G T 5 D P R O G R A M M E R'S

R E F E R E N C E

M A N U A L



DIVISION OF BENDIX AVIATION CORPORATION

5630 ARBOR VITAE STREET, LOS ANGELES 45. CALIFORNIA

TABLE OF CONTENTS

		Page
1.	The Bendix G-15 Digital Computer: Introduction	1.
2.	Principles of Machine Computation	2
_•	2.1 Machine Arithmetic	
	2.1.1 Binary Numbers and Arithmetic	2 2 4
	2.1.2 Binary-Coded Decimal	4
	2.1.3 Sexadecimal Notation	4
	2.2 Concept of a Program	6 6 7 8 8
	2.3 Organization of a Problem for Machine Solution	6
	2.3.1 Subroutines and Libraries	7
	2.3.2 Scaling	8
	2.3.3 Flow Charting	8
3.	Orientation on the G-15	10
-	3.1 G-15 Major Functional Components	10
	3.1.1 Input and Output Devices	11
	3.1.2 Memory with Recirculation	11
	3.1.3 Information Flow	14
	3.2 Programming and Coding for the G-15	14
	3.2.1 Commands	14
	3.2.2 Programming	17
	3.2.3 Minimizing the Access Time	18
	3.2.4 Checking and Debugging	19
	3.3 Examples of Application	21
4.		23
	4.1 Addition and Subtraction	23
	4.2 Multiplication and Division	27
5.	Commands for Double Precision Arithmetic Operations	32
	5.1 Double Precision Numbers and Commands	32
	5.2 Double Precision Addition and Subtraction	33
	5.3 Double Precision Multiplication and Division	34
6.	Copy and Block Operations	38
	6.1 Copy Operations	38
	6.1.1 Commands for Copy Operations	39
	6.1.2 Effect of C Codes in Commands with D	31 41
	6.1.3 Copy Operations Involving Two-Word	1.0
	Registers	42 45
	6.2 Block Operations	45 46
	6.2.1 Block Copy Commands	40
	6.2.2 Block Commands for Addition and	48
	Subtraction	-+0

TABLE OF CONTENTS (Continued)

		Page
7.	Command Sequence Control 7.1 Test Commands 7.2 Loops 7.3 Subroutines 7.4 Command Line Selection 7.5 Drum Mapping	51 51 55 55 56 58
8.	Input-Output Operations 8.1 Getting Information Into and Out of Computer 8.2 Regular Output Commands 8.3 Regular Input Commands 8.4 Commands for Regular Input-Output Control 8.5 Output Format	62 62 64 66 68 70
9.	Extract Operations 9.1 Extraction: A Logical Operation 9.2 Extract Commands	73 73 7 4
10.	Shift and Normalize Operations 10.1 Commands for Shifting, Single Precision 10.2 Double Precision Shift Commands 10.3 Normalize Commands	77 77 80 82
11.	Steps in the Solution of a Problem	83

LIST OF TABLES

		Page
Table 1.	Equivalent Representations of Numbers	5
APPENDIX		
Table A-l.	G-15D Basic Command List Arithmetic Operations: Single Precision Arithmetic Operations: Double Precision Arithmetic Operations: Block Commands Copy Operations Command Sequence Control Extract Operations Input-Output Operations Commands for Optional Auxiliary Equipment	87 88 89 90 90 91
Table A-2.	Components of a Decimal Command	92
Table A-3.	Command Conversion	93
Table A-4.	G-15D Special Commands: D=31	94
Table A-5.	Format Characters	95
Table A-6.	Punched Tape Characters	95
Table A-7.	Typewriter Control Keys and Switches	97
Table A-8.	General G-15D Operating Procedures	98
Table A-9.	G-15D Operating Procedures with Program Preparation Routine (PPR)	99
Table A-10.	Typical Steps in Using Program Preparation Routine	100
Table A-11.	Instructions for Program Preparation Routine	רסר

LIST OF FIGURES

		Page
Fig. 1	Typical Flow Chart	9
Fig. 2	Schematic Representation of Information Flow in the G-15	10
Fig. 3	Cross Section of the G-15 Magnetic Drum	13
Fig. 4	Converted Command in Binary Form	15
Fig. 5	G-15 Command in Decimal Form	16
Fig. 6	Three G-15 Commands for the Addition of Two Numbers	18
Fig. 7	Indicator Light Panel	20
Fig. 8	Cut and Fill Application to Highway Construction	21
Fig. 9	Steps in the Solution of a Problem	83
APPENDIX		
Fig. A-1	Typewriter Keys and Switches	96

LIST OF EXAMPLES

		Page
Example 1.	Addition and Subtraction, Single Precision	26
Example 2.	Addition and Subtraction, Minimum-Access Coded	27
Example 3.	Multiplication and Division, Single Precision	30
Example 4.	Arithmetic Operations, Double Precision	36
Example 5.	Block Addition and Subtraction, Double Precision	50
Example 6.	Command Sequence Control: Test Commands, Loops, Command Line Selection, Use of Subroutines	59
Example 7.	Snift, Single Precision	79

1. THE BENDIX G-15 DIGITAL COMPUTER: INTRODUCTION

The Bendix G-15 is a compact and powerful internally - programmed general-purpose digital computer of unusual versatility.

To appreciate the usefulness of such a computer as a practical mathematical tool, it is first necessary to become familiar with the technical jargon customarily employed by those familiar with the machine and its operation. An introduction to the principles, operation and application of the Bendix G-15 is included here to provide a general background upon which to build programming, coding and operational skills.

Terms such as "electronic brain" lead one to expect a superhuman mechanical monster which not only understands all the problems but knows how to solve them. Actually, the programmer must know enough about the problem and the procedure for solving it to prepare a detailed set of directions for each operation the machine is to perform. Then the machine will perform efficiently just the tasks planned for it.

An automatic digital computer is designed to execute a sequence of operations of ordinary arithmetic, to make simple decisions and to record the results in a useful form. A "general-purpose" computer can handle an almost unlimited variety of problems for which the solution can be reduced to a succession of these basic operations. In computers which are classified as analog devices, the variations in a physical quantity (such as the presence of a gas) are represented by continuous changes in some other quantity. The components and behavior of the real or proposed system under study are analogous to those of another system which is regarded as a model. In digital computers, which deal only with numbers, the model is mathematical and the behavior of the physical system is determined by the results of operations in arithmetic.

Therefore, describing the Bendix G-15 as a general-purpose digital computer implies that it is a mathematical tool applicable to widely diversified tasks involving manipulations with numbers. To understand how the computer works, something should be known about the number system in which internal computation is done.

2. PRINCIPLES OF MACHINE COMPUTATION

2.1 MACHINE ARITHMETIC

Probably because ten fingers are standard human equipment, and because counting or tallying on the fingers comes naturally, we have become accustomed to representing numbers by a set of ten characters or symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. The derivation from finger counting also leads to the term "digit" for any one of these symbols, and because there are ten digits the notation is called "decimal."

Another familiar notation, Roman numerals, employs alphabetical characters. Although satisfactory for counting years or chapters, this scheme is seen to be too awkward for arithmetic when you try to add MCMXIV to DXLVIII.

However, not even the decimal system is the most efficient when numbers are to be portrayed and manipulated by magnetic or electronic equipment. Design is simpler and components fewer when advantage is taken of the opportunity to represent numbers with elements that can have only two different values instead of ten. It is easy to recognize that a voltage is either relatively high or low, that a current is either flowing or not, or that a magnet moves either north pole or south pole first. If the voltage, for example, is low at a particular time, this condition may be denoted by "O"; if the voltage is high, that state is described by "l". With these two symbols we have a "binary" number language. The binary digits, O and 1, are called "bits" -- an abbreviation for "binary digits."

2.1.1 BINARY NUMBERS AND ARITHMETIC

Any number can be expressed in binary (bit) form, as shown in Table 1. There are two basic advantages in expressing numbers in this way - (1) simple elements can be used to indicate each of the bits in the binary number, as has been mentioned, and (2) arithmetic is surprisingly easy when numbers are expressed in binary form. For example, we need remember only these two simple rules for performing addition:

Rule 1. O plus 1 is 1

Rule 2. 1 plus 1 is 0 and carry 1 to the next column left.

With these two rules let us add 10 and 11 (binary numbers equivalent to the decimal numbers 2 and 3):

The binary 101 is equindent to the decimal number 5, which is indeed the sum of 2 and 3. It is obvious that a computer that is required to "remember" only these two basic addition rules will be much simpler than one that must "remember" that the sum of 3 and 4 is 7, or 4 and 8 is 12, plus all the other rules for addition with decimal numbers.

For similar reasons, subtraction and multiplication are also much simpler in the binary system because fewer combinations are possible. For example, multiplication is performed with only the two following rules:

Rule 1.
$$1 \times 0$$
 is 0 Rule 2. 1×1 is 1

Let us multiply the same two binaries, 11 and 10:

It must be remembered that a given number, or the result of an arithmetic operation, is the same whether it is expressed in the decimal system whose base is ten or in the binary system whose base is two. Any number, integer or fraction, can be depicted by a succession of bits, with a binary point separating the integral part from the fractional. The number of places a binary digit is away from the binary point indicates the power of two it contributes, just as the position of a decimal digit signifies a power of ten. For example, the binary number 10.01 equals

$$1(2)^{1} + 0(2)^{0} + 0(2)^{-1} + 1(2)^{-2} = 2.25$$

2.1.2 BINARY - CODED DECIMAL

All arithmetic in the G-15 is performed on binary numbers. However, since it would be too laborious to translate decimal numbers into the binary language before they are fed into the computer, the conversion is accomplished automatically in two stages, using binary-coded decimal numbers.

A binary-coded decimal number expresses each digit in the decimal number in binary form. For example, the decimal number 22 in binary-coded decimal is 0010 0010, where each of the four-bit binary groups is equivalent to the decimal number 2. Similarly, the number 97 in binary-coded decimal is 1001 0111. As four bits are required to express the largest decimal digit, 9, each decimal digit is expressed as a group of four bits. The second column in Table 1 shows some equivalent representations of numbers in binary-coded decimal.

In the G-15, each decimal digit enters the computer in the form of its binary equivalent, as a group of four bits. Then this intermediate binary-coded decimal representation is converted in the machine to true binary. A reverse procedure permits binary results to be converted to decimal form for output.

2.1.3 SEXADECIMAL NOTATION

Sometimes a record is wanted of the unconverted binary contents of the computer. Then a kind of shorthand is used which involves a number system with a base of sixteen instead of two or ten. This base is used because a group of four bits can represent any one of sixteen different numbers (zero to fifteen). By assigning a symbol to each of these combinations we arrive at a notation called sexadecimal (usually hex in conversation because nobody wants to abbreviate sex). The symbols in the sexadecimal language are the ten decimal digits and, on the G-15 typewriter, the letters u, v, w, x, y and z. These are arbitrary markings; other comuters may use different alphabet characters for these last six digits.

Equivalent forms of some numbers in decimal, binary-coded decimal, binary and sexadecimal are shown in Table 1.

Table 1. EQ	UIVALENT REPRE	SENTATIONS OF	NUMBERS.
Decimal	Binary-coded Decimal	Binary S	Sexadecimal
012345678901231567890122222890332	0000 0010 0010 0010 0010 0110 0110 011	0 1 10 10 11 100 101 100 101 100 101 100 101 100 101 100 101 100 101 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 1	0123456789uvwxyz01123456789uvwxyz011211456789uvwxyz012213

2.2 CONCEPT OF A PROGRAM

A complete set of commands to direct the computer's step by step operation must be stored in the memory before the automatic process begins. This list of coded instructions is called a "routine" and is also known as a "program." A program, in the broader sense, is a plan for the solution of a problem. A complete program includes plans for transcription of data, coding for the computer and plans for the effective use of the results. Thus the program becomes, in a sense, one of the functional components of the computer, along with the hardware of the input and output system, the memory, the arithmetic section and the circuitry for communication and control.

"Programming" is preparing a program. "Operating" the computer means actually manipulating the hardware: turning on the master switch, depressing certain buttons and typewriter keys, moving certain switches and typing in the program. A "programmer" is a person who plans the program and specifies the printing format. He may also do the numerical analysis and systems analysis, integrating the computer into the system. He may or may not actually operate the computer or code the program.

2.3 ORGANIZATION OF A PROBLEM FOR MACHINE SOLUTION

The automation of mathematics pays off by eliminating the duplication of effort for repetitive processes. Once a computer program has been prepared, it can be used over and over again each time the same type of problem is to be solved. The preparation of a program consists of: (1) compiling a complete and accurate statement of the problem; (2) making a numerical analysis to select suitable mathematical procedures; (3) drawing a flow diagram (Fig. 1) to indicate the arrangement of major blocks of commands; (4) planning the detailed steps of operation; (5) assigning locations in the computer storage for data and commands; (6) writing each command in decimally coded form; (7) finding and correcting the mistakes made in working through the first six steps.

Operation of the computer itself is introduced first in step (7): the decimal commands written in step (6) are transcribed on the input typewriter and automatically converted to binary words on entry into the computer; then a tape is punched, containing the program in sexadecimal form for rapid re-entry. After the inevitable mistakes or "bugs" in the program have

been detected and eliminated, little manipulation is needed to load commands and data into the computer for production runs.

A problem must be stated before it can be solved; an automatic machine can't ask, "What do I do now?" when it hits a snag in the middle of computation. A computer can make several kinds of simple decisions and, as a result, select between alternative sequences of ensuing commands; but unless the statement of a problem is initially complete, some conditions are likely to be overlooked. Not only must equations be written to express the relationships among the quantities involved in a computation, but every conceivable situation must be anticipated and its consequences set forth explicitly.

In some types of problems there is little or no uncertainty about the best method of solution; in others, however, various formulas or numerical techniques may be available. There are several good ways, for example, to approximate the solution of a set of differential equations or to invert a matrix, and each procedure has features to recommend it in certain instances. In these cases, depending more on the character of the problem than on the details of the computer, the application of the mathematical techniques of numerical analysis aids in the choice of the most suitable method.

2.3.1 SUBROUTINES AND LIBRARIES

Much of the labor required to prepare and debug a new program can be saved by incorporating previously checked blocks of commands called "subroutines". These are filed in a library of punched tapes from which masters can be withdrawn and reproduced for compilation with other segments to form a complete program. Included in a typical library would be subroutines to convert decimal input data to binary form and binary memory contents to decimal output; to evaluate square roots, trigonometric and other elementry functions; to perform numerical integration, matrix operations, and so forth.

In addition to subroutines, which become component parts of other programs, complete routines are available to reduce significantly the work of getting a problem ready for the G-15.

Although there is no substitute for an accurate statement of the problem and careful planning of solution methods, many clerical details can be relegated to the computer itself. A program of the type called "interpretive" permits commands to be written in a much simplified form. After an entire

program written in the simplified pseudo-code has been stored in the computer, each command is read in turn and interpreted as an operation to be performed by a subroutine. Valuable savings in programming time and effort are achieved at the cost of some computing speed, a compromise which enables relatively inexperienced personnel to utilize the G-15 effectively for tasks not repeated often enough to warrant an investment in more sophisticated techniques.

Whether an interpretive routine for simplified coding is used or not, the seven steps of program preparation listed above are fairly typical, not only for the G-15 but for most general-purpose digital computers, both large and small.

2.3.2 SCALING

Another aspect of analysis intimately associated with most digital computers is the matter of "scaling"; that is, making sure that all numbers handled in the computer retain significance while fitting into words of fixed length. One of the techniques involves the use of a number representation called "floating point", where the position of the decimal or binary point is specified by a special part of the number, like the characteristic of a logarithm. The G-15 has been designed to facilitate arithmetic with numbers of this form, not by inclusion of elaborate circuitry or expensive components, but through the operation of an unusually efficient programmed subroutine. This subroutine forms an integral part of the interpretive routine mentioned previously, with the result that scaling is taken care of automatically.

2.3.3 FLOW CHARTING

Once a problem has been properly stated, the numerical procedures established and scaling settled, the actual writing of a detailed program is facilitated by the use of a flow chart (Fig. 1) showing the sequence of major steps and alternative branches dependent on conditions encountered as the computation progresses. The complexity of a problem determines how elaborate its flow diagram need be, not only to aid the programmer but also to serve as a map of the completed work. Sometimes the operations included in one block of a comprehensive diagram are displayed in more expanded detail on a separate flow chart.

