain the partial result $1/2(\frac{a}{x_n} - x_n)$.

NEWTON'S METHOD FOR FINDING $\sqrt{x}$

Instruction		Effect
1.	ca 201	
2.	ts 203	
3.	ca 200	a
4.	dv 203	a/x_n in B-register
5.	sl 15	a/x_n in AC
6.	su 203	$a/x_n - x_n$
7.	sr 1	$1/2(a/x_n - x_n) = x_{n+1} - x_n$
8.	ts 204	
9.	ad 203	x_{n+1}
10.	ts 203	
11.	cm 204	$\lvert x_n - x_{n+1} \rvert$
12.	su 202	
13.	cp next job	
14.	sp 3	Return for calculation of next x_n
⋮		
⋮		
⋮		
Data		
200.	a	
201.	$x_1 = 1 - 2^{-15}$	
202.	2^{-14}	
203.	_ _ _,	Used for x_n, finally $\sqrt{a}$
204.	_ _ _,	Used for $1/2(\frac{a}{x_n} - x_n)$

F. GENERAL NATURE OF PROGRAMS FOR SOLVING SIMULATANEOUS EQUATIONS AND FOR INTEGRATION AND DIFFERENTIATION

F.1. SIMULATANEOUS ALGEBRAIC EQUATIONS

Simultaneous algebraic equations may be solved on WWI by a variety of methods. The investigation of such methods is a problem in numerical analysis, and no attempt will be made here to consider them since they represent essentially a mathematical analysis problem rather than a programming problem. Once an appropriate method is selected for a problem, the programming consists of scale factoring to avoid overflow and expressing the process by a sequence of instructions. However, the choice of method must take into consideration such things as whether the method will always converge on the solution, the accumulation of round-off error, required storage space, and operating time.

Simply to suggest the nature of the problem, consider the set of simultaneous linear equations:

$$a_1x + b_1y = c_1$$

$$a_2x + b_2y = c_2$$

Several methods of solution suggest themselves. A direct substitution method would be to solve the first equation for $y = f(x, a_1, b_1, c_1)$. Substitute this expression for y in the second equation and solve it for $x = f(a_1, b_1, c_1, a_2, b_2, c_2)$. Find a similar expression for y. Then the program would consist simply of the direct evaluation of these expressions for x and y in terms of a, b, c. This method resolves itself into the same process as the evaluation of x and y by the method of determinants or even by more general matrix manipulation methods. In some problems these latter methods may be appropriate for WWI.

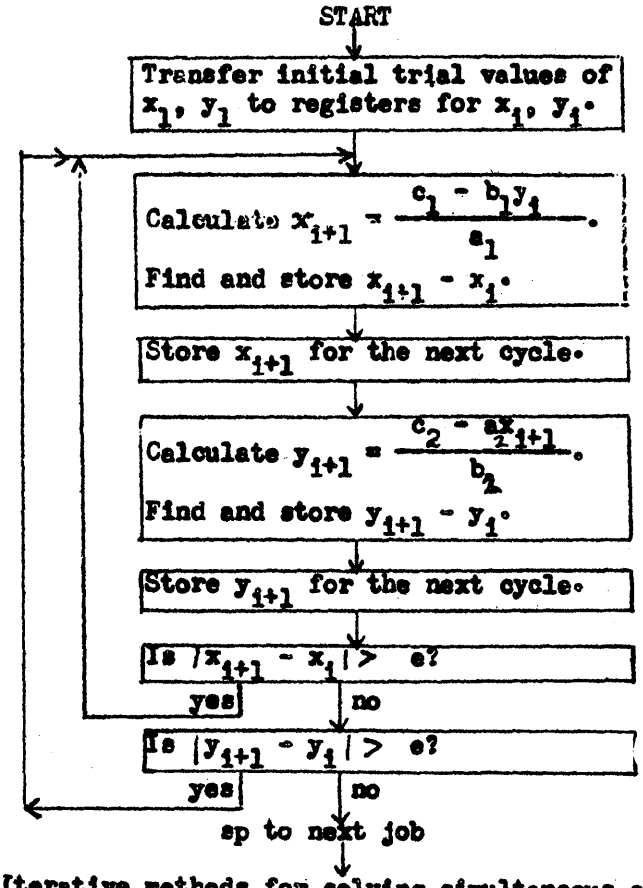
A less direct method of solution is the iterative one using the relations;

$$x_i = \frac{c_1 - b_1y_i}{a_1}$$

$$y_i = \frac{c_2 - a_1x_i}{b_2}$$

where the subscript i refers to the ith successive trial for the solution of x and y. Initial trial values x_1 and y_1 are assigned to x_i and y_i, from which are found $x_2, y_2, x_3 \cdots x_n, y_n$ successively. The iterative

procedure is stopped when both of the differences $|x_{i+1} - x_i|$ and $|y_{i+1} - y_i|$ become less than the maximum allowable error, e. This may be done by applying the cp to $|x_{i+1} - x_i| - e$, and to $|y_{i+1} - y_i| - e$.

