intercom programming

FOR THE

BENDIX G-15

COMPUTER

WYANT and HOWELL

intercom programming

FOR THE
BENDIX G-15
COMPUTER

LINSLEY WYANT
Mathematics Department
Cabrille College
Watsonville, California

JOHN M. HOWELL
Mathematics Department
Los Angeles City College
Los Angeles, California

WM. C. BROWN COMPANY PUBLISHERS
135 SOUTH LOCUST STREET • DUBUQUE, IOWA

NOTES	LOCATION	K	OP	ADDRESS		ACCUMULATOR
position paper	1200		30	00	05	?
clear and add a = (1251)	1		42	12	51	a
type a and tab	2		33	21	01	a
add a	3		43	21	01	2a

Manufactured by WM. C. BROWN CO. INC., Dubuque, Iowa
Printed in U. S. A.

Preface

With the use of digital computers in business and industry enormously increasing, colleges, universities, and even high schools have been establishing courses in the programming of these machines. Many institutions have acquired computers for use by students in such courses, and the Bendix G-15 is one of the machines that is being rather widely used in this way. One problem in developing these educational programs, however, has been the small number of textbooks available. While several books have been written describing the techniques of digital computing within the framework of a hypothetical machine, only a very few have been designed around an existing computer. There now seems to be a strong need for this latter type of book since so many schools have installed machines. Furthermore, the ideas basic to digital computing seem best understood through the actual use of a computer.

This book, then, is designed to provide the fundamental concepts necessary in digital computer programming and to serve as a text for courses in which a Bendix G-15 is available to the class. It is hoped, too, that anyone interested in digital computing, and the Bendix Intercom systems in particular, will find the book useful.

The material was developed from two distinct courses taught at Los Angeles City College, one for students majoring in Mathematics, Science, or Engineering, and the other for Business Administration and Social Science students. Both are offered as sophomore courses. An important feature of both is a weekly laboratory session for every student during which he learns to operate the computer and to carry through the solution of problems he has programmed. Experience so far indicates that this activity is of very great value in learning the material.

The first seven chapters of this book include the fundamental topics which have been covered in the courses. Chapters 8, 9, and 10 explain some interesting additional features of the Intercom systems and examples of typical applications. This material could be used to supplement a course as found appropriate. The Appendices contain reference material including cards which may be clipped from the book for quick reference.

It is hoped that the many flow charts, worked examples, and sample examinations will be found particularly helpful. The number of possible exercises is virtually unlimited; those included have been successfully used and are generally designed to keep down the output time required in lab sessions.

Those familiar with computing will note that the Intercom systems operate entirely in floating point so that fixed point programming with its problem of scaling is not included in this book. Also, it is not possible in the Intercom systems to modify commands by direct arithmetic operations, so that this technique, too, has been necessarily omitted. Students who learn Intercom, however, should not experience difficulty in learning these additional techniques if necessary. With the trend toward floating point, index registers, and compilers, these omissions are now less significant than they would have been five or more years ago.

We are very grateful to Dr. Rosella Kanarik for teaching from the book in preliminary form and for offering many valuable suggestions and corrections. We also wish to thank Mr. Marshall Elder and Mr. Glenn James for detecting errors and offering suggestions, and Mr. and Mrs. George Wyant for their work in editing our grammar. Our thanks to Mr. Howard Mark of Bendix Computer for reviewing the manuscript and to the Bendix Corporation for permission to use some of their material.

The authors wish to express special appreciation for the valuable contributions of Mrs. Joan Jack who assumed the responsibility for the production of the typographical portions of the text.

The Bendix G-15 Computer

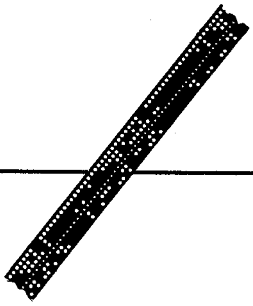


Table of Contents

TABLE OF CONTENTS (continued)

CHAPTER PAGE

Chapter 1

Introduction to Computers

1.1 Background

An important objective of this course is to achieve a rational understanding of digital computers which, while unquestionably remarkable, are perhaps not so mysterious as the popular press would have us believe. Whether or not computers "think" is a perilous question but, at least at present, it seems inappropriate to term them "giant brains." In any case, it may help to view them as a natural development of our search for more efficient and rapid means of numerical calculation.

The earliest efforts to ease the burden of computation were not mechanical. The most dramatic of these was doubtless the invention of logarithms. Others, less well known, began to anticipate mechanization. One such device was "Napier's Bones," a set of numbered bars invented in the late 16th century by John Napier whereby he attempted to simplify tedious multiplications.

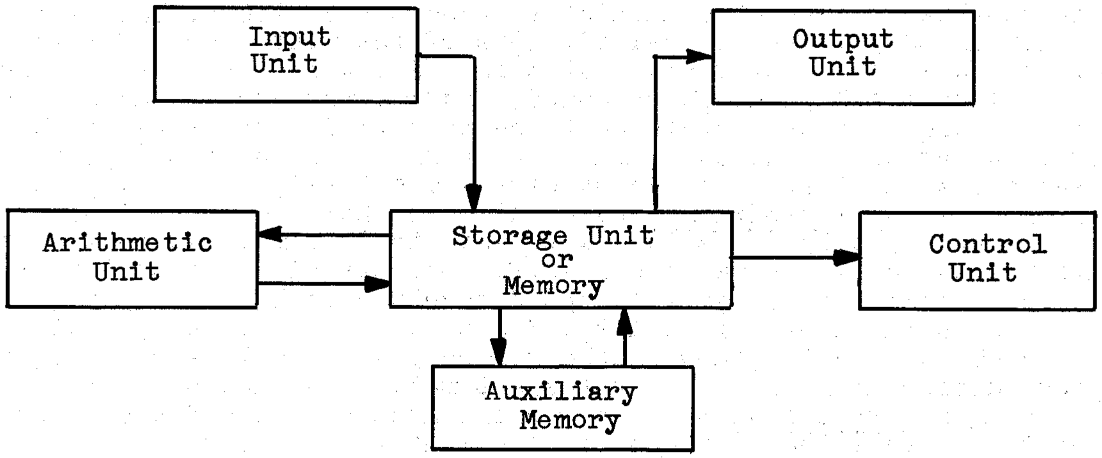
The mechanical stage began with the invention of a numerical-wheel calculator by Pascal in 1642. This device led to the modern adding machines and desk calculators with which most of us are familiar if for no other reason than our frequent trips through the check stand of a supermarket. While these mechanical counters are very useful, they have, from the point of view of control, at least one major weakness. It is necessary to keep telling them what to do. Even if the same series of calculations is to be repeated hundreds of times, the desk calculator must be actuated by the operator at each and every step, and in a lengthy problem this requirement can be prohibitive.

Thus we come to the development of means of automatic control. The problem is somehow to pre-set a machine to repeat a series of calculations automatically as many times as desired. One of the most successful ways of doing this has been the plug board system in which wires are inserted into a panel rather like a telephone switchboard, and the pattern of these wires determines the sequence of operations to be performed. The data may be made available as a pattern of holes punched in cards or in paper tape, so that the numbers become available to the machine sequentially. Such a calculator is called an *externally programmed* calculator in that the control of its operation is, at least by comparison with what is to come, external to the functioning hardware of the machine itself. Because of their ability to operate automatically and at greater speed than the desk calculator, literally thousands of such machines were built and many are still in use. But like the simple desk calculator, they in their turn possess a less obvious but very serious disadvantage. Once set the automatic sequence of operations cannot be changed *automatically*. The machine must be stopped and reset. The removal of this defect led to the digital computers of today and to the stored-program concept which gives them their enormous power. The understanding of this concept will be developed throughout this book. Simply stated, it means that the manner of instructing the computer will be *internally* stored prior to calculation and these stored "instructions" will be subject to modification during automatic operation. A full appreciation of this powerful idea can only be achieved through the actual programming and use of a computer.

Before proceeding, it may be well to point out an important distinction. We have been outlining the development of digital computers but the analogue computers deserve mention as well, although we will not be concerned with them in this book. Briefly, digital computers count in distinct steps whereas analogue machines measure quantities. A familiar and simple example of a device which uses the digital principle is the abacus in which numbers are handled by positioning counters to represent the digits involved. The abacus is exact in that it works with integers. By comparison, a slide rule is a simple analogue "computer" in that it represents numbers by measuring off their logarithms on ruler-like scales and is accurate only within the limitations of these measurements. Thus a ten inch slide rule can be graduated to give three significant digits in general, but a scale about sixty inches long is required to increase this to four significant figures. In recent years numerous mechanical and electronic analogue devices have been built and they have found particular application in the solution of differential equations. The theoretical basis of analogue machines is entirely different from that of digital computers. Analogue machines have far less general application than digital computers.

1.2 The Schematic Arrangement of a Computer

It is not at all necessary to know anything about electronics to program and operate computers. It is very useful, however, to have in mind their functional components and an understanding of the purpose of each. This material will apply to all computers and provides a schematic framework in which to view them. Throughout the discussion the reader should refer to the diagram below which shows the relationships between the various components. Actually, there is interaction between all units of the computer, but for simplicity we have shown here only lines of flow of information.



Schematic Diagram of a Computer.
Arrows show basic paths of information flow.

1.2 (continued)

Central to the operation of the modern computer is the storage unit or memory. Functionally this is nothing more than a set of "pigeonholes" into which groups of characters may be placed. In some machines these characters can only be numbers, but in others letters and punctuation symbols may be stored as well. Each pigeonhole is called a location and each location has a unique numerical address. The contents of a location are called a word. The memory is, of course, used to hold data for the problem being solved, but it is also used to hold numerically coded instructions which, when obeyed in sequence, will cause the machine to perform the desired operations. These numerically coded instructions make up a program and computers in which the program is held in the memory along with the data are stored-program computers.

The means by which the memory storage is accomplished will not be described in detail. Several methods have now been perfected. The Bendix G-15 employs a magnetic drum, a type of memory in very common use. The cylindrical drum revolves (in the Bendix G-15 at 1800 rpm) such that its curved surface passes continuously under units called read-write heads. These heads magnetize locally the drum surface and the pattern of the magnetized areas determines the number stored. Conversely, the heads can sense the magnetized pattern and translate it into a series of electrical pulses. Thus the computer "reads," "writes," and "remembers." Further details of the G-15 memory unit will be given in Chapter 2. Another widely used type of storage unit is the magnetic core memory in which a series of small doughnut-shaped iron cores are magnetized to indicate a series of numbers.

Many computers have, in addition to the regular memory, an auxiliary memory, usually of large capacity (say 100,000 words or more), and frequently in the form of magnetic tape units. Here again, information is stored as a pattern of tiny magnetized areas, but this time on the specially coated surface of plastic tape which is wound on large reels rather like movie film.

The reader may discern that these storage devices are logically binary; that is, the magnetized areas can have only one of two states, either "north-south" or "south-north." Hence the number system with base two, called the binary system, is usually used in computers so that the only digits to be stored are zeros and ones. These binary digits are abbreviated "bits" in computer jargon. Although the Bendix G-15 is basically binary, the Intercom systems to be discussed in this book allow the programmer to work exclusively with numbers to the base ten. Therefore, we will not go into the binary number system, but the reader is advised that it, together with certain other number systems, is frequently of importance in computing work.

As the diagram indicates, data travels from the memory to the Arithmetic Unit for processing. Which memory locations are involved and the nature of the operations performed are determined by the stored instructions which are interpreted sequentially in the Control Unit. The arrows on the diagram indicate the basic paths linking these components to the memory.

Clearly, some arrangement must be made for placing numbers in the memory and for getting numbers out. This is accomplished by the input and output units. On the G-15 an electric typewriter is provided and numbers typed on it may be transmitted to the memory. Conversely, numbers in the memory may be automatically typed out on this typewriter. A second input mode is available in the form of a photoelectric paper tape reader. This unit translates a pattern of punched holes into electric pulses which are in turn stored on the drum. The computer has a paper tape output unit which punches information from the memory as a pattern of holes in the tape.

1.2 (continued)

There are many other input and output methods in use on various computers. Punched cards are very commonly used and, as in the case of punched paper tape, units must be provided to translate the pattern of holes in the card to electrical pulses and to translate a sequence of pulses into punched holes. For output, several types of printing units are available as well as such specialized devices as the cathode ray tube for drawing graphs.

These various components and their associated circuitry make up the hardware of a computer. Many books are available describing computer hardware in detail both from an electronic and a logical design point of view.

1.3 Basic Computer Operations

From the point of view of circuitry, there are remarkably few things a digital computer can do. Naturally, it can add numbers and this turns out to be the basic operation for which the machines are designed. In fact, the other three arithmetic operations are accomplished by addition with the assistance of some circuitry which can shift numbers with respect to each other and form what is called the complement of a number. These ideas will be illustrated in some examples.

It is easy to see how multiplication may be done. Consider the problem 4172 x 213. The machine process may be compared with the usual hand method as follows:

```
By hand             By machine

  4172                4172
x  213              x  213
 12516                4172
 4172                 4172
8344                  4172
888636               4172
                    4172
                    4172
                    888636
```

Except for shifting the multiplicand to take care of place value, only addition circuitry is required.

Subtraction is less obvious. Methods vary, but generally some form of complementation is involved. The tens complement will be used as an example, and we will imagine that our machine can hold three decimal digits in each memory location. (This is not the case in the Bendix, but it will serve as a simple illustration.) In this case the complement of a number x is defined as 1000 - x. Suppose we wish the machine to subtract 135 from 467 and, of course, obtain 332. This would be accomplished by adding the complement of 135 to 467 thus:

$$467 + (1000 - 135) = 467 + 865 = 1332$$

1.3 (continued)

Since our machine has a word length of three digits we take the last three, namely 332, and obtain the correct answer. If the difference were negative, the answer would have to be complemented. For example, 86 - 271 = -185. In the machine:

$$86 + (1000 - 271) = 86 + 729 = 815$$

Then 1000 - 815 = 185 which are the digits in the answer. Obviously, in both these cases provision must be made for the correct algebraic sign to accompany the answer. Also, it is emphasized that these examples are intended merely to indicate the basis used to accomplish subtraction by adding. The details vary with each machine and in many, as in the Bendix G-15, the arithmetic is actually done in base two. Finally, the reason this complementation method is used is that it is cheaper to include the circuitry to complement in connection with the adder unit than it is to build a separate subtracter.

Finally, division is accomplished by repetitive subtraction. For example:

By hand

```
      312
14 )4370
    42
     17
     14
      30
      28
       2
```

By machine

```
      312
14 )4370
   -14
    29
   -14
    15
   -14
     17
    -14
      30
     -14
      16
     -14
       2
```

In addition to the four arithmetic operations, digital computers have a primitive decision-making ability. Usually this takes the form of testing a number (often a computed result) to see if it is positive or negative or zero. In most machines this test is a yes or no alternative. Thus in the Bendix G-15 we have a "transfer on minus" instruction. At this point two possible paths or sequences of instructions will be available. If the quantity tested is negative the computer will follow one path, if zero or positive it will follow the other. The precise form of these tests depends on the particular machine, but all stored-program computers have them and they are largely responsible for the remarkable versatility of modern computers.

1.4 Computers Compared

There are several ways of classifying the computers now in use. Perhaps the most basic comparisons would be in price, speed, and memory size. Sometimes, too, machines are designated "scientific" or "business" according to the area of application in which they are expected to find widest use, but the distinction is far from absolute and, in fact, machines like the IBM 709 have been used in all types of work. For our purposes it will be sufficient to place the Bendix G-15 in perspective at the present time of writing.

The G-15 is a small machine. Its basic price of $49,500 is low as computer costs go, although it has several competitors above and below that figure. Its storage capacity of 2,176 words is modest now that the largest machines have in excess of 32,000 words. The Bendix G-15 memory can be expanded by the addition of magnetic tape units, but, of course, so can the memories of the larger computers. Speeds are rather difficult to compare in that different operations require varying amounts of time. Generally the best way is to compare add times (the time required to add two words). The minimum in the Bendix G-15 is .00054 secs. = .54 millisecs. A millisecond, abbreviated as ms., is .001 seconds. However, this speed is not always achieved in the Bendix G-15. For example, the minimum add time of the Intercom 500 system discussed in this book is 87 ms. Other machines competitive with the Bendix G-15 have generally comparable speeds although recent announcements indicate more speed for the money is on the way. The IBM 650, a machine generally described as medium in size, has a minimum add time of .48 ms., while the add time of one of the largest and most recent machines, the IBM 7090, is .0048 ms., but, of course, this computer is in the over one million dollar class.

One further distinction should be mentioned. Computers vary considerably in "command structure," a term referring to the form of the numerically coded instructions which they are designed to obey. Typically, a command or instruction consists of an operation code calling for a certain manipulation by the machine and one or more addresses specifying the operands (numbers) involved in the manipulation. The command structure is distinguished by the number of addresses contained in one command. Many computers are designated single-address machines in that only one operand is specified per command. There are two and three address machines as well, while the SWAC computer at U.C.L.A. is a four address machine. The Bendix G-15 has a three address command structure. It is probably fair to say that the one address system is the most common.

Most computer manufacturers make available additional attachments called "peripheral equipment" for their machines and Bendix G-15 is no exception. Magnetic tape units for use as auxiliary storage have already been mentioned. Alternative input-output equipment available for the G-15 includes high-speed paper tape punches and readers, punched card input and output devices, a graph plotter, and an alphanumeric typewriter. Finally, there is a special device for handling differential equations called a Digital Differential Analyzer. No peripheral equipment will be described in this book, however.

In summary, the Bendix G-15 is to be viewed as a small computer of moderate speed, but one that has found wide application in business and industry and in which the basic principles are the same as the largest machines. Moreover, it is certainly true that once competence is achieved in programming one machine the rest are relatively easy. Thus, the Bendix G-15 and its Intercom programming system provide a very satisfactory introduction to the fundamentals of digital computing.

1.5 Applications

Digital computers are being used in such a large variety of applications that a comprehensive list is impossible here. Some of the broad areas of use are in the aviation industry, civil engineering, machine tool design, the petroleum industry, missile design, nuclear research, mathematical analysis, and the huge field of business accounting applications often referred to as data processing.

A few specific examples may be of interest. In highway construction a problem involving considerable caluclation arises in planning for cuts and fills. After a projected highway has been surveyed, a calculation is made of the volume of earth to be removed from cuts and to be provided for fills. Computers have been very successfully employed in making these calculations from the survey data. In the aviation industry the calculations stemming from data provided by wind tunnel tests are being conveniently done by computers. Further, in the mathematical analysis of data it is often desirable to fit a curve to a set of points by the method of least squares and this problem can be comfortably handled by machine. In the business field, computers are used for payroll processing, billing, inventory control, actuarial calculations, and so on.

In addition to these broad areas of use, digital machines have been programmed to perform in many bizarre ways which have captured the popular fancy. The game of checkers has been very successfully set up on several machines and a great deal of work has been done on chess. Reportedly, the computers play a fairly good game. There have been some successful efforts to program the machines to "learn" by their mistakes, too. Language translation has been developing for several years and will undoubtedly be done to some extent by machines in the future. Of course, the ingenuity of man is behind these rather striking developments and it is safe to say that much more is to come than we have yet imagined.

Chapter 2

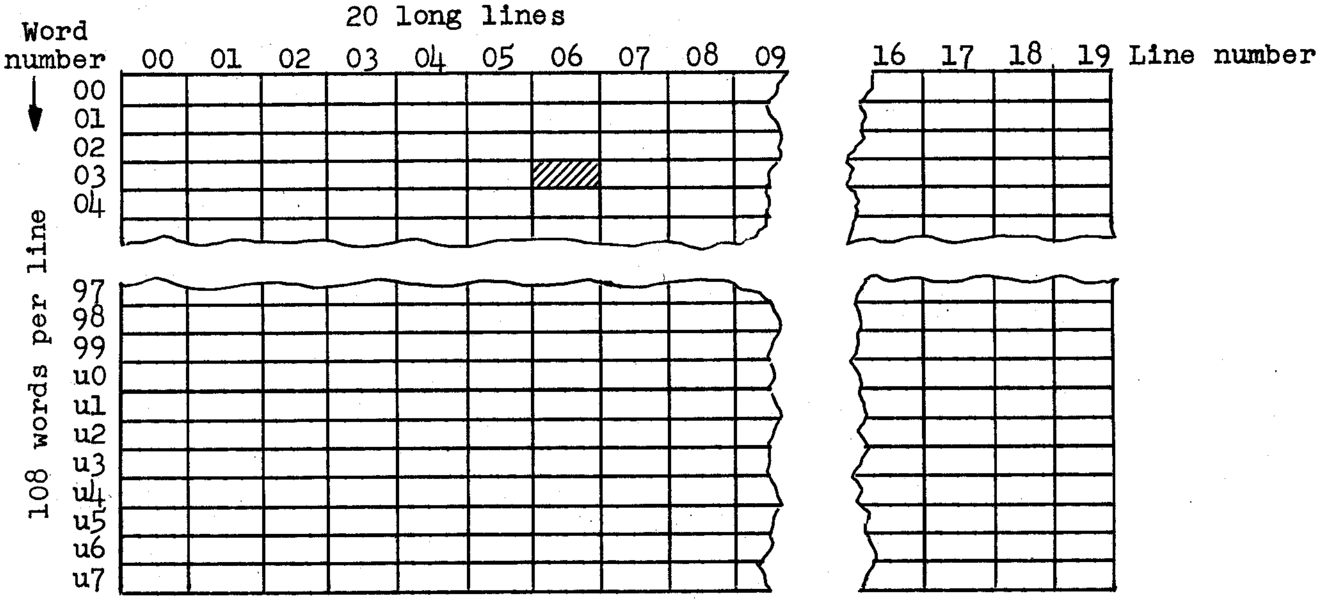
The Bendix G-15 Computer

2.1 Description of the Computer

Memory. As indicated in Chapter 1, this computer uses a magnetic drum for its internal memory. This drum is divided into bands around the drum called lines or channels. The lines are divided into segments called words. Some lines are called long lines, containing 108 words, but we shall be concerned with only 100 of these at present. Some lines are short lines containing one, two or four words. At present we shall be concerned with only a portion of one of these, the accumulator. This is the location at which the results of arithmetic operations are stored.

A location in memory is specified by a four-digit numoer called an address. The first two digits refer to a channel or line on the drum and the last two digits refer to a word position. We shall designate a location henceforth by ADDR or CHWD. These four letters stand for the four digits of the location. Remember that a location is like a pigeonhole into which a command or an item of data may be placed. It will occupy that position until replaced by another item.

The long lines of the memory may be diagrammed schematically by "unrolling"the surface of the drum.



For simplicity, many of the lines and words have not been marked in. The shaded section has address 0603.

2.1 (continued

This computer is referred to as a STORED-PROGRAM or an INTERNALLY-PROGRAMMED computer, since commands are stored in and executed from memory. This memory is a volatile memory. That is, when the computer is turned off, the contents of memory are lost.

Intercom. An intercom program is an interpretive program which occupies a certain portion of memory. It converts the computer from a three-address system to a one-address system, which is much easier to use than the machine language of the computer. Three Intercom systems will be considered here.

Intercom	Locations used by Intercom	Available to programmer	Accumulator address
500 (single precision)	0000 to 0899	0900 to 1899	2173
1000 Single Precision	0000 to 0699	0700 to 1899	2101
1000 Double Precision	0000 to 0899	0900 to 1899	2100

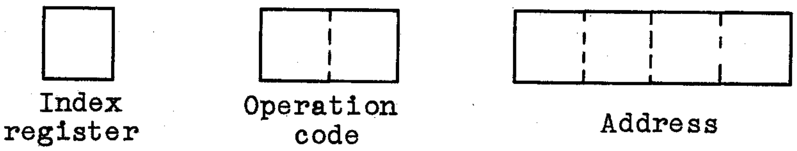
Single precision (SP) Intercoms work to five significant figures and double precision (DP) Intercom works to twelve significant figures.

Input-Output. Input and Output on this computer are either by the typewriter or by punched paper tape (see Section 1.2). Usually a program is entered into the computer from the typewriter. After it has been tested it is punched on paper tape by the computer, so that it may be used over and over again.

Since input by tape is much faster than by the typewriter, even incomplete or incorrect programs are punched on tape. These may be entered into the computer at a later time and completed or corrected. It may take half an hour to type one channel of commands and/or data, but one channel of tape can be entered into the computer in less than a minute.

2.2 Commands

Command Structure. Intercom commands are entered into the computer as seven-digit numbers. We shall refer to these as KOPADDR or KOPCHWD. The form of a command is as follows:



The first digit represents an index register. If no index register is used, the first digit is zero and need not be written. Index registers will be discussed in Chapter 6.

2.2 (continued)

The next two digits of the command represent the operation to be performed. This portion of the command causes the computer to perform arithmetic operations, logical operations, wait for input or initiate output. Operation codes will be discussed as they are introduced and are summarized in the Appendix.

The last four digits of the command are usually the address or location to which the operation code applies. With certain operation codes, however, this portion of the command has a special meaning.

Command Sequence. Commands are normally obeyed in numerical sequence, but transfer commands are available to change the order when desired. The transfer may be unconditional, or contingent upon the value of some calculated quantity or the contents of an index register.

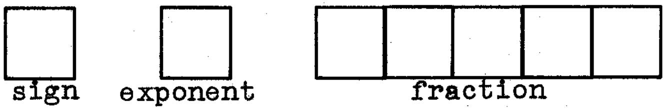
Commands may be stored at and executed from any available address.

2.3 Data

Fixed Point and Floating Point Data. Numbers as we usually think of them (for example, 123.45), will be called fixed point data. The computer, however, works with floating point numbers. The above number would be represented as 53.12345. This device allows a much greater range of numbers to be carried in the memory than could be carried if the computer was confined to fixed point numbers.

A floating point number consists of a whole number or integral portion and a decimal or fractional portion. To convert a fixed point number to a floating point number, move the decimal point until it is just to the left of the most significant digit. This is the fraction part of the floating point number. If the decimal point was shifted to the left, add the number of places shifted to 50. This is the integer portion of the floating point number. If the decimal point was shifted to the right, subtract the number of places shifted from 50.

Data Structure. Although numbers are carried in the computer in binary form, it is convenient to think of them in decimal form. A single-precision number then is carried as seven digits and a sign. Two of the digits are the whole number or exponent part of the floating point number and five digits are the fraction part.



To convert a floating point number to a fixed point number, write the decimal part of the floating point number, subtract 50 from the integer part, move the decimal point this many places to the right if positive. If the subtraction results in a negative number, write this many zeros to the left of the number and place the decimal point to the left of these. Here are some examples of some fixed point numbers and the corresponding floating point numbers.

2.3 (continued)

Fixed point	Floating point	Fixed point	Floating point
34.74	52.3474	34500000.	58.345
1267.365	54.1267365	-23.97	-52.2397
-1.23	-51.123	.0000123	46.123

Range of Values. Either fixed point or floating point data may be entered into, or typed out by, the computer. In a single precision, five significant digits may be used. In double precision, twelve significant digits may be used. If fixed point data is entered, it is automatically converted to floating point by the computer and stored in this manner. A single precision number with sign may be stored at any address. A double precision number must be stored at an even-numbered address and will also occupy the next odd-numbered address. The double precision Intercom accomplishes this "double storage" automatically.

In either single or double precision, as many as seven digits to the left of the decimal point may be entered or typed out in fixed point. The limit of seven is due to the nature of the computer and the Intercom program. In floating point, a number as large as 10^{37} or as small as 10^{-37} may be entered. During output, a number larger than 10^{7} will be typed as a floating point number even though fixed point output was called for. In output, the number of places after the decimal point in fixed point may be selected by the programmer. Remember that even though fixed point input or output is used, the computer carries all numbers internally as floating point numbers. A part of the Intercom program automatically performs this conversion.

Numbers larger than 10^{38} are considered too large to be handled by the computer, and it will stop if a number larger than this is encountered. Numbers smaller than 10^{-38} will be considered zero.

2.4 Example of an Intercom Program

Following is an example of an Intercom program. Commands used here will be explained in detail in Chapter 3. For the time being, notice that the program consists of some commands and some data placed in a portion of memory available to the programmer.

Notes	Location	Command K	 OP	 CH	 WD	Contents of accumulator
position paper	0900		30	00	02	?
clear and add a	0901		42	09	52	a
add b	0902		43	09	53	a + b = x
type accumulator	0903		33	21	01	x
halt	0904		67	00	00	x

Location	Data
0952	a
0953	b

2.4 (continued)

This program when executed will calculate and type x = a + b. We may select for values of a and b any values such that both and their sum is in the range of the computer.

2.5 Starting the Computer

When the computer is turned on, it may be checked for proper operation by the use of a test routine which is provided in a punched tape magazine.