3.1.1 INPUT AND OUTPUT DEVICES

Obviously the numbers involved in the operations a computer performs must somehow be fed into the machine. Standard G-15 input equipment includes an electric typewriter (specially modified) and a photoelectric reader which accepts 200 characters (sexadecimal digits or special control symbols) per second from punched tape. Permanent records of intermediate or final results of computations may be obtained by automatic operation of the typewriter at about eight characters per second or by a tape punch unit at a higher speed. Up to four magnetic tape units (Model MTA-2) available as optional auxiliary equipment, provide for magnetic tapes as input or output media. Various other optional input-output devices are available: these include punched card adapters (Models CA-1 and CA-2), a unit for handling alphanumeric information (Model AN-1), a machine tool control unit (Model AN-2), a graph plotter (Model PA-2), and a Flexowriter (Model 35-4, Friden Calculating Machine Company) for independent preparation and interpretation of tape. Another accessory, the digital differential analyzer (Model DA-1), provides for the simplified programming of differential equations.

3.1.2 MEMORY WITH RECIRCULATION

At the heart of any automatic digital computer is some device in which numbers can be retained to become available when needed in the course of calculations. This component, called the "memory" or storage, consists, in the G-15, of a cylinder or drum whose surface is thinly coated with a magnetic material. About a foot in diameter and a few inches wide, it rotates at 1800 rpm. Binary numbers are written on its surface by magnetizing a succession of spots around the circumference, using one polarity to represent "0" and the opposite polarity to represent "1". A group of 29 consecutive bits is called a "word."

Numbers involved in arithmetic operations in the G-15 may be either one or two words long, with one bit of the 29 or 58 used to indicate the sign (0 for +; 1 for -). A one-word number (expressed in 28 bits) can be as large as 268,435,455. A "double precision" or two-word number of 57 bits is equivalent to about 17 decimal digits. The G-15 is outstanding in its facility to perform double-precision arithmetic, thus retaining significant figures through long chains of computations.

Words stored in the G-15 memory are arranged serially on the drum in circumferential tracks or "lines" (Fig. 3). As the drum rotates, the magnetized spots in each line pass under a stationary read head which transmits, to other elements in the computer, signals representing the information stored in its line. Amost immediately, either the same or new signals are sent to a write head and recorded for another ride around the drum. Between the two heads associated with each line there is an erase magnet which causes the drum surface passing under it to be cleared (i.e., magnetized with the same polarity as "0"), so that new writing will not be superimposed on the old. This process of reading information at one point on the drum surface and rewriting it at another is termed "recirculation". It leads to certain advantages in circuitry.

As one consequence of the recirculation process, it can be seen that the length of a line (that is, the number of words it contains) depends directly on the circumferential distance from the write head to the read head (Fig. 3). Thus, lines of different lengths are made possible by different spacings between heads. Since a word written on the drum becomes available only when it passes under a read head, the "access time" (delay until a desired word arrives) is less for words stored in shorter lines.

The main portion of the G-15 memory consists of twenty "long" lines, numbered 00 through 19, each containing 108 words. There are five "short" 4-word lines, one of which is reserved for special functions during input and output processes, the other four being available for rapid-access storage. They are numbered 20 through 23.

In order to represent the 108 word locations using only two digits, 100 is represented by u0, 101 by u1, ..., 106 by u6 and 107 by u7. Word availability is cyclic: word 00 is available immediately after word u7 of a long line. In the short lines, the words are numbered 00 through 03. Word 00 is available immediately after word 03 in a 4-word line.

To locate a word of information stored on the drum, it is necessary to specify the number of the line the word is in and its position in that line. For example, 08.32 designates word 32 in line 8. Such a location number is called an "address".

The contents of a location are indicated by placing parentheses around the address. For example, (08.32) designates the contents of 08.32.

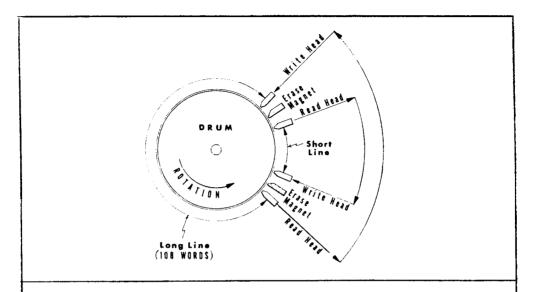


Fig. 3. Cross section of the G-15 magnetic drum. Each of 20 "long" lines, or circumferential tracks, contains 108 words of 29 bits. Rapid-access storage consists of four "short" 4-word lines; the arithmetic registers are a 1-word line and three 2-word lines. The read and write heads for the shorter lines are staggered between those for the long lines.

During internal machine computation, all numbers in the computer are assumed to be greater than -1 and less than +1. This range is obtained by carrying a scale factor along with the number.

Numbers are normally stored in memory in the form of absolute value and sign. A change of form occurs when negative numbers are involved in addition or subtraction operations. The change, called complementation, may be defined as the result of subtracting the absolute value of a negative fraction from one and retaining the negative sign. Note that complementing the complement (recomplementation) restores the negative number to its original form.

Arithmetic operations are performed in four "registers" which are also lines on the drum. Three of these registers are 2-words long and one is of 1-word length. A number may be directed to enter the 1-word register or

one of the 2-word registers, either replacing or adding to the previous contents of the line. The 2-word registers are used also in multiplication and division operations with either single or double precision numbers.

The AR and PN registers are "accumulators"; that is, each has associated electronics circuitry that enables it to perform addition and subtraction. The PN register also holds the Product in multiplication and the Numerator in division. The MQ register holds the Multiplier in multiplication and the Quotient in division. The ID register holds the multiplicand or "Icand" in multiplication and the Denominator in division.

The remaining portion of memory is occupied by lines, some permanently recorded, used for control information and timing signals by which all operations in the computer are synchronized.

3.1.3 INFORMATION FLOW

Words in corresponding positions in all lines are read and written together. One line may be selected as a "source", another as a "destination"; a word read from the source is written in the destination. Meanwhile, all lines except the one chosen as the destination continue to recirculate, their contents unchanged. The "transfer" or copying of a word, one bit at a time, from a source to a destination, is accomplished in 0.27 millisecond. This word time is the unit for measuring the duration of most G-15 operations.

As soon as one operation has been completed, a new source and destination may be selected. The next operation begins either immediately or when a designated word is available at the read head, and it may continue for one or more word times. Flexibility in the number of words copied in one operation is an unusual feature contributing to the efficiency of the G-15.

3.2 PROGRAMMING AND CODING FOR THE G-15

It is obvious that the machine, in order to function at high speed, must be controlled automatically, according to a sequence of operations established in advance and stored internally. To accomplish this, the contents of a binary word can be interpreted by the control circuits as a "command".

3.2.1 COMMANDS

Upon being read, a command is transferred to a special register in the memory, where it is retained until the action it specifies has been performed.

Various groups of bits within the word are decoded to specify (1) a source, (2) a destination, (3) when an operation starts and how long it lasts, (4) in which of certain ways words may be modified or delayed during transfer, and (5) where the next command is to found. In storage, a command cannot be distinguished in form from any other word; it may even be modified arithmetically as a number. A word serves as a command only after it has been so designated by the preceding command. Commands may be altered during the course of computation by the program itself. Since in the computer the form of a command is numerical, the numbers which make up the command may be operated upon in the same way as any other numerical information.

The set of commands and constants that constitute a program may be entered into the G-15D in decimal form. The decimal commands are automatically converted before use by a previously inserted "service" program into binary form and condensed to fit into the 29 bits of one word (Fig. 4).

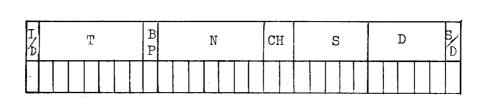


Fig. 4. Converted Command in Binary Form. In a 29-bit word designated as a command, the first bit at left (I/D) signifies whether the command is immediate (I=0) or deferred (D=1); the next 7 bits (T) specify the time of operation: when a deferred operation begins or an immediate operation ends; N is the location of the next command; CH is the characteristic operation to be performed; S is the source; D is the destination; S/D the precision (single S=0, double D=1). The BP bit, if a "1", indicates a break point.

One such service program, the "Program Preparation Routine", is furnished on punched tape with the computer. This routine and its use is described in detail in the operating manual. Thus, the commands in machine language programming are normally written in the decimal form (Fig. 5) understood by the Program Preparation Routine.

Locati o n				Comm	and.		
L	P	T or Lk	N	С	S	D	BP
Range From: 0 0 To: u 6	*	0 0 u 7	0 0 u 6	o 7	0 0 3 1	0 0 3 1	**

Fig. 5. G-15 Command in Decimal Form *Prefix of u or w may be required; otherwise, P is left blank. **Break point is indicated by a minus sign, -, if desired; otherwise, BP is left blank.

The components of a command in decimal form are designated by the following symbols:

- L = Location or word position of a command.
- P = Prefix
- T = Timing number: specifies the word position of the address or duration of execution.
- L_k = A number equal to the sum of the word position of the command plus k. (If the sum is greater than 107, subtract 108.) L_k designates when an immediate or block operation is to end.
- N = Word time during which next command is to be read.

- C = Code number specifying precision and characteristic. A single precision (SP) number consists of 28 bits and sign; a double precision (DP) number has 57 bits and sign. The characteristic determines if and how numbers may be modified or changed during transfer or copy operations.
- S = Source number: specifies a line or register as a source or serves as an operation code when D = 31.
- D = Destination number: specifies a line or a register as a destination or, if D = 31, that the source number has a special meaning.
- BP = Break Point: computation can be halted automatically after the execution of a breakpointed command during program checking to examine intermediate results.

 Break points, though indicated, can be ignored during production runs.
- ST = Source address (line 8, word T).
- DT = Destination address (line D, word T).

3.2.2 PROGRAMMING

With the flow diagram as a guide, the detailed steps of computer operation are planned in terms of individual commands. If two numbers are to be added, for example, three commands are usually required (Fig. 6). The first number is brought into an accumulating register to replace its former contents; the second number is then copied into the same register so as to add to the first; finally, the sum is stored in memory, either to be used later in the computation or to be converted to the binary-coded decimal form for typeout.

A G-15 command (in machine language, as contrasted to simplified coding) provides for the transfer or copy of a word or more from one memory line to another; i.e., from a source to a destination. However, some operations, such as multiplication, division, initiation of input or output activities, and others, do not involve such a transfer per se; these operations are specified by commands in which a special code takes the place of a destination number and a source code indicates the operation.

L	P	T or L _K	N	C	S	D	ВР	NOTES
00		o1	02	1	20	28		$\mathbf{a} = (20.01) \xrightarrow{+} \mathbf{AR}_{c}$
02		03	011	1	20	29		$b = (20.03) \xrightarrow{+} AR+$
OLI		05	06	1	28	07		$c = (AR) \xrightarrow{+} 07.05$

Fig. 6. Three G-15 commands for the addition of two numbers and the storage of their sum. (1) the command in Loc. 00 causes the number "a" to be copied from its storage location (line 20, word 01) to the accumulator register AR. The destination number 28 provides that "a" replaces the previous contents of AR. (2) the command in Loc. 02 copies the number "b" from line 20, word 03, to AR. The use of destination number 29 causes an addition to "a", the previous contents of AR. (3) Loc. 04 contains a command which stores "c", the sum of "a" and "b", in line 07, word 05. The next command would be placed in location 06. The time required to read and execute these three commands is about 1.6 milliseconds.

3.2.3 MINIMIZING THE ACCESS TIME

In operation, the computer is always in one of the four basic machine states:

- (1) Read command
- (2) Wait to execute command
- (3) Execute command
- (4) Wait for next command

States (2) and (4) represent delays. Minimization of these delays by judicious assignment of storage locations for commands and data is called minimum-access coding.

A specific word on a long line of the drum is read, and hence available for use, once every 108 word times. A word time, the time required for 29 bits (or 1 word) to pass under the read head, is equal to 0.27 milliseconds. A drum cycle requires 108 times 0.27 or 29 milliseconds. The average access time for a word on a long line is one

half of this, or 14.5 milliseconds. However, programming for minimum access greatly reduces this time.

A word on a short line containing 4 words becomes available every fourth word time or 1.08 milliseconds (4 times 0.27). Average access time for short lines is 0.54 milliseconds. There are 27 ($108 \div 4$) opportunities to read a word stored in a short line during a single drum cycle.

The ID, PN and MQ registers each hold two words; each word is available every other word time. The AR register holds one word of information which is accessible during any word time.

Minimum-access coding, in addition to reducing delays, may make use of the short lines and registers for temporary storage to further reduce access time.

For the G-15 programs with minimum delay times are just as easy to write as less efficient ones. In most cases, there need be no pauses between reading a command, executing it, and reading the next command. With access optimized, the three commands in the addition example can be read from memory and executed in about 1.6 milliseconds.

3.2.4 CHECKING AND DEBUGGING

Whenever a new program is written, it can be expected not to work at the first trial because even the best programmer, being human, is inherently fallible. To detect his mistakes it is essential that a "test case" be prepared in order that intermediate, as well as final, results may be verified. Usually the sample will be much shorter than a typical productive computation, for a single pass through an iterative sequence of commands tests them as effectively as many repetitions. It is important, though, that no condition or branch of the program be neglected during the check-out process.

Automatic operation of the G-15 can proceed in either of two modes, selectable by an external switch. In production runs, it will compute until a command to halt is reached; in checking a program, it can proceed through a sequence of commands at high speed until a programmed break point is recognized. Then the contents of any

part of memory can be typed out for examination, after which computation may be resumed until the next break point or a command to halt appears. It is possible also to cause the G-15 to execute a single command at a time. When computation stops after a halt command, a break point, or a single cycle operation, the operator can read from a small panel of neon lights (Fig. 7) information about the last command the computer has executed.

Special service routines have been developed to facilitate debugging. A "lister" automatically types out the commands in a line of programming in decimal form. A "tracer" automatically types out the commands of a program in decimal form in the order of execution together with a type-out of the result of each step.

To summarize the procedures for using an automatic digital computer, there are three principal activities; (1) problem statement and analysis, (2) program writing and debugging, (3) productive computation.

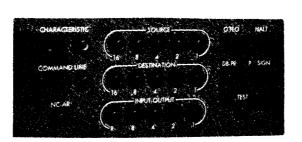


Fig. 7. Indicator light panel.

Neon lights provide information about the last command the G-15 has executed when computation has stopped after a halt command, a break point or a single cycle operation.

3.3 EXAMPLES OF APPLICATION

There have been so many allusions, in both technical and popular journals of the past decade, to the spectacular achievement of million-dollar computers that you may wonder whether real capability and versatility can be found in a modest tool like the \$49,500 G-15. Actually, the variety of applications to which it is suited is so large as to defy any attempt at completeness in a brief listing. To illustrate its usefulness, therefore, two examples have been chosen because they can be understood without profound knowledge of specialized technology or advanced mathematics. The first application, to highway construction, illustrates how the G-15 can save more than 90% of the time and cost otherwise required for calculations that must be performed dozens of times per mile of road.

Building a highway requires the movement of earth: low places must be filled; ground above roadbed level must be cut and removed. The cost of such work depends, of course, on the volumes of the cuts and fills, which can be calculated from survey readings. A typical calculation, which requires 30 to 45 minutes for each cross section when performed manually, can be accomplished with automatic type-out of results by the G-15 in about 15 seconds. Allowing an average of $1\frac{1}{4}$ minutes for the transcription of surveyor's field notes on the input typewriter, the G-15 shows an advantage in elapsed time of as much as 30 to 1.

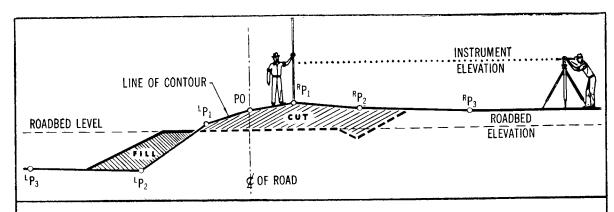


Fig. 8. Cut and fill application to highway construction. A G-15 can compute and record the volumes of cuts and fills from survey readings in a fraction of the time required for manual calculations, while avoiding the introduction of human errors.