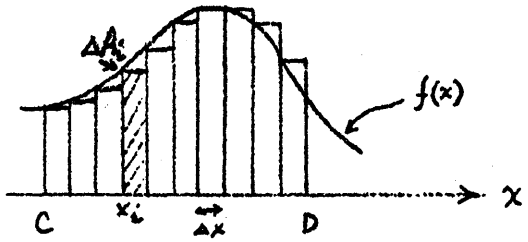


Iterative methods for solving simultaneous equations require longer computer operating time, but may be easier to set up and program, and in some cases provide the only practical means for solving large systems without undue round-off error.

F.2. INTEGRATION

Integration and differentiation may both be approximated in WWI by replacing these continuous processes by discrete discontinuous numerical procedures--the only kind that purely digital devices can handle.

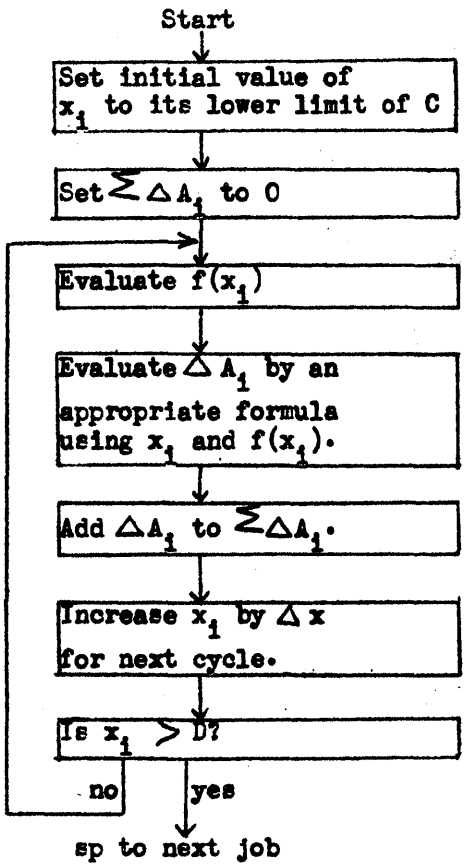
Integration may be approximated in WWI by various processes of summation. For instance, the area of the figure below could be approximated by the standard methods of summing narrow rectangular elements of the area, by the trapezoidal rule, or by Simpson's parabolic rule. These methods for approximating integration involve a series of steps, each of which may be handled by WWI; i.e., the evaluation of f(x), multiplication to find the area of a finite strip, and addition for the summation of these elements.



The diagram indicates the approximation made by summing narrow rectangular strips. Here the area between C and D is $\sum_{C}^{D} f(x)\,\Delta x$. The difference between this and the actual area $\int_{C}^{D} f(x)\,dx$ is represented by the area of the small segments between the top of the rectangles and the curve of f(x). This error, called truncation error, is inherent in all discrete methods for approximating continuous processes. The truncation error may be reduced by reducing the discrete interval Δx or by using more accurate approximation formulae. The former usually increases computing time and round-off errors; the latter requires more storage and is harder to program.

Cyclical programs seem most natural for evaluating integrals on WWI. The flow diagram shows the general procedure for a program to find the area under a curve f(x) between the limits x = C to D. Let the

total accumulated area be represented by $\Sigma\Delta A_i$, where ΔA_i is the incremental elements of area taken at the value of x equal to x_i.



The result $\sum_{C}^{D} \Delta A_i$ will be left in the register used to contain $\Sigma\Delta A_i$.

More complicated integrations will call for programs of this general form.