The procedure to turn on and check the computer is:

1. Place the "Test Routine" magazine on the photo-reader. The tape in the magazine must be rewound.

2. Put the Enable, Punch and Compute switches on the typewriter base in the center (off) positions.

3. Turn on the Start switch.

   Wait for the AC meter to read 6.3 volts or 100% and the amber AC light to become bright.

4. Press the Reset button until the red DC lamp lights.

   Wait until the photo-reader light remains off and the green "Ready" lamp lights.

5. Move the Compute switch to GO.

   The number "1" will be typed out. Wait for the display panel neons to remain steady.

6. Type "0 0 0 0 0 0 5(tab)s".

   Wait for the photo-reader light to remain off and the display panel neons to remain steady.

7. Type "0 0 0 0 0 0 6(tab)s".

   Bells ring at repeated intervals to signify successful procedure of each test in the routine.

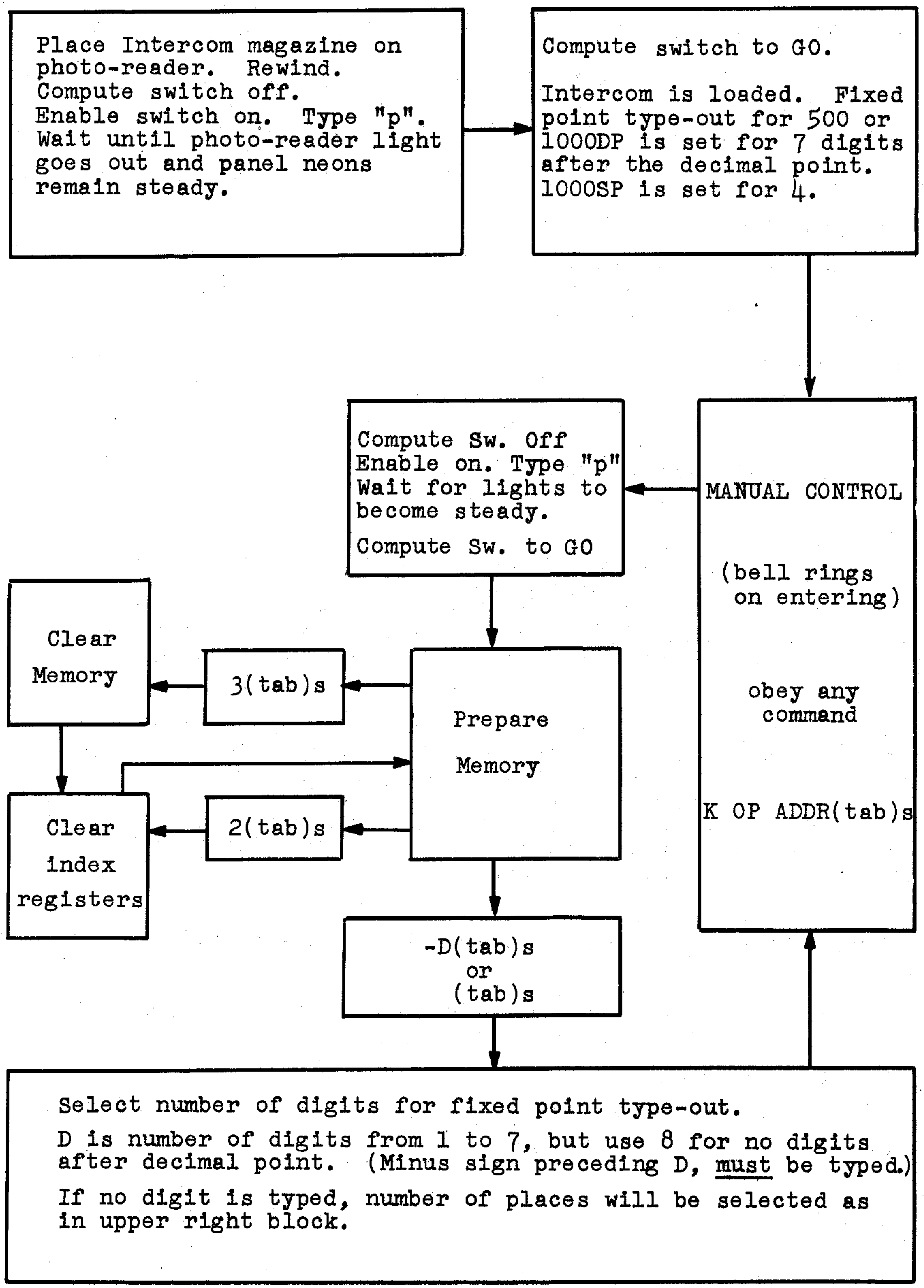
   Proper computer operation is indicated if no type-out occurs before the following is typed out:

```
- 1 1 2 2 3 3 4   4 4 5 5 6 6.7   7 7 8 8 9 9
- u u v v w w x   x x y y z z.0       2 3 4 5
```

8. At completion of the type-out put the Compute switch to the center position, rewind, and remove the "Test Routine" magazine.

2.6 Loading Intercom

An Intercom tape can be loaded into the computer by following the steps below. Memory or index registers may be cleared. The number of digits following the decimal point in fixed point type-out may be selected.



Place Intercom magazine on photo-reader. Rewind. Compute switch off. Enable switch on. Type "p". Wait until photo-reader light goes out and panel neons remain steady.

Compute switch to GO.

Intercom is loaded. Fixed point type-out for 500 or 1000DP is set for 7 digits after the decimal point. 1000SP is set for 4.

MANUAL CONTROL

(bell rings on entering)

obey any command

K OP ADDR(tab)s

Compute Sw. Off Enable on. Type "p" Wait for lights to become steady.

Compute Sw. to GO

Select number of digits for fixed point type-out.

D is number of digits from 1 to 7, but use 8 for no digits after decimal point. (Minus sign preceding D, must be typed.)

If no digit is typed, number of places will be selected as in upper right block.

2.7 Operation of Intercom

Some commands used in storing data and commands are:

	From manual mode	Change of state or sequence
Store commands starting at ADDR	50ADDR(tab)s	050ADDR//(tab)s
Store fixed point data starting at ADDR	51ADDR(tab)s	051ADDR//(tab)s
Store floating point data starting at ADDR	52ADDR(tab)s	052ADDR//(tab)s
Start automatic operation at ADDR	69ADDR(tab)s	069ADDR//(tab)s
Return to manual control		0670000//(tab)s
Punch channel CH on paper tape	39CH00(tab)s	
Read paper tape into channel CH	55CH00(tab)s	
Obey any command	KOPADDR(tab)S	KOPADDR//(tab)s

In all of the above, (tab) is not to be typed. This indicates that the tab key on the typewriter is depressed.

Manual. When the computer is in the manual mode, which we will refer to simply as "manual," any command may be executed by typing KOPADDR(tab)s. For example, the program of 2.4 could be executed by storing the data as indicated below and then typing the commands in order as they are given in the program. However, most programs are executed from the Automatic mode.

Automatic Operation. Automatic operation may be started by the command given at the beginning of this section. Automatic operation will continue until a halt command is encountered in the program, or the Compute switch is put to the OFF position, or an error is encountered. If the Compute switch is put to the OFF position and then returned to the GO position, automatic operation will be resumed. This is useful for positioning paper or stopping in a demonstration.

Return to Manual. Return to manual can be accomplished from any state or mode by:

1. Move the Compute switch to BP. WAIT FOR THE DISPLAY NEONS TO REMAIN STEADY! Move the Compute switch to OFF.
2. Hold the Enable switch on and type "scf".
3. Release the Enable switch and put the Compute switch to GO.
4. If bell does not ring, reload Intercom.

2.7 (continued)

Storing Commands. In order to store commands, we must inform the computer that we wish to store commands and where the first one is to be located. For example, if we are in manual and wish to store a command in 0900, we type 500900(tab)s. The computer will type 900 and wait for a command to be typed. If we type 300002(tab)s, the computer will verify .0300002, execute a carriage return, type 901 and wait for the next command, etc. This will continue until we notify the computer to do something else.

Care should be taken in entering commands since the computer will not know what to do if an erroneous command is executed. This may cause loss of Intercom. If an error is detected before tht tab key is depressed, we may type a few zeros, the correct command then (tab)s. Otherwise we should return and correct the command. A command stored in any location replaces the previous contents of that location.

Storing Fixed Point Data. If we are in manual, and wish to store the number 2.37 in location 0950, we would type 510950(tab)s. The computer would type 950 and wait for a number to be entered. We would type 2/37(tab)s. In entering fixed point data, the "slash" key is used in place of the decimal point. (It is possible to wire the typewriter so that the period key is used for the decimal point.) The computer would verify in floating point, typing 51.23700, execute a carriage return and type 951, wait for data to be entered, etc. If double precision data is being entered, the typewriter would type 952 instead of 951. This process would continue until we change to another state or mode or change sequence. The previous contents, if any, of this location would be lost.

Storing Floating Point Data. If we wished to store the above number as a floating point number at the same address, we would type 520950(tab)s. After the computer typed 950, we would type 51237(tab)s. In floating point input, the decimal point is not typed. Otherwise, operation is the same as above.

Change of State or Sequence. If we were storing commands, and the last one had been stored at 0932, the computer would verify, execute a carriage return, type 933, and wait for a command to be typed. If we now wished to change to storing fixed point data starting at 0950, we would type 0510950//(tab)s. The contents of 0933 would not be changed, and the typewriter would type 950, wait for data to be typed, etc., and would continue in this state until another change was called for.

2.8 Storing and Executing a Program

The following sequence of instructions will store and execute the program of 2.4. It is assumed that the computer is in manual at the start. Numbers typed by the computer are underlined. We will take a = 2.37 and b = 15.21.

2.8 (continued)

```
          500900  s   900  300002  s   .0300002
           901  420952  s  .0420952
           902  430952  s  .0430953
           903  332101  s  .0332101
           904  670000  s  .0670000
           905  0510952//  s  952  2/37  s  51.23700
           953  15/21  s  52.15210
           954  0690900//  s

                17.5800
```

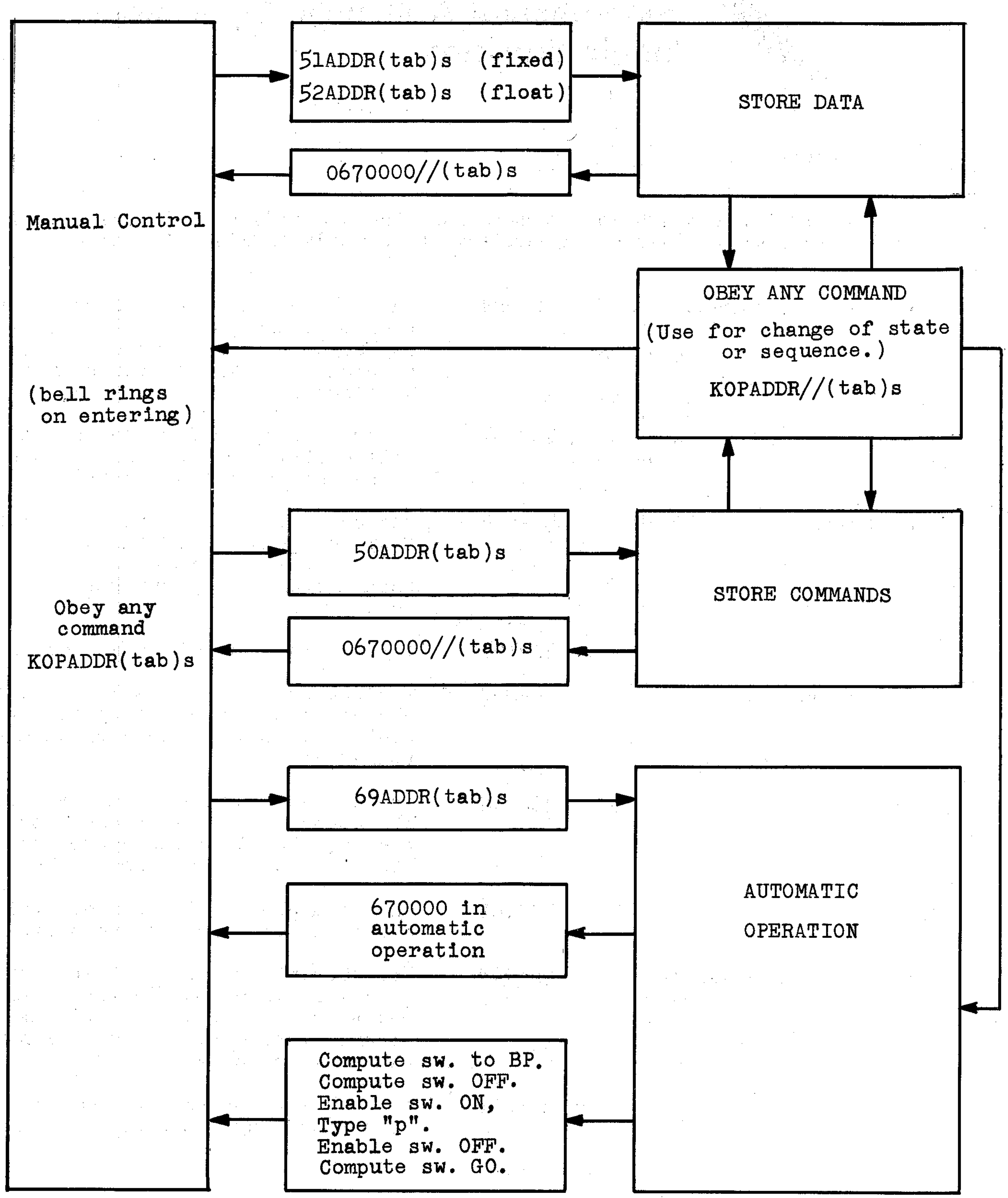
The same program, after data had been stored, executed from manual would look like this:

```
           420952  s  430953  s  332101  s    17.5800
```

Numbers or symbols underlined are typed by the computer, while those not underlined are typed by the operator.

2.8 (continued)

Changes of mode or state.



Chapter 3

Fundamental Arithmetic Operations Single Precision

3.1 Commands

In this and subsequent sections, the commands used in the example to follow will be explained. A summary of commands is given in the Appendix.

Clear and subtract	40ADDR	The contents of ADDR are subtracted from zero and put into the accumulator, replacing the previous contents of the accumulator. The contents of ADDR are unchanged.
	402101	After this operation, the accumulator will contain the negative of the value it had prior to the operation.
Subtract	41ADDR	The contents of ADDR will be subtracted from the contents of the accumulator and the result placed in the accumulator. The contents of ADDR are unchanged.
	412101	The accumulator will now contain zero.
Clear and add	42ADDR	The contents of ADDR will be placed in the accumulator replacing its previous contents. The contents of ADDR are unchanged.
Add	43ADDR	The contents of ADDR will be added to the contents of the accumulator and the sum placed in the accumulator. The contents of ADDR are unchanged.
	432101	The accumulator will contain twice the value it had prior to the operation.
Clear and add absolute value	45ADDR	The absolute value of the contents of ADDR will be placed in the accumulator, replacing the previous contents of the accumulator. The contents of ADDR will be unchanged.
	452101	The contents of the accumulator will be replaced by its absolute value.

3.1 (continued)

Store	49ADDR	The contents of the accumulator will be stored in ADDR replacing the previous contents of ADDR. The contents of the accumulator will be unchanged.
Position typewrite paper	30TBCR	The last four digits of this command are not a location. If CR is a number other than zero, the typewriter will execute CR carriage returns, that is return carriage to left margin and roll up CR spaces, and then will execute TB tab spaces. Tabs may be set as on any typewriter. If ALL tabs are set, two spaces will be given for each tab. If CR is zero, the carriage will move TB spaces on the same line.
Type fixed point data and tab	33ADDR	The contents of ADDR will be typed out in fixed point and the typewriter will stop on this line.
	332101	Type contents of accumulator as above.
Halt	670000	Last four digits always zero. Automatic computation will stop and control returned to manual.

Coding Sheets. Programs are usually written on forms called coding sheets. On these sheets, we indicate the location of the commands, the commands, the contents of the accumulator after each operation and notes to show what is done at each step. Neatness and orderliness are great helps in writing a good program. We will use the notation: a = (0927) to indicate that the number a is stored in location 0927.

NOTES	LOCATION	K	OP	ADDRESS	ACCUMULATOR

3.2 Example of Addition and Subtraction

The following program for Intercom 1000 SP will calculate and type:
$x = a - 3b - 2|c|$, where a = (0951), b = (0952), c = (0953).
(0954) is used for temporary storage.

Problem: $x = a - 3b - 2|c|$

NOTES	LOCATION	K	OP	ADDRESS		ACCUMULATOR
position paper	0900		30	00	02	?
clear and add b = (0952)	1		42	09	52	b
add b	2		43	21	01	2b
add b	3		43	09	52	3b
store 3b = (0954)	4		49	09	54	3b
clear and add $\|c\|$, c = (0953)	5		45	09	53	$\|c\|$
add $\|c\|$	6		43	21	01	$2\|c\|$
clear and subtract $2\|c\|$	7		40	21	01	$-2\|c\|$
subtract 3b	8		41	09	54	$-3b - 2\|c\|$
add a	9		43	09	51	x
type x	10		33	21	01	x
halt	11		67	00	00	x

Problems:

Write a program to calculate and type:

1. $x = a + 2b - 3c + 4|d|$
2. $x = 2a + 4b - 2c = 2(a + 2b - c)$
3. $x = a + |b - c|$
4. $x = a + b - |c - d|$
5. $x = - a - 2b - 2c = -(a + 2b + 2c)$
6. $x = a - 2(b+|c|) - b$. Compare with example above.

3.3 Commands for Multiplication and Division

Multiply	44ADDR	The contents of ADDR are multiplied by the contents of the accumulator and the product is placed in the accumulator replacing the previous contents of the accumulator. The contents of ADDR are unchanged.
	442101	The contents of the accumulator will be replaced by the square of its previous contents.
Divide	48ADDR	The contents of the accumulator will be divided by the contents of ADDR and the quotient placed in the accumulator. The contents of ADDR will not be changed. An error will result if the contents of ADDR is zero.
	482101	If the contents of the accumulator is not zero, the accumulator will contain unity after this operation.
Inverse divide	47ADDR	The contents of ADDR will be divided by the contents of the accumulator and the quotient placed in the accumulator. The contents of ADDR will not be changed. An error will result if the contents of the accumulator is zero.

Commands for output.

Type tabulating number	31TABL	TABL will be typed out and the typewriter will stop on this line. The last four digits of the command are not an address, but the actual number to be typed. Leading zeros are not typed.
Type fixed point data and tab	33ADDR	Fixed point number will be typed and typewriter will stop on this line.
Type fixed point data and return carriage	38ADDR	Fixed point number will be typed and typewriter will execute carriage return. (In Intercom 1000SP, when fixed point limit is exceeded, carriage does not return.)
Type floating point data and tab	32ADDR	Floating point number will be typed and typewriter will stop on this line.
Type floating point number and return carriage	34ADDR	Floating point number will be typed and typewriter will execute carriage return.

3.4 Example of Multiplication and Division

The following program for Intercom 1000SP will calculate:

$x = \frac{ac}{b} + \frac{b}{2a}$, where a = (1251), b = (1252), c = (1253). (1054) is used for temporary storage, a, b, c will be typed on one line and ac/b, b/2a, x on the next.

Problem: $x = \frac{ac}{b} + \frac{b}{2a}$

NOTES	LOCATION	K	OP	ADDRESS		ACCUMULATOR
position paper	1200		30	00	05	?
clear and add a = (1251)	1		42	12	51	a
type a and tab	2		33	21	01	a
add a	3		43	21	01	2a
inverse divide by b = (1252)	4		47	12	52	$\frac{b}{2a}$
type b and tab	5		33	12	52	$\frac{b}{2a}$
store b/2a = (1054)	6		49	10	54	$\frac{b}{2a}$
clear and add a	7		42	12	51	a
type c and return carriage	8		38	12	53	a
multiply by c = (1253)	9		44	12	53	ac
divide by b	10		48	12	52	ac/b
type ac/b	11		33	21	01	ac/b
type b/2a	12		33	10	54	ac/b
add b/2a	13		43	10	54	x
type x	14		38	21	01	x
halt	15		67	00	00	x

Problems: Write a program which will calculate and type:

1. $x = b^2 - 4ac$
2. $x = (a - 2bc)/3a$
3. $x = \frac{a}{b} - \frac{b}{a} = \frac{a^2 - b^2}{ab}$
4. $x = \frac{a - b}{a + b}$
5. $x = \frac{a^2 + b^2}{2c}$

3.5 Intercom 500

Intercom 500 is a single precision Intercom similar to, but, in general, faster than 1000SP. Speed will be increased if the following rules are observed.

1. Locate commands to be executed in word positions 20 to 43 of any channel.
2. Locate data in word positions 71 to 78 of any channel.
3. Use 2173 for address of the accumulator.

3.6 Commands for Input, Transfer and Halt

Gate for command	50ADDR	When this command is executed from automatic operation, the carriage will return and the computer will type ADDR and wait for a command to be entered. The command will be stored in ADDR, which should not be the same as the location of the command. After verification, the next command in sequence is obeyed. Note that this is the same operation code that was used from manual to store commands. It is usually desirable to follow this command with 300001 so that any subsequent output will start at the left margin.
Gate for fixed point data	51ADDR	When this command is executed from automatic operation, the carriage will return and the computer will type ADDR and wait for input of fixed point data. After verification, the next command will be executed. The value typed will replace the previous contents of ADDR. Note that this same operation code was used from manual to store data. It is usually desirable to follow this command with 300001 so that any subsequent output will start at the left margin.
Gate for floating point data	52ADDR	Same as above for floating point data.
Transfer	29ADDR	Take next command to be executed from location ADDR.
Breakpoint halt	680000	Last four digits always zero. The computer will stop. If the compute switch is moved to the center position and back to GO, the next command in sequence will be executed. This command is useful at the beginning of a program to position paper manually. It is useful in the middle of a program to check operation to that point.
Ring bell	630000	Last four digits always zero. In a long program this command is useful for keeping the operator awake.

3.7 Evaluation of a Polynomial

The following program for Intercom 500 will gate for x, then calculate:

$y = a + bx + cx^2 + dx^3 = a + x[b + x(c + dx)]$, type y, then return for type-in of another value of x. Note that the "nested form" requires fewer operations than the original form.

NOTES	LOCATION	K	OP	ADDRESS	ACCUMULATOR
transfer to 1020	1000		29	10 20	?
position paper	1020		30	00 04	?
gate for x = (1075)	21		51	10 75	?
clear and add x	22		42	10 75	x
mult. by d = (1074)	23		44	10 74	dx
add c = (1073)	24		43	10 73	c + dx
mult. by x	25		44	10 75	x(c + dx)
add b = (1072)	26		43	10 72	b + x(c + dx)
mult. by x	27		44	10 75	x[b + x(c + dx)]
add a = (1071)	28		43	10 71	y
type y	29		38	21 73	y
transfer to 1021	30		29	10 21	y

The location of the transfer command in 1000 is a precautionary measure since there is a tendency to start at the beginning of a channel. Automatic operation can be started at either 1000 or 1020.

Problems:

1. Use program of the above example to evaluate $y = 2x^3 - 3x^2 + x - 5$ for x = -3, 0, 2, -1, 5.
2. Use 1 to find root between 1 and 2 to three decimal places.
3. Write a program to evaluate a fourth degree polynomial.
4. Write a program to evaluate $y = ax + bx^3 + cx^5$.
5. Write a program to evaluate $y = a + bx^2 + cx^4$.
6. Write a program to evaluate
$$y = ax + b + cx^{-1} = (ax^2 + bx + c)/x,\ x \neq 0$$

3.8 Solution of Equations

The following program for 1000SP will solve the equations:

$$a_1x + b_1y = c_1$$
$$a_2x + b_2y = c_2, \quad b_1 \neq 0$$

for x and y, using $x = N/D$, $N = (b_2c_1/b_1)-c_2$, $D = (a_1b_2/b_1)-a_2$, $y = (c_1 - a_1x)/b_1$. Results will be typed: 1 x
2 y

Problem: $a_1x + b_1y = c_1$, $a_2x + b_2y = c_2$

NOTES	LOCATION	K	OP	ADDRESS		ACCUMULATOR
position paper	1300		30	00	02	?
clear and add b_2 = (1404)	1		42	14	04	b_2
mult. by c_1 = (1402)	2		44	14	02	b_2c_1
div. by b_1 = (1401)	3		48	14	01	b_2c_1/b_1
sub. c_2 = (1405)	4		41	14	05	N
store N = (1406)	5		49	14	06	N
clear and add b_2	6		42	14	04	b_2
mult. by a_1 = (1400)	7		44	14	00	a_1b_2
div. by b_1	8		48	14	01	a_1b_2/b_1
sub. a_2 = (1403)	9		41	14	03	D
inverse div. by N	10		47	14	06	x
type 1	11		31	00	01	x
type x and ret.	12		38	21	01	x
mult. by a_1	13		44	14	00	a_1x
clear and sub. a_1x	14		40	21	01	$-a_1x$
add c_1	15		43	14	02	$c_1 - a_1x$
div. by b_1	16		48	14	01	y
type 2	17		31	00	02	y
type y	18		38	21	01	y
halt	19		67	00	00	y

3.8 (continued)

Problems:

1. Write a program to solve two equations in two unknowns using $x = (b_2c_1 - b_1c_2)/(a_1b_2 - a_2b_1)$ instead of the equation of above example.

2. Write a program to solve the equations: $\frac{a_1}{x} + \frac{b_1}{y} = c_1$, $\frac{a_2}{x} + \frac{b_2}{y} = c_2$

3. Write a program to solve the equations: $\frac{a_1}{x} + b_1y = c_1$, $\frac{a_2}{x} + b_2y = c_2$

4. Write a program to solve 3 equations in three unknowns.

5. Solve equations: $a_ix^{-1} + b_iy^{-1} + c_iz^{-1} = d_i$, i = 1, 2, 3.

3.9 Similar Programs

Similar programs can be executed by writing a program once, then changing it by storing different commands at a given address depending on the entry location. In the following program, there are three different entries to the program. If one would imagine that the program represented here by the commands in locations 0900 to 0904 is a long one, he will see that considerable effort can be saved by writing a program once and then changing it by storing a command in a given location.

In this section we illustrate a technique whereby commands may be moved from one memory location to another. This is accomplished by placing the command in the accumulator by means of a 42 operation code and then storing the contents of the accumulator in a given location by means of a 49 operation code. In the following example, this is done at 0905 and 0906 and again at 0908 and 0909. Arithmetic operation codes other than 42 and 49 should not be used with commands in Intercom. In Section 4.5 we will use this same technique in double precision where a little more care must be exercised.

Assume that x = (0915), y = (0916). If we enter at 0905, x + y will be typed out. If we enter at 0908, xy will be typed out. But if we enter at 0913, the computer will type 902, halt and wait for a command to be typed.

If we then type:	The computer will verify and then the following will be typed out:
410916(tab)s	$x - y$
480916(tab)s	x/y
470916(tab)s	y/x
440915(tab)s	x^2

3.9 (continued)

Example of similar programs by alternative entries

NOTES	LOCATION	K	OP	ADDRESS		ACCUMULATOR
	0900		30	00	03	?
	1		42	09	15	x
	2		00	00	00	depends on entry location
	3		33	21	01	"
	4		67	00	00	"
entry for x + y	5		42	09	11	"
	6		49	09	02	
	7		29	09	00	
entry for xy	8		42	09	12	
	9		49	09	02	
	10		29	09	00	
	11		43	09	16	
	12		44	09	16	
entry for "other"	13		50	09	02	
	14		29	09	00	

Problems:

Write a program which will:

1. Type either x^2 or 2x depending on which of two entry locations are selected.
2. Type x, x^2 or x^3 depending on entry location.
3. Gate for x, gate for operation, gate for y, then type result.
4. Type x, then y or type x + y, depending on entrance.
5. Type 1, 2, 3 on a line or type 1, 2, 3 one under the other.

Chapter 4

Logical Operations
Double Precision

4.1 Double Precision

In Section 2.3 we discussed the form of data used in the Intercom systems and stated the distinction between single and double precision. In Chapter 3 all programs were written in single precision but in this chapter some will be in double precision. The student must then remember that only even-numbered addresses may be used for data storage, since the double precision system automatically uses the next odd-numbered location for part of the 12 significant figures. If an odd numbered address is designated for data, the computer will type 5 periods and ring 5 bells, indicating an error. Command storage is unchanged and the machine obeys commands in sequence just as in single precision. All the operation codes explained so far are valid. Finally, refer to the table of Section 2.1 and note the locations available for programming and the address of the accumulator in each system. With each example we will state which system is being used.