In addition to quantities which change infrequently, such as the width of the road, the following data enter the calculations for each station: (1) survey rod readings and the distance of the readings from a base line, (2) station number (3) height of the roadbed, (4) height of the instrument, (5) left and right side slope ratios, (6) depth of left and right ditches. From these data are calculated and recorded: (1) depth of cut or fill at left and right slope stakes, (2) horizontal distance of left and right slope stakes from the center line, (3) "cut" volume between preceding station and present station, (4) cumulative total of "cut" volumes, (5) "fill" volume between preceding station and present station, (6) cumulative total of "fill" volumes, (7) net difference between cut and fill, (8) cumulative difference between "cut" and "fill" volumes, adjusted to include a shrinkage factor. Included in the calculations are such operations as a determination of the point of intersection between the land contour and the road elevation, decisions whether each element of volume is a cut or a fill, and many others. Applications of the Bendix G-15 to this and other problems in highway construction and traffic control will conserve public funds while expediting the creation and improvement of transportation facilities.

The automation of an industrial process is a second example of the practical use of the G-15. Here the computer is coordinated with other units in a system designed and produced by the Bendix Aviation Corporation for the automatic control of machine tools. Numerical data describing the contours and tolerances of work to be produced by a milling machine are tabulated from engineering drawings, together with information about radius and feed rate of the cutting tool. These data, after being transcribed on a punched tape, are fed into the G-15 to enter calculations resulting in the specification of the path of the tool center, with components of motion in either two or three dimensions. Output from the G-15 consists of a second punched tape which has the calculated results arranged in the special form required by a machine-tool-control unit. This tape contains all the information necessary to direct the milling machine in high-precision automatic production.

Other numerous and varied examples could be cited to illustrate both present and future applications of the digital computer in private industries and public agencies. The two cases above indicate the increasingly important contributions of an automatic general-purpose computer of moderate size to the automation of processes involving the application of practical mathematics.

4. COMMANDS FOR SINGLE PRECISION ARITHMETIC OPERATIONS

4.1 ADDITION AND SUBTRACTION

Notes	Commands	Descriptions
$(S.T)$ \rightarrow AR_{C} $(S.T)$ \rightarrow AR_{C} $I(S.T)$ \rightarrow AR_{C} $I(S.T)$ \rightarrow AR_{C} $I(S.T)$ \rightarrow AR_{C} $I(S.T)$ \rightarrow AR_{C} $I(S.T)$ \rightarrow AR	T N 1 S 28 T N 1 S 29 T N 2 S 28 T N 2 S 29 T N 3 S 28 T N 3 S 29 T N 1 28 D	Clear AR and add (S.T) Add (S.T) to (AR) Clear AR and add absolute value of (S.T) Add absolute value of (S.T) to (AR) Clear AR and subtract (S.T) Subtract (S.T) from (AR) Store sum or difference from AR into D.T

Discussion

$(S.T) \xrightarrow{+} AR_c$ TN1 S 28

The AR register will be cleared to zero. The number in S.T will be entered into the AR register in preparation for an addition or a subtraction. If the number is negative, it will enter AR in complementary form, ready for an addition. A number stored in AR in "normal" form (absolute value and sign) may be put into suitable form for an addition by the command T N 1 28 28. The contents of S.T remain unchanged in storage.

$(S.T) \xrightarrow{+} AR+ T N 1 S 29$

The number in S.T will be added to the number in AR. The contents of S.T remain unchanged. Addition of a negative number is accomplished by first complementing it and then adding this complement. Further additions and subtractions may be performed on the results in the AR register. If the sum is greater than 1, or less than -1, or equals ± 1, an "overflow" occurs; this may be tested as described later. If the resulting sum in AR is negative, it will be in complementary form. Therefore, additions and subtractions should be followed by a "Store sum or difference from AR" command which will automatically put the stored sum in the form of absolute value and sign.

T.N.1.26.28

$(s.T) \longrightarrow AR_c$ TN2 S28

The AR register will be cleared to zero. The absolute value of the number in S.T will be copied into AR. The sign bit will be 0 as for a positive number.

$$(S.T) \longrightarrow AR+ T N 2 S 29$$

The absolute value of the number in S.T will be added to the number in AR.

$$(S.T) \longrightarrow AR_c$$
 T N 3 S 28

The AR register will be cleared to zero. The number in S.T will be entered into AR with the sign changed and will be complemented if the resulting sign is negative. (See discussion of "Clear and add to AR" command.) In effect, the number in S.T will be subtracted from zero and entered into AR. This command should be followed by another arithmetic operation involving the AR register.

$(S.T) \longrightarrow AR + T N 3 S 29$

The sign of the number from S.T will be changed as it goes to AR and the resulting number, in complementary form if negative, will be added to the number in AR. (See discussion of "Add to AR" command.) In effect, the number in S.T will be subtracted from the number in AR. Since the difference in AR may not be in the form of absolute value and sign, this command should be followed by another arithmetic operation involving AR or by a "Store sum or difference from AR" command which will recomplement a negative result.

$(AR) \xrightarrow{+} D.T$ TN 1 28 D

The result of addition or subtraction will be entered from register AR into D.T. If the result is negative, it will be in AR in complementary form, and this command will automatically recomplement so that the result will be stored in D.T in the form of absolute value and sign. After the result has been stored in D.T, it remains in AR as well (in complementary form if negative).

Example 1: Addition and Subtraction, Single Precision

Write a program to find and store the sum, s, of a - b + |c|. Assume that a is in line 12, word 05; b is in line 07, word 20; c is in line 07, word 36; and the sum is to be stored in line 06, word u0.

Statement of the problem:: s = a - b + |c|

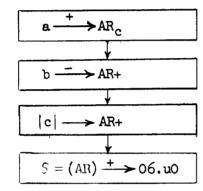
Storage locations: a = (12.05)

b = (07.20)

c = (07.36)

s = (06.u0)

Flow chart:



Coding of the program:

Word Position of Command			Comma	nd (Line <u>00</u>		
L	P	$\overset{\Gamma}{\circ_{\Gamma_{k}}}$	N	С	ន	D	BP	notes
00		05	01	1	12	28		$a = (12.05) \xrightarrow{+} AR_{c}$
01		20	02	3	07	29		$b = (07.20) \longrightarrow AR+$
02		36	03	2	07	29		$ c = (07.36) \longrightarrow AR+$
03		uO	04	1	28	06		$x = (AR) \xrightarrow{+} 06.u0$
04								

This program is not minimum-access coded.

Example 2: Addition and Subtraction, Single Precision, Minimum-Access Coded

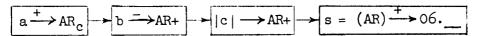
Rewrite the program of Example 1 so that storage locations of commands and data take into account the considerations for minimum-access coding. Let first command be in 00.00.

Statement of the problem: s = a - b + |c|

Storage locations: a = (12.)or

b = (07.) 03 word positions to be c = (07.) 05 chosen when coding. s = (06.) 07

Flow chart:



Coding of the program

Line 00

L	P	$^{\mathrm{T}_{\mathrm{Or}}}_{\mathrm{Lk}}$	N	С	S	D	BP	NOTES AND EXPLANATION			
00	? .	01	02	1	12	28		a = (12.01) +>AR _c Command is read at word time 00; earliest time available to execute command is 01. Therefore, let T=01, making a=(12.01). Earliest time available to read next command is 02, so N = 02.			
02		03	0,4	3	07	29		b = (07.03) -> AR+ Command is read during 02, the N of the previous command, and executed during word time 03. Next command may be read during 04.			
014		05	06	2	07	29		c = (07.05) -> AR+ Command is read during 04 and executed during 05. Next command read during 06.			
06		07		1	28	06		$s = (AR) \xrightarrow{+} 06.07$ Command is read during 06 and executed at 07. Earliest time for next command is 08, but its operation may require another location.			

4.2 MULTIPLICATION AND DIVISION

Notes	Comman					Descriptions
Clear	L ₃	N	0	· 23	31	Clear ID, MQ, PN, sign (IP)
(c, m)	Todd	N	0	S	25	Load multiplicand or divisor
$(S.T) \longrightarrow ID_1$	Teven	N	6	s	25	hoad multiplicate of divisor
(0, 11)>>>	T _{odd}	N	0	S	24	Load Multiplier
(S.T)	Teven	N	6	S	24	Dog warmbire
(C T) DN	Todd	N	0	S	26)	Load numerator
	Teven	N	6	s	26	and maner a sor
Multiply	56	N	0	24	31	$(ID_1)x(MQ_1) \longrightarrow PN_{0,1} (Location odd)$
Divide	57	N	ı	25	31	$(PN_1)*(ID_1) \longrightarrow MQ_0$ (Location odd)
$(PN_1) \longrightarrow D.T$	Todd	N	0	26	D	Store product in D.T
$(MQ_O) \longrightarrow D.T$	Teven	N	0	24	D	Store quotient in D.T

Discussion

Clear L₃ N O 23 31

The contents of the ID, MQ and PN registers will be cleared to zero. IP will also be cleared to zero. IP is a special storage element (a flip-flop) which holds one bit of information: the sign bit associated with multiplication or division. The two-word registers should be cleared before loading them for multiplication. However, when a multiplicand is loaded into ID from an even word location (Teven), the clear command is unnecessary. The two-word registers must be cleared before loading them for division. This command operates during word times L1 and L2.

$$(s.T) \longrightarrow ID_1 \begin{cases} T_{odd} & N \odot S & 25 \\ T_{even} & N & 6 & S & 25 \end{cases}$$

The absolute value of the number in S.T will be copied into the odd half of the ID register, ID1, ready to serve as the multiplicand in multiplication or as the divisor in division.

The sign bit from S.T will enter IP, replacing the previous contents of IP. If T is odd, the command must be in the form T N O S 25. This command will also clear the odd half of the PN register, PN₁, and will operate during word time T. If T is even, the command must be in the form T N 6 S 25, S must be less than 28, and this command will operate during word times T and T+1. This form of the command will also cause the even side of ID to be cleared, both sides of PN to be cleared, and the information originally stored in AR to be lost. ID should be loaded before loading MQ (multiplier) in multiplication, or before loading PN (numerator) in division. ID should not be loaded from the MQ or PN registers.

$$(s.T) \longrightarrow MQ_1 \begin{cases} T_{odd} & N \text{ o s 24} \\ T_{even} & N \text{ 6 s 24} \end{cases}$$

The absolute value of the number in S.T will be copied into the odd half of the MQ register, MQ1, ready to serve as the multiplier in multiplication. The sign bit from S.T will be added to (IP): a positive sign (0) will leave (IP) unchanged; a negative sign (1) will change (IP). If T is odd, the command must be in the form T N C S 24, and it will operate during word time T. If T is even, the command must be in the form T N 6 S 24, S must be less than 28, and this command will operate during word times T and T+1. Also, if T is even, the even side of MQ will be cleared and the information originally stored in AR will be lost. MQ should be loaded after loading ID for a multiplication. MQ should not be loaded from the ID or PN registers.

$$(s.T) \longrightarrow PN_1 \begin{cases} T_{odd} & N \circ S & 26 \\ T_{even} & N \circ S & 26 \end{cases}$$

The absolute value of the number in S.T will be copied into the odd half of the PN register, PN₁, ready to serve as the numerator in division. The sign bit from S.T, if positive, will leave (IP) unchanged; a negative sign will change (IP) from 0 to 1 or from 1 to 0. If T is odd, the command must be in the form T N O S 26, and it will operate during word time T. If T is even, the command must be in the form T N 6 S 26, S must be less than 28, and it will operate during word times T and T+1. Also, if T is even, the even side of PN will be cleared and the information originally stored in AR will be lost. PN should be loaded after loading ID for a division. PN should not be loaded from the MQ or ID registers.

Multiply 56 N O 24 31

The number in ID will be multiplied by the number in MQ, and the product will be developed in the entire PN register, PNO,1. The sign bit associated with this product is in IP. The location of this command must be odd, and it will operate during word times L1 through L56. The contents of ID and MQ will be altered during multiplication.

Divide 57 N 1 25 31 or 57 N 5 25 31

The number in PN will be divided by the number in ID, and the quotient will be developed in the even half of the MQ register, MQO. The sign bit associated with this quotient is in IP. The location of this command must be $\underline{\text{odd}}$, and it will operate during word times L_1 through L_{57} . The contents of PN will be altered during division, but the contents of ID remain unchanged. (PN) should be smaller in magnitude than (ID). The least significant bit of the quotient will always be equal to 1; this approximation to rounding will usually be more accurate than truncation.

 $(PN_1) \longrightarrow D.T$ $T_{odd} N O 26 D$

The most significant 28 bits of the product, from the odd half of the PN register, together with the sign bit from IP, will be copied into a storage location, D.T. D should not be a two-word register; T must be odd.

 $(MQ_O) \longrightarrow D.T$ $T_{even} N O 24 D$

The quotient, from the even half of the MQ register, together with the sign from IP, will be copied into a storage location, D.T. D should not be a two-word register; T must be even.

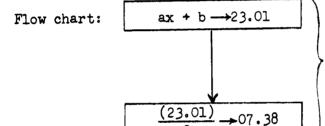
Example 3: Multiplication and Division, Single Precision Write a program to find and store $y = \frac{ax+b}{c}$.

Assume that a is in line 21 word 01, b is in line 21 word 03, c is in line 06 (word location to be chosen conveniently), x is in line 23 word 00, and y is to be stored in a convenient location in line 07. First command is to be in line 00 word 00.

Statement of the problem: $y = \frac{ax+b}{c}$

Storage locations: a = (21.01) b = (21.03)c = (06.82) x = (23.00) $y \rightarrow 07.38$ Word locations for

c, y, and ax+b chosen when coding.



A more detailed breakdown with a step corresponding to each command appears in the NOTES column of the coding sheet.

Coding of the program, together with recommended form of writing NOTES, is on the facing page.



G-15 D

Los Angeles 45, California

Prepared by <u>G. M. Silvern</u>

Page <u>1</u> of <u>1</u> Date:_____

....

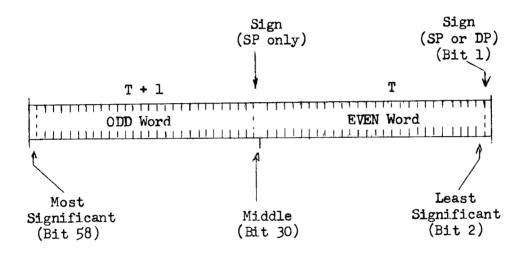
PROGRAM PROBLEM:				3) $y = \frac{ax+b}{c}$								Line
\8	1	2	3	L	P	T	N	C	S	D	ВP	NOTES
4	5	.	7	00		04	06	6	23	25		$x = (23.00) \longrightarrow ID_1$
8	9	10	¥	06		09	11	0	21	24		$a = (21.01) \longrightarrow MQ_{1}$
12	13	14	15	11		56	68	0	24	31		Multiply
16	17	18	19	68		69	70	0	26	28		$\mathbf{a}_{\mathbf{x}} = (PN_1) \longrightarrow AR$
20	21	22	23	70		71	72	ı	28	28		(AR) -+ →AR _c
24	25	26	27	72		75	76	1	21	29		$b = (21.03) \xrightarrow{+} AR+$
28	29	30	31	76		77	78	1	28	23		$ax + b = (AR) \xrightarrow{+} 23.01$
32	33	34	35	78		81	81	0	23	31		Clear MQ + 10 neg.
36	¥	38	39	81		82	84	6	06	25		c = (06.82)→ID ₁
40	41	42	43	84		85	87	0	23	26		$ax + b = (23.01) \rightarrow PN_1$
44	45	46	47	87		57	37	1	25	31		Divide
48	49	50	51	37		38	-	0	24	07		$y = (MQ_O) \longrightarrow 07.38$
52	53	54	55									
56	57	58	59									
60	61	62	63	50				6		25	H	
64	65	66	67	06		09	1)	0	10	24		
58		70	71									
72	73	74	75					ļ				
186	77	***	79									.1
80	M	82	. 83									
84	85	86	97	1			 	ļ	 			
88	89	90	91							 		
92	93	94	95					<u> </u>				
96 uo	97 U1	98 U 2	99 U3				·	_		<u> </u>	-	
			uj									
U/4	u5	116							•			

FORM BCD 108-5-0

5. COMMANDS FOR DOUBLE PRECISION ARITHMETIC OPERATIONS

5.1 DOUBLE PRECISION NUMBERS AND COMMANDS

A single precision (SP) number consisting of 28 bits and sign occupies one word in memory; a double precision (DP) number consists of 57 bits and sign and occupies two consecutive words. The less significant 28 bits and the sign of the DP number are stored in location T, and the more significant 29 bits in location T+1, where T must be even.