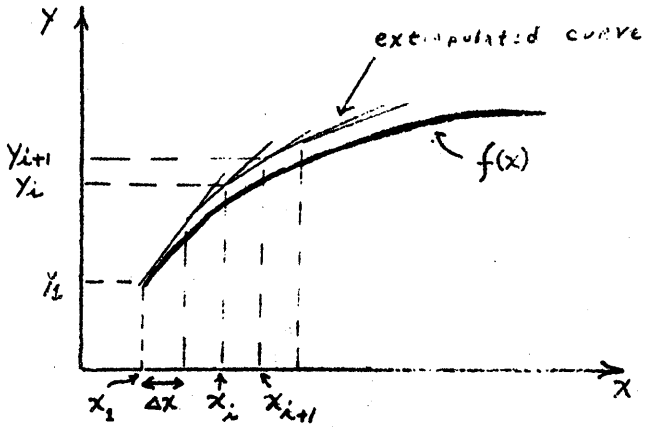
F.3. DIFFERENTIATION

Differential equations, like integration, describe continuous processes which must be transformed to forms which can be treated by discrete processes. Among these are difference equations, extrapolation formulae, and power series expansions. The selection of appropriate methods is more a matter of mathematical analysis than of programming, and again only a relatively simple but important example will be given in the next section.

Simultaneous differential equations are handled by first transforming them to simultaneous algebraic equations.

A program to solve a set of simultaneous differential equations by the use of linear extrapolation formulae is given in the next section. In general, extrapolation formulae carry an approximation to a curve along short segments of lines parallel to the curve which is being approximated.

Consider that the first derivative $(\frac{dy}{dx})$ at a point on the curve of a given function f(x)-- defined by differential equations-- is the value of the slope of the tangent to the curve at that point. The slope at this point may also be expressed as the ratio of finite increments $(\frac{\Delta y}{\Delta x})_i$. These finite increments are used to replace the differentials in the differential equations.



$$y_{i+1} = y_i + (\frac{dy}{dx})_i \Delta x; \quad (\frac{dy}{dx})_i = \text{slope found by evaluating derivative of } f(x)$$

$$x_{i+1} = x_i + \Delta x \qquad ; \Delta x = \text{constant}$$

G. SIMULATION AND DISPLAY

G.1. SOLUTION OF DIFFERENTIAL EQUATIONS DESCRIBING MOTION AND OSCILLOSCOPE DISPLAY OF THE PATH

This example of the handling of a set of differential equations will also be used to illustrate simulation of a physical process by WWI. Use of the output display orders for controlling an oscilloscope will be introduced.

The differential equations of a projectile, or any moving body in space, acted on by the force of gravity only are:

$$\frac{d^2y}{dy^2} = -g \text{ (acceleration of gravity)}$$

$$\frac{dx}{dt} = v_x \text{ (constant horizontal velocity component}$$

Linear extrapolation formulae used to find successive points (x_i, y_i) of the path of motion at times t_i are:

$$x_{i+1} = x_i + v_x \Delta t = \text{x position component}$$

$$y_{i+1} = y_i + (v_y)_{i+1} \Delta t = \text{y position component}$$

$$(v_y)_{i+1} = (v_y)_i - g\Delta t = \text{vertical velocity component}$$

The program solves these equations for x_{i+1}, y_{i+1} and displays the corresponding points on an oscilloscope. It further displays a horizontal axis at $y = 0$ and arranges to have the path of motion represent that of a body bouncing on the horizontal axis. The coefficient of restitution R is equal to $-\frac{(v_y)_a \text{ (after bounce)}}{(v_y)_b \text{ (before bounce)}}$ The gp is used to determine when the value of y_i becomes negative. Then the corresponding downward value of $(v_y)_b$ is multiplied by R to get $(v_y)_a$ which will be directed upward since R is negative. A sketch of the display follows the program.