4.2 Conditional Transfer Commands. Flow Charts

In this and subsequent sections we will study ways of using the decision-making ability of the computer. The idea is simple. The sequence of commands obeyed by the machine will depend on the outcome of certain numerical tests it will make using data we provide. Let us illustrate this first in a rather artificial example. We will store two numbers in memory and then cause the machine to compare them for algebraic size. This will demonstrate the basic ideas and then later sections will show their practical use. The following new commands will be used:

Transfer on positive or zero	20ADDR	If the accumulator contains zero or a positive number the next command obeyed will be taken from location ADDR. If the accumulator contains a negative number the next command obeyed will be the one following in normal sequence.
Transfer on negative	22ADDR	If the accumulator contains a negative number the next command obeyed will be taken from location ADDR. If the accumulator contains zero or a positive number the next command obeyed will be the one following in normal sequence.

4.2 (continued)

Transfer on zero	23ADDR	If the accumulator contains zero the next command obeyed will be taken from location ADDR. If the accumulator contains a non-zero number the next command will be taken in normal sequence.

In using these commands remember that the machine is simple-minded. If it does transfer out of normal sequence it will then continue obeying instructions located in sequence after the one to which it transferred. For example if the machine encounters in location 1431 the instruction 221766 and if the accumulator contains a negative number, the next instruction obeyed will be the one in location 1766, and the one after that will be the one in 1767, etc., until another transfer is encountered. It is, of course, up to the programmer to see that meaningful commands are located in all necessary locations.

There is a useful technique in writing programs containing several conditional transfer instructions. Remeber that as soon as you write 20, 22, or 23 in the OP column there are two possible locations for the next instruction which will be obeyed. Both of these alternative paths or branches must be programmed. Usually it is best to leave blank the address of the conditional transfer instruction and continue writing the instructions to be obeyed in normal sequence. When that branch is finished select an available address for the first instruction of the other branch, write it in the blank address part of the conditional transfer and in the location column at a fresh space on the coding sheet, and proceed to program the instructions to be obeyed should control be transferred.

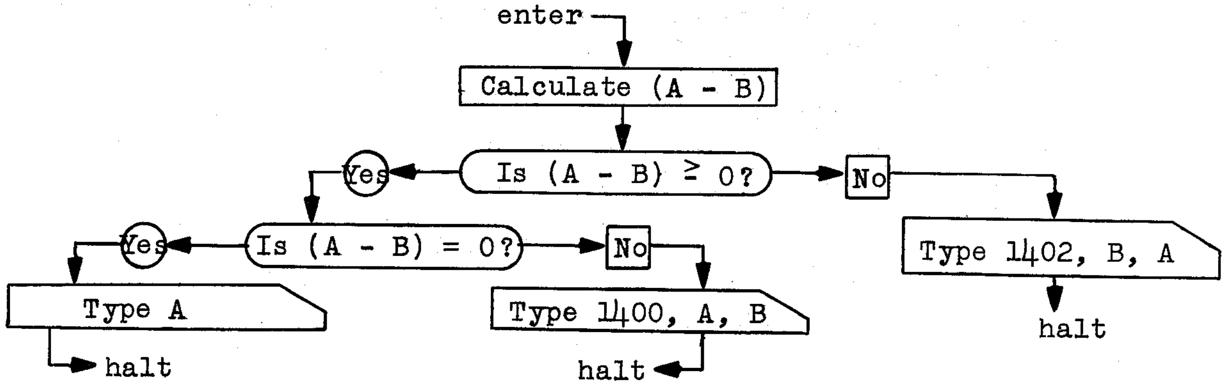
Example. Tests for positive, negative, and zero.

This program for Intercom 1000 Double Precision will test the two numbers A = (1400) and B = (1402) and will do the following:

If A > B, type location of A, type A, type B.
If A = B, type their common value.
If A < B, type location of B, type B, type A.

Before writing the program let us outline the steps involved. This is conveniently done in a form called a flow chart showing at least each major step as a separate block and their sequence by means of arrows connecting these blocks. After this example we will say a few more things about flow charts.

FLOW CHART



4.2 (continued)

Problem: Determine which of two numbers is greater.

NOTES	LOCATION	K	OP	ADDRESS		ACCUMULATOR
position paper	1500		30	00	02	?
clear and add A = (1400)	1		42	14	00	A
subtract B	2		41	14	02	A - B
transfer if accum ≧ 0	3		20	15	08	A - B
type 1402	4		31	14	02	A - B < 0
type B	5		33	14	02	A - B < 0
type A	6		38	14	00	A - B < 0
halt	7		67	00	00	A - B < 0
transfer if accum = 0	8		23	15	13	A - B ≧ 0
type 1400	9		31	14	00	A - B > 0
type A	10		33	14	00	A - B > 0
type B	11		38	14	02	A - B > 0
halt	12		67	00	00	A - B > 0
type A	13		38	14	00	A - B = 0
halt	14		67	00	00	A - B = 0

Problems:

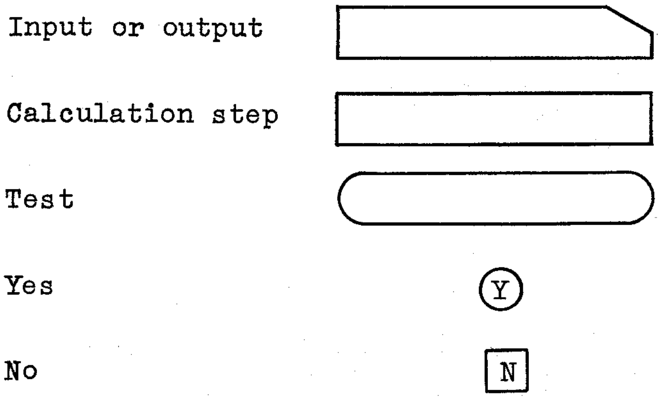
1. Three unequal numbers are in consecutive locations in channel 14; write a program to type the largest, then next largest, then smallest.

2. As 1, but type location of largest number and then the largest number.

4.3 A Further Word about Flow Charts

While it is true that many problems can be programmed without the aid of a flow chart, experienced programmers use them regularly because they afford a concise, clear analysis of the way the computer will tackle the problem. Moreover, the coding of problems involving many transfers is enormously simplified once a good flow chart has been drawn. Generally, it will be found that complex problems requiring considerable thought before coding are best analyzed in flow chart form. The student should make the effort to learn the techniques of flow charting even if he feels the problems do not always require it.

Throughout this book the flow chart symbols will be standardized as follows:



A good flow chart will always clearly show the beginning or entrance for the problem and the halt(s) or exit(s) from the problem. Also, the question involved in a test should be precisely stated such that the answer is either yes or no. Further pointers will be given in subsequent sections.

4.4 The Loop Concept. Data Modification

In digital computing a loop is a set of instructions which are obeyed repeatedly, probably with certain alterations before each repetition. Normally this set forms a part of a larger program. The term data modification means that the data being used in a given problem will be changed or generated by the computer during automatic computation. These important ideas will be illustrated in the next two examples.

Each example is accompanied by an essential aid to good programming called a storage allocation chart. This chart is used to list quantities and their locations which are relevant to the program. This list is typically not completed prior to coding but rather will be used to note down quantities needed as coding progresses. Thus, if at some point in coding we find we need a constant of 10, we would decide on a location for it and note this down on the list. Thus the storage allocation chart is invaluable when the time comes to place the required numbers in the correct locations prior to automatic computation. Notice that we distinguish carefully between constants and variables. In this connection these familiar words refer to the contents of a memory location. That is, if a certain location, say 1650, contains a 5 at the beginning of computation and the 5 remains in that

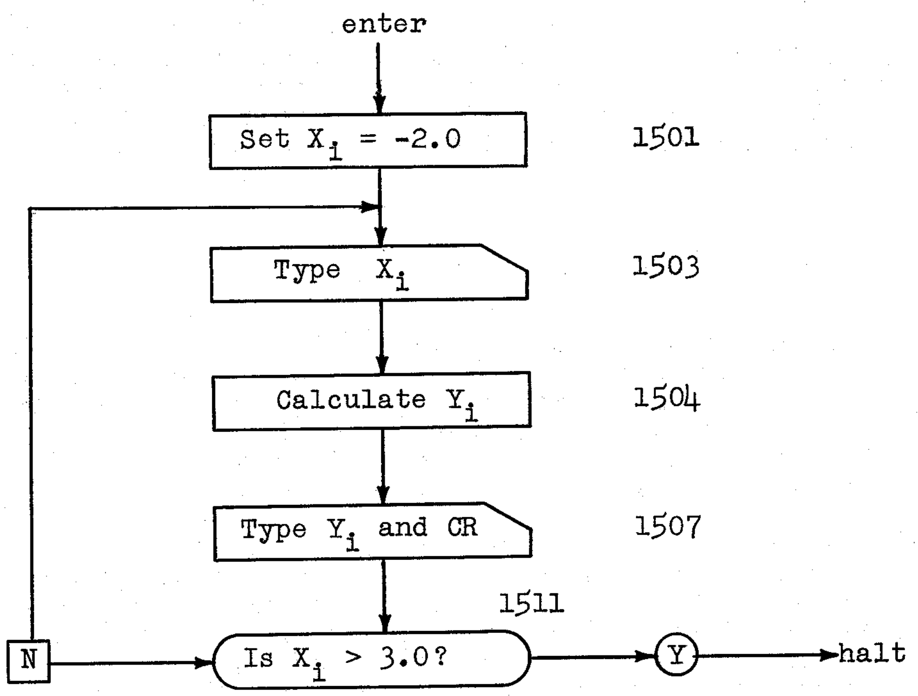
4.4 (continued)

location throughout the entire process of automatic computation, 1650 and 5 will be listed in the constants column. But if the 5 is to be replaced by another quantity during computation, then 1650 must be listed as a variable. This points up the fact that the contents of the locations listed as variables must be set up properly prior to each use of the program. Otherwise, incorrect results may be obtained. This initial setup should be done in the program so that the computer will itself automatically set up these values. The student should carefully observe this in the examples of this section.

In studying the examples, if the flow charts are not at first clear, study the programs and then return to the charts. Some of the notation is new and it will take a little time to understand it. Strive to think sequentially in the way that the machine obeys its instructions.

Example 1. Using Intercom 1000 DP, evaluate y = ax + b for x = -2.0, -1.9, -1.8, ..., 3.0. Type out rows of the form: x_i y_i (This is really a rather general problem in that we evaluate a linear function on a specified interval of the independent variable where the variable takes values in arithmetic sequence. a and b may be any desired values.)

FLOW CHART



4.4 (continued)

Problem: $y = ax + b$

NOTES	LOCATION	K	OP	ADDRESS		ACCUMULATOR
Position paper	1500		30	00	02	?
Set x_i	1		42	15	98	-2.0
	2		49	15	80	-2.0
Type x_i	3		33	15	80	
Calculate y_i	4		42	15	80	x_i
	5		44	15	96	ax_i
	6		43	15	94	$ax_i + b$
Type y_i	7		38	21	00	"
Increment x_i	8		42	15	80	x_i
	9		43	15	92	x_{i+1}
	10		49	15	80	$x_{i+1} \rightarrow x_i$
Test x_i	11		41	15	90	$x_i - 3.05$
	12		22	15	03	"
halt	13		67	00	00	"

STORAGE ALLOCATION CHART

Constants			Variables		
Loc.	Symbol	Quantity	Loc.	Symbol	Starting value
1598		-2.0	1580	x_i	-2.0
1596	a				
1594	b				
1592	Δx	0.1			
1590		3.05			

Some explanation of the example is in order. Toward this end the blocks of the flow chart have been numbered with the command location of the program which begins the manipulation in the block. This cross referencing is, incidentally, a very useful scheme as one often has to refer back and forth from flow chart to program.

Since we plan to have the sequence of x's calculated by the machine, we will establish an "x cell" which is nothing more than a specific location in the memory in which we can repeatedly place these values during computation. Thus this location contains a variable. In block 1501 we show that this variable must be set initially at -2.0.

4.4 (continued)

Blocks 1503 through 1511 constitute the loop. It is here that the computer will cycle repeatedly through the same set of instructions. If the symbol x_i is given its initial value -2.0, it will be seen that during the first run through we are calculating y_1. In block 1507, C.R. stands for carriage return.

Block 1508 indicates that we must *increment* x_i. That is, we must take the current x_i (the first time $x_i = -2.0$ but not thereafter), add $\Delta x = 0.1$ and store the result back in the x_i cell. The symbolism $x_{i+1} \rightarrow x_i$ is best read as "x_{i+1} becomes x_i." Thus we calculate the new value to be used in the computation and place it where the previously used value was stored.

In block 1511 we test to see if x_i (which, remember, is the just incremented value) exceeds 3.0, the last value with which we wish to calculate. If it does, we halt computation. If $x_i < 3.0$ we must return to the first step in the loop and run through it again, of course with the new x_i. Note carefully that at the test if $x_i = 3.0$ we are *not* done for we have just calculated with $x_i = 2.9$. The student should follow these details on the flow chart and the program until he sees clearly the development of the problem.

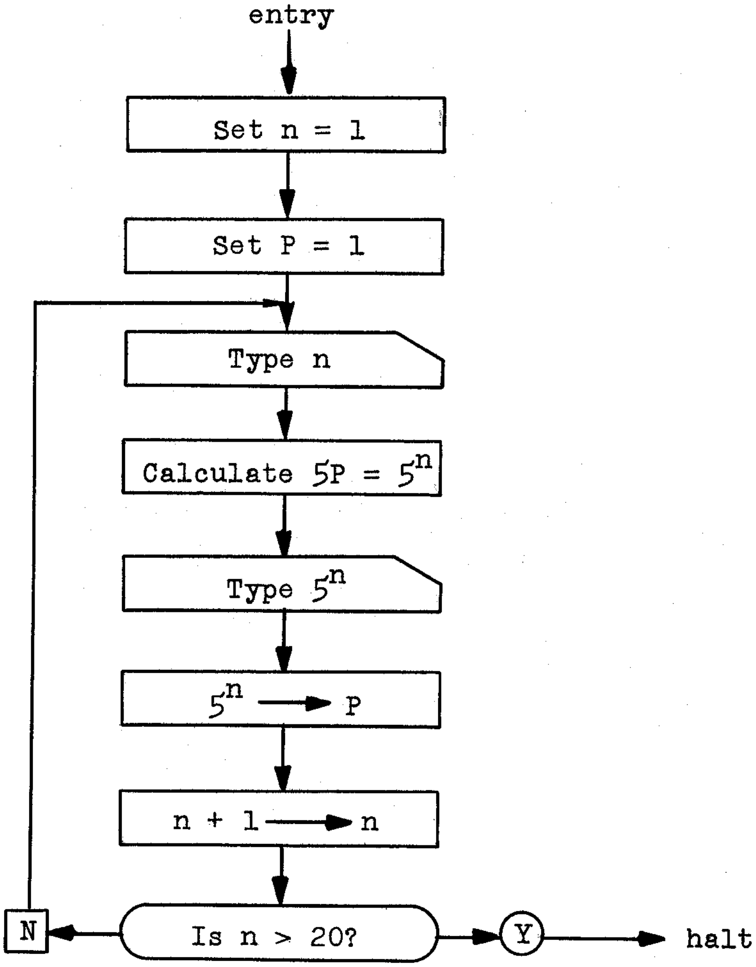
It will be noticed that in making the test we subtracted 3.05. This value is called the *testing constant* and care must be exercised in choosing it properly. A good method is to follow through the loop mentally using the final desired x_i. Notice that if we had used 3.0 as the testing constant the machine would have halted too soon. For example, when $x_i = 3.1$ in this problem there may be a small error due to the fact that the computer calculates and rounds off in base two. Thus a testing constant of 3.1 might not give zero when subtracted from $x_i = 3.1$ as computed by the machine, and the computer would take one more run through the loop.

Finally, it is well to note that a loop generally involves the following logical sections: *Set-up*, *Compute*, *Increment*, and *Test*. The last three sections need not be in the order given but this order possesses certain advantages and in general we will use it. In particular, it fits the use of index registers to be presented in Chapter 6.

4.4 (continued)

Example 2. Using Intercom 1000DP calculate 5^n for n = 1, 2, ..., 20. Type rows of the form: n 5^n

FLOW CHART



4.4 (continued)

Problem: n, 5^n

NOTES	LOCATION	K	OP	ADDRESS		ACCUMULATOR
Position paper	0900		30	00	02	?
	1		42	09	30	1
Set P	2		49	09	50	1
Set n	3		49	09	52	1
Type n	4		33	09	52	
	5		42	09	50	P
Calculate 5P	6		44	09	32	5P = 5^n
Type 5^n	7		38	21	00	"
$5^n \rightarrow P$	8		49	09	50	"
	9		42	09	52	n
	10		43	09	30	n + 1
	11		49	09	52	n + 1 → n
	12		41	09	34	n - 21
Test n	13		22	09	04	
	14		67	00	00	

STORAGE ALLOCATION CHART

Constants			Variables		
Loc.	Symbol	Contents	Loc.	Symbol	Starting value
0930		1	0950	P	1
0932		5	0952	n	1
0934		21			

This program illustrates the use of a product cell in which we store the next higher power of 5 on each run through the loop. In the flow chart this cell is symbolized by P. Notice the importance of setting P = (0950) to the value one initially, so that the first multiplication gives 5 times 1 = 5. Of course, it is essential to store the calculated power in the "P" cell before transferring back for another run through the loop. This step is shown on the flow chart as $5^n \rightarrow P$.

4.4 (continued)

Similarly, we call cell 0952 in this program a sum cell, for its contents are increased by the addition of one each time around. Thus the machine counts the number of multiplications and types out the correct n. Again it is important to note the necessity of storing back in the n cell the incremented value.

Notice carefully the use of 21 as the testing constant. The student should follow through the loop mentally to see that this is the correct value to use. Incidentally, we are not concerned here with a round-off problem for n is an integral variable. Thus we can safely depend on having zero in the accumulator when the computer subtracts the testing constant of 21 from the n = 21 which was calculated.

The student should pick out in this program the four logical sections of the loop: Set-up, 0900 - 0903; Compute, 0904 - 0907; Increment, 0908 - 0911; Test, 0912 - 0913.

Problems:

1. Rewrite the example of Section 3.7 to evaluate the polynomial for $x = -3.0$ to $+ 3.0$ with $\Delta x = 0.5$. Type out as in example 1 above.

2. Given that $\sin x \approx x - \frac{x^3}{3!} + \frac{x^5}{5!}$ where x is an angle in radian measure, calculate the sines of .1, .2, ···, .9, radians. Type rows of the form: x sin x. (Note that this is only an approximation. The error for any x, however, is less than $\frac{x^7}{7!}$).

3. Write a loop to find $S = \sum_{i=1}^{15} x_i y_i$ where $x_1 = 2$ and $\Delta x = 0.5$, $y_1 = -6$ and $\Delta y = 0.3$.

4. Calculate $A_n = P(1 + r)^n$ for n = 1, 2, ···· Type out rows of the form: n A_n. Gate for input of P, r, and the final value of n.

4.5 Command Modification by Replacement

In Section 3.9 we saw that commands could be moved about in the memory during automatic computation. We may now apply this technique to cause the computer to build its own computation loop. The example, in which we will calculate x + y and x - y, is very artificial in that no sensible programmer would use this scheme if all he had to do was add and subtract two numbers. In large involved programs, however, this method of "reshuffling" commands is sometimes useful. Therefore, we emphasize that our objective here is to demonstrate a particular technique in a simple, uncluttered problem.

The program is written for double precision and it is necessary to recall that two locations are used for data. Thus, when the computer obeys 421464 the contents of 1464 and 1465 are placed in the accumulator. In moving commands, therefore, we must realize that two at a time will be moved. Moreover, any 42 and 49 operations must have even numbered addresses. The student should carefully examine how these requirements have been handled in the example.

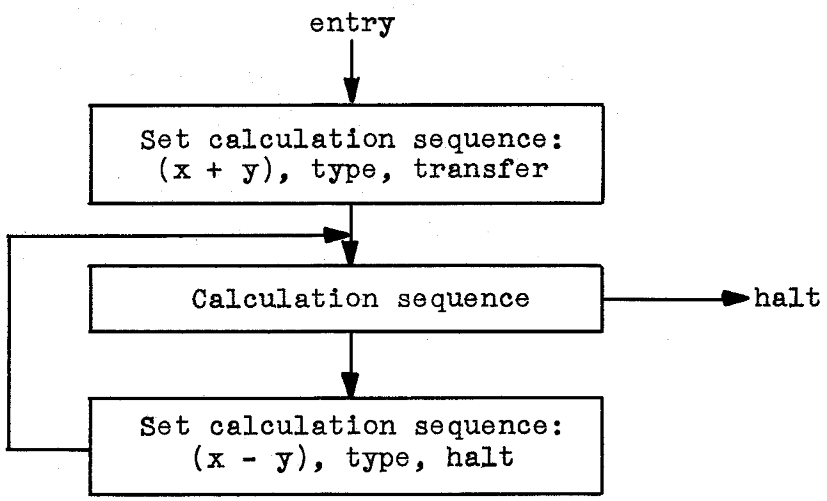
4.5 (continued)

Observe also that commands appear in the storage allocation chart. These are the quantities which will be copied appropriately into locations 1410 through 1413 during automatic computation. They are to be entered manually from the typewriter as commands. Note that it is unnecessary to store manually any commands in 1410 through 1413.

A flow chart precedes the example, although it is probably not necessary in this case and may not even be very informative. Remember that flow charts are convenient tools to be employed by the programmer as he finds necessary.

Example. Intercom 100DP. Given x = (1460), y = (1462); write a computation loop that will be successively altered to type out (1) x + y, then (2) x - y.

FLOW CHART


entry
Set calculation sequence: (x + y), type, transfer
Calculation sequence
halt
Set calculation sequence: (x - y), type, halt

4.5 (continued)

PROGRAM

NOTES	LOCATION	K	OP	ADDRESS		ACCUMULATOR
	1405		30	00	02	?
Set sequence for x + y, transfer	6		42	14	64	x + y sequence
	7		49	14	10	
	8		42	14	68	type and transfer sequence
	9		49	14	12	
Set automatically	10					
	11					
	12					
	13					
Set sequence for x - y, halt	14		42	14	66	x - y sequence
	15		49	14	10	
	16		42	14	70	type and halt sequence
	17		49	14	12	
	18		29	14	10	

STORAGE ALLOCATION CHART

Constants			Variables		
Loc.	Symbol	Value	Loc.	Symbol	Starting value
1460	x				
1462	y				
1464		421460			
1465		431462			
1466		421460			
1467		411462			
1468		382100			
1469		291414			
1470		382100			
1471		670000			

Problems:

1. Write a program in Intercom 1000DP that will build its own computation loop to calculate and type (1) xy, (2) x/y, (3) y/x.

2. Do as in problem 1, for (1) xy, (2) x^2y^2, (3) x^2/y^2.

4.6 Sample Examination

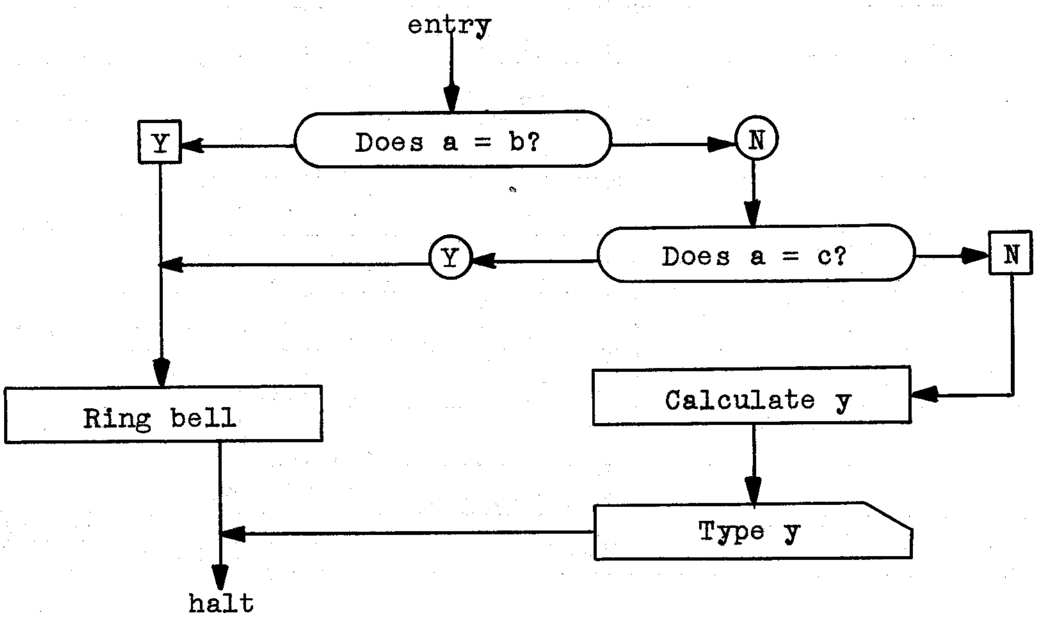
1. Write a program in Intercom 1000SP to solve the following problem:

Three numbers are in storage as follows: a = (1600), b = (1601), c = (1602). If $a \neq b$ and $a \neq c$, calculate $y = \frac{a}{a - b} + \frac{c}{a - c}$, type y, and halt. If a = b or a = c, ring bell, and halt. Use only one halt instruction and place it at the end of your program.

Include a flow chart and storage allocation chart.

SOLUTION:

FLOW CHART



4.6 (continued)

PROGRAM

NOTES	LOCATION	K	OP	ADDRESS		ACCUMULATOR
Position paper	1400		30	00	02	?
	1		42	16	00	a
	2		41	16	01	a - b
transfer if a = b	3		23	14	13	"
	4		47	16	00	a / (a - b)
	5		49	16	03	"
	6		42	16	00	a
	7		41	16	02	a - c
transfer if a = c	8		23	14	13	"
	9		47	16	02	c / (a - c)
	10		43	16	03	y
type y	11		38	21	01	y
	12		29	14	14	y
ring bell	13		63	00	00	
halt	14		67	00	00	

STORAGE ALLOCATION CHART

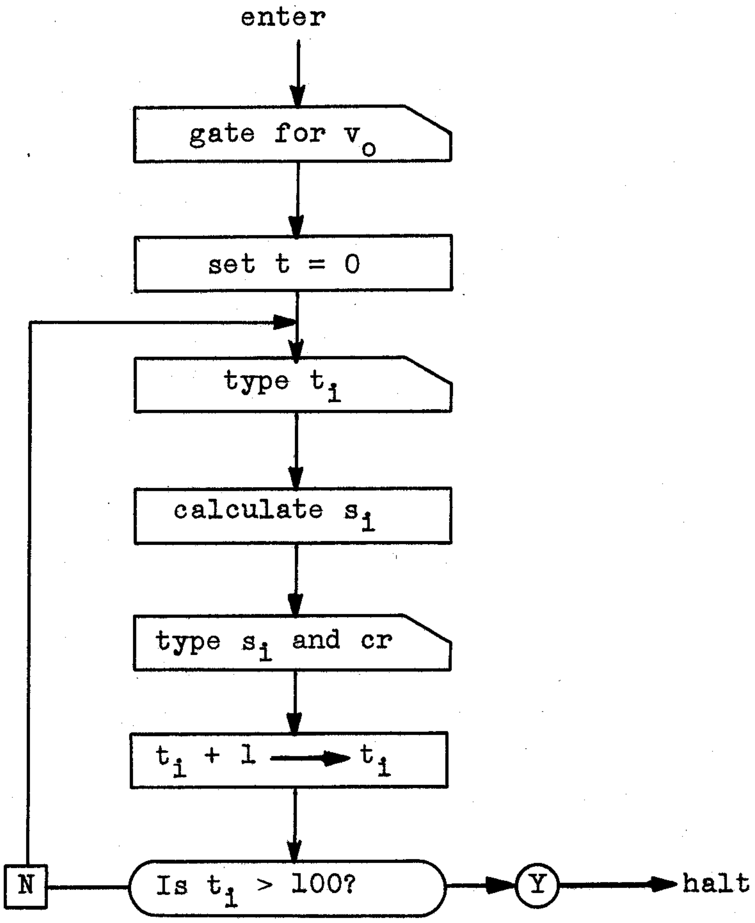
Constants		
Loc.	Symbol	Value
1600	a	
1601	b	
1602	c	
1603		Temporary storage

4.6 (continued)

2. Using Intercom 1000DP write a program to calculate $s = v_o t + \frac{1}{2}gt^2$ for $t = 0, 1, 2, \cdots, 100$. Gate for v_o. Type rows of the form: t_i s_i

SOLUTION:

FLOW CHART



4.6 (continued)

Problem: $s = v_o t + \frac{1}{2}gt^2$

NOTES	LOCATION	K	OP	ADDRESS		ACCUMULATOR
Gate for v_o	1716		51	17	94	?
Position paper	7		30	00	02	?
Generate zero	8		41	21	00	0
Set $t_i = 0$	9		49	17	98	0
Type t_i	20		33	17	98	
	1		42	17	98	t_i
	2		44	17	96	$\frac{1}{2}gt_i$
Calculate t_i	3		43	17	94	$v_o + \frac{1}{2}gt_i$
	4		44	17	98	$v_o t_i + \frac{1}{2}gt_i{}^2$
Type s_i	5		38	21	00	"
	6		42	17	98	$t_i + 1$
Increment t_i	7		43	17	92	$t_i + 1 \rightarrow t_i$
	8		49	17	98	$t_i - k$
	9		41	17	90	"
Test t_i	30		22	17	20	
Halt	31		67	00	00	

STORAGE ALLOCATION CHART

Constants			Variables		
Loc.	Symbol	Value	Loc.	Symbol	Starting value
1796	g/2	16	1798	t_i	0
1794	v_o				
1792		1			
1790	k	101			

Chapter 5 Debugging

5.1 Basic Principles

It is almost inevitable that long complicated problems will not work the first time they are run, and frequently enough even the short simple ones fail to function properly at first. There are many types of errors that can creep into a program and the process of finding them is called "debugging."