Most of the commands for arithmetic and other operations on single precision numbers can be modified slightly to operate on double precision numbers. In the DP commands for operations other than multiply, divide, shift, and normalize, the C code is four greater than in the corresponding SP command. This causes the execution of the command to continue for two word times (the first even, the second odd), recognizing a sign bit during the even word time only.

Additions and subtractions of double precision numbers require a 2-word arithmetic register; PN is used for this purpose. The DP operations of multiply, divide, shift, and normalize are performed in the same registers as the corresponding SP operations, except that both halves of the registers are necessarily involved. The T numbers in the commands for these latter operations must be even, and generally are greater than in the corresponding SP commands, but the C codes may be the same.

5.2 DOUBLE PRECISION ADDITION AND SUBTRACTION

(T must be even; $00 \le S \le 23$; $00 \le D \le 23$; each operates for 2 word times, T and T+1)

Notes	Commands	Descriptions
(S.T, T+1) → PN (S.T, T+1) → PN+ (S.T, T+1) → PN+ (S.T, T+1) → PN+ (S.T, T+1) - PN+ (PN) → D.T, T+1	TN5 S 26 TN5 S 30 TN4 S 26 TN6 S 30 TN7 S 30 TN5 26 D	Clear PN and add (S.T,T+1) Add (S.T,T+1) to (PN) Clear PN and add abs. value of (ST,T+1) Add abs. value of (S.T,T+1) to (PN) Subtract (S.T,T+1) from (PN) Store sum or diff. from FN into D.T,T+1

Discussion

$$(S.T,T+1) \xrightarrow{+} PN$$
 T N 5 S 26

The PN register will be cleared to zero. The double precision number in S.T, T+1 will be entered into PN in preparation for an addition or subtraction. T must be even.

The double precision number in S.T, T+1 will be added to (PN). T must be even.

The PN register will be cleared and the absolute value of the double precision number in S.T, T+1 will be copied into PN. The sign bit in PN will be 0 (positive). If the sign bit of the number in S.T, T+1 is negative, it will change (IP), but this will not affect the sign in addition or subtraction. T must be even.

$$(S.T,T+1) \rightarrow PN+ TN6S30$$

The absolute value of the double precision number in S.T,T+1 will be added to (PN). T must be even.

$$(S.T,T+1) \xrightarrow{-} PN+ T N 7 S 30$$

The double precision number in S.T, T+1 will be subtracted from (PN). I must be even.

$$(PN) \xrightarrow{+} D.T, T+1$$
 TN 5 26 D

The result of a double precision addition or subtraction in PN will be stored in D.T, T+1 in the form of absolute value and sign. T must be even.

5.3 DOUBLE PRECISION MULTIPLICATION AND DIVISION

(T must be even; 00\(S \le 23; 00\le D \le 23)

Notes	Commands	Descriptions
$(S.T,T+1) \longrightarrow ID$ $(S.T,T+1) \longrightarrow MQ$ $(S.T,T+1) \longrightarrow PN$ Multiply Divide $(PN) \longrightarrow PN$ $(PN) \longrightarrow D.T,T+1$ $(MQ) \longrightarrow D.T,T+1$	T N 4 S 25 T N 4 S 24 T N 4 S 26 V4 N O 24 31 V6 N 1 25 31 T N 4 26 26 T N 4 26 D T N 4 24 D	Load multiplicand or divisor Load multiplier Load numerator (ID)x(MQ)

Discussion

$(S.T,T+1) \longrightarrow ID$ TN4 S 25

A double precision multiplicand or divisor will be loaded from S.T, T+1. Its sign enters IP and its absolute value enters the ID register, replacing the prior contents of each. PN will be entirely cleared, so the "clear" command (L3 N 0 23 31) is not necessary before loading the registers for DP multiplication. Before division, however, certain bits of the 2-word registers must be zero; hence, the registers should be cleared before a divisor is loaded. ID should be loaded before MQ or PN. T must be even; execution times are T and T+1.

$$(S.T,T+1) \longrightarrow MQ$$
 TN4 S 24

A double precision multiplier will be loaded into MQ from S.T, T+1 while its sign combines with the sign of the multiplicand to form the correct product sign in IP. T must be even; execution times are T and T+1.

$$(S.T,T+1) \longrightarrow PN$$
 T N 4 S 26

A double precision numerator will be loaded into PN from S.T,T+1. Its sign combines with the sign of the divisor to form the correct quotient sign in IP. T must be even; execution times are T and T+1.

 $(ID)_X(MQ) \longrightarrow PN$. This command differs from that for single precision multiplication only in T, the length of time during which it operates. For efficiency, $N \longrightarrow L_7$. The location must be odd.

Divide v6 N 1 25 31 or v6 N 5 25 31

 $(PN)+(ID)\longrightarrow MQ$. This command differs from that for single precision division only in T, the duration of its execution. The quotient will fill MQ. For efficiency, $N \longrightarrow L_9$. The location must be odd.

 $(PN) \longrightarrow PN$ T N 4 26 26

This command will prepare a product for double precision addition in PN. The sign in IP will be copied into the register; if it is negative, (PN) will be complemented. (IP) does not change. T must be even.

 $(PN) \longrightarrow D.T, T+1$ TN4 26 D

The sign from IP and the absolute value of a double precision product from PN will be stored in D.T, T+1. T must be even.

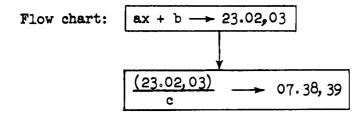
 $(MQ) \longrightarrow D. T, T+1$ T N 4 24 D

The sign from IP and the absolute value of a double precision quotient from MQ will be stored in D.T, T+1. T must be even.

Example 4: Arithmetic Operations, Double Precision

Given: a = (21.00,01) b = (21.02,03) c = (06.82,83)x = (23.00,01)

Find $y = \frac{ax+b}{c}$ store in 07.38,39



Coding: (see coding sheet on facing page)

Note that storing y in 07.u0, ul would be more efficient.

The selection of command locations was based on the word positions given for $\ensuremath{\text{c}}_{\circ}$



Los Angeles 45, California

Page 1 of 1 Date:_____

G-15 D Program Problem

4) $y = \frac{ax+b}{c}$

Prepared by G. M. Silvern (Double Precision)

Line 00

PROG	RAM	PROB	LEM:	4) y		С	(Dodo	1.0 1	1001			Line 00
0	1	2	3	L	P	or L _k	N	C	S	D	ВP	NOTES
4	5	6	7	47		48	50	4	23	25		$x = (23.00,01) \longrightarrow ID$
8	9	10	11	50		52	57	4	21	24		a = (21.00,01)
12	13	14	15	57		v 4	64	0	24	31		Multiply
16	17	18	19	64		66	68	4	26	26		$ax = (PN) \longrightarrow PN$
20	21	22	23	68		70	72	5	21	30		b = (21.02,03) + PN+
24	25	26	27	72		74	78	5	26	23		$ax + b = (PN) \longrightarrow 23.02,03$
28	29	30	31	78		81	81	0	23	31		Clear
32	33	34	35	81		82	84	4	06	25		c = (06.82,83) → ID
36	37	38	39	84		86	89	4	23	26		$ax + b = (23.02, 03) \longrightarrow PN$
40	41	42	43	89		v 6	98	1	25	31		Divide
44	45	46	47	98		38		4	24	07		$y = (MQ) \rightarrow 07.38,39$
48	49	50	51									
52	53	54	55									
56	57	58	59									
60	61	62	63									
64	65	66	67									
68	69	70	71	ļ. 								
72	73	74	75	ļ							 	
76	77	78	79									
80	81	82	83							ļ		
84	85	86	87					<u> </u>				
88	89	90	91	<u> </u>								
92	93	94	95	ļ 	ļ	<u> </u>		_		ļ	_	
96	97	98	99	ļ		<u> </u>		ļ	ļ		<u> </u>	
uo	u1	u2	u3			ļ	-	_	ļ		<u> </u>	
U4	u5	u6										

6. COPY AND BLOCK OPERATIONS

6.1 COPY OPERATIONS

The purpose of a copy operation is usually one of the following:

- (1) Distribution of input data to permanent storage.
- (2) Entry of numbers into arithmetic registers.
- (3) Storage of the result of an arithmetic operation.
- (4) Relocation of data for more efficient accessibility.
- (5) Placement of data in preparation for output.

To accomplish these purposes, numbers are sometimes transmitted without delay or change in form; sometimes they are modified in passage from the source line to the destination; and sometimes they are routed via AR, either to interchange the contents of AR with the contents of a storage location or to introduce a one-word delay in transmission. The "C" code in a command determines whether or not information will pass through AR or be modified in transit. Information stored in the source line is never altered or lost unless, because of modification or delay via AR, the same line is selected as the destination.

Among the arithmetic operations previously discussed, those with D<31 may be considered special cases of copies, in accordance with (2) and (3) above. In the following discussion of basic copy commands, certain limitations on S and D are imposed only to exclude commands which are discussed specifically elsewhere in this manual. Copies involving two-word registers are discussed separately.

6.1.1	COMMANDS	FOR	COPY	OPERATIONS

Notes	Commands	Descriptions
$(s.T) \longrightarrow D.T$	TNOSD	00\(\sigma \subseteq 23, \text{ or S=28; 00\lefta D=23, } \) or D=28. Copy a single precision number into D.T
(S.T) -+ >D.T	TNlsD	or D=28. Copy a positive number into D.T; enter complement of a neg. number into D.T
$ \left\{ \begin{array}{l} (AR) \longrightarrow D.I \\ (S.I) \longrightarrow AR \end{array} \right\} $	TN2SD	(00\S\23; 00\D\23 \[\text{Interchange storage with (AR)}\] (00\S\23; 00\D\23
$ \begin{cases} (AR) \longrightarrow D \cdot T \\ (S \cdot T) \xrightarrow{+} AR \end{cases} $	T N 3 S D	Interchange storage with (AR), entering complement of (S.T) into AR if (S.T) < 0
(S.T, T+1)—→D.T, T+1	T _{even} N 4 S D	{00≤S≤23; 00←D≤23 Copy a double precision number (00≤S≤26; 00←D≤26
(S.T,T+1)-+D.T,T+1	T _{even} N 5 S D	Sopy a pos. double prec.num- ber; enter complement of neg. double prec. number

Discussion

$$(S.T) \rightarrow D.T$$
 TNOSD $(00 \le S \le 23, \text{ or } S = 28; 00 \le D \le 23, \text{ or } D = 28)$

This general form of the copy command, with C=0, will copy a single precision number from S.T into D.T, replacing the previous contents of D.T. There will be no change in the form of the number and no delay. (S.T) will not be destroyed by this operation.

Note that the contents of any location may be copied into any other location with a different T address by two commands: Copy from source to AR (T_1 N O S 28), followed by Copy from AR to destination (T_2 N O 28 D).

$$(S.T) \xrightarrow{+} D.T$$
 TN1SD $(00 \angle S \angle 26, \text{ or } S=28; 00 \angle D \angle 26, \text{ or } D=28)$

This command, with C=1, will copy a positive number into D.T, or enter the complement of a negative number into D.T, with no delay. This command may be used for the temporary storage of a number in a two-word register, providing that the same form of command is used when the number is returned to a longer line: $(S.T_1)$ - $\stackrel{+}{\longrightarrow}$ two-word register, and two-word register $\stackrel{+}{\longrightarrow}$ D.T₂. In this pair of commands, both T₁ and T₂ must be even or both must be odd.

$$\begin{cases}
(AR) \longrightarrow D \cdot T \\
(S \cdot T) \longrightarrow AR
\end{cases}$$
T N 2 S D $(00 \le S \le 23; 00 \le D \le 23)$

This command, with C=2, will cause the original contents of AR to be copied into D.T, replacing the previous contents of D.T, while at the same time the contents of S.T are being copied into AR. There will be no change in the form of the numbers. (S.T) will not be destroyed by this operation unless S=D. This command, called "Interchange storage with (AR)", will operate for word time T only.

If either S or D or both \geq 28, and D \neq 31, then this command will copy into D.T the absolute value of (S.T).

$$\begin{cases} (AR) \longrightarrow D.T \\ (S.T) \stackrel{+}{\longrightarrow} AR \end{cases} T N 3 S D \quad (00 \leq S \leq 23; 00 \leq D \leq 23)$$

This command, with C=3, will cause the original contents of AR to be copied into D.T, while at the same time a positive number in S.T will be copied into AR or a negative number in S.T will be entered into AR in complementary form. This command will be executed during word time T only.

If either S or D or both≥28, and D≠31, then this command will change the sign coming from the source. If the new sign is positive, the absolute value of the number will be copied into D.T; if the new sign is negative, the complement of the number will enter D.T.

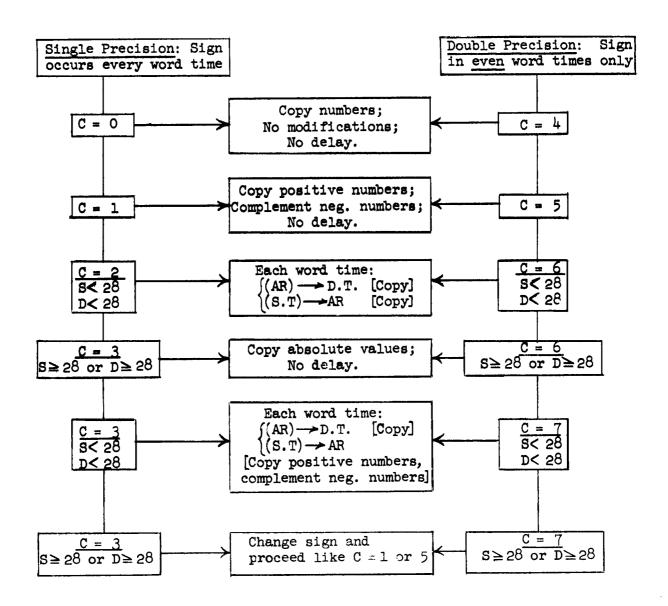
$$(S.T,T+1) \longrightarrow D.T,T+1$$
 $T_{even} N \downarrow S D$ $(00 - S - 23; 00 - D - 23)$

This command, with C=4, will copy a double precision number from S.T,T+1 into D.T,T+1. This command will operate during word times T and T+1, but otherwise it is similar to the corresponding single precision operation: T N O S D. T must be even.

$$(S.T,T+1) \xrightarrow{+} D.T,T+1$$
 $T_{even} N 5 S D (00 \le S \le 26; 00 \le D \le 26)$

This double precision command, with C=5, is similar to the corresponding single precision command: T N 1 S D. A positive double precision number will be copied; a negative DP number will be complemented. T must be even.

6.1.2 EFFECTS OF C CODES IN COMMANDS WITH D < 31



6.1.3 COPY OPERATIONS INVOLVING TWO-WORD REGISTERS

The two-word registers MQ (line 24), ID (line 25), and PN (line 26) are unique among G-15 memory lines, not only because of their function in arithmetic operations, but also because provision is made for the external storage of a bit representing the sign of their contents. The chief function of this one-bit storage element (IP) is to combine properly the signs of factors entering a multiplication or division to yield the correct sign for the product or quotient. The basic commands necessary to program these operations have already been discussed; below is a complete summary of the conditions governing the entry of numbers into the two-word registers, the transmission of their contents to storage, and the effects of commands to copy information from one register into another.

С	CODE	SOURCE	DESTINATION	EFFECT	
		00≤S ∠23 or 27∠S≤31	D=25 (ID)	Sign copied into IP; abs. value copied into ID. During each word time, the corresponding half of PN is cleared.	
		S=24 (MQ) S=25 (ID) S=26 (PN)	D=25 (ID)	(IP) unchanged; abs. value copied into ID. During each word time, the corresponding half of PN is cleared.	
		00 ≤ S ≤ 23 D=24 (MQ) or or 27 ≤ S ≤ 31 D=26 (PN)		(IP) unchanged by negative sign; abs. value copied into destination; No other register affected.	
	D=0	S=24 (MQ) S=25 (ID) D=24 (MQ S=26 (PN)	D=57 (MÅ)	(IP) unchanged; Absolute value copied into MQ; No other register affected.	
	C=4	S=24 (MQ) or S=25 (ID)	D=26 (PN)	(IP) unchanged; Absolute value copied into PN; No other register affected.	
	!	S=26 (PN)	D=26 (PN)	(IP) unchanged, but copied into sign bit in PN. (PN) unchanged if positive, complemented if negative. No other register affected. Normally C=4.	
		S=24 (MQ) 00∠D∠23 S=25 (ID) or S=26 (PN) 28∠D∠30		(IP) unchanged, copied into sign bit in destination with absolute value copied from the source register.	
		S=24 (MQ) S=25 (ID) S=26 (PN)	D=27	(IP) unchanged; Absolute value of the source contents tested for non-zero.	