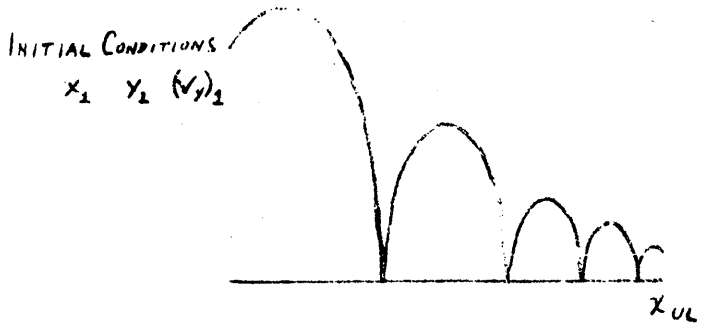
BOUNCING BALL DISPLAY

1.	ca 207	x_1
2.	ts 201	
3.	ca 208	y_1
4.	ts 202	
5.	ca 209	$(v_y)_1$
6.	ts 203	
7.	ca 201	x_i
8.	ad 210	$x_{i+1} = x_i + v_x \Delta t$
9.	qh 201	Set horizontal deflection of oscilloscope to x_{i+1}, transfer x_{i+1} to register #201.
10.	ca 212	+ 0
11.	qd 212	Set y deflection to + 0, display point $(x_{i+1}, + 0)$ to form a point on the horizontal axis.
12.	cs 202	y_i } Determine if last point calculated lies above or
13.	cp 17	} below x axis and proceed accordingly
14.	ca 203	$(v_y)_b$
15.	mr 204	$R(v_y)_b = (v_y)_a$ = new $(v_y)_i$ directed upward
16.	ts 203	
17.	ca 203	$(v_y)_i$
18.	su 205	$(v_y)_{i+1} = v_{yi} - g\Delta t$
19.	ts 203	
20.	mr 206	$(v_y)_{i+1}\Delta t$
21.	ad 202	$y_{i+1} = y_i + (v_y)_{i+1}\Delta t$
22.	qd 202	Display point of path at (x_{i+1}, y_{i+1})
23.	cm 201	$\lvert x_i \rvert$
24.	su 211	$\lvert x_i \rvert - x_{UL}$
25.	cp 5	If $\lvert x_i \rvert < x_{UL}$, return to calculate next point on path
26.	sp 1	Return to repeat display
201.	x_i	
202.	y_i	
203.	$(v_x)_i$	
204.	R	$= \left\lvert \frac{(v_y)_a}{(v_y)_b} \right\rvert$
205.	$g\Delta t$	
206.	Δt	
207.	x_1 }	
208.	y_1 }	Initial conditions
209.	$(v_y)_1$ }	
210.	$v_x\Delta t$	Δx
211.	x_{UL}	Upper Limit of x_i (greater than x_1 in magnitude)
212.	+ 0	y value of horizontal axis

Of course actual numerical values for the stored constants must be used in the final program given to RWI. These values would be scale-factored and proportional to those of the simulated process.

A program such as this could include formulas for modifying the initial conditions of position and velocity so as to control the actual path of motion during operation of the program. For instance a relationship between the final conditions at x_{UL} and the initial conditions could be programmed, which would serve to adjust the initial conditions to meet certain desired final conditions at x_{UL}. Further, it would be possible to have the program provide for the variation of x_{UL} with externally introduced data. In this way the program could be considered to both <u>simulate</u> and <u>control</u> a physical process.

At any rate, the parabolas displayed as successive points on the oscilloscope screen represent solutions of the original differential equations for different sets of initial conditions.



Results of programs may be displayed in a variety of other ways. At present results may be typed on a typewriter or punched on paper tape. These latter two forms of output require special conversion subroutines which are available.

H. USE OF STORED TABLES

H.1. GENERAL PROCEDURE

The general procedure for looking up a value stored in tabular form in WWI storage is given here. Consider that a stored table of values consists of n values which are functions of x, x ranging from 1 to n. The program begins at a point where a value of x equal to x_i is already in the accumulator. The last order leaves the corresponding function, $f(x_i)$, in the accumulator. The values of x_i are scale factored to $x_i \times 2^{-15}$.

Instructions		Effect	Data	
14.	- - -	x_i in AC	215.	ri 215
15.	ad 215	$x_i + 215$	216.	$f(x_i=1)$
16.	td 17		217.	$f(x_i=2)$
17.	ca - - -	$f(x_i)$ in AC	218.	$f(x_i=3)$
			215+n.	$f(x_i=n)$

Notice in addition to the table of values that it is necessary to store also an address to indicate the beginning of the table.

Normally it requires less storage space to evaluate an analytic function than it does to store a table of its values unless only a few values are needed. On the other hand, the use of tables may save considerable operating time, which may be needed in certain control applications. A method for interpolating between values in a stored table of analytic functions is given in the next section.

Tables must be used for looking up non-analytic functions of x, i.e., values or words which are identified by the value of x but which cannot be evaluated from x in an equation. For example, such tables are used for conversion programs to get from one coding system to another. It is with the use of such a table that WWI converts decimal and binary characters to binary when a program is read into storage. The same conversion tables are used to convert results back to decimal and alphabetical characters when results are being read out of the computer. Such tables are also needed in all applications involving accounting, statistical analysis, correlation of data, etc.