Perhaps the first major rule in avoiding errors on the G-15 is to keep an eye on the verifications typed by the computer when storing commands and data. These are true verifications in that the machine stores the information and then reads and types what it stored. It is possible to type one thing and store another, though this sort of machine error is extremely rare. Sometimes by accident two typewriter keys may be hit almost simultaneously and the character that prints will not be the one that is stored. This will show up in the verification. On the whole, however, the operator will find that if he is careful and accurate he will succeed in entering his commands and data as written. The trouble then is that he may not have written them correctly.

Upon starting automatic computation all sorts of things can happen, including the happy possibility that everything works fine. But it can be that the program will run and obtain wrong answers. This, of course, presupposes that the correct answers are known. If they are not, as is generally the case, a trial run should be made with simple numbers so that the machine answer(s) can be checked by hand. There are several things that can be done in the case of wrong answers. First, one should proofread the verifications of the type-in and make sure that all necessary data have been entered. Then, if the problem is not too long, one can follow it through step-by-step writing down exactly what the accumulator contains after each instruction. If the trouble is still not found it will be necessary to employ the techniques to be discussed in Sections 5.3 and 5.4.

A more frequent difficulty upon starting automatic computation is that the instructions are so wrong that the machine does not finish the program. For example, control may be transferred improperly so that the computer starts taking commands from locations not used by the programmer. This may lead to an error indication (see Section 5.2) or, more likely, it will destroy a portion of the Intercom program. Then the Intercom tape must be re-entered before anything can be done.

If all this seems depressing to the reader he will find that, while debugging is a major problem, experience on the machine will go far to lighten the burden. Moreover, there are techniques built into the Intercom systems which are extremely helpful in diagnosing errors. Before outlining these techniques, we suggest that the student cultivate a reasonable degree of humility. It is all too easy to blame errors on the computer. While this may occasionally be the case, it is far more likely that the trouble lies with the programmer and operator.

5.2 Error Indications

During automatic computation, certain errors will cause the machine to halt and, in the Intercom 1000 systems, type five periods interspersed with the ringing of bells. In the Intercom 500 system the computer halts, rings three bells, and types "zOz yOu" to show these errors. After either such indication the operator should put the Compute switch to the center (off) position and then to BP. The computer will then type the address of the command causing the error. The computer may then be returned to manual control as described in Section 2.7. Or, if desired, computation may be resumed by moving the Compute switch back to GO.

Error indications are caused by the following:

1. In all three Intercom systems,

   Any internal computation in which the result exceeds 10^{38}
   Square root of a negative number (see Chapter 7)
   Division by zero
   Log of zero or log of a negative number (see Chapter 7)

2. In Intercom 1000 DP only,

   An arithmetic or data input command with an odd-numbered address
   Incrementing a command until the word position of an address exceeds 99 (see Chapter 6)

5.3 Memory Interrogation

It is frequently useful to determine the contents of specified memory locations. In using the procedures below, THE COMPUTER MUST BE IN MANUAL CONTROL.

Data. Type any of the four type-out instructions, 33, 38, 32, or 34, followed by the address of a location containing data or by the address of the accumulator. Since the computer obeys any instruction typed from manual control, it will type out the contents as specified. If the address of a location containing a command is given there will be a type-out, but it will not be meaningful.

Commands. Type 35 followed by the address of a location containing a command. With Intercom 1000 SP or DP the contents will be typed out in the form KWDOPCH. With Intercom 500 the type-out will be in the standard form KOPCHWD. If the address of a location containing data is given there will be a type-out, but it will not be meaningful.

Hexadecimal Number. Type 37 followed by the address of a location which contains either a command or data. With Intercom 500, the contents of the location will be typed out in hexadecimal form exactly as it is stored.

Control. With all three Intercom systems if the instruction 060000 is typed, the computer will type the address of the last command obeyed in the automatic computation mode.

5.4 Automatic and Manual Tracing

It is possible to execute a stored program one instruction at a time. This is useful if a certain portion of a program is suspected to be causing errors. One can work through this portion, interrogating the memory when desired. The procedure is:

FROM MANUAL CONTROL,

1. Put the Compute switch to BP.
2. Type "69ADDR(tab)s" where ADDR is the location of the first command to be executed.
3. Put the Compute switch to the center position and back to BP.
4. Repeat step 3, for each command in the program. Each time the switch is moved back and forth one command will be executed.

Throughout, the computer is in the automatic operating mode. To execute the remainder of the program without halting between commands, put the Compute switch to GO. To revert to the manual mode, follow the procedure outlined in Section 2.7.

The Intercom systems are equipped with a trace routine which may be used to list commands as they are obeyed and the contents of the accumulator after each command. The output format is:

Location Command Contents of accumulator in floating point

(In Intercom 500 only, the third column is typed only when there is a change in the accumulator contents.)

In essence, then, this routine causes the machine to display its behavior thus allowing the operator to watch the numbers being calculated as well as the transferring of control. The process, however, is fairly slow and should only be used if other efforts to find errors have failed.

An important feature of the trace routine is that it is selective; that is, the operator may specify whether he wishes all commands to be traced or only certain ones. It is emphasized that in either case the computer obeys all commands in the program during tracing. The selection is accomplished in step 6 below by typing either (tab)s or two seven character selectors.

The first selector consists of the digits in their appropriate positions which specify the commands to be traced. The remaining positions are to contain zeros. For example, if one wishes to trace all 49 operations the first selector would be 0490000.

The second selector consists of a combination of z's and zeros. The z's must be typed in the same positions as the digits of the first selector which are to identify the commands to be traced.

Examples:

Commands to be traced	First selector	Second selector
All 49 operations	0490000	0zz0000
All involving channel 14	0001400	000zz00
All 20 operations	0200000	0zz0000
All transfers of control	0200000	0z00000

5.4 (continued)

To Use the Trace Routine. The computer must be in the manual operating mode. The Intercom magazine must be on the photo-reader with the tape at the manual control position.

1. Put the Compute switch in the center (off) position. Hold the Enable switch ON and type p. Release the Enable switch and wait for the photo-reader light to remain off.

2. Put the Compute switch to GO. Wait for the neon indicator lights to remain steady.

3. Type "1(tab)s". Wait for the photo-reader light to remain off.

4. Type "(tab)s". Wait for the bell to ring signaling that the computer has returned to the manual mode.

5. Type "610000(tab)s". Wait for the input-output neons to be in the configuration O●●OO.

6. To list every command, type "(tab)s". To list selected commands, type "FIRST SELECTOR (tab) SECOND SELECTOR (tab)s". The bell will ring signaling that the computer has returned to the manual mode.

7. Type "69ADDR(tab)s" where ADDR is the location of the command at which computation is to begin. The computation, with listing of selected commands, will proceed.

To Terminate Listing.

1. Put the computer in the manual mode (Section 2.7).

2. Type "620000(tab)s". This command terminates listing and the computer is now in the manual operating mode.

It is important to know that in the tracing mode the computer may be halted and computation restarted as many times as desired and it will continue to trace. Furthermore, steps 5 and 6 above may be reused from manual control whenever it is desired to change the selection.

This point becomes clear if one understands that steps 1 - 4 accomplish the *reading-in* of the trace routine while 5 and 6 specify *how* it is to function. Obviously, it is only necessary to read in the routine once.

Finally, with Intercom 1000 *Single Precision* the reading-in of the trace routine destroys the fixed-point input routine so that until the Intercom program itself is re-read the computer cannot obey a 51 operation code. This means that any 51 gates in the program must be replaced with 52 codes before tracing and all data must be entered in floating point form. This difficulty does *not* exist with the Intercom 1000 DP and the Intercom 500 systems.

5.5 Correction of Stored Programs

The foregoing sections have outlined methods of detecting errors in programs which have been entered into the memory. It is natural to inquire next how such errors should be corrected once they have been found. The answer, of course, depends on the type of error and, since there are so many possible types, we will not attempt to give an exhaustive discussion, but rather we will indicate what may be done in the most frequently occurring situations. Much must of necessity be left to the ingenuity of the programmer.

If a single command is found to be incorrect in the memory, it is only necessary to re-store that command. Use the 50 operation code from manual control and specify the location of the particular command in question. When the computer types back the address of the location, type the correct command followed by (tab)s and the previous contents will be replaced by the correction. Similar remarks apply to items of data which are incorrectly stored. The whole point is that from manual control the operator may store anything anywhere he wishes following the procedures of Section 2.7.

Often, however, errors can only be corrected by inserting commands which have been left out. As an example, suppose we have in the memory a program starting at 1310 and ending at 1390, and we find it necessary to insert the commands 421400 and 441408 between the commands located in 1322 and 1323. Obviously, it would involve considerable labor to move everything from 1323 - 1390 into 1325 - 1392 in order to free two locations for the insertion. A better method is to "patch" the program using unconditional transfers. We change the contents of 1323 to cause a transfer to an unused section of the memory where we place the commands to be inserted, usually followed by the command which was in 1323. Then we transfer back to 1324. The situation may be illustrated as follows, where arbitrary commands are listed in the relevant locations:

```
        Before                                       After
 Loc.   OP  ADDR                               Loc.  OP  ADDR
 1310   30  0002                               1310  30  0002
         .                                            .
         .                                            .
         .                                            .
 1322   49  1410                               1322  49  1410
 1323   49  1412        Transfer to patch      1323  29  1391
 1324   42  1404                               1324  42  1404
         .                                            .
         .                                            .
         .                                            .
 1390   67  0000                               1390  67  0000
                                             ( 1391  42  1400
                                   Patch    {  1392  44  1408
                                             ( 1393  49  1412
                        Transfer back          1394  29  1324
```

If possible, patches should be placed in the same channel(s) as the original program for convenience in punching a tape containing all commands. Finally, it is essential to write on the coding sheets clearly and completely all changes before they are made. This practice helps to make the changes accurately and also provides an invaluable record of the program as it is in the memory. Remember, the corrections may be wrong and one must know what they were in order to do any further debugging.

Chapter 6

Index Registers

6.1 The Command Modification Problem

One of the most powerful features of digital computers is their ability to modify the quantities being used in a problem during automatic computation. Data modification as defined and illustrated in Section 4.4 is an example of this ability. Further, we saw in Section 4.5 how the computer can be made to alter automatically the sequence of commands in the program before obeying them. This procedure was termed "command modification by replacement" and is only of limited usefulness. Now consider a more typical situation requiring command modification.

Suppose we have 100 numbers and we wish to form their sum. We could store them sequentially in the memory using Intercom SP or DP instructions depending on the precision required. We could then store 100 instructions in another part of the memory, the first one being a "42" (clear and add) operation and the remaining 99 instructions being "43" (add) operations. Of course, we would also want to type out the answer and halt. The process of storing this program, however, would be very laborious. Obviously, it is also a very repetitious program. If we could use just one "43" operation and cause the computer repeatedly to alter its address part and then obey it (i.e. form a loop) we could effect an enormous saving in the number of locations occupied by the program in the memory. It is possible to do this in the Intercom systems using a technique called <u>indexing</u>, employing things called <u>index</u> <u>registers</u>.

Because indexing is so important let us display more specifically the problem discussed above. Of course, the reader may not understand all the details of the second program. We are attempting only to clarify the command modification problem and to arouse curiosity as to the way the index registers function.

Suppose, then, that using Intercom 1000SP, 100 numbers are stored in locations 1700 - 1799. The first scheme suggested to form their sum might look as follows:

```
Loc.  K OP ADDR
1500    30 0002
   1    42 1700
   2    43 1701
   3    43 1702 )
         .      }
         .      }  This gap contains 95 commands!
         .      )
  99    43 1798
1600    43 1799
   1    34 2101
   2    67 0000
```

6.1 (continued)

Here is an indexed program which will accomplish the same thing:

```
                          Loc.  K OP ADDR

                          1500    30 0002
                             1  1 73 0000 )
                             2  1 72 0098 )   These four instructions
                             3  1 71 0001 )   set up the index register
                             4  1 70 0000 )
   Set accumulator           5    42 1700    (
                            {6  1 43 1701    { This 1 instruction accom-
   Calculation loop         {7  1 76 1506 <--{ plishes the incrementing,
                             8    34 2101    { testing, and transferring.
                             9    67 0000
```

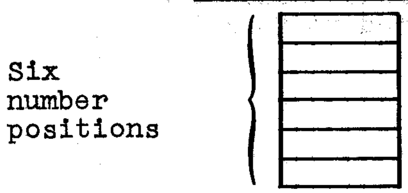
For most purposes the 10 instructions of the second method are preferable to the 103 instructions required by the first. However, the first program would be executed more rapidly than the second, since fewer instructions would actually be executed. It should thus be appreciated that index registers are very powerful tools; in fact, they have many uses in addition to the one illustrated above. Some of these uses will be shown in later sections. First it is necessary to understand how the registers function.

6.2 The Intercom Index Registers

Perhaps the best way to explain index registers is from a schematic point of view; that is, to emphasize the way they operate rather than how the Intercom programs cause the manipulations to take place. It may be helpful to state, however, that each index register is nothing more than six locations in the memory, but it is best at first not to worry about where they are. Further information on this point is given in Chapter 8, as well as a discussion of certain special features of the Intercom 500 registers.

An index register, then, should be thought of as being made up of six distinct number positions. A simple-minded diagram displaying this idea and the names of the six numbers follows:

An index register



Name of number	Abbreviation
Word base	WB
Word difference	WD
Word limit	WL
Channel base	CB
Channel difference	CD
Channel limit	CL

In general each number consists of two decimal digits but sometimes the WB and the CB may be larger than this. These exceptions will be discussed later.

There are available ten of these registers designated 1, 2, 3, •••, 9, u. When using a register in a program its symbol is written in the K column of the coding sheet for each relevant command.

6.2 (continued)

We will soon see that whenever a register is used it is necessary to set up its initial values. There are six operation codes for this purpose, one for each of the six number positions. The address part of commands containing these codes is used to provide the values to be set in the register. A symbol must be placed in the K position of these commands to designate which of the ten registers is involved. The operation codes are:

		When the command is executed:
Set Word Base	K 70 00WB	The word base of index register K will be set to the two digits found in the WB position of the command.
Set Word Difference	K 71 00WD	The word difference of index register K will be set to the two digits in the WD position of the command.
Set Word Limit	K 72 00WL	The word limit of index register K will be set to the two digits found in the WL position of the command.
Set Channel Base	K 73 CB00	The channel base of index register K will be set to the two digits found in the CB position of the command.
Set Channel Difference	K 74 CD00	The channel difference of index register K will be set to the two digits found in the CD position of the command.
Set Channel Limit	K 75 CL00	The channel limit of index register K will be set to .the two digits found in the CL position of the command.

There are two additional operation codes whose execution affects the designated register. But they do a great deal more besides and it is imperative to have a precise understanding of their function. Therefore we will discuss them in detail, together with examples, in Sections 6.3 and 6.4.

Before proceeding, let us consider how index registers may be used to modify commands. It should be quite clear by now what will happen when the machine obeys a command such as 43ADDR. Suppose instead that the command is indexed; that is, one of the symbols 1, 2, ..., 9, u, appears in the K position. When the computer obeys such a command the following happens:

(1) The digits of the CB and WB of register K are added in the control unit to ADDR thus:

$$\begin{array}{r} \mathrm{ADDR} \\ +\underline{\mathrm{CBWB}} \end{array}$$

(2) The resulting number, often called the effective address, becomes the address used in obeying the command.

(3) Neither the contents of the register involved nor the indexed command as recorded in the memory are altered by the execution of the command.

6.2 (continued)

As an example suppose the computer obeys the instruction 1 43 1701 located in 1506, and suppose that at this time register 1 is as follows:

	I. R. 1
WB	08
WD	01
WL	98
CB	00
CD	00
CL	00

The effective address of the command becomes 1701 + 0008 = 1709, so that the command obeyed will contain the address 1709. Hence the computer will add to the accumulator the contents of memory location 1709. After the command is executed, I.R. 1 will look exactly as above and the contents of 1506 will remain as 1431701.

It should be obvious that whenever a command is indexed, the programmer must be certain that the CB and WB of the register designated contains what he wants it to contain. It is astonishing how mar7 mistakes can be made in indexing, and the student is advised to re-read this section after studying Sections 6.3 and 6.4.

6.3 Modifying the Word Position of a Command

In order to write a program in which the word position of a command will be repeatedly modified, we require the following instruction:

Increment word base K 76 ADDR

Since this command is so important and does so much, we will state how it functions step by step. Moreover, the order of these steps is extremely important. They are:

In the index register designated by K,

(1) The WD is added to the WB and the sum replaces the previous WB.

(2) The WB (remember, it has just been incremented) is compared with the WL, and

If the WB ≦ WL, the computer takes its next instruction from location ADDR (that is, the location specified in the address portion of the 76 command).

If the WB > WL, the computer takes its next instruction in normal sequence (that is, from the location following that of the 76 command).

6.3 (continued)

As an example suppose the computer obeys the instruction 1 76 1506 located in 1507, and suppose that at this time register 1 is as follows:

	I. R. 1
WB	08
WD	01
WL	98
CB	00
CD	00
CL	00

After the command has been executed we have:

	I. R. 1
WB	09
WD	01
WL	98
CB	00
CD	00
CL	00

Since 09 < 98 (WB<WL), the next instruction obeyed will be the one in 1506.

The reader should now return to the example of Section 6.1 and observe the behavior of register 1 as the commands are obeyed. Note that after the instructions in 1500 - 1504 are obeyed we have:

	I. R. 1
WB	00
WD	01
WL	98
CB	00
CD	?
CL	?

The machine will then cycle repeatedly through the loop formed by the commands in 1506 and 1507. On each cycle the effective address of the 1431701 instruction will be incremented by 1, for the word base of I.R. 1 is so incremented each time the 1761506 instruction is obeyed. Thus the sequence of numbers stored from 1700 - 1799 will be added.

Finally, when I.R. 1 contains a WB of 97, as it eventually must, the 43 command will have an effective address of 1798. The computer will then obey the 76 command and transfer for the final time to 1506. This is because the WB of 97 is incremented to 98, the comparison of WB and WL gives 98= 98, and when WB = WL the next command comes from the address part of the 76 instruction.

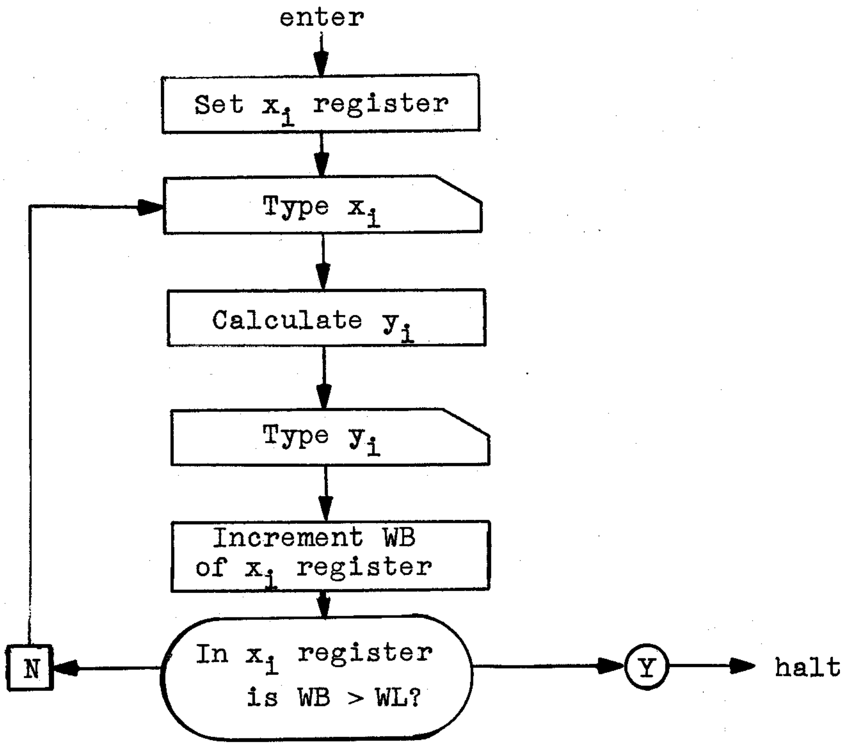
We then have the final run of the loop in which the effective address of the 43 command is 1799, the WB of 98 is incremented to 99, the comparison yields 99>98, and the next command comes from 1508 since for the first time WB>WL. The student should follow these details carefully visualizing the behavior of the register and remembering that the 1431701 command in 1506 never changes as recorded in the memory. In fact, the only variable is the WB of register 1. Once set initially the WD, WL, and CB never change. Note that it was not necessary to set the CD or CL. This point will become clear in Section 6.4.

6.3 (continued)

We can now unify all the foregoing ideas in a further example. The student should follow carefully all the details of this example and also compare it with Example 1, Section 4.4. As an additional aid, following the program, the condition at certain times of the register in use will be shown. Note also that, since we are using double precision, the word difference of the register is set at two. Finally, note the selection of 78 as the word limit. The best way to determine this value is to trace through the problem mentally for the final run. Thus we see that since the last effective address involving x_i must be 1378 the last WB value used must be 78. The computer then encounters the 76 command, increments 78 to 80, and compares 80 with the WL. Since we now desire WB>WL, we see that the WL must be either 78 or 79.

Example. In Intercom 1000DP evaluate $y = ax + b$ for 40 values of x.
Type out rows of the form: x_i y_i

FLOW CHART



STORAGE ALLOCATION CHART

Constants			Variables
Loc.	Symbol	Value	
1300	x_i		
.	.		
.	.		
.	.		
1378	x_{40}		
1398	a		
1396	b		

6.3 (continued)

Problem: $y = ax + b$

NOTES	LOCATION	K	OP	ADDRESS	ACCUMULATOR
	1600		30	00 02	?
Set x_i register	1	8	73	00 00	"
	2	8	72	00 78	"
	3	8	71	00 02	"
	4	8	70	00 00	"
Type x_i	5	8	33	13 00	"
Calculate y_i	6		42	13 98	a
	7	8	44	13 00	ax_i
	8		43	13 96	$ax_i + b$
Type y_i	9		38	21 00	"
Increment & test	10	8	76	16 05	"
	11		67	00 00	"

Index Register 8

	Initial set-up	During last run through the loop	When computer halts
WB	00	78	80
WD	02	02	02
WL	78	78	78
CB	00	00	00
CD	?	?	?
CL	?	?	?

Problems:

1. Write a program in Intercom 1000SP to evaluate $y = ax^3 + bx^2 = cx + d$ for 15 values of x. Type out as in the example above.

2. Do as in 1. for Intercom 1000DP.

3. Given 10 values of x and 10 values of y use Intercom 1000DP to calculate the 10 products $x_i y_i$ and the $\Sigma x_i y_i$. Type out 10 rows of the form $x_i y_i$ and then in the eleventh row type $\Sigma x_i y_i$.

4. Given $x_1 = (1600), \cdots, x_{12} = (1622)$ and $y_1 = -2.0$, $\Delta y = 0.4$, calculate $x_i + y_i$ for $i = 1, 2, \cdots, 12$. Gate for y_1 and Δy. Type out rows of the form x_i y_i $x_i + y_i$. Use an index register to get out of the loop.

6.3 (continued)

Problems: (cont.)

5. Re-write problem 4 using a testing constant and the variable y_i to get out of the loop. (Note that this problem neet not contain a 72 instruction.)

6.4 Modifying the Channel Position of a Command

In this section the remaining operation code affecting the contents of an index register will be introduced. This command operates analogously to the 76 operation described in Section 6.3.

Increment Channel Base K 77 ADDR

When this command is executed the following steps occur in the order given:

In the index register designated by K,

(1) The CD is added to the CB and the sum replaces the previous CB.

(2) The CB (just incremented) is compared with the CL and,

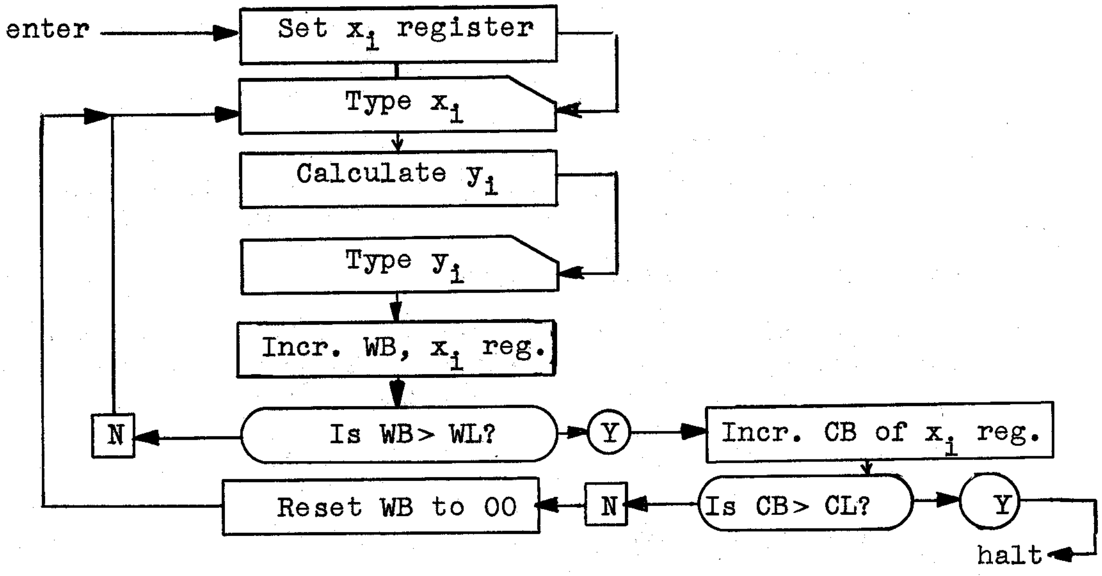
If the CB ≦ CL, the computer takes its next instruction from location ADDR (that is, the location specified in the address portion of the 77 command).

If the CB > CL, the computer takes its next instruction in normal sequence (that is, from the location following that of the 77 command).

The 77 command is typically used when there are too many items of data to store all in one channel. The following example illustrates this and is followed by some explanation of the new ideas involved.

Example. Evaluate y = ax + b for 100 double precision values of x. Type out rows of the form x_i y_i.

FLOW CHART



6.4 (continued)

STORAGE ALLOCATION CHART

Constants			Variables
Loc.	Symbol	Value	
1300	x_i		
.	.		
.	.		
.	.		
1498	x_{100}		
1598	a		
1596	b		

Problem: $y = ax + b$

NOTES	LOCATION	K	OP	ADDRESS		ACCUMULATOR
	1500		30	00	02	?
	1	4	75	01	00	"
	2	4	74	01	00	"
Set x_i register	3	4	73	00	00	"
	4	4	72	00	98	"
	5	4	71	00	02	"
	6	4	70	00	00	"
Type x_i	7	4	33	13	00	"
	8		42	15	98	a
Calculate y_i	9	4	44	13	00	ax_i
	10		43	15	96	$ax_i + b$
Type y_i	11		38	21	00	"
Incr. WB & test	12	4	76	15	07	"
Incr. CB & test	13	4	77	15	06	"
	14		67	00	00	

The student should notice that the operation of this program for the first 50 values of x_i is similar to the example of Section 6.3. For this portion of the problem the contents of I.R. 4 may be displayed as follows:

6.4 (continued)

	Initial set-up	During 50th run through loop	Just after execution of the 76 for the 50th time.
WB	00	98	100
WD	02	02	02
WL	98	98	98
CB	00	00	00
CD	01	01	01
CL	01	01	01

← Note the 3 digit no.