When the operation copies an absolute value (C=2 or 6 and S or D equals or exceeds 28), IP is not changed or otherwise involved; and PN is not cleared if $S \leq 28$ and D=25 (ID).

When the destination is 30 (PN+), the sign bit enters the register without affecting IP, regardless of C.

C CODE	SOURCE	DESTINATION	EFFECT
	00 ≤S≤ 23 or S=27	D=25 (ID)	Sign copied into IP; absolute value copied into AR. EVEN word time: IDo and PNo cleared. ODD word time: (AR) copied into ID1; PN1 cleared. If C=6, 29 bits from source copied into AR, (IP) unchanged.
	S=24 (MQ) S=25 (ID) S=26 (PN)	D=25 (ID)	(IP) unchanged; absolute value copied into AR. EVEN word time: IDo and PNo cleared. ODD word time: (AR) copied into ID1; PN1 cleared. If C=6, 29 bits from source copied into AR.
C=2	or D=26 (PN S=27 D=26 (PN S=24 (MQ) D=24 (MQ) S=25 (ID) or	D=24 (MQ) or D=26 (PN)	(IP) changed by negative sign; absolute value copied into AR. EVEN word time: MQO or PNO cleared. ODD word time: (AR) copied into MQ1 or PN1. If C=6, 29 bits from source, copied into AR, (IP) unchanged.
or C=6		I	(IP) unchanged; absolute value copied into AR. EVEN word time: MQO or PNO cleared. ODD word time: (AR) copied into MQ1 or PN1. If C=6, 29 bits from source copied into AR.
	S=24 (MQ) S=25 (ID) S=26 (PN)	00∠⊅∠23	(IP) unchanged, copied into bit 1 of AR with absolute value copied from source register. If C=6, 29 bits copied from source into AR during odd word time. (AR) copied into destination each word time.
	S=24 (MQ) S=25 (ID) S=26 (PN)	D=27	(IP) unchanged; absolute value copied into AR. If C=6, 29 bits copied from source into AR during odd word time. (AR) tested for non-zero each word time.

A command with C=1, 3, 5, or 7 operates on the contents of a two-word register, either as source or destination, exactly the same as on the contents of any other memory line. If C is odd, IP is not involved: Its content is unchanged when MQ, ID, or EN is the destination, and it does not contribute the sign when one of these registers is the source. A sign is stored in Bit 1 of the register itself: in each word if C=1 or 3, or in only the even word if C=5 or 7. Also, if D=25 (ID), then PN is not cleared.

To summarize copies involving two-word registers in another manner, the following rules may be applied:

- 1. Even characteristics involve the IP flip-flop when a two-word register is either a source or a destination, but not when both source and destination are two-word registers (see rule 3).
- 2. Odd characteristics never involve the IP flip-flop; sign is taken from bit 1 of source and put into bit 1 of destination.
- 4. With a C code of O or 4 and destination ID, PN is cleared during each word time of operation.
- 5. With a C code of 2 or 6, a copy via AR is performed with the following differences:
 - a. During an even word time, a two-word register destination is cleared instead of getting the contents of AR.
 - b. If the destination is ID, PN is also cleared during each word time of operation.
 - c. If S or D equals 28, the command is an "add absolute value" and the usual rules apply.
- 6. Rules for IP flip-flop: (even C, source not a two-word register)
 - a. If destination is ID, IP takes the sign of the source.
 - b. If destination is MQ or PN, the sign of the source is added to the sign in the IP.

6.2 BLOCK OPERATIONS

A G-15 command contains information which not only specifies the operation to be performed, but also controls the time of its execution. In some cases, the operation begins immediately after the word time corresponding to the location of the command; in other cases, the execution is deferred until a time later in the 108 word-drum cycle.

A command in which D=31 is normally immediate in operation. The commands for multiplication and division initiate operations which last for as many word times as specified by the number in the T position of the commands. In the shift and normalize commands, this number is an upper limit for the number of word times of execution. These four commands (S=24,25,26,27;D=31) constitute a group in which the T number is relative; i.e., it indicates the length of the operation, regardless of the specific word time in which the execution begins. In other immediate commands with D=31, such as "clear" (discussed above), a number L_k is written in place of T. This specifies the first word time following the end of the execution which then occurs during word times L_1 , L_2 , ..., L_{k-1} .

For a command with D < 31, the operation is normally immediate only if $T=L_1$; otherwise, it is normally deferred. In either case, the execution lasts for one word time, T, if the operation is single precision (C < L) or if T is odd; it lasts for two word times, T and T+1, if the operation is double precision $(C \ge L)$ and T is even. The time of execution is independent of the location of the command.

One of the virtues of the G-15 is the possibility, by slight modification of a command with D<31, to extend its execution time so as to cause it to operate on a sequence, or block, of numbers stored in consecutive locations in the same line of memory. The command must be located exactly one word position earlier than the first of the sequence of numbers, though not necessarily in the same line. The modification consists of including a prefix "u" with the command and specifying T as a number one greater than the location of the last word entering the operation. A command so modified can be written to operate on a block of any length from one to 108 single precision, or 54 double precision, numbers.

For example, if the sum of the single precision numbers in the block stored in consecutive locations 08.35, 08.36, ..., 08.44 is to be added to the contents of AR, this may be accomplished by one command in location 34: u 45 N 1 08 29. In general, if a command in L_0 is written u L_{k+1} N C S D, its execution will occur during word times L_1 , L_2 , ..., L_k . If D=31, the prefix "u" is unnecessary and may be omitted.

Note that the execution times for a block command do not depend on whether the command is to operate on single or double precision numbers. The distinction between single precision (SP) and double precision (DP) in a block command consists in the recognition of signs every word time for SP $(C < \downarrow)$, but only during even word times for DP $(C \ge \downarrow)$. Resultant complementation, if it is to occur, will affect individual words in a block containing negative SP numbers or pairs of words in a block containing negative DP numbers.

6.2.1 BLOCK COPY COMMANDS

Notes	Commands	Descriptions
$(s.L_1,,L_k) \longrightarrow p.L_1,,L_k$	u \mathbf{L}_{k+1} N O S D \langle	0045423; 004D423. Block copy; Copy block of k numbers from line S into corresponding positions of line D.
$(S.L_1,,L_k) \longrightarrow D.L_2,,L_{k+1}$ $(AR) \longrightarrow D.L_1; (S.L_{k+1}) \longrightarrow AR$	u L _{k+2} N 2 S D	00≤S≤23; 00≤D≤23. Block copy k numbers with delay of 1 word time.
N.T-→18	L ₁ N 1 31 31	Copy number track into line 18.

Discussion

$$(S.L_1,...,L_k) \longrightarrow D.L_1,...,L_k$$
 u L_{k+1} N O S D $(00 \le s \le 23; 00 \le p \le 23)$

This command, with C=O, will copy a block of k single precision numbers from line S into the corresponding positions of line D, with no change of form. If L_0 is odd and k is even, then $\frac{K}{2}$ double precision numbers will be copied. The contents of line S remain unchanged. This command operates during word times L_1 through L_k .

Example: Clear line 19.

L	P	$\mathtt{T}_{\mathtt{L}_{\mathbf{k}}}^{\mathtt{or}}$	N	С	S	D	вР	NO TES
47		50	50	0	23	31		Clear 2-word registers
50	u	51	N≥51	0	24	19		0 = (MQ) → 19

The first command clears both words in MQ (as well as ID and PN). The second command operates for 108 word times. The odd-numbered positions in line 19 are cleared by copying zero from MQ1; the even-numbered positions are cleared by copying (MQ_0) .

During each word time that this block command operates, the contents of AR will be copied into line D while being replaced in AR by the contents of line S. Thus, after two word times, the original contents

of AR have been copied into $D.L_1$, $(S.L_1)$ have been copied into $D.L_2$, and AR contains a copy of $(S.L_2)$. The shift in word position between S and D is called a precession. S and D may be the same line, but not necessarily so.

Example:	Command	in	loc.	71:	u	77	N	2	21	21.
----------	---------	----	------	-----	---	----	---	---	----	-----

Word Time	Initial (21)	Final (AR)	Final (21)	Word Position
71	d	х	d	03
72	a	-> a	x	00
73	ъ	b	a	01
74	с —	- c	ъ	02
75	d	d	c	03
76	x	- x	d	00
77	a	x	a	01

The table illustrates the operation by showing the contents of line 21 and AR before, during, and after the execution of the command. The entire contents of line 21 are precessed one word position; the original contents of AR are restored. There is no change in the form of any number involved.

$NT \rightarrow 18$ $L_1 N 1 31 31$

This special command will copy into line 18 the contents of a line (not otherwise addressable) called the Number Track. The contents of the Number Track, loaded automatically from punched tape when the DC power is turned on, are used by the control circuits to identify the word times corresponding to word positions in the memory lines. This command makes the Number Track available to a programmer for checking purposes or for convenient output to a new punched tape. Line 18 must be cleared before this command is given; otherwise, the Number Track is superimposed upon the contents of line 18. Superposition does not occur in any other copy operation.

6.2.2 BLOCK COMMANDS FOR ADDITION AND SUBTRACTION (0045423)

Notes	Commands	Descriptions
$\sum_{i=1}^{k} (S.L_{i}) \xrightarrow{+} AR+$	u L_{k+1} N 1 S 29 $\left\{$	Add k numbers $(S.L_1) + + (S.L_k)$, to (AR)
$\sum_{j=1}^{k} (S.L_{j}) \longrightarrow AR+$	u L_{k+1} N 2 S 29 \langle	Add the abs. values of k numbers, $ (S.L_1) $ ++ $ (S.L_k) $, to (AR)
$\sum_{i=1}^{k} (S.L_i) \xrightarrow{\longrightarrow} AR+$	u L _{k+1} N 3 S 29	[Subtract k numbers, $[(S.L_1) + + (S.L_k)]$, from (AR)
$\sum_{i=1}^{k} (s.L_{2i-1},L_{2i}) \xrightarrow{+} PN+$	u L_{2k+1} N 5 S 30	Add k double precision numbers, (S.L ₁ ,L ₂) ++ (S.L _{2k-1} ,L _{2k}), to (PN) (Location, L ₀ , odd) (L _{2k+1} is even)
$\sum_{j=1}^{k} (s.L_{2j-1},L_{2j}) \longrightarrow PN+$	u L_{2k+1} N 6 S 30	Add the abs. values of k double precision numbers to (PN) (Loc. odd)
$\sum_{i=1}^{k} (S.L_{2i-1}, L_{2i}) \xrightarrow{\longrightarrow} PN+$	u L _{2k+1} N 7 S 30	Subtract the sum of k double precision numbers from (PN) (Location odd)

Discussion

$$\sum_{i=1}^{k} (S.L_i) \xrightarrow{+} AR+ \qquad u L_{k+1} N 1 S 29$$

The sum of k single precision numbers, (S.L1) + (S.L2) +...+ (S.Lk), will be added to the contents of AR.

$$\sum_{i=1}^{k} |(s.L_i)| \longrightarrow AR+ \qquad u L_{k+1} N 2 S 29$$

The sum of the absolute values of k single precision numbers, $|(S.L_1)| + |(S.L_2)| + \ldots + |(S.L_k)|$, will be added to (AR).

$$\sum_{i=1}^{k} (S.L_i) \xrightarrow{-} AR+ \qquad u L_{k+1} N 3 S 29$$

The sum of k single precision numbers, $\left[(S.L_1) + (S.L_2) + ... + (S.L_k)\right]$, will be subtracted from (AR).

$$\sum_{i=1}^{k} (S.L_{2i-1}, L_{2i}) \xrightarrow{+} PN+ u L_{2k+1} N 5 S 30$$

The sum of k double precision numbers, $(S.L_1,L_2) + (S.L_3,L_1) + \dots + (S.L_{2k-1},L_{2k})$, will be added to the contents of PN. The location of this command must be odd; L_{2k+1} must be even.

$$\sum_{i=1}^{k} (s.L_{2i-1},L_{2i}) \longrightarrow PN+ u L_{2k+1} N 6 S 30$$

The sum of the absolute values of k double precision numbers, $(S.L_1,L_2) + (S.L_3,L_4) + \dots + (S.L_{2-1},L_{2k})$, will be added to (PN). The location of this command must be odd; L_{2k+1} must be even.

$$\sum_{i=1}^{k} (S.L_{2i-1},L_{2i}) \xrightarrow{\longrightarrow} PN+ u L_{2k+1} N 7 S 30$$

The sum of k double precision numbers, $(S.L_1,L_2) + (S.L_3,L_1) + \ldots + (S.L_{2k-1},L_{2k})$, will be subtracted from (PN). The location of this command must be odd; L_{2k+1} must be even.

Example 5: Block Addition and Subtraction, Double Precision

Given:
$$x_1 = (20.00,01), x_2 = (20.02,03)$$

 $y_1 = (12.08,09), y_2 = (12.10,11), ..., y_{20} = (12.46,47)$

Find the sum of the 20 double precision numbers, $y_1 + y_2 + \cdots + y_{20}$, and subtract x_1 and x_2 from this sum. Store the result in 21.00,01.

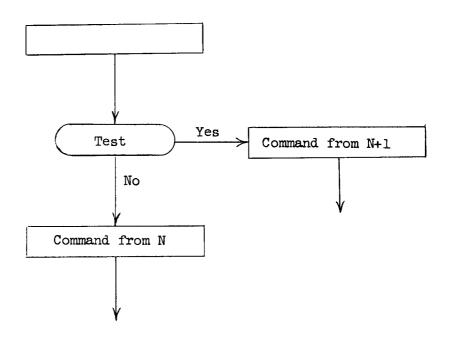
L	P	or L _k	N	С	s	D	BP	NOTES
04		07	07	0	23	31		Clear
07	u	48	49	5	12	30		$\sum_{i=1}^{20} (y_i) = (12.08, 0946, 47)$ $\stackrel{+}{=} PN+$
49	u	54	54	7	20	30		$x_2 + x_1 = (20.02, 03) + (20.00, 01) \xrightarrow{-} PN+$
54		56	N ≥58	5	26	21		(PN) + ≥ 21.00,01

7. COMMAND SEQUENCE CONTROL

7.1. TEST COMMANDS

Each command has within it the location of the next command, N. The next command will normally be read from word N of the same command line.

Under certain conditions, however, it is possible to deviate from this sequence and take the next command from location N+1 of the command line. When such a test command is executed, the location of the next command depends upon whether or not the test conditions are met. If the condition does not exist, the next command is taken from N; if the condition does exist, the next command is from N+1. Thus, the computer has the ability to make comparisons, choices and decisions, the succeeding sequence of commands depending upon the result of the test.



Test Commands

Notes	Commands	Descriptions
(S.T) → test	TNO S 27	(Test for non-zero (single word) (If (S.T) = 0, next comm. from N (If (S.T) ≠ 0, next comm. from N+1
$ \begin{cases} (AR) \longrightarrow \text{test} \\ (S.T) \longrightarrow AR \end{cases} $	TN2 S27	{Test (AR) for non-zero, then copy (S.T) into AR [S < 28]
		{Test for non-zero (double pre- cision). T must be even.
AR sign → test	L ₂ N O 22 31	Test for (AR) negative If (AR) not negative, next command from N If (AR) negative, next command from N+1
Ready> test	иио 2 8 31	Test for end of regular input- output operation. If not ready, next command from N If ready, next command from N+1 Set N=LO to repeat test until ready
Overflow -> test	L ₂ N O 29 31	Test for overflow If no overflow since last test, next command from N If overflow, next command from N+1
PUNCH sw. → test	T N 1 17 31	Test for PUNCH switch ON T=L ₁ rings bell; T=L ₂ does not N=L ₋₁ repeats test if switch ON N=L ₀ repeats test if switch off

Discussion

TNO S 27

In this test for non-zero, if the contents of S.T are equal to zero, then the next command will be taken from word N of the command line; if $(S.T) \neq 0$, then the next command will be taken from word N+1 of the command line.