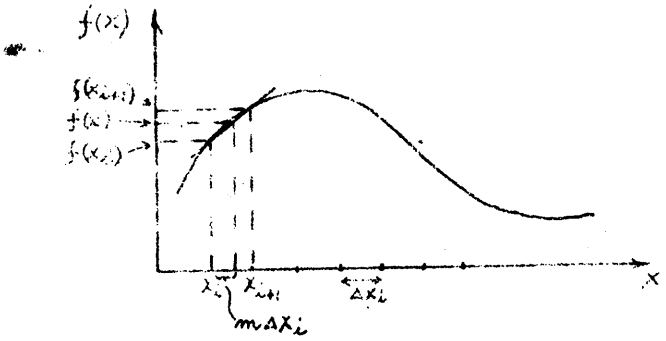
H.2. LINEAR INTERPOLATION

The formula for finding an unknown function of x, f(x), which lies between two known functions of x, $f(x_i)$ and $f(x_{i+1})$, by linear interpolation is:

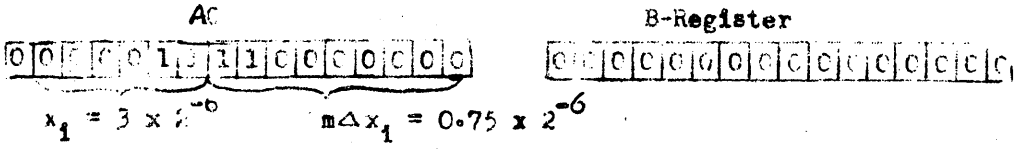
$$f(x) = f(x_i) + m\left[f(x_{i+1}) - f(x_i)\right]$$

where $m = \dfrac{x - x_i}{x_{i+1} - x_i} = \dfrac{x - x_i}{\Delta x_i}$ is

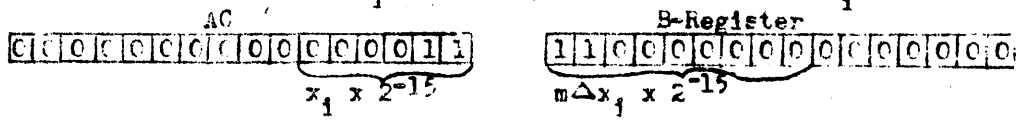
the fraction of the interval Δx_i that is covered by the interval $(x-x_i)$.



In the program given here using this interpolation formula, values of x_i are considered to be positive integers. The value of $x = x_i + m\Delta x_i$ for which f(x) is to be obtained by interpolation is represented in a register as $x \times 2^{-6}$. Up to 2^6 or 64 values of x_i may be handled in this manner. For the particular value of x three-quarters of the way between $x_i = 3 \times 2^{-6}$ and $x_{i+1} = 4 \times 2^{-6}$, x would appear in the AC as:



where $\Delta x_i = 2^{-6}$. With this representation of x in the AC, the instruction sr*9 (shift everything in the AC and B-Register 9 digit positions to the right without roundoff) would leave $x_i \times 2^{-15}$ in the AC, and $m \times 2^{-15}$ in the B-Register. Then $x_i \times 2^{-15}$ may be used to find $f(x_i)$ and



$f(x_{i+1})$, as was done in section H.1. The sl 15 instruction (shift everything left 15 digit positions) would then put the fraction m in the AC. Thus the three values $f(x_i)$, $f(x_{i+1})$, and m, required to find f(x) from the extrapolation formula, are all made available.

Linear Interpolation in a Table of n Values ($n \geq 64$)

	Instruction		Effect
Prepare to find $f(x_i)$, $f(x_{i+1})$, m.	1.	ca 200	x
	2.	sr *9	$x_i (x\ 2^{-15})$ in AC, m x 2^{-15} in B-Register
	3.	ad 203	address of $f(x_i)$
	4.	td 11	
	5.	td 13	
	6.	ad 200	
	7.	td 10	address of $f(x_{i+1})$
	8.	sl 15	m in AC
	9.	ts 202	
Find f(x) by interpolation formula	10.	ca _ _ _	$f(x_{i+1})$
	11.	su _ _ _	$f(x_{i+1}) - f(x_i)$
	12.	mr 202	$m\left[f(x_{i+1}) - f(x_i)\right]$
	13.	ad _ _ _	$f(x) = f(x_i) + m\left[f(x_{i+1}) + f(x_i)\right]$
	14.	ts 202	
	15.	sp next job	
		⋮	
Data			
	200.	ri 1	
	201.	x	value of x for which f(x) is desired
	202.	_ _ _	receives m, then f(x)
	203.	ri 204	address of beginning of table
	204.	$f(x_i (x\ 2^{-15}) = 0)$	
	205.	$f(x_i = 1)$	
	206.	$f(x_i = 2)$	tabulated functions
		⋮	
	204+n.	$f(x_i = n)$	

HS:jl

Signed Hrand Saxenian
Hrand Saxenian

Approved
J. W. Forrester

Attached:

A-36229	1a
A-36230	6a
A-35676	8a
A-36237	14a
A-36234	15a
A-36232	15b
A-36233	15c
A-36231	15d
Short Guide to Coding	A viii
Revisions in the Whirlwind I Order Code	A ix

APPENDIX

		Page
A.	BINARY NUMBERS IN WWI	A 1
B.	DECIMAL-TO-BINARY CONVERSION	A iv
C.	SHIFTING BINARY NUMBERS IN WWI	A vi
D.	PROGRAMMING IN OCTAL (BASE 8) FORM	A vii
E.	ORDER CODE	A viii

A. BINARY NUMBERS IN NWI

The decimal system of numbers is based on the use of 10 separate digits for counting. These are the integers 0 through 9. Correspondingly, the binary system of numbers is based on the use of 2 digits, 0 and 1.