Since 100 > 98 (WB > WL), the next instruction obeyed will be the one in 1513. Note that this is the *first* execution of the 77 command. This will cause the CB of register 4 to become 01, (CB + CD → CB) and then the comparison of CB versus CL will take place. Since 01 = 01, (CB = CL), the next instruction comes from 1506, which is the location written in the address part of the 77 instruction. Since we had the foresight to place the "Set Word Base" (70) instruction last in the set-up section of the program, the computer is thus made to *reset the WB* of register 4 to 00. It then continues with the instruction in 1507 and the observant reader will see that it is now "trapped" in the computation loop for another 50 cycles. But since the CB is now 01 the x_1's stored in channel 14 will be used. Finally the machine reaches the halt command when it gets by both the 76 and 77 commands on the last run of the second 50.

We conclude with diagrams of index register 4 for the second 50 runs:

	During 51st run	During 100th run	After 100th execution of the 76 instr.	After 2nd execution of the 77 instr.
WB	00	98	100	100
WD	02	02	02	02
WL	98	98	98	98
CB	01	01	01	02
CD	01	01	01	01
CL	01	01	01	01

Problems:

1. Rewrite the foregoing example for the following storage of 30 values of x:

$x_1 = (0900)$ $x_{11} = (1000)$ $x_{21} = (1100)$

$\vdots$ $\vdots$ $\vdots$

$x_{10} = (0918)$ $x_{20} = (1018)$ $x_{30} = (1118)$

2. Do problem 3 Section 6.3 for the following storage of 20 x's and 20 y's:

$x_1 = (1300)$ $y_1 = (1400)$ $x_{11} = (1500)$ $y_{11} = (1600)$

$\vdots$ $\vdots$ $\vdots$ $\vdots$

$x_{10} = (1318)$ $y_{10} = (1418)$ $x_{20} = (1818)$ $y_{20} = (1838)$

6.4 (continued)

Problems: (cont.)

3. Do as in 2 for the following storage:

$$x_1 = (1700) \quad y_1 = (1720) \quad x_{11} = (1800) \quad y_{11} = (1820)$$
$$\vdots \qquad \vdots \qquad \vdots \qquad \vdots$$
$$x_{10} = (1718) \quad y_{10} = (1738) \quad x_{20} = (1818) \quad y_{20} = (1838)$$

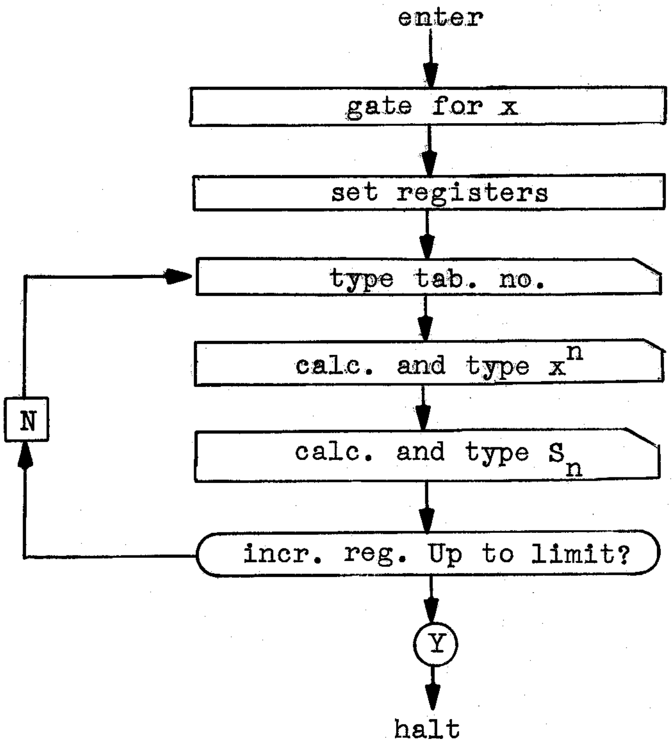
6.5 Use of Index Registers for Typing Tabulating Numbers

The following programs in Intercom 1000DP will gate for x then type:

$$n,\ x^n,\ S_n = x_1 + x_2 + \cdots x_n$$

the first for n = 1 to 20, and the second for n = 1 to 199.

In the first program the constants 0 and 1 are generated in the computer, while in the second they are stored. This was done to illustrate the difference between the two methods.



When not using the channel portion of an index register, setting the channel base to zero is a precautionary measure.

6.5 (continued)

Problem: n, x^n, $S_n = x_1 + x_2 + \cdots + x_n$, 20 values

NOTES	LOCATION	K	OP	ADDRESS	ACCUMULATOR
position paper	1000		30	00 04	?
gate for x	1		51	21 00	x
space	2		30	00 02	x
store x = (1050)	3		49	10 50	x
div. by x	4		48	21 00	1
store 1 = x^{n-1} = (1052)	5		49	10 52	1
sub. 1	6		41	10 52	0
store 0 = S_{n-1} = (1054)	7		49	10 54	0
set channel base to 0	8	1	73	00 00	0
set word base to 0	9	1	70	00 00	0
set word difference to 1	10	1	71	00 01	0
set word limit to 19	11	1	72	00 19	0
type tab. no.	12	1	31	00 01	0 or S_n
clear and add x^{n-1}	13		42	10 52	x^{n-1}
mult. by x	14		44	10 50	x^n
store x^n	15		49	10 52	x^n
type x^n	16		33	21 00	x^n
add S_{n-1}	17		43	10 54	S_n
store S_n	18		49	10 54	S_n
type S_n (cr)	19		38	21 00	S_n
increment register	20	1	76	10 12	
halt	21		67	00 00	

6.5 (continued)

Problem: n, x^n, $S_n = x_1 + x_2 + \cdots + x_n$, 199 values

NOTES	LOCATION	K	OP	ADDRESS	ACCUMULATOR
position paper	1000		30	00 04	?
gate for x = (1050)	1		51	10 50	?
clear and add 1 = (1052)	2		42	10 52	1
store 1 = x^{n-1} = (1056)	3		49	10 56	1
clear and add 0 = (1054)	4		42	10 54	0
store 0 = S_{n-1} = (1058)	5		49	10 58	0
set channel limit to 1	6	1	75	01 00	0
set channel diff. to 1	7	1	74	01 00	0
set channel base to 0	8	1	73	00 00	0
set word limit to 99	9	1	72	00 99	0
set word diff. to 1	10	1	71	00 01	0
set word base to 1	11	1	70	00 01	0
space	12		30	00 02	0
type tab. no.	13	1	31	00 00	0
clear and add x^{n-1}	14		42	10 56	x^{n-1}
mult. by x	15		44	10 50	x^n
store x^n	16		49	10 56	x^n
type x^n	17		33	21 00	x^n
add S_{n-1}	18		43	10 58	S_n
store S_n	19		49	10 58	S_n
type S_n carr. ret.	20		38	21 00	S_n
increment word base	21	1	76	10 13	"
set word base	22	1	70	00 00	"
increment channel base	23	1	77	10 13	"
halt	24		67	00 00	"

6.5 (continued)

Problems:

1.	Compound interest and present value of 1.	n	$(1+i)^n$	$(1+i)^{-n}$
2.	Compound interest and present value of A.	n	$A(1+i)^n$	$A(1+i)^{-n}$
3.	Constant percentage depreciation of 1.	n	$(1-r)^n$	$1-(1-r)^n$
4.	Constant percentage depreciation of A.	n	$A(1-r)^n$	$A[1-(1-r)^n]$
5.	Power and reciprocal of power.	n	x^n	x^{-n}
6.	Reciprocal of power and sum.	n	x^{-n}	$S_n = x^{-1} + x^{-2} + \ldots x^{-n}$
7.	Factorial and sum.	n	$x!$	$S_n = 1! + 2! + \ldots n!$

6.6 Use of Index Registers with Data

The following program for Intercom 1000DP will find the average and average of squares of numbers in 1800, 1802, ... to 1818. Results will be typed out: $i \quad x_i \quad x_i^2$

$$\Sigma \frac{x}{n} \quad \Sigma \frac{x^2}{n}$$

Problem: Average and average of squares

NOTES	LOCATION	K	OP	ADDRESS		ACCUMULATOR
position paper	1500		30	00	02	?
	1	1	73	00	00	"
	2	1	72	00	10	"
	3	1	71	00	01	"
	4	1	70	00	01	"
set registers	5	2	73	18	00	"
	6	2	72	00	18	"
	7	2	71	00	02	"
	8	2	70	00	00	"
clear and add 0 = (1820)	9		42	18	20	0
store 0 = Σx = (1822)	10		49	18	22	0
store 0 = Σx^2 = (1824)	11		49	18	24	0

(program continued on next page)

6.6 (continued)

NOTES	LOCATION	K	OP	ADDRESS		ACCUMULATOR
type tab. no.	1012	1	31	00	00	?
clear and add Σx	13		42	18	22	Σx
add next x	14	2	43	00	00	Σx
store back	15		49	18	22	Σx
type x	16	2	33	00	00	Σx
clear and add x	17	2	42	00	00	x
mult. by x	18		44	21	00	x^2
type x^2	19		38	21	00	x^2
add Σx^2	20		43	18	24	Σx^2
store back	21		49	18	24	Σx^2
incr. reg. 1	22	1	76	15	23	Σx^2
incr. reg. 2	23	2	76	15	12	Σx^2
space	24		30	05	02	Σx^2
clear and add Σx	25		42	18	22	Σx
div. by 10 = (1826)	26		48	18	26	$\Sigma x/n = \bar{x}$
type $\bar{x}$	27		33	21	00	$\Sigma x/n = \bar{x}$
clear and add Σx^2	28		42	18	24	Σx^2
div. by 10	29		48	18	26	$\Sigma x^2/n$
type $\Sigma x^2/n$	30		38	21	00	$\Sigma x^2/n$
halt	31		67	00	00	"

Problems:

1. Write a program to calculate the average of 50 double precision numbers in 1800 to 1898. Type the result.

2. Write a program to calculate the average of 50 double precision numbers in 1750 to 1848. Type the result.

3. Same as 1 for 100 single precision numbers in 1800 to 1899.

4. For x's in 1800, 1802, ... and y's in 1600, 1602, ... (double precision), write a program to find and type:

   $i \quad x_i \quad y_i \quad x_i y_i \quad x_i^2 \quad y_i^2$

   then the sums of these.

6.7 Output Format Control with Index Registers

The following portion of a program will type 20 lines with a space after each 5. Here it is assumed that part of the program is before 1030 and part between 1038 and 1070.

Problem: Type 20 lines with space after each five lines.

NOTES	LOCATION	K	OP	ADDRESS	ACCUMULATOR
	1030	1	73	00 00	
	31	1	72	00 19	
	32	1	71	00 01	
	33	1	72	00 00	
	34	2	73	00 00	
	35	2	72	00 04	
	36	2	71	00 01	
	37	2	70	00 00	
	38	1	31	00 01	
	1070	1	76	10 72	
	71		67	00 00	
	72	2	76	10 38	
	73		30	00 01	
	74		29	10 37	

Problems:

1. Rewrite the above example to type 30 lines with a space after each three lines.

2. Rewrite one of the problems of Section 6.5 to space after each five lines.

6.8 Summary

In Intercom 1000DP it is possible to get an error indication using index registers (see Section 5.2). This will happen if the word position of an address is incremented beyond 99. Thus if the WB of I.R. 3 is 100 and we execute an indexed command, as 3431700, an error indication will be given. Note carefully that it is not an error to increment the WB to 100 or more as has been done in previous examples. It is the attempt to execute a command with an effective address whose word position is more than 99 that causes the error. Thus if I.R. 2 has a WB of 60 and the machine encounters the command 2491570 an error indication will be given.

Index registers seem to give more difficulty than any other concept in digital computing. There is no simple cure for those who find this to be the case. Re-reading Sections 6.1 - 6.4 many times may help. Make an effort to think precisely. Ask yourself at each step in a program what the register contains. Above all, learn exactly what the machine does when it executes a 76 or a 77 command, and when it executes an indexed command like 7421400.

It may help, too, to visualize the index register as a counter. The variables are the word base and the channel base and they count up in increments specified by the constant word difference and channel difference respectively. The word limit and channel limit are also constants.

In the last analysis there is no substitute for experience. Putting problems on the computer and making mistakes is one of the best ways to learn if one takes the trouble to debug the problem and think carefully about the mistakes which are found.

6.9 Sample Examination

Double precision values of x and y are stored as follows:

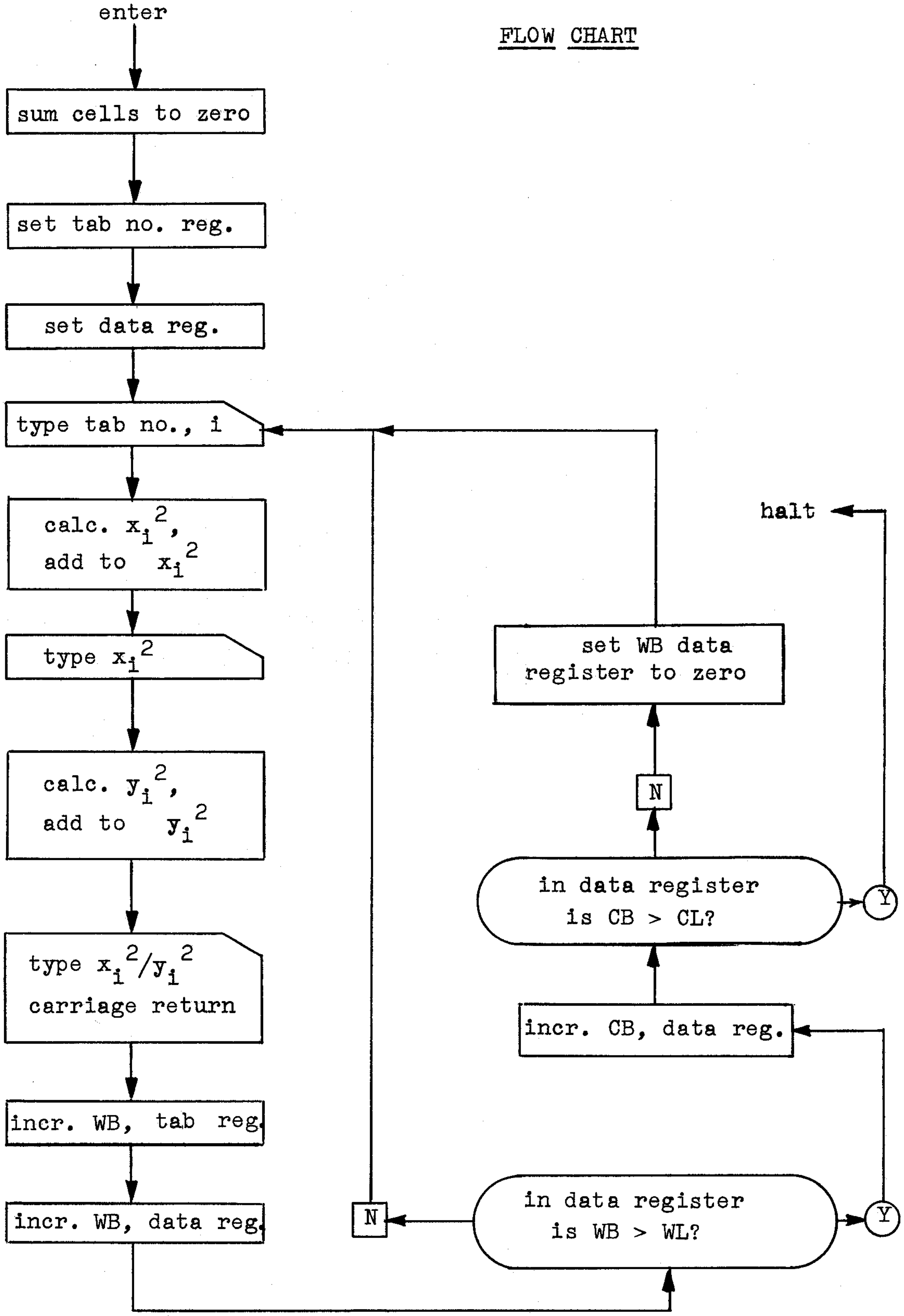
$$\begin{array}{ll}
x_1 = (0900) & y_1 = (0902) \\
x_2 = (0904) & y_2 = (0906) \\
\vdots & \vdots \\
x_{25} = (0996) & y_{25} = (0998) \\
x_{26} = (1000) & y_{26} = (1002) \\
x_{27} = (1004) & y_{27} = (1006) \\
\vdots & \vdots \\
x_{50} = (1096) & y_{50} = (1098)
\end{array}$$

Type out 50 rows of the form:

$$i \qquad x_i^2 \qquad y_i^2 \qquad x_i^2/y_i^2$$

Underneath the last 3 columns, type the sum of each column.
Halt the machine.
Include a flow chart and a storage allocation chart.

6.9 (continued)


enter
FLOW CHART
sum cells to zero
set tab no. reg.
set data reg.
type tab no., i
calc. x_i^2, add to x_i^2
type x_i^2
calc. y_i^2, add to y_i^2
type x_i^2/y_i^2 carriage return
incr. WB, tab reg.
incr. WB, data reg.
in data register is WB > WL?
N
Y
incr. CB, data reg.
in data register is CB > CL?
Y
halt
N
set WB data register to zero

6.9 (continued)

STORAGE ALLOCATION CHART (not including x's and y's)

	Loc.	Symbol	Starting value
	1198	x_i^2	0
	1196	y_i^2	0
	1194	x_i^2/y_i^2	0
	1192	temporary	storage
	1190	temporary	storage

Problem: i, x^2, y^2, x^2/y^2

NOTES	LOCATION	K	OP	ADDRESS		ACCUMULATOR
	1100		30	00	02	?
Generate zero	1		41	21	00	0
Set sum cells	2		40	11	98	"
	3		49	11	96	"
	4		49	11	94	"
Set tab no. register	5	1	73	00	00	"
	6	1	71	00	01	"
	7	1	70	00	00	"
Set data register	8	2	75	01	00	"
	9	2	74	01	00	"
	10	2	73	00	00	"
	11	2	72	00	97	"
	12	2	71	00	02	"

(program continued on next page)

6.9 (continued)

NOTES	LOCATION	K	OP	ADDRESS		ACCUMULATOR
Set data reg. base zero	1113	2	70	00	00	
Type tab no.	14	1	31	00	01	
	15	2	42	09	00	x_i
	16		44	21	00	x_i^2
Calc., type, save,	17		33	21	00	
accumulate x_i^2	18		49	11	92	
	19		43	11	98	$x_i^2 + \Sigma x_i^2$
	20		49	11	98	"
	21	2	42	09	02	y_i
	22		44	21	00	y_i^2
Calc., type, save,	23		33	21	00	"
accumulate y_i^2	24		49	11	90	"
	25		43	11	96	$y_i^2 + \Sigma y_i^2$
	26		49	11	96	"
	27		42	11	92	x_i^2
Calc., type,	28		48	11	90	x_i^2/y_i^2
accumulate x_i^2/y_i^2	29		38	21	00	"
	30		43	11	94	$x_i^2/y_i^2 + \Sigma x_i^2/y_i^2$
	31		49	11	94	
Incr. tab no. register	32	1	76	11	33	
Incr. WB, data reg.	33	2	76	11	14	
Incr. CB, data reg.	34	2	77	11	13	
Space	35		31	00	00	
	36		33	11	98	
Type sums	37		33	11	96	
	38		38	11	94	
halt	39		67	00	00	

Chapter 7

Subroutines

7.1 General Description

A subroutine is a program which will obtain a frequently needed result and which is designed so that it may be used as often as necessary in conjunction with another program. For example, one often requires the square root of a quantity. Since the Bendix, like most computers, does not have a square root command, a subroutine is written to obtain the square root. Once checked out and placed on paper tape, the subroutine is always available for use. Computing installations maintain a large file or library of subroutines which the programmers use as necessary. Also, most computer manufacturers include with their machines a basic subroutine library. In particular, Bendix provides for use with the Intercom systems a library of the most frequently required routines. The use of this library will be discussed in Sections 7.3 through 7.5.

There are some important fundamental ideas which arise in connection with subroutines. Obviously, provision must be made for placing any required subroutines in the memory along with the program which uses them. On the Bendix this is easily done by having the subroutines punched on paper tape and then reading in the tapes as necessary. Of more interest are the problems involved in designing subroutines so that they may be conveniently used. A little thought will indicate that provision must be made for transferring control from *any point* in the main program to the subroutine and the subroutine must then be able to return control *to that point*. This process might take place many times during computation, each time from a different point in the main program. The Intercom systems are designed to handle this problem so that one may write his own subroutines if desired. We will discuss this point in Section 7.2.

Another major problem is that subroutines must be designed so that the programmer may conveniently make available to the subroutine all parameters on which it is to operate. In the case of a square root subroutine for example, the radicand might be placed in a definite location prior to transferring control to the subroutine. The subroutine, then, would be designed simply to take the square root of whatever quantity is in this location. The instructions required to set up the parameters for the subroutine are usually referred to as the *calling sequence*.

Finally, subroutines are to be thought of as convenient tools for the programmer. If it is found that a certain calculation is to be performed many times in solving a problem, a subroutine may well be used. If one is already written so much the better; if not, it will usually pay to write one. These points will become clearer after reading the next sections.

7.2 Writing Subroutines in Intercom

There are some commands in the Intercom systems which are specifically designed for writing subroutines. They are called "mark place and transfer" commands.

Mark Place and Transfer I	K 26 ADDR	Control is transferred to the command in location ADDR. The address of the 26 command is stored in a special register.
Return to Marked Place I	16 0000	Control is transferred to the command located immediately subsequent to the last 26 command. This command has no meaning unless it is preceded in the program by a "Mark Place and Transfer I" command.
Mark Place and Transfer II	K 28 ADDR	Control is transferred to the command in location ADDR. The address of the 28 command is stored in a special register.
Return to Marked Place II	18 0000	Control is transferred to the command located immediately subsequent to the last 28 command. This command is meaningless unless it is preceded in the program by a "Mark Place and Transfer II" command.

The whole point here is that the 26 or 28 commands may be located anywhere and the computer will unerringly return to the command following in sequence when it encounters a 16 or 18 command respectively. The student should note carefully the distinction between this transferring scheme and the simple unconditional transfer command 29.

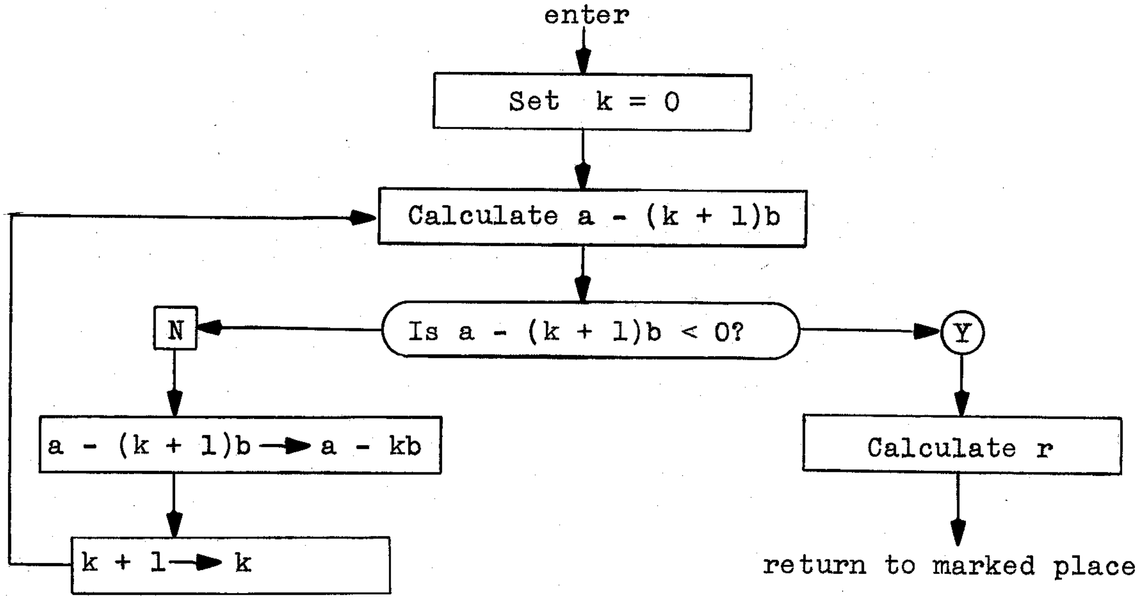
Example. Write an Intercom 1000 DP subroutine to perform division such that the integral part of the quotient is obtained separately from the remainder.

Let	a = dividend	k = integral part of quotient
	b = divisor	r = remainder

We have $a/b = k + r/b$ or $a = kb + r$. Our plan will be to subtract b from a, repeatedly, counting the number of subtractions until the result first becomes negative. The count will be k, and r will be obtained by adding b to the negative result.

7.2 (continued)

FLOW CHART



It would now seem appropriate to select storage locations for some of the quantities involved except for the fact that we wish the routine to be as compact as possible. Therefore we will write the program using symbols for some of the addresses. Then, once it is known how many locations the program requires, we can assign the next sequential cells for data storage.

Program

Notes	Loc.	K	OP	ADDR	Accumulator
Set k = 0 {	0900		41	2100	0
	1		49	k	"
Calculate a - (k + 1)b {	2		42	a	a - kb
	3		41	b	a - (k + 1)b
Is a - (k + 1)b neg.?	4		22	0910	"
a - (k + 1)b → a - kb	5		49	a	"
	6		42	k	k
k + 1 → k	7		43	one	k + 1
	8		49	k	"
	9		29	0902	"
	10		43	b	r
	11		16	0000	"

We can now make the following assignments:

Constants			Variables		
Loc.	Symbol	Value	Loc.	Symbol	Starting value
0912	one	1	0914	k	0
0918	b		0916	a	the dividend

7.2 (continued)

In order to place the subroutine in the memory, and so on tape, one of course must replace the literal symbols by the above values.

Finally, the specifications for using the routine may be stated as follows:

1. Subroutine uses locations 0900 - 0919.
2. Before transferring control, place a in 0916 and b in 0918.
3. To enter subroutine, use 26 0900.
4. Location of results: k in 0914, r in 2100.

As an example of the use of this subroutine, suppose that at some point in a program we have calculated a quantity x and we wish to know if it is divisible by, say, 31. Assuming 31 = (1454), x is in the accumulator, and our next command location is 1327, we may proceed as follows:

Notes	Loc.	K	OP	ADDR	Accumulator
		.			
		.			
		.			x
x → a	1327		49	0916	x
31 → b	8		42	1454	31
	9		49	0918	31
	30		26	0900	31
Transfer if r = 0	31		23		
		.			
		.			
		.			

We would now do whatever might be desired. Note the existence of a branch depending on whether or not the remainder is zero. Of course, the integral part of the quotient is in 0914 and could be obtained if desired.

It may be well to emphasize that before such a program (of which this last is a piece) is run, it is necessary to load the subroutine into channel 09.

Problems.

1. Write a subroutine to reduce any angle in radians to a positive angle less than 2π such that the trigonometric functions of the original angle and the final angle are the same. Include a clear statement of specifications with your solution.

2. Write a subroutine to accomplish the multiplication of two complex numbers. Note that a complex number a + bi must be stored in two separate locations, one for a and the other for b. Thus the subroutine will require four locations for initial data, and two locations for results.

3. Extend the subroutine in 2 to accomplish complex division. Have a separate entry for this but use the same storage locations as in 2 for the initial data and for the results. The entire subroutine for both multiplication and division should employ locations in only one channel.

7.3 The Intercom Subroutine Library

The subroutines made available by Bendix for use with the Intercom systems may be divided into two classes: (1) Mathematical Subroutines and (2) Auxiliary Equipment Subroutines. The latter are required in using such accessories as a Flexowriter, magnetic tape, or IBM card equipment, and will not be discussed here.

The Mathematical subroutines are normally supplied in a separate tape magazine, one magazine for each of the three Intercom systems. (Actually there are two appendices for Intercom 1000 DP. See problem 4, Section 7.5 for an explanation.) These are referred to as appendix magazines or appendix tapes. Below is given a table displaying the mathematical subroutines in each appendix tape. One horizontal block corresponds to one subroutine which fits in one channel of the memory, but note that one routine may do several distinct things; the one which is done is a function of the word position used when transferring control to the subroutines. This point is illustrated in the next sections, which explain the use of the subroutine library and the meanings of the various items in the tables.