This test command is particularly useful for leaving a loop when a tally is kept. The tally is compared with the final count desired. As long as the difference is not zero, the next command from N+1 effects a return to the loop. When the difference equals zero, the next command is taken from N and a new sequence of commands is followed.

$$\begin{cases}
(AR) \longrightarrow \text{test} \\
(S.T) \longrightarrow AR
\end{cases}$$
T N 2 S 27

This command will test (AR) for non-zero (as in the previous command) and then will copy (S.T) into AR. S must be less than 28. This command will operate during word time T.

$$(S.T, T+1) \longrightarrow \text{test}$$
 $T_{\text{even}} N + S = 27$

This double precision command will test (S.T, T+1) for non-zero, in the same manner as the corresponding single precision command. It will operate for word times T and T+1; T must be even.

AR sign
$$\rightarrow$$
 test L₂ N 0 22 31

The sign bit of AR will be tested for non-zero [(AR) negative] If this sign bit is zero [(AR) positive or zero], then the next command will be taken from word N of the command line; if the sign bit is <u>not</u> zero [(AR) negative], then the next command will be taken from word N+1 of the command line. This command will be executed during word time L_1 .

This test command is useful in determing when a limit has been reached by detecting a change of sign.

Ready → test

N N O 28 31

The "ready" state is a condition such that no input or output operation is in progress and that any input or output operation may be initiated.

This command will test for the end of a regular input or output operation. If the input-output circuitry is still processing earlier information, the next command will be taken from location N of the command line. If N=I_O, this is the command itself, so that if written I_O I_O 0 28 31, this test will be repeated as many times as necessary until the information has been completely processed. If the input-output circuitry is ready to handle new information, the next command will be executed from location N+1 (or $\rm L_1)$ of the command line.

Overflow --- test

L₂ N O 29 31

An overflow condition exists when, as a result of addition, subtraction, or division, the contents of AR, PN or MQ are incremented beyond the limitation that they may not equal or exceed 1 in magnitude. The overflow indicator light will go ON when an overflow condition exists and will remain ON until an overflow test is made.

In the test for overflow command, if there has been \underline{no} overflow since the last overflow test (the contents of the registers have remained between +1 and -1), then the next command will be taken from location N of the command line. If an overflow has occurred, the next command will be executed from location N+1 and the overflow indicator light will be turned off. This command will be executed during word time L_1 .

Setting \mathbb{L}_{1} (one less than the location) will turn the overflow light off whether or not there has been an overflow.

PUNCH sw. → test

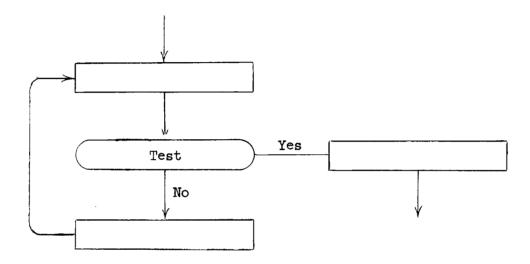
T N 1 17 31

This command will test the position of the PUNCH switch for ON. If the switch is off, the next command will be taken from location N of the command line; if the switch is ON, the next command will be taken from N+1 of the command line. This command will operate during word times L_1 through T-1.

This test is most valuable when it is used to alert the operator to turn the switch on or off. If T is made equal to L_1 , this command will ring a bell in addition to testing the punch switch; $T = L_2$ will not cause the bell to ring. If N is made equal to L_1 (one less than the location), the test will be repeated if the switch is ON, and it will continue to repeat until the switch is sturned off. If $N=L_0$, the test will be repeated if the switch is off, and it will continue to repeat until the switch is turned ON.

7.2 LOOPS

It is possible to select as the next command one that had previously been executed, causing the sequence of commands to form a <u>loop</u>. In order to leave the loop after going through it the proper number of times, it is necessary to include one or more tests. When the tested condition changes, the exit is made from the loop and a new sequence of commands may be followed.



7.3 SUBROUTINES

A subroutine is a sequence of operations stored apart from the main routine and called in when necessary. The subroutine may be a service routine supplied by Bendix. A list of available subroutines together with their specifications, flow charts, coding sheets and other information, is provided with the computer. Additional subroutines are composed by the programmer when needed.

In using a subroutine, the input data called for in the subroutine is first stored in the proper locations according to the specifications, and then control is transferred to the first command of the subroutine as specified. A return command at the end of the subroutine returns control to the main routine.

7.4 COMMAND LINE SELECTION

Commands must be executed from one of eight lines, called <u>command lines</u>, or from the AR register. The command lines are long lines 00, 01, 02, 03, 04, 05 and 19, and short line 23. However, insofar as possible, line 05 should be reserved for certain useful service routines which are executed from this line. Also, words 00 through 03 of lines 02 and 03 serve special format functions to be described later, and these words should not be used for commands. A command may be stored in any memory location, but it must be transferred to one of the specified command lines or to AR before it can be obeyed by the computer.

In order to deviate from the sequence of taking succeeding commands from the same command line, a command may be given to transfer control to a new command line. This may be necessary when there are too many commands for one line or when a subroutine already stored in another line is to be used.

Notes	Commands	Descriptions
Mark → C	w T N C 21 31	Next command from C.N; Mark word T
	$\begin{cases} L_2 & L_1 & C & 20 & 31 \\ N & N & C & 20 & 31 \end{cases}$	(Next command from C.T. (marked word in line C)
Return C	•	Next command from C.N (Next command from C.N
	(r N C 50 31	if L ₂ N marked T; other wise from C.T
N.C. from AR	r ⁵ N 0 31 31	Next command from AR.N
Halt	L ₂ N 0 16 31	Halt
Discussion		
	M M G O1 31	

$$Mark \longrightarrow C \qquad w T N C 21 31$$

This command will select a new command line and will "mark" T (remember word position T) to be used if the "Return" command below is given at a later time. The next command will then be taken from location N of new line C. C ranges from O to 7 according to the following table:

C	0	1	2	. 3	14	5	6	7
Line	00	01	02		04	05	19	23

All subsequent commands will be taken from line C until another command is specified.

T should be noted for later use with the return command. The command is executed during word time T. In the case where $T=L_1$, the form w L_1 N C 21 31 would cause an unnecessary extra drum revolution, so that in this event it would be better to omit the prefix and use the block form L_2 N C 21 31.

Return \longrightarrow C L₂ L₁ C 20 31

This form of the return command will cause the next command to be taken from location T (the marked word specified in the last "Mark" command given) of line C (a line specified by the return command). All subsequent commands will be taken from line C until another command line is specified.

Line C may or may not be the same command line as that being used when T was marked.

This return command may be used as an exit command from a subroutine with control returned to the main control line. This command should not be break-pointed. Single cycle operation may also cause the next command to be taken from the wrong location.

Return -- C L₂ N C 20 31

This form of the "Return" command is conditional and depends upon the relationship between L_2 , N and the marked T. If $L_2 \le N$ and N \le marked T, the next command will be taken from location N of line C. Otherwise, the next command will be taken from the marked word T of line C.

When L_k is made equal to N, the next command will be taken from location N of line C. Thus, the form N N C 20 31 can be used as an unconditional transfer; taking the next command from C.N without leaving a mark or losing any existing mark.

If any form of the return command is single cycled, or if it is break-pointed and BP operation is used, the next command will be taken from C.T and the halt will occur after the execution of the command which follows the return command.

N.C. from AR L₂ N O 31 31

At word time N the contents of AR will be read and then executed as if it were a command in location N of the original command line. After the command in AR has been obeyed, control will return to the command line containing the "N.C. from AR" command.

This command is often used when a command has been modified in AR by addition or subtraction before it is to be executed, or when one of several commands is selected by the program in accordance with existing conditions.

L₂ N O 16 31

The "Halt" command will cause computation to stop. To resume operation, the COMPUTE switch must be switched to the middle position and then back to the GO or BP position. Computation will resume with the command in the location N specified by the halt command.

Break-point operation: The computer may be programmed to come to a halt after performing a specific command. To order the computer to do so, a minus sign (-) is added as a suffix to the command and the COMPUTE switch is set to the "BP" position.

The "COMPUTE" switch has three positions: BP, or break-point; the middle position, which is HALT; and "GO", which is the normal position in which computation occurs. If the switch is set to "GO", computation will proceed and any coded break-points in commands (minus signs) will be disregarded. If the switch is set to "BP", computation will proceed but will cease after a command coded for break-point is obeyed.

To re-continue computation after a break-point has been reached, the switch is set to "HALT" and then again to "BP" or "GO". The computer will then obey the command in the "N" address of the break-pointed command.

7.5 DRUM MAPPING

A record should be kept of where commands and constants are located, where the various subroutines are stored, and where data and other information are to be found in memory. This drum mapping should be planned along with the coding of the program. A Memory Allocation Work Sheet prepared by Bendix is also useful in keeping track of the locations of commands, constants, data, intermediate and final results and temporary working storage.

Halt.

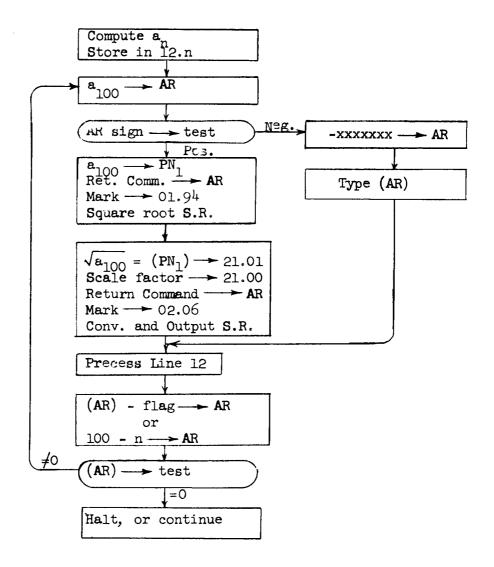
Example 6: Command Sequence Control: Test Commands, Loops Command Line Selection, Use of Subroutines

Given: 100 values of a_n obtained and stored in words 1 through u0 of line 12.

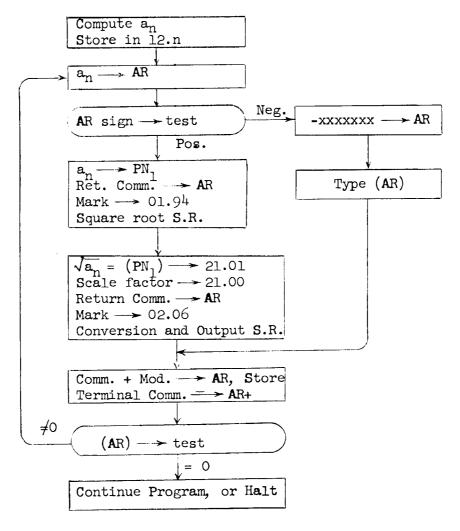
Find and type the 100 values of $\sqrt{a_n}$

Storage locations: $a_1 = (12.01)$, $a_2 = (12.02)$, ..., $a_{100} = (12.00)$ Square root subroutine is in line 01 Output subroutine (including binary to decimal conversion) is in Line 02.

Flow Chart, Method 1: Precessing Line 12



Flow Chart, Method 2: Modification of Command



Coding of Method 2 is on the facing page.

Computer Los Angeles 45, California

Page 1 of 1

G-15 D

Prepared by <u>G. M. Silvern</u>

PRO	PROGRAM PROBLEM: 6) Van											
0	×	X	×	L	P	or L _k	N	C	S	D	BP	
X	5	X	7									Compute and store an's. Scale factor -> 22.00
X	X)4	11	011		05	09	0	00.	22		Initial Comm. = (00.05) → 22.01
X	13	X	M	09		13	15	0	22	28		(22.01) —→ AR _C
16	17	18	19	15		17	17	0	31	31		N.C. from AR
20.	21	22	23	[AR [(17)		01	87	0	12	28		$\begin{bmatrix} a_1 = (12.01) \longrightarrow AR_c \end{bmatrix}$
24	25	26	27	87		89	89	0	22	31		AR sign —→ test
28	29	30	31	89		91	92	2	00	26		Pos. $a_n = (AR) \longrightarrow PN_1$ Ret. Command = $(00.91) \longrightarrow AR_c$
32	33	34	35	90		96	97	0	00	28		$\frac{\text{Neg.}}{-\text{xxxxxxx}} = (00.96) - \rightarrow AR_c$
36	37	38	39	97		99	02	0	08	31		Type (AR)
40	41	42	43	02		02	02	0	28	31		Ready> test
44	45	46	47	92		94	94	1	21	31		T = 93 Mark —→ 01.94 (√S.R.)
48	49	50	51	u0		ul	u2	0	26	21		$\sqrt{a_n} = (PN_1) \longrightarrow 21.01$
52	53	54	55	u2		ι 1 4	u 5	0	22	21		Scale factor = (22.00)—→21.00
56	57	58	59	u5		u6	01	0	00	28		Ret. Command = $(00.u6) \rightarrow ARc$
60	61	62	63	Ol	W	C	06	2	21	31		Mark → 02.06 (Output S.R.)
64	65	66	67	03		05	06	0	22	28	ا	Comm. = $(22.01) \longrightarrow AR_c$
68	69	70	71	06		07	08	1	00	29		Modify = (00.07) + AR+
72	73	74	75	08		09	10	0	28	22		Command + Modify = $(AR) \rightarrow 22.01$
76	77	78	79	10		11	12	3	00 -	29		Terminal Comm.= (00.11) \longrightarrow AR+
80	81	82	83	12		13	14	2	22	27		(AR) → Test ≠0 (22.01) → AR. Reenter loop —
84	85	86)H(14		16	00	0	1 6	31		Halt
88	89	90	91	05		01	87	0	12	28		Initial Command: a1=(12.01) -> ARc
92	93	94	95	11		ul	87	0	1.2	28		Terminal Comm ≤ alol=(12.ul)→ARc
96 DQ	94X U1	98	99	91		u0	u0	0	21.	31		Return Command, \(\subseteq \sin \subseteq \subseteq \subseteq \subseteq \subseteq \subseteq \s
•)) 2	u3	u6 07		38 01	37 00000	0	20	31		Return Command, Output S.R.
U4	945	u6		96			XXXXX					

8. INPUT-OUTPUT OPERATIONS

8.1 GETTING INFORMATION INTO AND OUT OF THE G-15

Commands and input information may be fed into the computer from a specially modified typewriter or from punched tape. The results of computation may be typed out and/or punched on paper tape. Also available but not necessary, are special input-output accessories: magnetic tape units (up to four units may be used, each with 300,000 word storage), punched card input-output converter, graph plotter or graph plotter/follower (for use with DA-1 Digital Differential Analyzer), and alphanumeric punched tape accessory.

Internal computation may proceed simultaneously with the insertion or read-out of information. However, certain lines should not be programmed for use during input or output. Line 23 is used by the computer during any input and during magnetic tape or fast punch output; therefore, line 23 should not be programmed for storage during times which may coincide with the execution of those operations. Line 19 is involved in input, and either line 19 or the AR register is involved in output.

Information read into the computer goes into line 19. Input enters into the lower numbered word positions of the line. Four words of information are read into locations 00 through 03; all words in line 19 are then shifted to the next higher numbered group of four word locations; and the process is repeated. The shifting is automatic and need not be programmed.

The read out of information is from either line 19 or the AR register. The order of read out is from highest numbered to lowest numbered word and from most significant to least significant digit. For line 19, the contents of word u7 are read out first; all the other words in the line are shifted to the next higher numbered word locations; and the process is repeated. A special control register (called the format register) controls the form or appearance of the typeout. Certain digits may be ignored, tabs and carriage returns may occur and periods may be inserted into the copy through format control. Format will be discussed later in more detail. The codes for the format characters are listed in the appendix (Table A-5) together with "standard" formats. Words 00 through 03 of line 02 are used to hold the format words which determine the form of typeout from line 19; words 00 - 03 of line 03 hold the format words which determine the form of typeout from the AR register. The line or register is cleared of its contents by the readout. The line 19 output operations is self-halting only after line 19 has been cleared of all non-zero information.