The number 5097.23 in the decimal system may be represented in more explicit form, namely:

$$5(10^3) + 0(10^2) + 9(10^1) + 7(10^0) + 2(10^{-1}) + 3(10^{-2})$$

The two forms are identical except that each power of ten given in the latter form is implied in the first form by the number of places to the left or right of the decimal point of its coefficient. Use of the second form of the number is hardly necessary in the decimal system, but it is helpful in explaining the binary system, or any other number system.

The following is an example of a binary number:

1110.101

It also may be represented in a corresponding explicit form, using powers of 2 instead of 10, and 2 possible coefficients instead of 10, thus:

$$1(2^3) + 1(2^2) + 1(2^1) + 0(2^0) + 1(2^{-1}) + 0(2^{-2}) + 1(2^{-3}).$$

This is equivalent to:

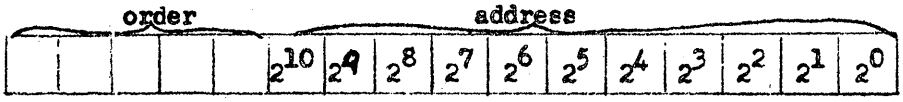
$$1(8) + 1(4) + 1(2) + 0(1) + 1(1/2) + 0(1/4) + 1(1/8).$$

Adding these factors together gives 14.625 as the decimal representation of the original binary number, 1110.101.

Binary numbers in NWI registers are either positive integers in the case of addresses in instructions, or less than one in magnitude

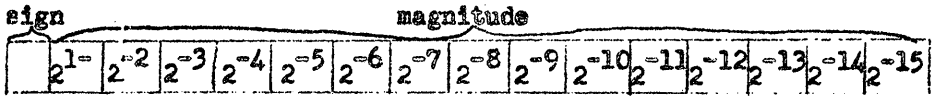
in the case of numerical values to be used in calculations. In other words, the binary point is considered to be at: (1) the extreme right of the 11 digit positions available for addresses; (2) the extreme left of the 15 digits available for magnitudes of words representing numbers.

Thus, as in the example of the more explicit form of representing binary numbers, when the digit positions for an address contain a 1 they represent the values indicated:



Range of addresses held in one register
0 to 2047 in steps of 1

Correspondingly for words representing numbers:



Range of positive and negative numbers
0 to $1-2^{-15}$ in steps of 2^{-15}

Scale factoring permits the representation of a practically unlimited range of numbers.

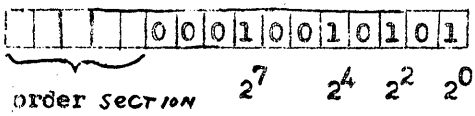
Example of an address in a register follows:

The address 149 can be represented as:

$2^7 + 2^4 + 2^2 + 2^0$ (= 128 + 16 + 4 + 1 = 149)

In binary form this is 10010101.

In a register the address 149 would be:



Example of number:

The number + .375 can be represented as:

$2^{-2} + 2^{-3}$ (= .250 + .125 = .375)

This is represented in a register as:

The negative number - .375 is represented by changing all the 0's to 1's and 1's to 0's of its positive representation:

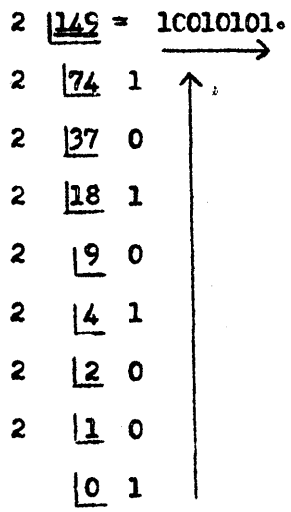
Because of this convention, the number 0 can be represented as + 0 with 16 0's or as -0 with 16 1's.

B. DECIMAL TO BINARY CONVERSION

The preceeding examples of an address and a number in a register indicate a method for decimal to binary conversion. The address 149 was converted to binary form by first finding the largest power of 2 (2^7) which would fit into 149; next was found the largest power of 2 (2^4) which would fit into 21, the difference between 149 and 2^7, etc. It was shown that the sum of these powers of 2 equaled the original number to be converted. Given these powers of 2, the binary number was written out by placing 1's in the corresponding binary digit position to the left of the binary point.