Table I. Library Subroutine Specifications

Subroutine	Word position for entry	N (loading code no.)	
		SP	DP
Fraction Selector	See Table II	1 CHu0	1 CHu0
Square Root	97	2 CHu0	2 CHu0
$\mathrm{Log}_{10}\ x$	71	3 CHu0	3 CHu0
$\mathrm{Log}_e\ x$	17		
$\mathrm{Log}_2\ x$	08		
e^x	22	4 CHu0	4 CH00
2^x	08		
10^x	72		
Sin x (degrees)	39	5 CHu0	5 CH00
Sin x (radians)	42		
Cos x (degrees)	23		
Cos x (radians)	26		
Arctan x (radians)	24	6 CHu0	6 CH00

Table II. Fraction Selector Entries

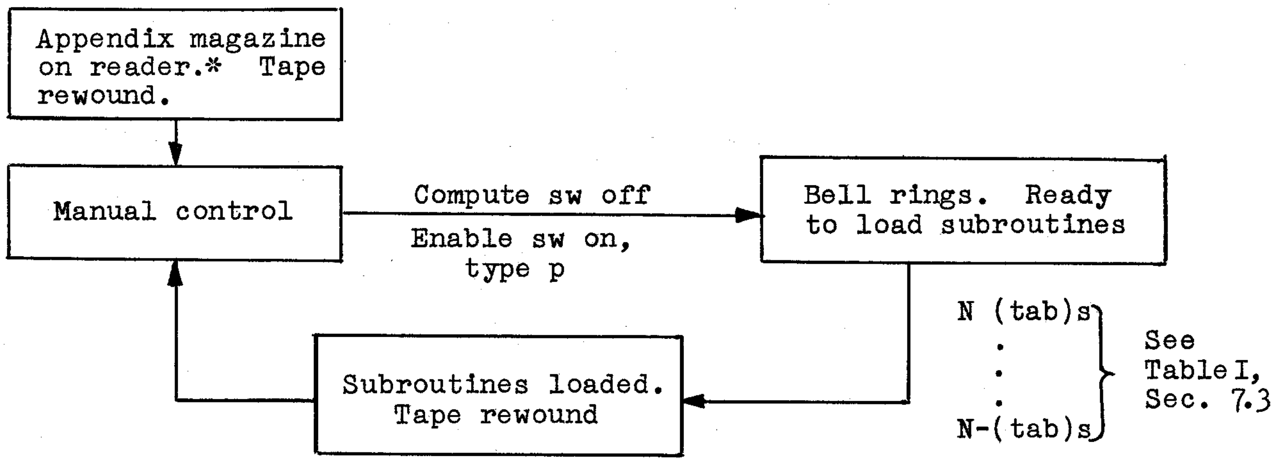
No. of decimal places to be typed out	Word position for entry
0	08
1	01
2	02
3	03
4	04
5	05
6	06
7	07

Table III. Storage Limitations

If the N value of a subroutine ends in 00, when the subroutine is:	
Placed in CH	Do not use index register
09	1
10	2
11	3
12	4
13	5
14	6
15	7
16	8
17	9
18	u

7.4 Loading Library Subroutines

A library subroutine may be placed in and will operate correctly from any channel of the Intercom memory. The loading procedure is outlined in the chart below. The symbol N represents a five digit number. The first digit is a code for the subroutine, the next two are the channel into which the subroutine is to be loaded and the last two are either u0 or 00.


Appendix magazine on reader.* Tape rewound.
Manual control
Compute sw off
Enable sw on, type p
Bell rings. Ready to load subroutines
N (tab)s
.
.
.
N-(tab)s
See Table I, Sec. 7.3
Subroutines loaded. Tape rewound

*With Intercom 500, the subroutines are usually included with the basic tape. In this case it is not necessary to change magazines. Merely follow the instructions from manual control.

Note that one or more subroutines may be loaded at "one shot," by typing one loading code number N, followed by (tab)s, for each routine desired. The computer will retain these N's but will not load until an N is followed by the minus sign (tab)s, at which point all subroutines specified by the retained code numbers will be loaded. The following examples should clarify this point.

Example 1. We wish to load Intercom 1000 DP subroutines as follows:

Logarithm	CH 10
Square root	CH 12
Sine-Cosine	CH 17

At the appropriate point for typing N we type:

```
3 10u0 (tab)s
2 12u0 (tab)s
5 1700 -(tab)s
```

The order of these is immaterial except that the last N must be followed by the minus sign, then (tab)s.

Example 2. We wish to load the Intercom 1000 SP exponential subroutine in channel 18.

At the appropriate point for typing N we type 4 18u0 -(tab)s.

7.5 Programming with Library Subroutines

The transfer of control to a library subroutine is accomplished with a special command.

Perform subroutine K 08 CHWD

The library subroutine in channel CH with word entry position WD is executed using the value in the accumulator as the argument. The result is placed in the accumulator and control is returned to the location next in sequence to that of this command.

Exception: Use of the fraction selector subroutine does not affect accumulator.

In addition to this command, there are some special perform subroutine commands (see problems 4 and 5). At this point, however, the student should concentrate on learning the 08 command, for it is of most general use.

Clearly, the 08 operation has no meaning unless the channel designated does, in fact, contain a subroutine with an entry at WD. But, as we learned in Section 7.4, library subroutines may be placed in any channel of the Intercom memory. Hence in programming we have complete freedom to place the needed subroutines wherever convenient with the exception shown in Table III, Section 7.3. When a certain channel is selected for a subroutine this should be recorded in the storage allocation chart.

As a simple example of subroutine programming, suppose at some point in a program we have an angle x in degrees in 1426 and we wish to find sin x and store it in 1480. We select an unused channel, say 17, and reserve it for the sine-cosine subroutine. The program steps might then be:

```
Loc.  K OP ADDR       Accumulator
      .
      .
      .
        42 1426            x
        08 1739          sin x
        49 1480            "
      .
      .
      .
```

where the locations for the commands depend on the rest of the program.

A more complicated example of subroutine programming follows.

7.6 Example of Use of Subroutines

The following program for 1000 DP will calculate and type:

$$x_i \quad x_i^2 \quad x_i^3 \quad \sqrt{x_i} \quad \sqrt[3]{x_i}$$

from $x_i = x_o$ to $x_i = x_n$ by increments of Δx where x_o, x_n, Δx are numbers with one decimal place, $x_n > x_o$.

Storage Allocation Chart

Constants			Variables		
Loc.	Symbol	Value	Loc.	Symbol	Starting value
0948	x_o		0950	x_i	x_o
0952	x_n				
0954	Δx				
0956	3	3			
CH.	Subroutine				
10	fraction selector				
11	square root				
12	logarithm				
13	exponential				

7.6 (continued)

Program for x, x^2, x^3, $\sqrt{x}$, $\sqrt[3]{x}$.

NOTES	LOCATION	K	OP	ADDRESS		ACCUMULATOR
position paper	0900		30	00	02	?
cl. & add x_o	01		42	09.	48	x_o
set $x_i = x_o$	02		49	09	50	x_i
cl. & add x_i = (0950)	03		42	09	50	x_i
select 1 decimal pl.	04		08	10	01	x_i
type x_i	05		33	21	00	x_i ✓
mult. by x_i	06		44	21	00	x_i^2
select 2 dec. pl.	07		08	10	02	x_i^2
type x_i^2	08		33	21	00	x_i^2 ✓
mult. by x_i	09		44	09	50	x_i^3
select 3 dec. pl.	10		08	10	03	x_i^3
type x_i^3	11		33	21	00	x_i^3
select 7 dec. pl.	12		08	10	07	x_i^3 ✓
cl. & add x_i	13		42	09	50	x_i
find $\sqrt{x_i}$	14		08	11	97	$\sqrt{x_i}$
type $\sqrt{x_i}$	15		33	21	00	$\sqrt{x_i}$ ✓
cl. & add x_i	16		42	09	50	x_i
find log	17		08	12	71	log x_i
div. by 3 = (0956)	18		48	09	56	1/3 log x_i
exp. 10	19		08	13	72	$\sqrt[3]{x_i}$
type $\sqrt[3]{x_i}$	20		38	21	00	$\sqrt[3]{x_i}$ ✓
cl. & add x_i	21		42	09	50	x_i
add Δx	22		43	09	54	$x_i + \Delta x$
store $x_i + \Delta x$	23		49	09	50	$x_i + \Delta x$
sub x_n	24		41	09	52	$x_i + \Delta x - x_n$
trans if acc < 0	25		22	09	03	$x_i + \Delta x - x_n \geqq 0$
halt	26		67	00	00	$x_i + \Delta x - x_n \geqq 0$

7.6 (continued)

Problems.

1. Write a program to type:

   $x \quad \sin x \quad \cos x \quad \tan x$

2. Write a program to type:

   $x \quad \text{arc} \sin x \quad \text{arc} \tan x$

3. Given eight values of x, calculate $\sqrt[n]{x}$ for n = 2, 3, ..., 6. Output format:

$$\begin{array}{lll} n & & \\ 1 & x_1 \cdots & \sqrt[n]{x_1} \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ 8 & x_8 \cdots & \sqrt[n]{x_8} \end{array}$$

Five such blocks with 3 spaces between each block and 1 space between n and the 8 rows.

Type n with no digits after the decimal point but the x's and their roots with 7 places.

4. In Intercom 1000DP, Appendix II subroutines use CH 05 instead of CH 19 and use of subroutine can be made while output is in progress, hence will sometimes save some time. These subroutines use OP 02 instead of 08. Rewrite the illustrated example to use 02 subroutines.

5. In Intercom 500, all subroutines are executed from line 05, OP code 02 or 08 may be used. In addition, if entry is at word 00, OP code w3 will be faster than the above. The square root subroutine has an additional entry at 00. Rewrite the illustrated example for Intercom 500.

Chapter 8

Some General Examples

In the following sections, notes and contents of the accumulator will not be included with all programs. Filling in these details is left as an exercise for the student.

It will also be noted that the notes and flow charts for the programs in this and following chapters tend to be briefer than heretofor. This has been done purposely to encourage the reader to learn to interpret programs which are not so completely spelled out. Remember that flow charts and notes are only aids to be used to the extent felt necessary by the individual. But remember, also, that short cuts in programming may lead to extra hours of debugging.

8.1 Programs for punching, loading, and/or typing programs.

```
0960  1751300
  61  1740100
  62  1730900
  63  1700000
  64  1390000
  65  1770964
  66   670000

0970  1751300
  71  1740100
  72  1731000
  73  1700000
  74  1550000
  75  1770974
  76   670000

0980   300002
  81  1730900
  82  1720037
  83  1710001
  84  1700000
  85  1310000
  86  1350000
  87   300001
  88  1760985
  89   300002

0990   630000
  91   630000
  92   680000
  93  1720052
  94  1710001
  95  1700040
  96  1310000
  97  1380000
  98  1760996
  99   670000
```

Assume that a program for 1000SP has commands in 0900 to 0937 inclusive and data in 0940 to 0952 inclusive, there are subroutines in channels 10 to 13 inclusive. The commands of this section are added to the program.

If we type from manual:	The following will happen.
690960(tab)s	A tape for the entire program will be punched.
Tape on photo-reader, 550900(tab)s 690970(tab)s	The program will be read into the computer.
690980(tab)s	Commands of the program will be typed out. Bell will ring twice. Breakpoint halt.
Compute switch off, then back to go.	Data of the program will be typed out.

Problem.

Add commands to a program already written to punch, load, and/or type the program.

Note.

In Intercom 500, commands will be typed out in usual form KOPCHWD, but in Intercom 1000, commands are typed out as KWDOPCH.

8.2 (continued)

Problems:

1. Write a program to type: 105, 104, ... 96.
2. Write a program to calculate and type:
   x° sin x° cos x° tan x° $(90^{\circ} - x^{\circ})$
   for x = 0 to 45
3. Write a program to calculate and type:
   K Ka (10 - K)a 10 - K
   for K = 1 to 10
4. Write program for a page of trig. tables for a particular number of degrees. Type out functions for each minute.

8.3 Block Copy Operations

Block copy operations (OP 81) are possible with Intercom 1000DP and 500. The execution of command K 81 CHWD will copy the contents of words u2 to u7 and 00 to (WD - 1) of channel CH into the corresponding words of channel (08 + K). If CH is 29, the words of channel (08 + K) will be cleared to zero, in Intercom 1000DP. (Floating point zeros are not obtained in this way with Intercom 500.)

For example:

command	will copy	into
2 81 1508	15u2 to 15u7 and 1500 to 1507	10u2 to 10u7 and 1000 to 1007
3 81 2900	zeros	index register 3
5 81 29u2	zeros	index reg. 5 and ch 13

The following program will clear index register 1, clear ch. 10 except words u0 and ul, copy channel 11 except words u0 and ul into corresponding words of channel 13.

Intercom 500 has constants stored in words u0 and ul of almost all channels, so these should not be cleared or copied.

NOTES	LOCATION	K	OP	ADDRESS	ACCUMULATOR
position paper	0900		30	00 02	
clear index reg. 1	1	1	81	29 00	
clear ch 10 except u0, ul	2	2	81	29 u0	
copy ch 11 into ch 13 except u0, ul	3	5	81	11 u0	
halt	4		67	00 00	

8.3 (continued)

Problems:

1. Write a program to clear all index registers.
2. Write a program in channel 9 to clear channels 10 to 18 inclusive.
3. Write a program to set index register 1, then set index register 2 to 5 incl. with the same values.
4. Write a program to store the number 7 in all locations 0900 to 1899 incl.

8.4 Index Register Operation, Intercom 500

In Intercom 500, the word base and channel base may be set independently to any value as long as the total is not greater than 3199. The index register accumulator (IRA) has a memory location of 2202.

These operations are available:

OP09:	K09CHWD	copies [CHWD + (K)] into IRA
OP78:	K78000D	copies element of index register into IRA
OP79:	K79000D	copies IRA into element index register

D	element of register
0	word difference
1	word limit
2	word base
3	channel base
4	channel difference
5	channel limit

If the index register utilization subroutine is used, a floating point number may be used to set an index register. A value from an element of an index register may be stored as a floating point number.

If the index register utilization subroutine is stored in channel CH, the following portion of a program will convert the value in the word base of register 7 and store the value in 1772.

Problem: Convert WB of register 7 to floating point and store in (1772)

NOTES	LOCATION	K	OP	ADDRESS		ACCUMULATOR
		7	78	00	02	
			08	CH	01	
			49	17	72	

8.4 (continued)

The following portion of a program will convert the floating point number (an integer) in 1375 and store this in the word limit of register 3.

NOTES	LOCATION	K	OP	ADDRESS	ACCUMULATOR
			42	13 75	
			08	CH 00	
		3	79	00 01	

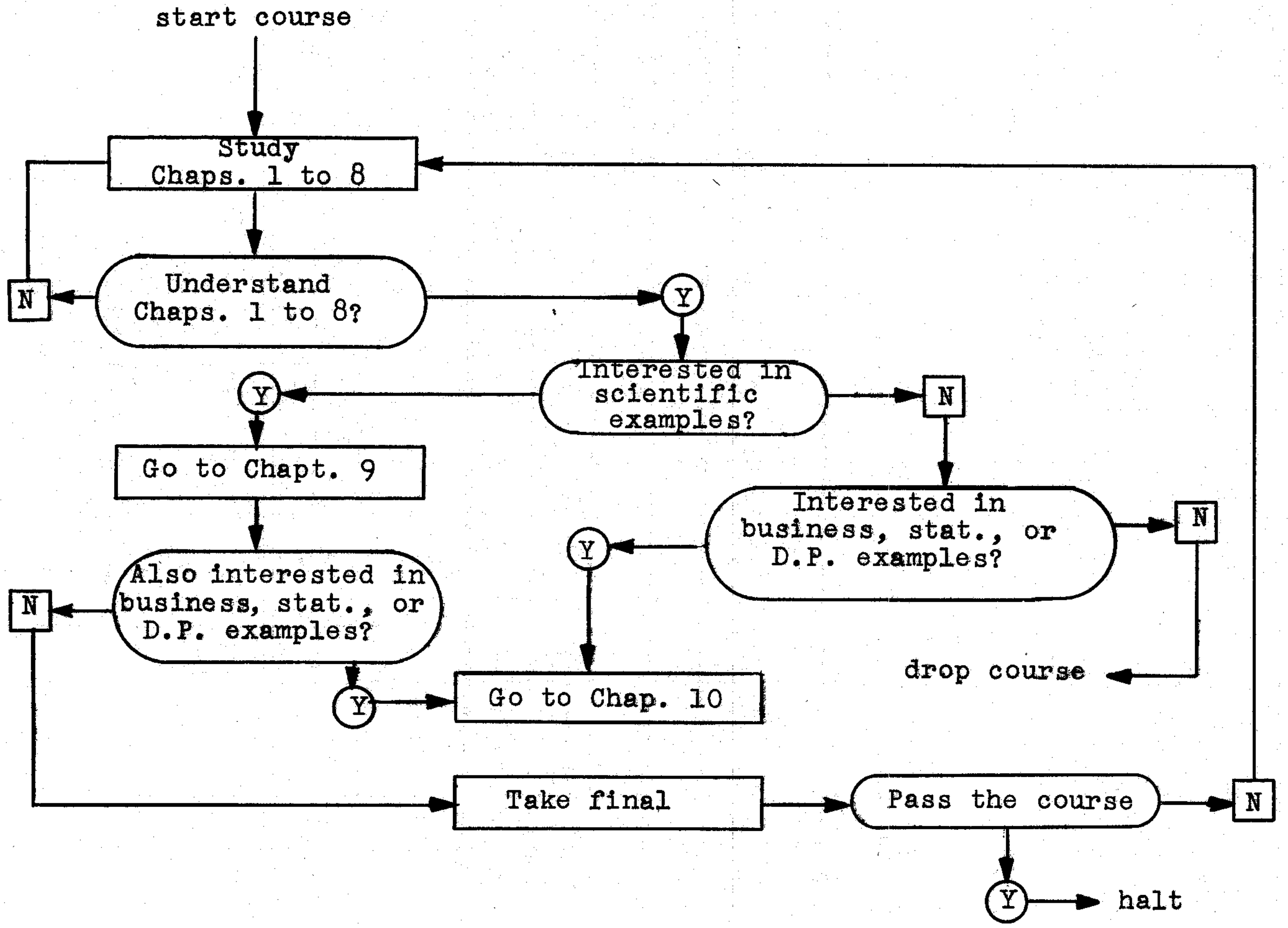
If the word base of register 3 contains 375 and the channel base contains 1172, the execution of the command:

NOTES	LOCATION	K	OP	ADDRESS	ACCUMULATOR
		3	09	10 21	

enters 375 + 1172 + 1021 = 2568 into IRA.

8.5 An Important Flow Chart

Before continuing, the student should pause at this point and consider the following flow chart.


start course
Study Chaps. 1 to 8
Understand Chaps. 1 to 8?
N
Y
Interested in scientific examples?
Y
N
Go to Chapt. 9
Interested in business, stat., or D.P. examples?
Y
N
Also interested in business, stat., or D.P. examples?
N
Y
Go to Chap. 10
drop course
Take final
Pass the course
N
Y
halt

Chapter 9

Scientific Examples

9.1 Solution of Quadratic Equations

NOTES	LOCATION	K	OP	ADDRESS		ACCUMULATOR
transfer to 0920	0900		29	09	20	?
position paper	0920		30	00	02	?
cl and add a = (0970)	21		42	09	70	a
add a	22		43	21	73	2a
store 2a = (0973)	23		49	09	73	2a
mult. by c = 0972)	24		44	09	72	2ac
add 2ac	25		43	21	73	4ac
store 4ac = (0974)	26		49	09	74	4ac
cl and add b = (0971)	27		42	09	71	b
mult. by b	28		44	21	73	b^2
sub. 4ac	29		41	09	74	$b^2 - 4ac$
trans. if acc < 0	30		22	09	41	$b^2 - 4ac \geqq 0$
sq. rt. sub. in ch. 10	31		w3	10	00	$\sqrt{b^2 - 4ac} = \sqrt{D}$
store $\sqrt{D}$ = (0975)	32		49	09	75	$\sqrt{b^2 - 4ac} = \sqrt{D}$
cl and sub. b	33		40	09	71	-b
add $\sqrt{D}$	34		43	09	75	$-b + \sqrt{D}$
div. by 2a	35		48	09	73	r_1
type r_1	36		33	21	73	r_1
cl and sub. b	37		40	09	71	-b
sub. $\sqrt{D}$	38		41	09	75	$-b - \sqrt{D}$
div. by 2a	39		48	09	73	r_2
type r_2	40		38	21	73	r_2
halt	41		67	00	00	"

9.1 (continued)

The preceding program for Intercom 500 will solve the quadratic equation: $ax^2 + bx + c = 0$ for real roots $r_1 = (-b + \sqrt{D})/2a$, $r_2 = (-b - \sqrt{D})/2a$, $D = b^2 - 4ac \geqq 0$. If roots are complex, program will halt without type-out.

Problems:

1. Write a program to find complex roots as well as real roots for quadratic equations.
2. Write a program to solve equation $ax^4 + bx^2 + c = 0$
3. Write a program to solve $ax^{2/3} + bx^{1/3} + c = 0$
4. Use DeMoivre's theorem to solve equation $x^n = a + bi$.
5. Write a program to solve $x^3 + ax + b = 0$ for real roots.
6. Write a program for 1000DP to solve a quadratic equation. Gate for type-in of a, b and c.

9.2 Evaluation of an Integral

The following program for Intercom 1000DP will find approximation to the integral:

$$I = \int_a^b f(x)dx \approx \Delta x \mathop{}_{i=1} f(a+i\ \Delta x), \qquad x = \frac{b-a}{n}$$

The command in 0913 may be a transfer command (29) to location to find f(x) or mark place and transfer or 08 command to find f(x) by subroutine.

DATA ALLOCATION CHART

Constants			Variables		
Loc.	Symbol	Value	Loc.	Symbol	Starting value
0950	a		0958	x_i	a
52	b		60	n - i	n
54	n		66	$\Sigma f(x_i)$	0
56	Δx				
62	1	1			
64	0	0			

9.2 (continued)

Problem: $I = \int_a^b f(x)dx$

NOTES	LOCATION	K	OP	ADDRESS	ACCUMULATOR
position paper	0900		30	00 02	?
cl and add 0 = (0964)	1		42	09 64	0
store 0 = $\Sigma f(x_i)$ = (0966)	2		49	09 66	0
cl and add b = (0952)	3		42	09 52	b
sub a = (0950)	4		41	09 50	b - a
div by n = (0954)	5		48	09 54	Δx
store Δx = (0956)	6		49	09 56	Δx
add a	7		43	09 50	a + Δx
store a + Δx = (0958)	8		49	09 58	a + Δx
clear and add n	9		42	09 54	n
sub 1	10		41	09 62	n - 1
store n - k = (0960)	11		49	09 60	n - 1
cl and add x_i	12		42	09 58	x_i
find $f(x_i)$	13				$f(x_i)$
add $\Sigma f(x_i)$	14		43	09 66	$\Sigma f(x_i)$
store back	15		49	09 66	$\Sigma f(x_i)$
cl and add a + (k - 1) Δx	16		42	09 58	a + (k - 1) Δx
add Δx	17		43	09 56	a + k Δx
store back	18		49	09 58	a + k Δx
cl and add n - k	19		42	09 60	n - k
sub 1	20		41	09 62	n - k - 1
store back	21		49	09 60	n - k - 1
trans if acc $\geqq$ 0	22		20	09 12	n - k < 0
cl and add $\Sigma f(x_i)$	23		42	09 66	$\Sigma f(x_i)$
mult by Δx	24		44	09 56	I
type I	25		38	21 00	I
halt	26		67	00 00	I

9.2 (continued)

Problems:

1. Modify example 17 to use $i = 0$ to $n - 1$.
2. Write a program to evaluate an integral using trapezoidal rule.
3. Same using Simpson's rule.
4. Write a program for $f(x) = x-\cos x +.\ln x$ then find integral from $x = 1$ to 2.
5. Write a program to evaluate integral of e^{-x^2} from 0 to 1.

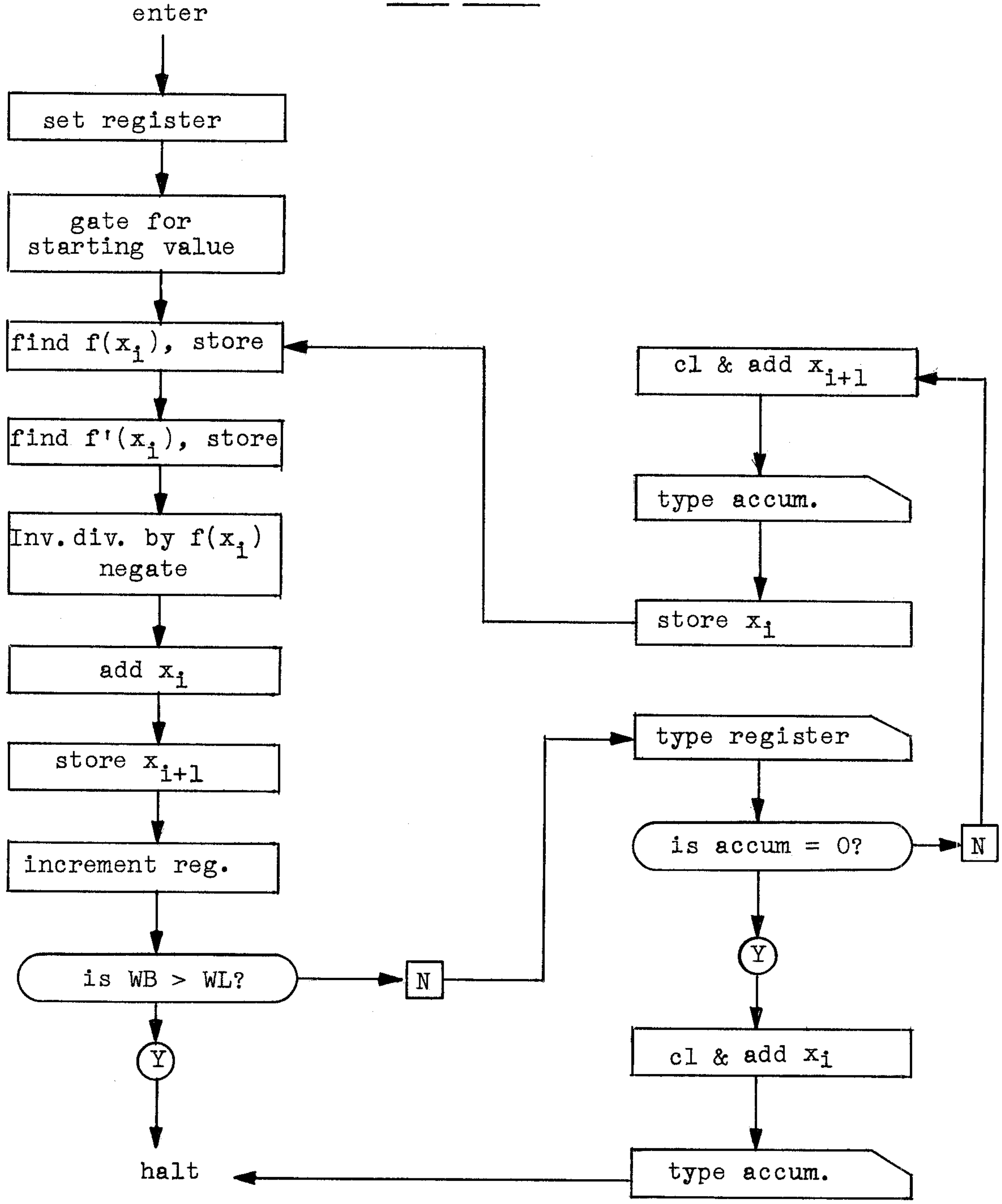
9.3 Newton's Method for Solving Equations

The following program for Intercom 500 uses Newton's method to find the roots of an equation, $x_{i+1} = x_i - f(x_i)/f'(x_i)$. As illustrated here, this program is used to solve the equation $x^2 - \cos x = 0$. This program could be used to find roots of some other equation by writing a program for f(x) starting at 1020 and concluding with the command 160000 and a program for f'(x) starting at 1120 and concluding with the command 180000.

Two samples of output are included. One has a starting point close to the root, the other with a starting point further away.