When a command is given to initiate an input or an output operation, the computer stays in the "execute" state just long enough to transmit a code number to the input-output unit, and then it reads the next command. The input-output unit interprets the code as an instruction and, after obeying the instruction, clears the code number from a static register. Since the input-output section operates independently of the rest of the computer, an input or output operation may proceed while the computer is carrying out computation.

Before any input-output operations may be initiated, however, the computer must be in the "ready" state. The "ready" state is defined as the state the computer is in after it has fully completed its last input-output operation, and therefore it is ready for the next one. The command "Test for Ready" should follow input. If two or more input-output operations are to be carried out in one program, the command "Test for Ready of Input-Output" should precede all but the first operation.

Commands are usually entered into the computer in decimal form and converted to binary form by a service routine, the "Program Preparation Routine" (PPR). This routine and how it is used is described in detail in the operating manual. However, a summary of the PPR instructions appears in Table A-11 of the Appendix, together with a summary of operating procedures with the PPR (Table A-9) and typical steps in using the PPR (Table A-10).

Data is usually entered into the computer in decimal form and converted to binary form by a subroutine incorporated into the program. The binary results of computation are normally converted to decimal form for output by the program or a subroutine incorporated into it.

External control may be exercised from the typewriter keyboard and the control switches mounted on the base of the typewriter. The functions of each of these keys and switches are summarized in Table A-7 of the Appendix; the operating manual contains more complete descriptions. This section will deal with the commands for input and output operations which can be programmed. Commands to be used with optional auxiliary equipment are listed in the Appendix and will be described in greater detail elsewhere in the literature for the specific equipment to which they refer, although some of the more common ones are included here.

8.2 REGULAR OUTPUT COMMANDS

Notes	Commands	Descriptions
Type (AR) Type (19) Punch (19) Punch cards M. tape write	L ₂ N O O8 31 L ₂ N O O9 31 L ₂ N O 10 31 L ₁ N O 11 31 woo N C O1 31	Type (AR) \(\) and punch with Type line 19 PUNCH switch ON Punch line 19 on tape Punch line 19 on cards Write on magnetic tape (C=# of magnetic tape unit)
File code	L ₅ N C 30 31	Write file code on magnetic tape. (C=# of m. tape unit)

Discussion

Type (AR) L₂ N 0 08 31

The contents of the AR register will be typed out under control of the AR format. If the standard AR format is used, AR will be cleared. If the PUNCH switch is ON, the contents of AR will also be punched on tape. The order of type-out is from the most significant to the least significant digit. The rate of type-out is approximately 8 characters per second.

The contents of line 19 will be typed out under control of the line 19 format. The order of type out is from the highest numbered to the lowest numbered word, and from the most significant to the least significant digit. The rate of type-out is approximately 8 characters per second. If the PUNCH switch is ON, the contents of line 19 will also be punched on tape, with the punch-out slowed down to occur simultaneously with the type-out at the same rate. Line 19 will be cleared by the execution of this command.

For each word typed out of 19.u7, the rest of line 19 is precessed one word, so that 19.u7 will then contain the next word to be typed. The output process will continue until all non-zero information has been typed out and line 19 contains only zeros.

The contents of line 19 will be punched on paper tape under control of the line 19 format, starting with word u7. The punch-out will continue until line 19 contains only zeros. The tape punch operates at approximately 17 characters per second. Line 19 will be cleared by the execution of this command.

Punch cards

Ll N O Ol 31

The contents of line 19 will be punched on IBM cards through an IBM 026 card punch and a Bendix CA-1 card adapter. A program card will control the punching. The rate of output is about 11 characters per second, with column skipping at the much faster rate of 80 columns per second. Line 19 will be cleared by the execution of this command.

Mag. tape write

w00 N C 01 31

The contents of line 19 will be written on magnetic tape at the rate of about 430 sexadecimal characters per second. When line 19 is empty, a stop code will be written on the tape.

If a "M. tape write" command follows a search magnetic tape command, at least 16 drum cycles must elapse after the "Ready" state is reached to allow the capstan drive motor to slow down. If this command follows a previous "M. tape write" or "Read m. tape" command, at least 15 word times must elapse between the time that the magnetic tape unit is stopped and the time that this command is given.

The number of the tape unit, 0, 1, 2 or 3, is put in the "C code" position of the command.

File code

L₅ N C 30 31

This command will cause a file code to be written on the magnetic tape. The tape will not be moved by this command, and it must not be moving when this command is given. The file code should be written between two blocks. The number of the tape unit involved, 0, 1, 2 or 3, is put in the "C" position of the command.

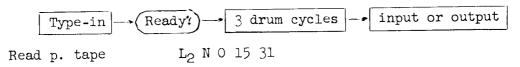
8.3 REGULAR INPUT COMMANDS

Notes	Commands	Descriptions
Type-in Read p. tape Read m. tape	L_2 N O 12 31 L_2 N O 15 31 L_2 N C 13 31	Read typewriter input Read punched tape (Read magnetic tape (C=# of mag. tape unit)
Read cards	L_1 N O 14 31	Read punched cards
Discussion		
Type in	L ₂ N O 12 31	

This command readies the computer to accept typewriter input. Information which is then typed will go into line 23. Up to four words may be filled, after which a slash (/) is necessary to copy the information from line 23 into line 19 before entering additional words. The striking of the "s" key will inform the computer that the input is ended. If the computer has been "sitting on a Ready' test", it will now continue with the next command.

The "Read type-in" command is also known as "Permit typewriter input", "Permit Type-in" and "Gate Type-in".

After a type-in is completed, at least three drum cycles must elapse before another input or output command can be obeyed.



The punched tape will be moved in the forward direction and read into line 19 until a stop code appears. Tape is read at approximately 200 characters per second. After this command, a Ready test must be programmed before information can be taken from line 19. This command may then be repeated to read in another block of tape.

A loading routine consists essentially of a "Read tape" command, followed by a "Ready test" and a "Store (19) into line x", repeated for each block of program tape to be loaded into the computer.

Read mag. tape Lo N C 13 31

The magnetic tape is run in the forward direction and read into line 19 until a stop code appears on the tape. File codes have no effect. Magnetic tape is read at approximately 430 characters per second.

If this command follows either a "M. tape write" or "File code" command, at least four drum cycles must elapse after the "Ready" state is reached between the two commands. After a search magnetic tape command, at least sixteen drum cycles must elapse after reaching the "Ready" state before a "read m. tape" command can be obeyed. At least 15 word times must elapse, after reaching the "Ready" state, between two "Read m. tape" commands.

The number of the tape unit is put in the "C Code" position of the command.

Read cards

L₁ N O 14 31

This command will cause punched cards to be read into line 19 from an IBM 026 unit with a CA-1 card adapter. Input rate is 17 characters per second.

8.4 COMMANDS FOR REGULAR INPUT-OUTPUT CONTROL

Notes	Commands	Descriptions
P. tape back M. tape search M. tape reverse	L_2 N O O6 31 L_{16} N C O5 31 L_{16} N C O4 31	Reverse punched tape Search magnetic tape, forward Search magnetic tape, reverse
Set ready	L ₂ N O OO 31	(C=# of magnetic tape unit) Set "Ready" for regular input- output
Ring bell	L ₁ N O 17 31	Ring bell (Special command)

Discussion

P. tape back L₂ N 0 06 31

This command to reverse punched tape will cause the punched tape to run in the reverse direction for one block. Although the photo reader light will come on, no information is read into the computer.

In order to reverse one block, the computer will actually read back for two stop codes and then read forward one. Thus, the tape will stop in the leader (blank tape) between two blocks, ready to read in or to read back again.

M. tape search L_{16} N C 05 31

The magnetic tape is searched for the next file code: the search speed is about six times as fast as the read magnetic tape speed.

The magnetic tape is run in the forward direction until a file code appears. The tape stops in front of the block following the file code. The information is not read into the memory. If this command follows a "Write on Magnetic Tape" or "Write File Code" command, at least four drum cycles must elapse after recording has ceased. If this command follows a "Search Magnetic Tape" or "Read Magnetic Tape" command, at least 15 word times must elapse after the computer has reached the "Ready" state.

The number of the magnetic tape unit involved is put into the "C" position.

The L_{16} N C 05 31 form of this command will locate and stop at the next file code unless it is very close to or under the read head. It will by-pass such nearby file codes but will stop at one which is at least four words away.

M. tape reverse L_{16} N C O4 31

The magnetic tape is searched for the previous file code: the speed of the reverse search is six times as fast as the reading speed.

The magnetic tape is run in the reverse direction until a file code appears on the tape. The tape stops in the block before the file code. To read the block following the file code, a "Search Forward" command should be given, followed by a "Read m. tape" command.

If this command follows a "M. tape write" or "File Code" command, at least four drum cycles must elapse after recording has ceased. If it follows a "M. tape search" or "Read m. tape" command, at least 15 word times must elapse after reaching the "Ready" state.

The number of the magnetic tape unit is put into the "C" position.

Set Ready L₂ N 0 00 31

This command stops any regular input or output that may be occurring and puts the computer in the "Ready" state. It will therefore be in a position to process new input or output information, even though a preceding input-output operation had not be properly terminated.

The "Set ready" command must be used cautiously, since in stopping inpur or output before completion it disturbs line 19. Thus, the input or output cannot later be resumed as if the "set ready" had not occurred.

This command is useful in a sequence of commands which cause leader to be punched between blocks of tape.

Ring bell L_1 N O 17 31

When this special command is reached, the bell will ring once.

This command should not be repeated more often than once every three drum cycles, since recovery time is necessary to get a clear chime. If the bell tries to ring once every drum cycle or every other drum cycle, it sounds more like a buzzer.

If the Punch switch test is given in the form L_1 N l 17 31, the bell will also ring. However, L_2 N l 17 31 will not cause the bell to ring since the one word time of execution, L_1 , is not sufficient to activate the bell driver. $T = L_1$ provides one drum cycle of execution time, which is ample to ring the bell.

8.5 OUTPUT FORMAT

Information typed out or punched out from the computer is under control of a block of four words called the format. If line 19 holds the output, its format is held in the first four words of line 02; when the AR register is to be typed, its format is held in the first four words of line 03. When an output command is executed, the format is copied from line 02 or line 03 into a special format register which then controls the output.

Word u7 of line 19 is the first to be processed in output; correspondingly, the first format character is taken from the three most significant bits of word 03, with succeeding format characters being taken from successive three bit groups.

Each format character is represented by a three bit code as shown in Table A-5 of the Appendix. The eight possible combinations of the three bits represent the following eight format characters:

- Sign (100): This character causes bit 1 of 19.u7 or AR (the bit which holds sign information) to be typed out as a space if zero (positive) or as a "-" sign if one (negative). In each word, the sign type-out should be first unless it is to be omitted altogether. (Internally, there is no shift of bits in the output line as a result of sign read out.)
- Digit (000): This character causes the information in the four most significant binary places of 19.u7 or AR to be read out as a single decimal or sexadecimal digit. The digit that is read is then shifted out of the computer. (Internally, the line is precessed in the more significant direction by four bits, with four zero bits entering the least significant positions of word 00.)
- Period (Oll): This character causes a period (decimal point) to be typed without affecting the information held in the output line. The period also stops zero suppression: that is, any zeros which follow the period are typed out. If leading digits to the left of the period are zeros, a space (suppressed zero) replaces the zero until the first non-zero digit is typed. Zeros following a non-zero digit or a period are not suppressed.

Wait (111): This character will inhibit the type-out of the next four-bit digit, but will shift the four bits out of the computer in the same manner as the digit character. It is useful in preventing the printing of digits which lack significance. The wait character will also cause zero suppression to start again, until another significant digit or period occurs.

Tab (110): This character causes the tab key to be pressed, and also causes the bit then in the most significant position to be shifted out of the computer. Thus, after seven digits (28 bits) have been typed out, the tab will shift out the bit which originally represented the sign.

Carriage

Return (010): This character causes the carriage return key to be pressed, and also causes the same one bit shift as the tab character. Thus, either a tab or a carriage return should normally follow each complete number.

End (001): This character will terminate an output from AR and put the computer in the "Ready" state. If line 19 is being typed or punched, the end character will terminate the output if line 19 is empty; but if there is any non-zero bit remaining in line 19, the end character will automatically be changed to a "reload" character.

Reload (101): The four words of format contain 116 bits which may be divided into a maximum of 38 format characters (three bits each). The format bits are precessed out of the format register as they are used. In the event that more than 38 format characters are needed, the format may be recopied into the format register from line 02 or line 03 by this reload character. This permits the typing or punching of up to a full line of 108 words with a single output command and without interrupting simultaneous computation. If the format in line 02 is changed before the reload is reached, the new format will control the remainder of the output. The reload character itself does not normally appear in the stored format, but results from an automatically changed end character.

In constructing a format, the three bit codes are strung out and then divided into sexadecimal words of seven digits and sign each. The first sexadecimal word belongs in word 03 of the format, followed by words 02, 01 and 00, in that order, if there are sufficient bits to occupy them. Anything may follow the end code, but zero bits are usually used to fill in the remainder of the word.

Two standard formats are often used for output from AR and line 19 These appear in Table A-5 in the Appendix.

A service routine to assist in the construction of formats is described in the operating manual.

9. EXTRACT OPERATIONS

9.1 EXTRACTION: A LOGICAL OPERATION

Although the unit of information transmitted from a source to a destination by a G-15 command is a word of 29 bits, it is sometimes necessary to consider individual bits or groups of bits within a word. This situation arises, for example, in programs for the conversion between the binary form of numbers in memory and the binary-coded decimal form involved in input and output operations, or in programs for arithmetic operations on numbers represented in "floating point" form. (A single precision floating point number may be represented by a fraction of, say, 20 bits and a sign, stored together in one word with eight bits specifying an integral power of the base 2 by which the fraction is multiplied.)

The separation of certain bits from the rest of a word is accomplished by an extract operation. At least two memory lines are involved in forming the resultant word: one line contains a number (called the "extractor") that specifies which bits are to be copied from the corresponding word in the other line. The extractor is composed of "one" bits in selected positions and "zero" bits elsewhere, and is usually a constant of the program. Both the extractor and the number on which it is to work must be stored in certain locations before the extract command is executed.

A typical extract operation is conveniently described in a notation borrowed from elementary symbolic logic (Boolean algebra). The expression a·b=c means that word c contains "l" bits in those positions where "l" bits occur in both numbers a and b, with "0" bits elsewhere. The expression $\overline{a} \cdot b$ =c means that c consists of "l" bits in those positions where "l" bits in b correspond to "0" bits in a, with other positions in c containing "0" bits. In this notation \overline{a} is read "not a", meaning a number having a "l" bit everywhere a has "0" and "0" wherever a has "l". The expression a+b=c means that "l" bits in c occur in positions corresponding to "l" bits in a or b or both. Note that $(a \cdot b) + (\overline{a} \cdot b) = (\overline{a} + a) \cdot b = b$.

```
For example, if a=.0000,0000,0000,0001,1111,1111,1111,1(binary)=-.0003zzz(hex) b=.1001,1000,0111,0110,0101,0011,0001,0(binary)= .9876531(hex) then a.b=.0000,0000,0000,0010,0101,0001,0001,0(binary)= .0002531(hex) a-b=.1001,1000,0111,0100,0000,0000,0000,0(binary)= .9874000(hex) a+b=.1001,1000,0111,0111,1111,1111,1111,1(binary)=-.9877zzz(hex)
```

In the following discussion of extract commands, examples are given in the briefer sexadecimal (hex) notation.

9.2 EXTRACT COMMANDS (D < 31)

Notes	Commands	Descriptions
(20·21) _T -> D.T	T N O 31 D	·
$(20.21)_{T} \rightarrow D.T$ $(20.21)_{T} + (20.AR)_{T} \rightarrow D.T$	T N O 30 D T N O 27 D	Single precision extraction
$\begin{cases} (02 \cdot PN)_T \rightarrow ID \\ and (02 \cdot PN)_T \rightarrow PN \end{cases}$	w T N 3 23 31	extraction
(20·21) → D.T,T+1	Teven N 4 31 D	
$(20 21) \rightarrow D.T, T+1$	Teven N 4 30 D	Double precision extraction
$(02 \cdot PN)_{T,T+1} \rightarrow ID$ and $(02 \cdot PN)_{T,T+1} \rightarrow PN$	w T _{even} N 7 23 31	extraction
(20 IR) - D.T	T N O 29 D	From special input register
$(20\cdot ID) \longrightarrow OR$	w T N O 18 31	To special output register

Discussion

$$(20.21)_{T} \rightarrow D.T$$
 T N O 31 D

This command will copy into location D.T a word composed of "l" bits in those positions where "l" bits occur in both (20.T') and (21.T'), with "0" bits elsewhere. (Here T' is the remainder obtained when T is divided by 4.) The command is executed during word time T. Although this command will operate on the contents of line 20 and line 21 in the same way, the extractor is usually stored in line 20 because of its use in other extract operations.