The number .375 was converted by the same process. Since .375 is less than 1, only negative powers of 2 fit into it, and these are represented to the right of the binary point.

A mechanized technique for handling this conversion for integer numbers is:



Considering the binary point to be above the first remainder, the result is 10010101 = 149. In this process the operator divides 2 into 149, puts the integer result below, and the remainder as 1 to the right of 74. Next he divides 2 into 74, puts the integer result below and since there is no remainder puts 0 to the right of 37. The process is continued until the last quotient is a 0.

The same thing is done with fractions, except that the decimal number and its remainder are divided by 1/2 (multiplied by 2) each time, and a 1 is used to indicate that the result of a step has exceeded 1, a 0 to indicate the result of a step is less than 1.

$$
\begin{array}{ll}
\underrightarrow{.0110} = .375 & (x2) \\
\downarrow\ 0(.750) & (x2) \\
\downarrow\ 1(.500) & (x2) \\
\downarrow\ 1(.000) & (x2) \\
\downarrow\ 0(.000) &
\end{array}
$$

This is equal to .0110, again considering the binary point to be above the first 0 or 1.

To convert a binary to a decimal number, simply add the decimal values of the powers of 2 represented by each 1 in the binary number.

C. SHIFTING BINARY NUMBERS IN WWI

WWI has orders which shift the position of a number either to the left or to the right in a register.

A shift of one digit position to the left corresponds to multiplying the original number by 2. Shifting to the right corresponds to division by 2. This is so because the shift is made relative to a fixed binary point. Correspondingly, shifting digits in the decimal system relative to the decimal point corresponds to multiplication or division by 10.

D. PROGRAMMING IN OCTAL (BASE 8) FORM

Binary-to-octal conversion is simpler and more rapid than is binary-to-decimal conversion. In the octal system of numbering, the eight digits 0 through 7 are used. Three binary digits are sufficient to represent any single octal digit.

Octal	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

The binary form of an octal number may be found directly from the above table, which is short enough to be readily memorized. For example, the octal number 316 is 011 001 110. Any binary number may be converted to octal by simply reading off from the left the octal digits (each consisting of three binary digits).

Programs for WWI are often prepared in octal form rather than in decimal form because of this greater ease of conversion.

The WWI control panel has indicator lights which give the binary contents of any desired register. The binary register contents may be readily recorded in octal form or readily checked against a program originally written in octal form.

For programs written in octal form, registers are numbered 0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, ... 75, 76, 77, 100, 101... Special program forms numbered octally are available at Project Whirlwind.

A SHORT GUIDE TO CODING

Using the Whirlwind I Code of October 1949

Electronic Computer Division
Servomechanisms Laboratory

Massachusetts Institute of Technology
Cambridge, Massachusetts

A SHORT GUIDE TO CODING

(using the Whirlwind I code of October 1949)

COMPUTER PROGRAMS

Program. A program is a sequence of actions by which a computer handles a problem. The process of determining the sequence of actions is known as programming.

Flow diagrams. A flow diagram is a series of statements of what the computer has to do at various stages in a program. Lines of flow indicate how the computer passes from one stage of the program to another.

Coded program. Programs and flow diagrams are largely independent of computer characteristics, but instructions for a computer must be expressed in terms of a code. A set of instructions that will enable a computer to execute a program is called a coded program, and the process of preparing a coded program is known as coding.

Orders and operations. Individual coded instructions are known as orders and call for specific operations such as multiply, add, shift, etc.

The computer code. The computer code described here is that of Whirlwind I, an experimental computer using binary digits, single-address order code, parallel operation, and electrostatic storage. It is expected that computers of this type will ultimately achieve an average speed of 50,000 operations per second.

COMPUTER COMPONENTS

Registers and words. A register has 16 digit positions each able to store a one or a zero. A word is a set of 16 digits that may be stored in a register. A word can represent an order or a number.

Arithmetic element. Arithmetic operations take place in the arithmetic element, whose main components are three flip-flop registers, the A-register, the accumulator, and the B-register (AR, AC, BR). The 16 digit positions of AR starting from the left are denoted by AR 0, AR 1, . . ., AR 15. Similarly for AC, BR. Words enter AC through AR; BR is an extension of AC.

Storage. The term "register" by itself refers to the main electrostatic storage, which consists of 2^{11} or 2048 registers, each of which is identified by an address. These addresses are 11-digit binary numbers from 0 to 2047. The computer identifies a register by its address.