9.3 (continued)

FLOW CHART



9.3 (continued)

PROGRAM

```
900   .0300002    934   .0420974
901   .1700000    935   .0382173
902   .1710001    936   .0490971
903   .1720020    937   .0290920
904   .1730000    938   .0420971
905   .0510971    939   .0382173
906   .0300002    940   .0670000
907   .0420971
908   .0290920    1020  .0491071
                  1021  .0081226
920   .0261020    1022  .0491072
921   .0490972    1023  .0421071
922   .0420971    1024  .0441071
923   .0281120    1025  .0411072
924   .0490973    1026  .0160000
925   .0402173
926   .0470972    1120  .0491171
927   .0430971    1121  .0081242
928   .0490974    1122  .0491172
929   .0410971    1123  .0421171
930   .1760932    1124  .0432173
931   .0670000    1125  .0431172
932   .1310000    1126  .0180000
933   .0230938
```

STORAGE ALLOCATION

(0971) = x_i

(0972) = $f(x_i)$

(0973) = $f\ (x_i)$

(0974) = x_{i+1}

(1071) = x_i

(1072) = $\cos x_i$

(1171) = x_i

(1172) = $\sin x_i$

sin-cos subroutine
in channel 12.

SAMPLES OF OUTPUT

```
690900  s

 971  /8   s 50.80000

  1            .8244700
  2            .8241300
  3            .8241300

690900  s

 971  /1   s 50.10000

  1           3.3852000
  2           1.4814000
  3            .9496100
  4            .8317200
  5            .8241600
  6            .8241300
  7            .8241300
```

Problems:

1. Write a DP program for Newton's method.
2. Write a program to find roots of a cubic equation by Newton's method.
3. Write a program to find roots of a cubic equation by Horner's method.
4. Write a program to find roots of $x^3 = e^{-x}$.
5. Write a program to find roots of $x^2 + \ln x = 0$.

9.4 Subroutine for Square Root

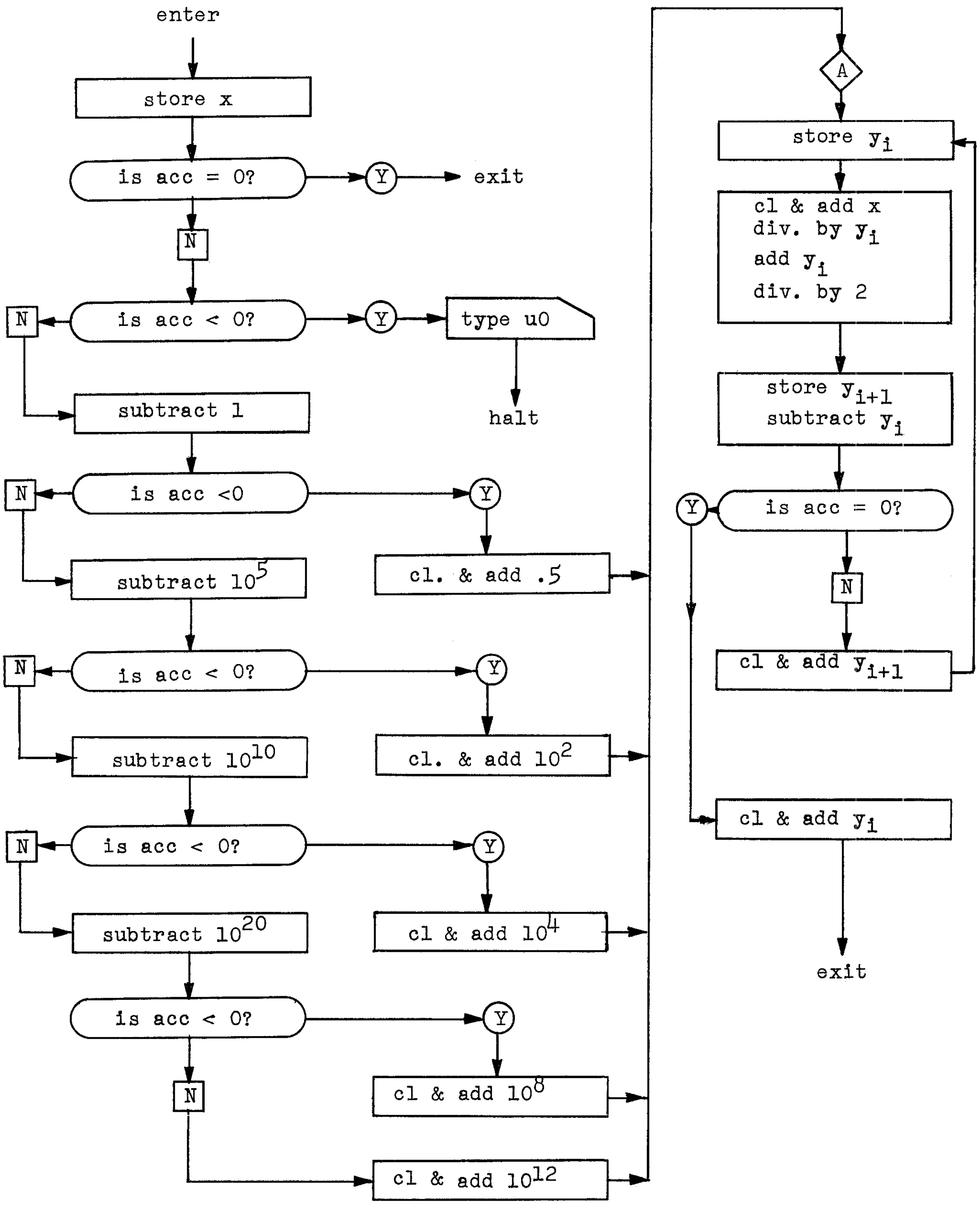
The following program written as a subroutine for Intercom 500 will find the square root of the number which is in the accumulator on entry. To use this subroutine: clear and add number of which the square root is desired. Then use the command 261500. If the number is negative, the computer will type u0 and halt. Otherwise control will be returned to the marked place with the square root in the accumulator. This subroutine is, of course, not as fast as the machine language subroutine, but it gives the student an opportunity to see how a subroutine works.

An alternate entrance is provided at 1535 so that this program may be used from manual. If we start automatic operation at 1535, the computer will gate for a number, then type square root, and then gate for another number.

The portion of the program up to the point A of the flow chart is to give the computer a convenient starting point. Because of the wide range of values used by Intercom, several starting values are used.

9.4 (continued)

FLOW CHART



9.4 (continued)

PROGRAM

$y = x \qquad y_{i+1} = \frac{1}{2}(y_i + x/y_i)$

```
1500   .0491571
1501   .0231532
1502   .0221533
1503   .0411583     1520   .0491572
1504   .0221519     1521   .0421571
1505   .0411574     1522   .0481572
1506   .0221517     1523   .0431572
1507   .0411575     1524   .0481582
1508   .0221515     1525   .0491573
1509   .0411576     1526   .0411572
1510   .0221513     1527   .0231531
1511   .0421581     1528   .0421573
1512   .0291520     1529   .0491572
1513   .0421580     1530   .0291520
1514   .0291520     1531   .0421572
1515   .0421579     1532   .0160000
1516   .0291520     1533   .03100u0
1517   .0421578     1534   .0670000
1518   .0291520     1535   .0512173
1519   .0421577     1536   .0261500
                    1537   .0382173
                    1538   .0291535
```

STORAGE

1571	x_i	1577	.5
1572	y_i	1578	100
1573	y_{i+1}	1579	10^4
1574	10^5	1580	10^8
1575	10^{10}	1581	10^{12}
1576	10^{20}	1582	2
		1583	1

Problems:

Write as a subroutine, a program to find:

1. Square root (double precision).
2. Sine and cosine.
3. Arctangent.
4. Exponential.
5. Logarithm.
6. Binomial coefficients.
7. Hyperbolic functions.

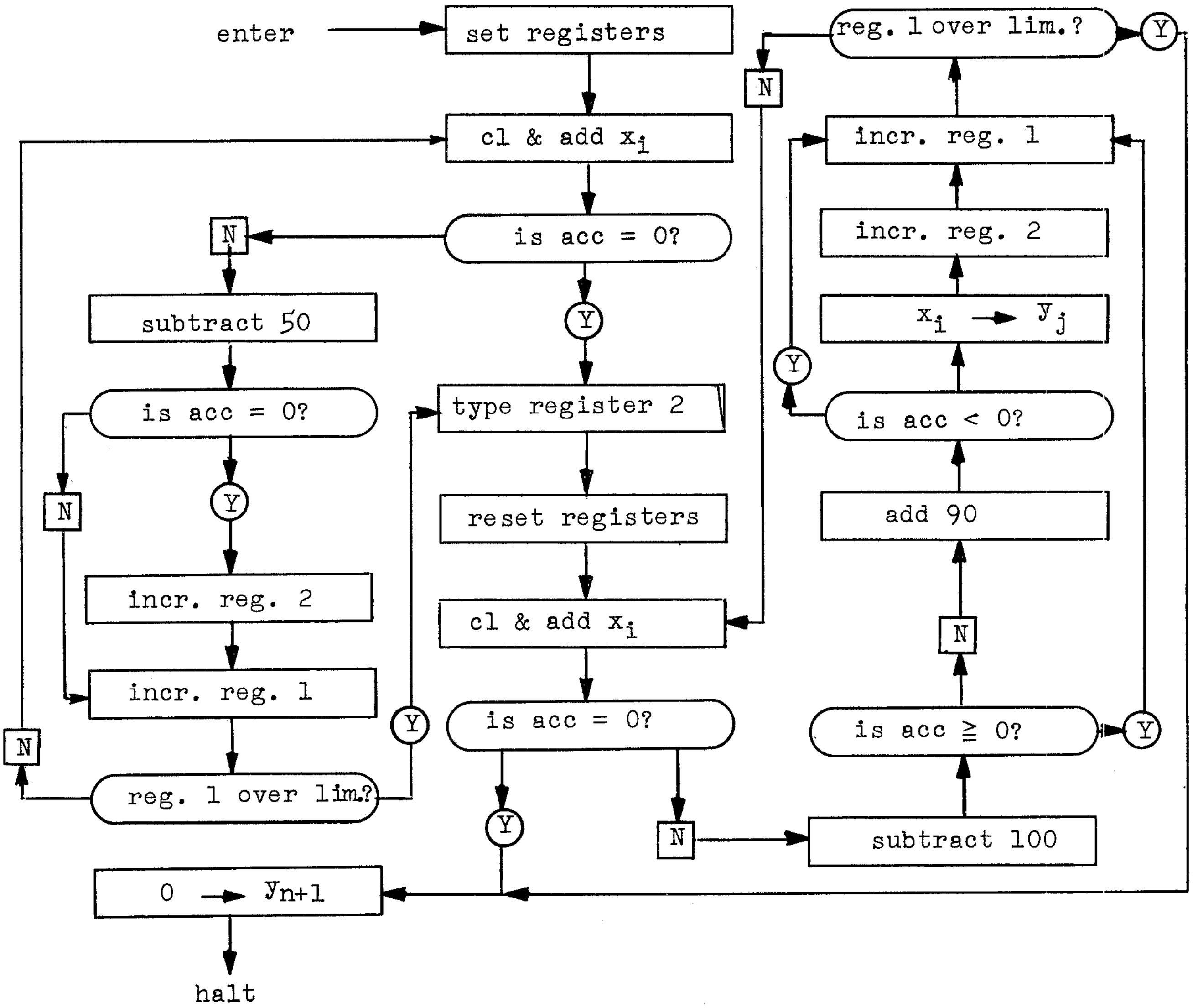
Chapter 10

Business, Statistical and Data Processing Examples

10.1 Search

One of the basic problems encountered in data processing is to search a set of data to determine if any are of a given value or in a certain range of values.

Assume that channel 10 has 100 non-zero values stored in it, or a smaller number of values with zero, as a flag, after the last one. The following SP program will determine how many of these values, if any, are exactly 50, and then store all those which are 10 or more but less than 100 in channel 11, and then store a zero after the last.



10.1 (continued)

```
  PROGRAM
0900  300002     0910   230920     0920  2310000     0930  2491100
   1 1700000       11   410971       21  1700000       31  2760932
   2 1710001       12   230914       22  2700000       32  1760923
   3 1720099       13   290915       23  1421000       33   420974
   4 1730000       14  2760915       24   230934       34  2491100
   5 2700000       15  1760909       25   410972       35   670000
   6 2710001       16   290920       26   200932
   7 2720099                         27   430973        STORAGE
   8 2730000                         28   220932     0971     50
   9 1421000                         29  1421000       72    100
                                                       73     90
                                                       74      0
```

Problems:

1. Write a DP program similar to the example of this section.

2. Starting balances of accounts 1100 to 1105 are in those locations. Numbers are stored in channel 10 in pairs. The even location is the account number and the following odd numbered location is the amount of the transaction. If the number of pairs is less than 50, zero is stored in the even location after the last one. Write a SP program to add amount of transactions to proper account numbers. Type account numbers and new balances.

3. Similar to problem 2, but transactions are on tape rather than in channel 10.

10.2 Sort

Another problem often encountered in data processing is that of sorting a set of numbers which are in random order and putting them in ascending or descending order.

The following SP program will arrange the numbers in the first 20 locations of channel 10 in ascending order.

This program accomplishes the sorting by successively comparing the number in two consecutive locations and placing the smaller in the lower numbered location. On each pass, numbers are shifted. After one less pass than the number of items, the items will be in correct order. The reader should make up an example similar to the following one, and follow it through step by step.

original order	first pass	second pass	third pass	fourth pass
7	5	5	5	4
5	7	7	4	5
8	8	4	7	7
9	4	8	8	8
4	9	9	9	9

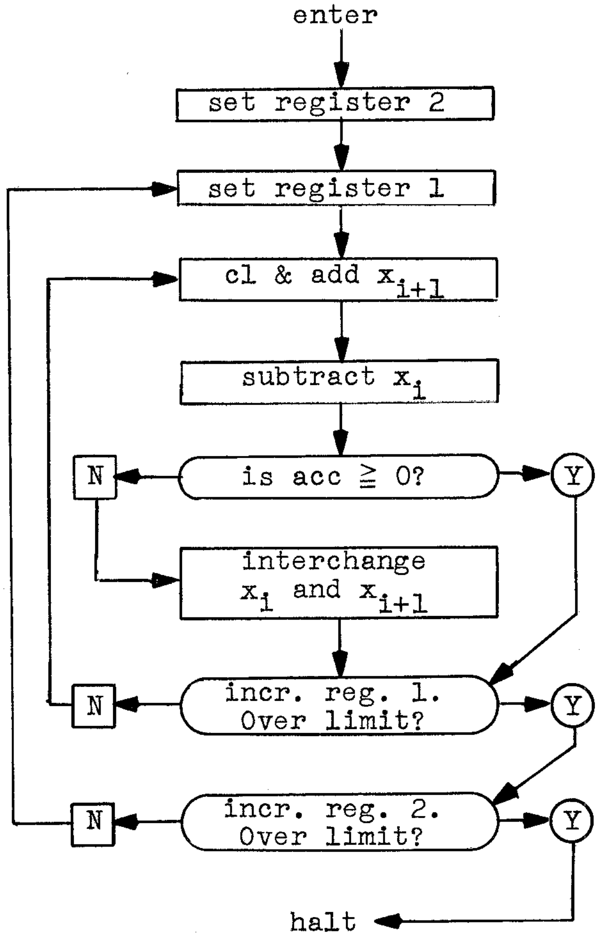
10.2 (continued)

Program

```
0913   2731000
  14   2720018
  15   2710001
  16   2700000
  17   1731000
  18   1720018
  19   1710001
0920   1700000
  21   1420001
  22   1410000
  23    200930
  24   1420000
  25    490910
  26   1420001
  27   1490000
  28    420910
  29   1490001
0930   1760921
  31   2760920
  32    670000
```

Storage

```
0910   Temp.
```



Problems:

1. Write a DP program to sort a set of data.

2. Write a sorting program which first finds the smallest number in the set, then finds the smallest in those remaining, etc.

3. Find the median of a set of values.

4. Write a program to find the largest (L) and the smallest (S) of a set of numbers, then use these to find the range (L-S) and the mid-range: $\frac{1}{2}$(L+S).

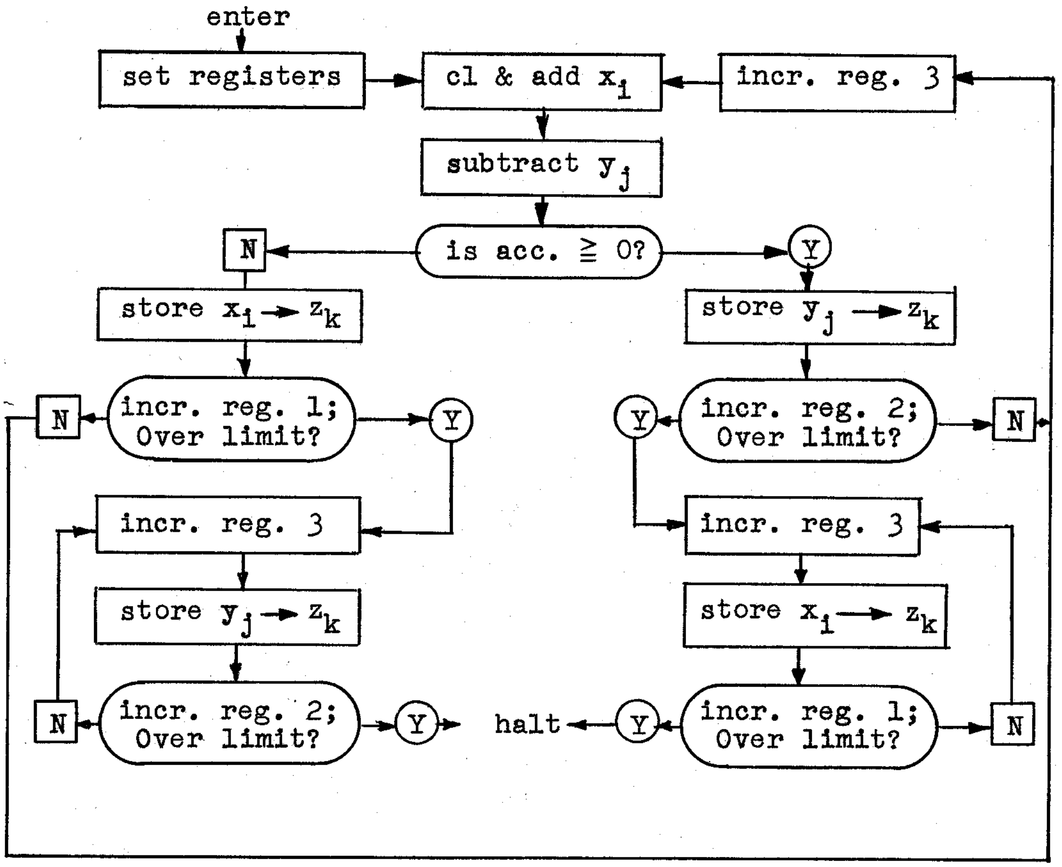
10.3 Merge

It is sometimes desirable to put two sets of data, which are each in order into a new set which is in order. This process is called merging or collocating. (Unfortunately, it is frequently called "collating," but this is a misuse of the word.)

Twenty items in ascending order are in channel 10 and 25 items also in ascending order are in channel 11. The following SP program will put these together in ascending order and store them in channel 12.

```
0900    1700000
   1    1710001
   2    1720019
   3    1731000
   4    2700000
   5    2710001
   6    2720024
   7    2731100
   8    3700000
   9    3710001
0910    3720099
  11    3731200
  12     290921

0920    3760921
  21    1420000
  22    2410000
  23     200932
  24    1420000
  25    3490000
  26    1760920
  27    3760928
  28    2420000
  29    3490000
0930    2760927
  31     670000
  32    2420000
  33    3490000
  34    2760920
  35    3760936
  36    1420000
  37    3490000
  38    1760935
  39     670000
```


enter
set registers
cl & add x_i
incr. reg. 3
subtract y_j
N
is acc. ≧ 0?
Y
store $x_i \rightarrow z_k$
store $y_j \rightarrow z_k$
N
incr. reg. 1; Over limit?
Y
Y
incr. reg. 2; Over limit?
N
incr. reg. 3
incr. reg. 3
store $y_j \rightarrow z_k$
store $x_i \rightarrow z_k$
N
incr. reg. 2; Over limit?
Y
halt
Y
incr. reg. 1; Over limit?
N

Problems:

1. Write an SP program to merge two sets of data which are in descending order.

2. Write a DP program to merge two sets of data.

3. Write a program to merge three sets of data.

4. Twenty pairs of numbers are in channel 10. They are arranged in order of magnitude of contents of the even locations. Twenty-five similar pairs are in channel 11. Merge them and store in channel 12.

10.4 Business Examples

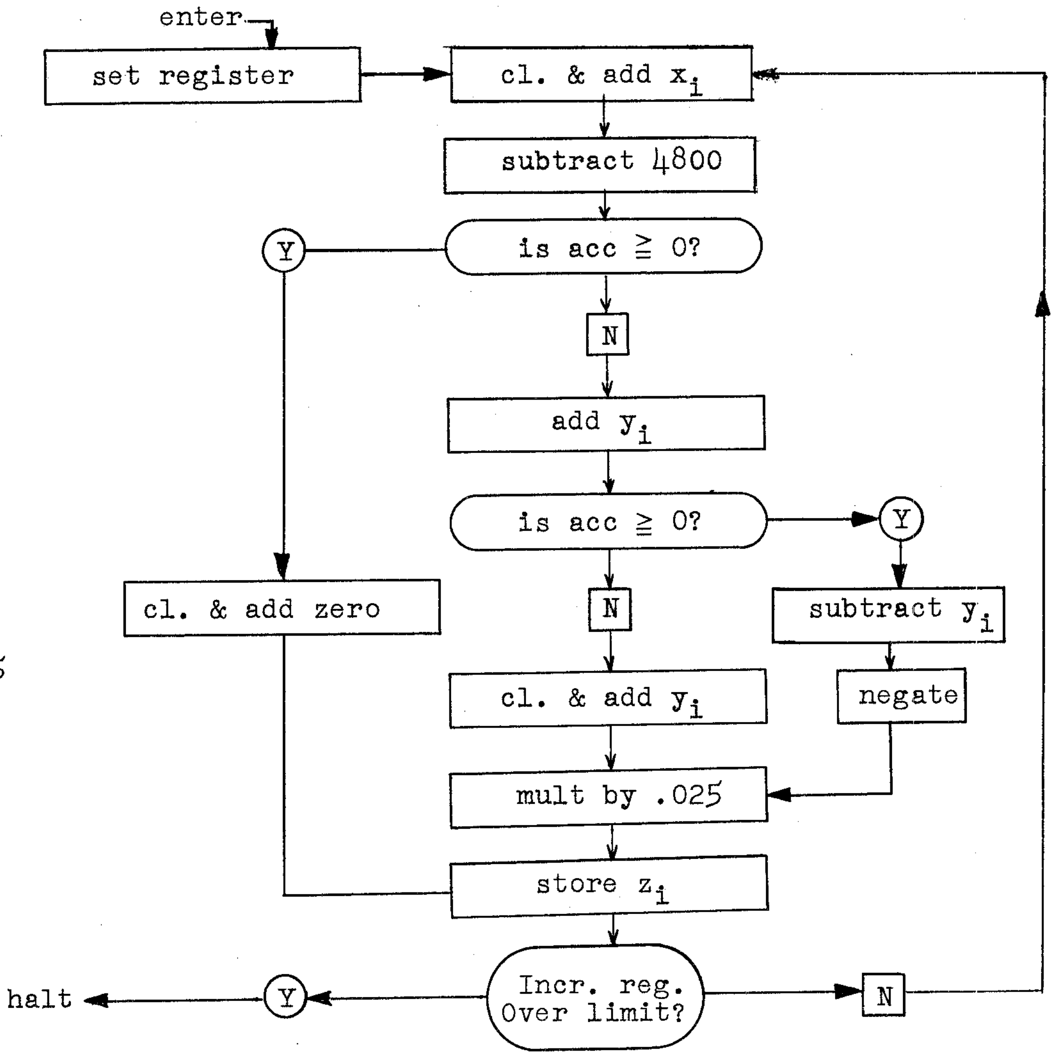
The Flim Z Building Co. has 25 employees. Their gross wages to date, not including this week's, are in channel 10 (x_i). This week's wages are in channel 11 (y_i). Social Security tax is 2½% of the first $4,800. The following DP program will calculate Social Security tax for each employee and store in channel 12 (z_i).

Program

```
0900   1700000
   1   1710002
   2   1720048
   3   1730000
   4   1421000
   5    410920
   6    200914
   7   1431100
   8    200916
   9   1421100
0910    440922
  11   1491200
  12   1760904
  13    670000
  14    420924
  15    290911
  16   1411100
  17    402100
  18    290910
```

Storage

```
0920   4800.
  22       .025
  24      0
```



Problems:

1. The Schock M. Goode Electric Co. has 100 customers. Meter readings for last month are in channel 10, this month in 11. Rates are $.08 per kwh. with a minimum of $2. Write a SP program to type meter reading, amount used and cost.

2. The R. E. Klein Furniture Co. stocks 100 items. Number of each item on hand at beginning of period, number sold, number added to stock, and cost are stored in separate channels. Write a SP program to type these numbers and the number of each item on hand at the end of the period, and the value at beginning and end of period. Include totals of values at beginning and end of period.

10.4 (continued)

Problems: (cont.)

3. The M. Bezzlar Trust & Loan Co. has 100 accounts. Starting balance is in channel 10. A tape has data in pairs. The even numbered location is account number, and the odd numbered location is the amount of the transaction. Zero is stored in the even location after the last transaction. Write a SP program to bring the accounts up to date and type the results.

4. Write a DP program for one of the above.

5. Modify problem 1 to include some accounts which are commercial with a rate of $.06 and minimum of $5.

6. Modify problem 2 to include a flag if number in stock falls below a certain minimum.

7. Consider the problem involved in a given business and write a program to solve it.

10.5 Finance Examples

Sometimes it is desired to drop the fractional part of a number. This can be done in Intercom with the following procedure. Clear and add N, add B, subtract B, where B is 65.4 in floating point for double precision and 56.6 for single precision. If N is a positive number, after this procedure, only the whole number part will be in the accumulator.

In business problems, we often wish to round off to the closest cent. The procedure would then be: Clear and add N, add .005, multiply by 100, add B, subtract B, divide by 100.

The following program written as a subroutine will find

$$(1+i)^n, \ (1+i)^{-n}, \ \frac{(1+i)^n-1}{i}, \ \frac{1-(1+i)^{-n}}{i} .$$

The logarithm subroutine is in channel 17 and the exponential subroutine is in channel 18. Steps in use of this subroutine are:

1. store n in 1628

2. clear and add i

3. transfer to subroutines:

for	use command
$(1+i)^n$	26 1600
$(1+i)^{-n}$	26 1605
$\frac{(1+i)^n-1}{i}$	28 1611
$\frac{1-(1+i)^{-n}}{i}$	28 1616

10.5 (continued)

Subroutine for $(1+i)^n$, $(1+i)^{-n}$, $\frac{(1+i)^n-1}{i}$, $\frac{1-(1+i)^{-n}}{i}$

NOTES	LOCATION	K	OP	ADDRESS		ACCUMULATOR
add 1 = 1630	1600		43	16	30	$1 + i$
log	1		08	17	71	$\log(1 + i)$
mult by n = 1628	2		44	16	28	$n \log(1 + i)$
exp.	3		08	18	72	$(1 + i)^n$
ret. to marked place I	4		16	00	00	$(1 + i)^n$
add 1	1605		43	16	30	$1 + i$
log	6		08	17	71	$\log(1 + i)$
mult by n	7		44	16	28	$n \log(1 + i)$
negate	8		40	21	00	$-n \log(1 + i)$
exp.	9		08	18	72	$(1 + i)^{-n}$
ret. to marked place I	10		16	00	00	$(1 + i)^{-n}$
store i = 1626	1611		49	16	26	i
trans to 1600 (mark I)	12		26	16	00	$(1 + i)^n$
sub 1	13		41	16	30	$(1 + i)^n-1$
div by i	14		48	16	26	$[(1+i)^n -1]/i$
ret. to mark II	15		18	00	00	"
store i = 1626	1616		49	16	26	i
trans to 1605 (mark I)	17		26	16	05	$(1 + i)^{-n}$
negate	18		40	21	00	$-(1 + i)^{-n}$
add 1	19		43	16	30	$1-(1 + i)^{-n}$
div by i	20		48	16	26	$[1 - (1+i)^{-n}]/i$
ret. to mark II	21		18	00	00	"

10.5 (continued)

Problems:

1. Write a program to find value at compound interest, present value, amount of an annuity of 1, present value of an annuity of 1, and the reciprocal of the latter. Gate for interest rate and number of periods.

2. Same as problem 1, but type for number of periods from 1 to 1200 for a given interest rate.

3. Write a program to type an amortization schedule.

4. Write a program to type a sinking fund schedule.

5. Write a program to type a depreciation schedule.

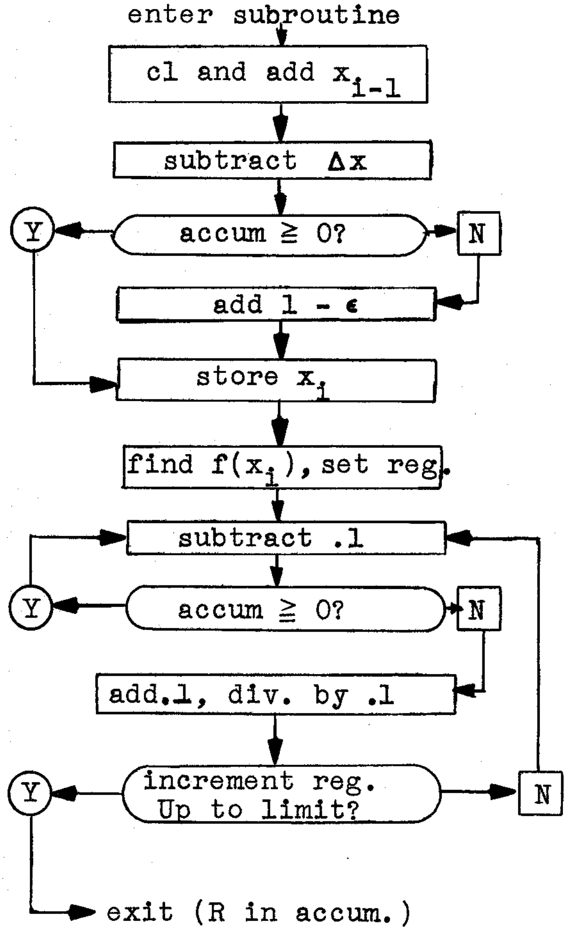
10.6 Random Numbers

Random numbers are used in statistical work. These numbers are often used to simulate sampling experiments. On the computer, it is easier to generate the random number than to store a table of such numbers. These numbers are usually generated by using some function and omitting a given number of the most significant digits.

The following program written as a subroutine to be used in any channel (CH) in 1000DP will find a random number regardless of contents of accumulator by using 26 CH 00. A starting value of x_o is first stored in CH 98, then subroutine is entered. This subroutine uses register u, and this should not be used for another purpose.

$$f(x) = \frac{x}{2} - \frac{x^2}{4} + \frac{x^3}{8}$$

Data:	(CH 88) = .1
	(CH 90) = 1
	(CH 92) = 2
	(CH 94) = $1-\epsilon$ = .999876789997
	(CH 96) = Δx = .100000098967
Storage:	(CH 98) = x
	(CH 86) = x/2



10.6 (continued)

Subroutine for random numbers

NOTES	LOCATION	K	OP	ADDRESS		ACCUMULATOR
cl and add x_{i-1}	CH00		42	CH	98	x_{i-1}
sub Δx = (CH96)	1		41	CH	96	$x_{i-1} - \Delta x = x_i$ if > 0
trans if accum ≧ 0	2		20	CH	04	$x_{i-1} - \Delta x$
add $1 - \epsilon$ = (CH94)	3		43	CH	94	$x_{i-1} - \Delta x + 1 - \epsilon$
store x_i	4		49	CH	98	"
div by 2 = (CH92)	5		48	CH	92	$x_i/2$
store x/2 = (CH86)	6		49	CH	86	"
sub 1 = (CH90)	7		41	CH	90	$x/2 - 1$
mult by x/2	8		44	CH	86	$x/2(-1 + x/2)$
add 1	9		43	CH	90	$1 + x/2(-1 + x/2)$
mult by x/2	10		44	CH	86	f(x)
	11	u	73	00	00	"
set	12	u	72	00	02	"
register	13	u	71	00	01	"
	14	u	70	00	00	"
sub .1 = (CH88)	15		41	CH	88	f(x) - k(.1)
trans if acc ≧ 0	16		20	CH	15	"
add .1	17		43	CH	88	"
div by .1	18		48	CH	88	"
incr. reg.	19	u	76	CH	15	"
return to mark I	20		16	00	00	"

10.6 (continued)

The following program will type random numbers found in the preceding subroutine, five on a line.

Problem: Type random numbers, five on a line.

NOTES	LOCATION	K	OP	ADDRESS		ACCUMULATOR
position paper	0900		30	00	02	?
gate for x_0	1		51	CH	98	?
space	2		30	00	02	?
	3	1	73	00	00	?
set	4	1	72	00	04	?
register	5	1	71	00	01	?
	6	1	70	00	00	?
random number	7		26	CH	00	R
type R	8		33	21	00	R
	9	1	76	09	07	"
	10		30	00	01	"
	11		29	09	06	"

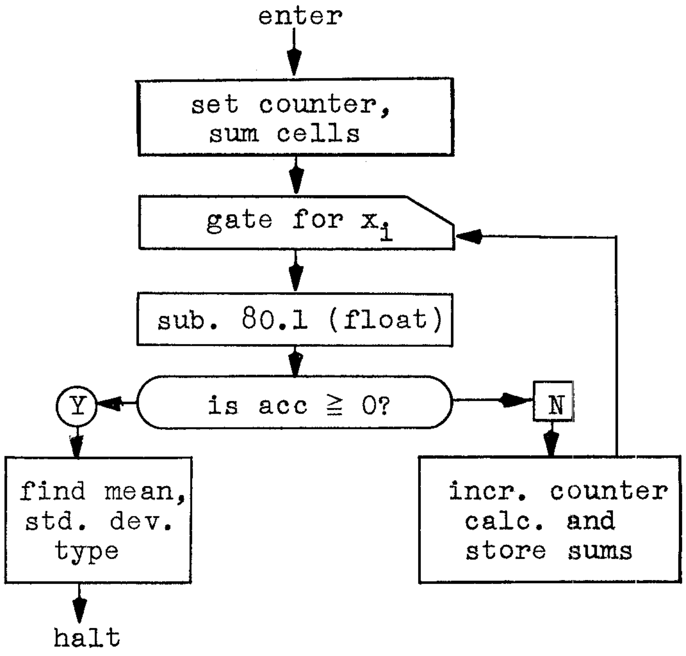
Problems:

1. Write a subroutine using sin x, $0<x<\pi/4$, to find random numbers.
2. Write a program using another function of x to find random numbers.
3. Write a program to simulate coin tossing.
4. Write a program to simulate die rolling.
5. Write a program to simulate sampling from a binomial population.
6. Write a program to test random numbers found above or to test one of the sampling problems.

10.7 Statistical Examples

The following DP program will calculate and type the average and standard deviation of a set of numbers. The computer will gate for type-in of the data. After the last 811 should be typed. Then calculations will be made and results typed out. A number larger than any in the data may be used to exit from a loop when the data may include zeros.

```
0900  300002     0923  480952
   1  420950       24  332100
   2  490952       25  442100
   3  490954       26  490958
   4  490956       27  420956
   5  512100       28  480952
   6  490958       29  410958
   7  410948     0930  081297
   8  200920       31  332100
   9  420952       32  670000
0910  430946
  11  490952     0946     1
  12  420958       48  80.1(float)
  13  430954       50    0
  14  490954
  15  420958     0952     n
  16  440958       54  Σx
  17  430956       56  Σx^2
  18  490956       58    x, m^2
  19  290905
0920  300002     sq. rt. sub. in
  21  330952     channel 12.
  22  420954
```



Problems:

1. Write a program to calculate a type the average and standard deviation of the data in channel 10.

2. Write a program to find correlation coefficient.

3. Write a program using random numbers to find random normal numbers using:

$$y = x - (2.30753 + 0.27061x)/(1 + 0.99229x + 0.04481x^2),$$

$$x = \sqrt{\ln (1/R^2)}, \quad 0 < R < .5$$

See "inverse error function" in Hastings, "Approximations for Digital Computers."

4. Use random normal numbers to simulate sampling experiments.

5. Write a program for test of hypotheses, Chi Square test, or other statistical test.

Appendix I

COMMAND LIST FOR INTERCOM 500 AND 1000

Special		
06	Type location of last command executed	(5.3)
08	Perform subroutine	(7.5)
Return		
16	Return to marked place I	(7.2)
18	Return to marked place II	(7.2)
Transfer		
20	Transfer if accumulator ≧ 0	(4.1)
22	Transfer if accumulator < 0	(4.1)
23	Transfer if accumulator = 0	(4.1)
26	Mark place and transfer I	(7.2)
28	Mark place and transfer II	(7.2)
29	Transfer control	(3.6)
Output		
30	Position typewriter paper	(3.1)
31	Type tabulating number	(3.3)
32	Type floating point and tab	(3.3)
33	Type fixed point and tab	(3.3)
34	Type float, return carriage	(3.3)
35	Type command	(5.3)
38	Type fixed, return carriage	(5.3)
39	Punch paper tape	(2.7)

Arithmetic		
40	Clear and subtract	(3.1)
41	Subtract	(3.1)
42	Clear and add	(3.1)
43	Add	(3.1)
44	Multiply	(3.3)
45	Clear and add absolute value	(3.1)
47	Inverse divide	(3.3)
48	Divide	(3.3)
49	Store	(3.1)
Input		
50	Store commands	(2.7)
50	Gate for command	(3.6)
51	Store fixed point data	(2.7)
51	Gate for fixed point data	(3.6)
52	Store floating point data	(2.7)
52	Gate for floating point data	(3.6)
55	Read punched paper tape	(2.7)
Start and Stop		
61	Start list (trace routine)	(5.4)
62	Stop list (trace routine)	(5.4)
63	Ring bell	(3.6)
67	Halt and return to manual	(3.1)
68	Breakpoint halt	(3.6)
69	Start automatic operation	(2.7)
Index registers		
70	Set word base	(6.2)
71	Set word difference	(6.2)
72	Set word limit	(6.2)
73	Set channel base	(6.2)
74	Set channel difference	(6.2)
75	Set channel limit	(6.2)
76	Increment word base, test	(6.3)
77	Increment channel base, test	(6.4)

Additional commands available with Intercom 500 or 1000DP

02	Perform subroutine	(7.5)	78	Clear and add index register	(8.4)
81	Block copy	(8.3)	79	Store index register accum.	(8.4)

Additional commands available with Intercom 500

09	Set index register accum.	(8.4)	w3	Perform subroutine	(7.5)
37	Type hexadecimal number	(5.3)			

Numbers in parentheses are section in which command is introduced.

Appendix II

GLOSSARY OF TERMS USED IN DIGITAL COMPUTING

Access time: The time required to locate a word in storage and transfer it to the arithmetic or control unit.

Accumulator: A register used to hold the result of arithmetic operations or to hold words to be tested. Specific use varies with each machine.

Adder: The electronic circuitry which will form the sum of two words.

Address: A number which identifies a unique storage location in the computer.

Alphanumeric machine: A computer capable of handling both alphabetic and numerical information.

Arithmetic unit: The portion of the machine which accomplishes the arithmetic operations.

Assembly routine: A routine which causes a computer to translate a program from symbolic language to machine language, such that there is a 1 to 1 correspondence between symbolic and machine language instructions. (See compiler)

Binary coded decimal: A method of representing decimal digits by a pattern of bits.

Binary numbers: Numbers in the number system with base two.

Bit: A binary digit, either 0 or 1.

Branching: The selection of one of two alternative instruction sequences according to the result of a numerical test or comparison during automatic computation.

Breakpoint halt: A halt within a routine, usually used for debugging.

Appendix II (continued)

Buffer: A register used for temporary storage to allow computation to continue while the stored data is otherwise used. Usually employed during input and output.

Calling sequence: A series of instructions to provide information (parameters) for a subroutine. These parameters may be the addresses of numbers or they may be the numbers themselves on which the subroutine will operate.

Character: A symbol transmitted within a computer by a combination of bits. Usually a decimal digit, letter, or punctuation mark.

Check sum: The numerical sum of a block of words. Often used to check the accuracy of input or output.

Code: (Verb) The act of writing coded instructions that a computer will follow in solving a problem.

Command: (See <u>instruction</u>)

Compiler: A routine which enables a computer to translate a program in symbolic language into machine language. One compiler instruction will in general require several machine language instructions to perform the required operation. (See assembly program)

Debug: (Verb) To eliminate errors from a program.

Double precision arithmetic: Arithmetic in which the computer uses 2 words for each piece of data allowing twice the usual significance.

Fixed point: The system in which all digits in the machine carry significance and the position of the decimal or binary point is fixed.

Floating point: The system of carrying numbers in a computer in 2 parts: (1) a fraction part (mantissa) and (2) an exponent (characteristic) which shows the power of ten by which the decimal part is to be multiplied to obtain the actual number. Contrast with fixed point.

Hexadecimal number system: The number system with base 16.

High-order digit: The digit at the extreme left end of a number

Appendix II (continued)

Index register: A register which acts as a counter to determine the number of times a given routine will be executed and as an address modifier to change the form of instructions before they are executed.

Input unit: The device which accomplishes the entry of numbers to the storage unit of the machine.

Instruction: A word which is to be interpreted by the computer such that a desired operation will take place.

Interpretive routine: A machine language routine which may be regarded as a series of subroutines, one of which will be executed upon encountering a "pseudo-instruction" in the program. The "pseudo-instruction" is a symbol not a part of the machine's standard vocabulary.

Language: A set of operation codes and the form in which they are written.

Linkage: A set of instructions which will cause the machine to leave the main program (usually to transfer to a subroutine) and return to the point from which it transferred.

Loop: A series of instructions to be executed over and over, usually with a change of parameters for each repetition.

Low-order digit: The digit at the extreme right end of a number.

Machine language: The language for which a computer is originally designed.

Memory: (See Storage Unit)

Microsecond: One millionth of a second. Abbreviated µs.

Millisecond: One one-thousandth of a second. Abbreviated ms.

Octal number system: The number system with base 8.

Off-line operations: Operations which are independent of the central processing unit of the computer.

On-line operations: Operations requiring the use of the central processing unit of the computer.

Operand: The word upon which an operation acts.

Operation code: A numerical or literal symbol which a computer can interpret such that a definite operation will be performed.

Output: Information received from a machine.

Program: A set of instructions which are designed to cause a machine to solve a problem.

Random access: A term used to describe a memory unit having the property that the contents of any location are immediately available.

Read: The action of a machine in translating recorded information into electrical impulses.

Register: A storage location either separate from or contained in the main storage unit which will hold a word of information.

Routine: A set of instructions designed to perform a definite operation.

Single precision: Arithmetic in which the computer uses one word for each distinct piece of data.

Storage unit: The unit of the machine which holds information.

Stored-program machine: A machine which obeys numerically coded instructions taken from its own storage unit.

Subroutine: A routine which accomplishes a frequently used result and designed so that it may be fitted into a program as desired.

Trace routine: A routine which monitors the execution of a program, usually causing to be typed or printed each instruction as it is obeyed as well as the contents of the accumulator at the conclusion of each instruction.

Transfer instruction: An instruction causing the machine to select alternative sequences of instructions.

Word: The contents of a storage location.

Word length: The number of characters that can be held in one storage location.

Write: The action of a machine in recording information.

Appendix III

Steps in Good Programming

I. Analysis.

1. State clearly the requirements of the problem including input, output, and computation.
2. Make a flow chart indicating the step-by-step development of the problem. Don't be afraid of too much detail. More steps now will save errors later.
3. Introduce a clear system of notation showing loops in general terms (i.e., x_i rather than x_1).
4. Explore the possibility of using subroutines for repetitive calculations.
5. When branching, state the question clearly such that the answer is either yes or no.
6. Remember the major steps in a loop are SET, COMPUTE, INCREMENT, TEST. The order of the last three may be changed, but the foregoing is recommended.
7. Don't forget that a loop must have an entrance and an exit.

II. Coding

1. As you code, keep a careful list of storage assignments distinguishing clearly between constants and variables.
2. Try to place constants in the same channel(s) as your program.
3. Start coding at the beginning of loops, writing the SET instructions (which precede the loop) last.
4. If the program is long, break it up into distinct sections and write each as a complete unit. Do not be afraid to use unconditional transfers to get from one section to another.
5. Write enough notes to show clearly the purpose of the command(s). Sometimes several may be bracketed and one note written for the section.
6. Keep track of the contents of the accumulator.
7. When using index registers check the behavior of the register for the first run of the loop and for the last run.
8. In using conditional transfer instructions leave ADDR blank, make a note of the transfer condition in the notes column, and proceed coding in sequence. Come back later and code the other branch. Decide on the numerical values of testing constants at the time of or after writing the transfer command, not before.
9. When making insertions (splicing) do not erase the replaced instruction but draw a line through it and write the new instruction at the side. Later, you may want to know what was there.
10. Code on the assumption that you will make mistakes. The clearer your work, the easier it will be to debug.
11. Remember that a computer is entirely devoid of intuition. Normally it does exactly what the programmer <u>tells</u> it to do. Unfortunately, this is not always the same thing as doing what the programmer <u>wants</u> it to do.

Index

INTERCOM
CODING
SHEET

NOTES	LOCATION	K	OP	ADDRESS	ACCUMULATOR

STORAGE
ALLOCATION
CHART

CONSTANTS			VARIABLES		
LOC.	SYMBOL	VALUE	LOC.	SYMBOL	INITIAL VALUE

CH.	SUBROUTINE	CH.	SUBROUTINE

CUT ON LINES FOR HANDY POCKET CARDS

COMMAND LIST FOR INTERCOM 500 AND 1000

Special
- 06 Type location of last command executed
- 08 Perform subroutine

Return
- 16 Return to marked place I
- 18 Return to marked place II

Transfer
- 20 Transfer if accumulator ≥ 0
- 22 Transfer if accumulator < 0
- 23 Transfer if accumulator = 0
- 26 Mark place and transfer I
- 28 Mark place and transfer II
- 29 Transfer control

Output
- 30 Position typewriter paper
- 31 Type tabulating number
- 32 Type floating point and tab
- 33 Type fixed point and tab
- 34 Type float, return carriage
- 35 Type command
- 38 Type fixed, return carriage
- 39 Punch paper tape

Arithmetic
- 40 Clear and subtract
- 41 Subtract
- 42 Clear and add
- 43 Add
- 44 Multiply
- 45 Clear and add absolute value
- 47 Inverse divide
- 48 Divide
- 49 Store

Input
- 50 Store commands
- 50 Gate for command
- 51 Store fixed point data
- 51 Gate for fixed point data
- 52 Store floating point data
- 52 Gate for floating point data
- 55 Read punched paper tape

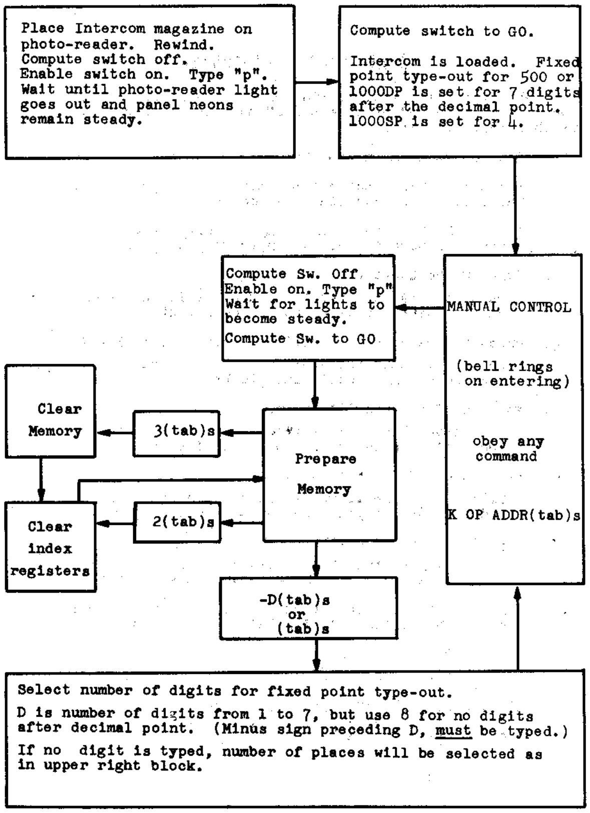
Start and Stop
- 61 Start list (trace routine)
- 62 Stop list (trace routine)
- 63 Ring bell
- 67 Halt and return to manual
- 68 Breakpoint halt
- 69 Start automatic operation

Index registers
- 70 Set word base
- 71 Set word difference
- 72 Set word limit
- 73 Set channel base
- 74 Set channel difference
- 75 Set channel limit
- 76 Increment word base, test
- 77 Increment channel base, test

Intercom 500 or 1000DP only
- 02 Perform subroutine
- 78 Clear and add index reg.
- 79 Store index reg. accum.
- 81 Block copy

Intercom 500 only
- 09 Set index register accum.
- 37 Type hexadecimal number
- w3 Perform subroutine

LOADING INTERCOM



Place Intercom magazine on photo-reader. Rewind. Compute switch off. Enable switch on. Type "p". Wait until photo-reader light goes out and panel neons remain steady.

Compute switch to GO.

Intercom is loaded. Fixed point type-out for 500 or 1000DP is set for 7 digits after the decimal point. 1000SP is set for 4.

MANUAL CONTROL

(bell rings on entering)

obey any command

K OP ADDR(tab)s

Compute Sw. Off Enable on. Type "p" Wait for lights to become steady.

Compute Sw. to GO

Prepare Memory

3(tab)s

Clear Memory

2(tab)s

Clear index registers

-D(tab)s or (tab)s

Select number of digits for fixed point type-out.

D is number of digits from 1 to 7, but use 8 for no digits after decimal point. (Minus sign preceding D, must be typed.)

If no digit is typed, number of places will be selected as in upper right block.

TABLE I. LIBRARY SUBROUTINE SPECIFICATIONS

Subroutine	Word position for entry	N (loading code no.)	
		SP	DP
Fraction Selector	See Table II	1 CHu0	1 CHu0
Square Root	97	2 CHu0	2 CHu0
Log_{10} x	71	3 CHu0	3 CHu0
Log_e x	17		
Log_2 x	08		
e^x	22	4 CHu0	4 CH00
2^x	08		
10^x	72		
Sin x (degrees)	39	5 CHu0	5 CH00
Sin x (radians)	42		
Cos x (degrees)	23		
Cos x (radians)	26		
Arctan x (radians)	24)	6 CHu0	6 CH00

TABLE II. FRACTION SELECTOR ENTRIES

No. of decimal places to be typed out	Word position for entry
0	08
1	01
2	02
3	03
4	04
5	05
6	06
7	07

TABLE III. STORAGE LIMITATIONS

If the N value of a subroutine ends in 00, when the subroutine is:	
Placed in CH	Do not use index register
09	1
10	2
11	3
12	4
13	5
14	6
15	7
16	8
17	9
18	u

TO USE THE TRACE ROUTINE.

The computer must be in the manual operating mode. The Intercom magazine must be on the photo-reader with the tape at the manual control position.

1. Put the Compute switch in the center (off) position. Hold the Enable switch ON and type p. Release the Enable switch and wait for the photo-reader light ro remain off.
2. Put the Compute switch to GO. Wait for the neon indicator lights to remain steady.
3. Type "1 (tab) s". Wait for the photo-reader light to remain off.
4. Type "(tab)s". Wait for the bell to ring signaling that the computer has returned to the manual mode.
5. Type "610000 (tab) s". Wait for the input-output neons to be in the configuration 0●●00.
6. To list every command, type "(tab)s". To list selected commands, type "FIRST SELECTOR (tab) SECOND SELECTOR (tab)s". The bell will ring signaling that the computer has returned to the manual mode.
7. Type "69ADDR(tab)s" where ADDR is the location of the command at which computation is to begin. The computation, with listing of selected commands, will proceed.

TO TERMINATE LISTING.

1. Put the computer in the manual mode (Section 2.7).
2. Type "620000 (tab)s". This command terminates listing and the computer is now in the manual operating mode.

STARTING THE COMPUTER

When the computer is turned on, it may be checked for proper operation by the use of a test routine which is provided in a punched tape magazine.

The procedure to turn on and check the computer is:

1. Place the "Test Routine" magazine on the photo-reader. The tape in the magazine must be rewound.
2. Put the Enable, Punch and Compute switches on the typewriter base in the center (off) positions.
3. Turn on the Start switch.
   Wait for the AC meter to read 6.3 volts or 100% and the amber AC light to become bright.
4. Press the Reset button until the red DC lamp lights.
   Wait until the photo-reader light remains off and the green "Ready" lamp lights.
5. Move the Compute switch to GO.
   The number "1" will be typed out. Wait for the display panel neons to remain steady.
6. Type "0 0 0 0 0 0 5(tab)s".
   Wait for the photo-reader light to remain off and the display panel neons to remain steady.
7. Type "0 0 0 0 0 0 6(tab)s".
   Bells ring at repeated intervals to signify successful procedure of each test in the routine.

   Proper computer operation is indicated if no type-out occurs before the following is typed out:

   ```
   - 1 1 2 2 3 3 4   4 4 5 5 6 6.7   7 7 8 8 9 9
   - u u v v w w x   x x y y z z.0       2 3 4 5
   ```
8. At completion of the type-out put the Compute switch to the center position, rewind, and remove the "Test Routine" magazine.

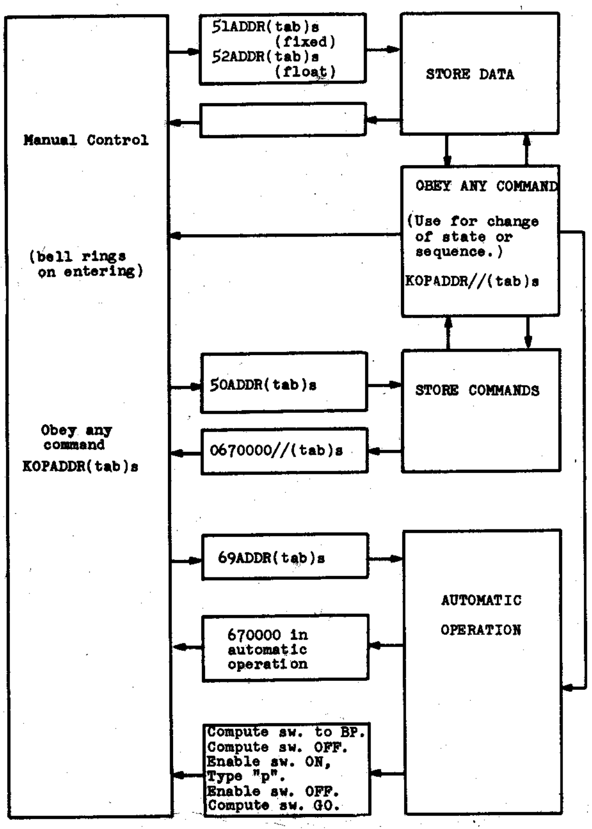
SOME COMMANDS USED IN STORING DATA AND COMMANDS ARE:

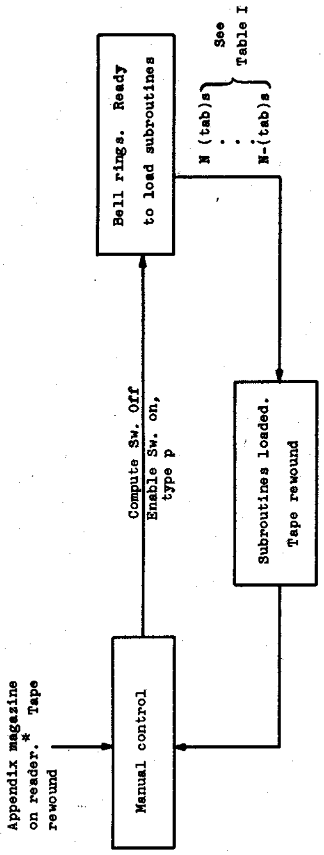
	From manual	Change of state or sequence
Store commands starting at ADDR	50ADDR(tab)s	050ADDR//(tab)s
Store fixed point data starting at ADDR	51ADDR(tab)s	051ADDR//(tab)s
Store floating point data starting at ADDR	52ADDR(tab)s	052ADDR//(tab)s
Start automatic operation at ADDR	69ADDR(tab)s	069ADDR//(tab)s
Return to manual control		0670000//(tab)s
Punch channel CH on paper tape	39CH00(tab)s	
Read paper tape into channel CH	55CH00(tab)s	
Obey any command	KOPADDR(tab)s	KOPADDR//(tab)s

RETURN TO MANUAL - Return to manual can be accomplished from any state or mode by:

1. Move the Compute switch to BP. WAIT FOR THE DISPLAY NEONS TO REMAIN STEADY! Put compute switch to GO.
2. Hold the enable switch on and type "scf".
3. Release the enable switch and put the compute switch to GO.
4. If bell does not ring, reload Intercom.

CHANGES OF MODE OR STATE





*With Intercom 500 the subroutines are usually included on the basic tape. In this case it is not necessary to change magazines. Merely follow the instructions from manual control.