Input-output. All information entering or leaving the computer is temporarily stored in the input-output register (IOR). The computer regulates the flow of information between the internal storage and IOR, and also calls for any necessary manipulation of external units. The descriptive names of the input-output orders were chosen for photographic film reader-recorder units, but the orders are applicable to other types of external equipment.

Control element. The control element controls the sequence of computer operations and their execution. Instructions are obtained from storage in the form of individual orders, each of which is represented by a single word.

Inter-connections. The four main elements (storage, control, arithmetic, and input-output) are connected by a parallel communications system, known as the bus.

REPRESENTATION OF ORDERS

Operation section. When a word is used to represent an order the first (left-hand) 5 digits, or operation section, specify a particular operation in accordance with the order code.

Address section. The remaining 11 digits, or address section, are interpreted as a number with the binary point at the right-hand end. In the majority of orders this number is the address of the register whose contents will be used in the operation. In orders sl, sr, the number specifies the extent of a shift; in rf, rb, the number specifies an external unit; in ri, rs, the address section is not used.

Example. The order ca x has the effect of clearing AC (making all the digits zero) and then putting into AC the word that is in the register whose address is x. If q is a quantity in some register, the order needed to put q in AC is not ca q but ca x, where x is the address of the register that contains q.

REPRESENTATION OF NUMBERS

Single-word representations. When a word is used to represent a number the first digit indicates the sign and the remaining 15 are numerical digits. For a positive number the sign digit is zero, and the 15 numerical digits with a binary point at their left specify the magnitude of the number. The negative -y of a positive number y is represented by complementing all the digits, including the sign digit, that would represent y. (The complement is formed by replacing every zero by a one and every one by a zero.) In this way a word can represent any multiple of 2^{-15} from $2^{-15} - 1$ to $1 - 2^{-15}$. Neither +1 nor -1 can be represented by a single word. Zero has two representations, either 16 zeros or 16 ones, which are called +0 and -0 respectively.

Overflow — increase of range and accuracy. With single-word representation the range is limited to numbers between $2^{-15} - 1$ and $1 - 2^{-15}$. Programs must be so planned that arithmetic operations will not cause an overflow beyond this range. The range may be extended by using a scale factor, which must be separately stored. Accuracy can be increased by using two words to represent a 30-digit number.

COMPUTER PROCEDURE

Sequence of operations. After the execution of an order the program counter in the control element holds the address of the register from which the next order is to be taken. Control calls for this order and carries out the specified operation. If the order is not sp or cp(-) the address in the program counter then increases by one so that the next order is taken from the next consecutive register. The sp and cp(-) orders permit a change in this sequential procedure.

Transfers. A transfer of a digit from one digit position to another affects only the latter digit position, whose previous content is lost.

Negative zero. The subtraction of equal numbers produces a negative zero in AC, except when AC contains +0, and -0 is subtracted from it.

Manipulation of orders. Words representing orders may be handled in the arithmetic element as numbers.

Procedure in the arithmetic element. The execution of an addition includes the process of adding in carries; this process treats all 16 digits as if they were numerical digits, a carry from AC 0 being added into AC 15. A subtraction is executed by adding the complement. Multiplication, division, shifting and round-off are all executed with positive numbers, complementing being performed before and after the process when necessary. For round-off the digit in BR 0 is added into AC 15.

NOTATION FOR CODING

Addresses. A coded program requires certain registers to be used for specified purposes. The addresses of these registers must be chosen before the program can be put into a computer, but for study purposes this final choice is unnecessary, and the addresses can be indicated by a system of symbols or index numbers.

Writing a coded program. Registers from which control obtains orders may be called action registers, and should be listed separately from registers containing other information, which may be called data registers. A coded program is written out in two columns; the first contains the index number of each action or data register, and the second column indicates the word that is initially stored in that register. In many cases part or all of a word may be immaterial because the contents of the register in question will be changed during the course of the program. This state of affairs is indicated by two dashes, for example, ca --.

The abbreviations RC, CR. Abbreviations used in referring to the register that contains a certain word or to the word in a certain register are

RC . . . = (Address of) Register Containing . . .
CR . . . = Contents of Register (whose address is) . . .

The symbol ri x. When an address forms part of an order it is represented by the last 11 digits of a word whose first 5 digits specify an operation. An address x that is not part of an order is represented by the last 11 digits of a word whose first 5 digits are zero, which is equivalent to specifying the operation ri. Thus the word for an unattached address x may be written ri x. It could also be written $x \times 2^{-15}$.