Example: If (20.03) = -.zzzzw00 and (21.03) = -.vwxyz67, then after the command 51 N O 31 25 is executed, ID_1 will contain .vwxyw00 and (IP) will be 1 (indicating a negative sign).

$$(\overline{20}:21)_{\text{T}} \longrightarrow \text{D.T}$$
 T N 0 30 D

This command will extract those bits from (21.T') which correspond in position to "0" bits in (20.T') and will copy the result, with "0" bits elsewhere, into D.T. (T' is the remainder when T is divided by μ_{\bullet}) The command is executed during word time T.

Example: If (20.03) = -.zzzzw00 and (21.03) = -.vwxyz67, then after the command 55 N O 30 28, AR will contain .0000367.

$$(20 \cdot 21)_{\text{T}} + (\overline{20} \cdot AR)_{\text{T}} \longrightarrow D.T$$
 T N 0 27 D

This command will copy into location D.T a word composed of bits from (21.T') which correspond in position to "l" bits in (20.T') and bits from (AR) which correspond in position to "0" bits in (20.T'). (T' is the remainder when T is divided by 4.) The command is executed during word time T.

Example: If (20.03) = -.zzzzw00; (21.03) = .9uvwxyz; and (AR), -.0004567; then after execution of the command 75 N O 27 22, location 22.03 will contain .9uvwx67.

$$\begin{cases} (O2 \cdot PN) \longrightarrow ID \\ and (O2 \cdot PN)_T \longrightarrow PN \end{cases} \qquad w T N 3 23 31$$

This command will extract and copy into ID the bits from PN which correspond in position to "1" bits in (02.T), clearing these bit positions in PN. In positions corresponding to "0" bits in (02.T), ID will be cleared and (PN) will be unchanged. (IP) will not be affected. The prefix "w" is necessary because, with D=31, this command would otherwise be executed as a block operation. (If $T=L_1$, the command may also be written L_2 N 3 23 31 for more efficient execution.)

Example: If (02.27) = -.000zzzz; (PN₁) = -.7654321; and (ID₁), .13579vx; then after execution of the command w 27 N 3 23 31, ID₁ will contain -.0004321 and PN₁ will contain .7650000.

These double precision extract commands will operate for two words T and T+1, in exactly the same manner as the corresponding single precision commands above. T must be even.

$$\begin{cases} (02^{\circ}PN) & T, T+1 \longrightarrow ID \\ \text{and } (02^{\circ}PN)_{T_{\circ}} & T+1 \longrightarrow PN \end{cases} \quad \text{w Teven 7 23 31}$$

This double precision extract command will operate during word times T and T+1 on the contents of both halves of PN and ID in exactly the same manner as the corresponding single precision command discussed above. If T=L1, then the command may also be written L3 N 7 23 31 for more efficient execution. T must be even.

$$(20 \cdot IR) \longrightarrow D.T$$
 TNO 29 D

This extract command will operate on the contents of a special <u>input</u> register which may be attached to the G-15D as optional auxiliary equipment.

 $(20 \cdot ID) \longrightarrow OR$

w T N O 18 31

This command will extract bits from line 20 for transmission to a special output register which may be attached to the G-15D as optional auxiliary equipment.

10. SHIFT AND NORMALIZE OPERATIONS

10.1 COMMANDS FOR SHIFTING, SINGLE PRECISION

Notes	Commands	Descriptions
$(s.T) \rightarrow ID_1$	$ \begin{bmatrix} T_{\text{odd}} & N & 0 & S & 25 \\ T_{\text{even}} & N & 6 & S & 25 \end{bmatrix} $	Load number to be shifted right in ID
$(s.T) \longrightarrow MQ_1$	$ \begin{cases} T_{\text{odd}} & \text{N O S 2L} \\ T_{\text{even}} & \text{N 6 S 2L} \end{cases} $	{Load number to be shifted left in MQ
	(T _p N 1 26 31	$\begin{cases} 2^{p} (MQ_{1}) \longrightarrow MQ_{1}, \\ 2^{-p} (ID_{1}) \longrightarrow ID_{1}, \\ T_{p}=2_{p} (Location odd) \end{cases}$
Shift	T _p N 0 26 31	$\begin{cases} 2^{p} & (MQ_{1}) \longrightarrow MQ_{1}, \\ 2^{-p} & (ID_{1}) \longrightarrow ID_{1}, \\ T_{p}=2_{p} & (\text{Location odd}) \end{cases}$ $\begin{cases} 2^{p} & (MQ_{1}) \longrightarrow MQ_{1}, \\ 2^{-p} & (ID_{1}) \longrightarrow ID_{1}, \\ p \cdot 2^{-28} \longrightarrow AR+, \\ T_{p}=2_{p} & (\text{Location odd}) \end{cases}$
	54 N O 26 31	$ \begin{array}{ll} (T_p=2_p & (\text{Location odd}) \\ (2p(MQ_1) \longrightarrow MQ_1, \\ 2-p(ID_1) \longrightarrow ID_1, \\ (1p) & (AR) = -p \cdot 2^{-28} \\ (1p) & (1p) & (1p) & (1p) & (1p) & (1p) \\ (1p) & (1p) \\ (1p) & (1p) $
(ID ₁)—D.T	Todd N O 25 D	Store result of right shift into D.T
(MQ_1) — D. T	Todd N O 24 D	(Store result of left shift into D.T

Discussion

$$(\mathbf{s.7}) \rightarrow \text{ID}_{1} \qquad \begin{cases} \text{T}_{\text{odd}} & \text{N O S 25} \\ \text{T}_{\text{even}} & \text{N 6 S 25} \end{cases}$$

The absolute value of the number in S.T will be copied into the odd half of the ID register, ID1, ready to be shifted right. The sign of the number is stored in IP. These commands are the same as those previously discussed for loading ID1 with a multiplicand or divisor.

$$(S.T) \longrightarrow MQ_1 \qquad \begin{cases} T_{\text{odd}} & N \text{ O S 24} \\ T_{\text{even}} & N \text{ 6 S 24} \end{cases}$$

The absolute value of the number in S.T will be copied into the odd half of the MQ register, MQ1, ready to be shifted left or normalized. In order to store the sign properly in IP, and to

prevent the introduction of extraneous bits from the even (less significant) half of the register during the shift, a "clear" command (L_3 N O 23 31) must be executed before MQ₁ is loaded. Otherwise, this command is the same as that previously discussed for loading MQ₁ with a multiplier.

Shift $T_D N 1 26 31$

This command will simultaneously shift the contents of ID p bit positions to the right and the contents of MQ p bit positions to the left, where $T_p=2p$. That is, $2^{-p}(ID) \rightarrow ID$ and $2^p(MQ) \rightarrow MQ$. Zero bits are introduced into the left end of ID and the right end of MQ; non-zero bits shifted beyond the ends of the registers are lost.

This command has no effect on AR or PN, nor is it affected by AR or PN.

The location of the shift command must be odd. It will operate during word times L1 through L2p. $(L_{2p}=L_0+T_p)$. $T_p \le 54$.

Shift $T_p N O 26 31$

This shift command with C=0 will cause the same shifts of ID and MQ as the shift command with C=1. However, in addition, as increment of 2^{-28} is added to the contents of AR for each bit position of the shift: $p \cdot 2^{-28} \rightarrow AR^+$. To avoid early termination of the shift, the initial contents of AR must differ (in absolute value) from 1 by at least $p \cdot 2^{-28}$: a positive number must not be incremented to overflow, and a negative number must not be increased to zero.

The location of the shift command must be odd. It will operate during word times L_1 through L_{2p} . $T_p \le 54$.

Shift 54 N O 26 31

This form of the shift command will permit the initial contents of AR to determine the number of bit positions the contents of ID and MQ are to be shifted. If AR initially contains the complementary form of $-p \cdot 2^{-28}$, as the result of either a subtraction or a "clear and subtract" command, then the contents of ID will be shifted right and the contents of MQ will be shifted left p bit positions. That is, $2^{-p}(ID) \longrightarrow ID$ and $2^p(MQ) \longrightarrow MQ$, where 0 . If <math>p > 27, then the numbers will be completely lost from ID1 and MQ1; hence, $T = 54 \ge 2p$. The location of the shift command must be odd. It will operate during word times L1 through L2p.

$$(ID_1) \rightarrow D.T$$
 T_{odd} N 0 25 D

The result of a right shift will be copied from ID1, together with the sign from IP, into storage location D.T. D must not be another two-word register, and T must be odd.

$$(MQ_1) \longrightarrow D_0 T$$
 $T_{odd} N O 24 D$

The result of a left shift will be copied from MQ1, together with the sign from 1P, into storage location D.T. D must not be another two-word register, and T must be odd.

Example 7: Shift, Single Precision

Given: y = (06.22)

Find: $\frac{y}{16}$; store in 21.03

L	P	or L _k	N	С	S	D	вр	notes
20		22	25	6	06	25		y = (06.22) ID ₁
25		08	34	1	2 6	31		Shift
34		35		0	25	21		$2^{-4y} = (ID_1)$ 21.03

Remarks:

Since $16=2^{l_4}$, $p=l_4$ and $T_p=08$ in the shift command. For efficiency, $N=25+8+1=3l_4$.

Note that division of a number by an integral power of 2 can be accomplished more efficiently by shifting the number in ID than by performing the divide operation. Multiplication by an integer which is a power of two can be accomplished by shifting a number left in MQ or by repeated additions in AR as a block command.

The shift and normalize commands are extremely useful in scaling.

10.2 DOUBLE PRECISION SHIFT COMMANDS

<u>Notes</u>	Commands	Descriptions
$(S.T,T+1) \rightarrow ID$	T N 4 5 25	Load number to be shifted right in ID.
$(S.T,T+1) \rightarrow MQ$	T N 4 5 24	{Load number to shifted left in MQ
	Tp N 1 26 31	$\{2^{p}(MQ) \rightarrow MQ, 2^{-p}(ID) \rightarrow ID, \{T_{p}=2_{p} \text{ (Location } \underline{odd})\}$
Shift	Tp N O 26 31	$\begin{cases} 2^{p}(MQ) \rightarrow MQ, 2^{-p}(ID) \rightarrow ID, \\ p \cdot 2^{-28} \rightarrow AR+ \\ Tp=2p \text{ (Location odd)} \end{cases}$
	v2 N O 26 31	$(2^{p}(MQ) \rightarrow MQ, 2^{-p}(ID) \rightarrow ID,$ Initially (AR)=-p·2-28, $0 \le p \le 56$ (Location odd)
(ID)→D.T,T+l	T N 4 25 D	Store result of right shift into D.T, T+1
$(MQ) \longrightarrow D.T, T+1$	T N 4 24 D	Store result of left shift into D.T,T+1

Discussion

$$(S.T, T+1) \rightarrow ID$$
 T N 4 S 25

The absolute value of the double precision number in S.T, T+1 will be copied into ID ready to be shifted right. The sign is stored in IP. This command is the same as that for loading ID with a double precision multiplicand or divisor. T must be even.

$$(S.T, T+1) \longrightarrow MQ$$
 T N L S 2L

The absolute value of the double precision number in S.T, T+1 will be copied into MQ ready to be shifted left. This command must be preceded by the "Clear" command in order to store the sign properly in IP. Otherwise, this is the same as the command for loading MQ with a double precision multiplier. T must be even.

Shift

T_p N 1 26 31

The contents of ID will shift p bit positions to the right and the contents of MQ will shift p bit positions to the left, where $T_p=2p$. This command is identical with that for the corresponding single precision shift, except that $T_p = 2p$. The location must be odd.

Shift

Tp N O 26 31

This shift command is identical with the corresponding single precision shift, except that Tp may be greater than 54: $\text{Tp} \leq \text{v2}$. This command is also similar to the double precision shift with C=1 with the additional feature of the tally added to (AR): $\text{p} \cdot 2^{-28} \longrightarrow \text{AR}+$.

Shift

v2 N 0 26 31

This form of the shift command is used when the shifting of double precision numbers is to be controlled by the initial contents of AR. Except that 0 , the operation is the same as that for single precision numbers. The location must be odd.

 $(ID) \longrightarrow D.T, T+1$ T N 4 25 D

The double precision result of a right shift will be stored, with the sign from IP, in D.T,T+1. T must be even.

(MQ) --> D.T, T+1 T N 4 24 D

The double precision result of a left shift will be stored, with the sign from IP, in D.T,T+1. T must be even.

10.3 NORMALIZE COMMANDS

Notes	Commands	Descriptions		
$(S_T) \longrightarrow MQ_1$		Load number to be normalized		
$(S.T,T+1) \longrightarrow MQ$	T _{even} N 4 S 24	Load DP number to be normalized		
Normalize,S.P.	54 NO 27 31	Normalize (MQ_1) , tally $\rightarrow AR + (Location odd)$		
Normalize, D.P.	v2 N O 27 31	Normalize (MQ), tally → AR+ (Location odd)		
$(MQ_1) \longrightarrow D.T$	$\mathtt{T}_{\mathrm{odd}}$ N O 2 $\mathtt{l}_{\mathtt{l}}$ D	Store result of normalize in D.T		
$(MQ) \longrightarrow D.T, T+1$	T _{even} N 4 24 D	Store result of D.P. normalize		
Discussion				
$(S.T) \longrightarrow MQ_1$ $(S.T,T+1) \longrightarrow MQ$	T_{odd} N O S 24, T_{even} N 4 S 24	T _{even} N 6 S 24		

The number to be normalized is loaded into MQ in the identical manner to loading MQ for a left shift.

The contents of MQ will be shifted left until the most significant bit of MQ₁ is 1. Only leading zero bits are lost. The contents of ID do not shift. A tally of the number of bit positions of shift is made by adding increments of 2^{-28} to the contents of AR, but the normalize operation is not terminated by overflow in AR.

This operation takes two word times for each leading zero shifted out of MQ_1 . Since more than 27 shifts in a single precision word is meaningless, T is usually made equal to 54. If $(MQ_1) = 0$, the operation will cease after 54 word times. If a "1" bit is encountered sooner. the operation will cease sooner.

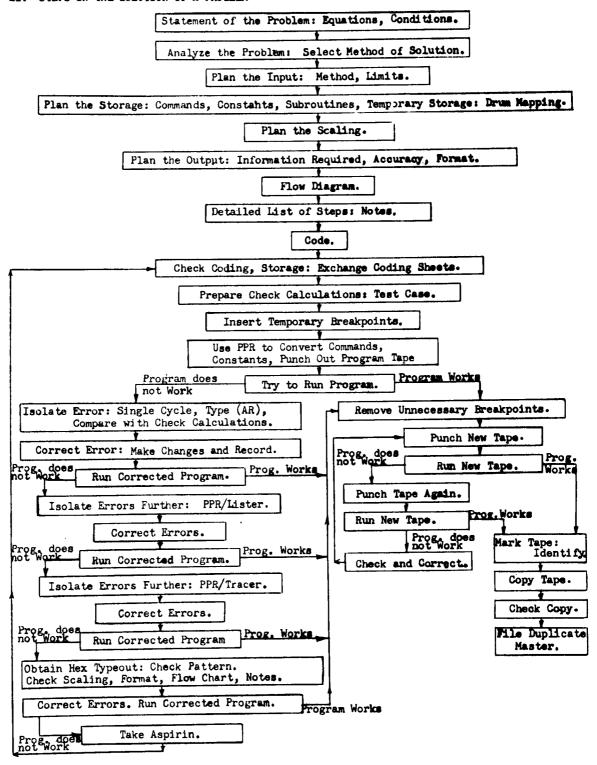
The double precision version of the normalize command, v2 N O 27 31, is the same as for single precision normalization, except that the duration of execution may extend to 112 word times (v2) and shift out up to 56 leading zeros.

The normalize operation is indispensable in floating point scaling. The location of the normalize command must be odd.

$$(MQ_1)$$
 — D.T T_{odd} N O 24 D (MQ) — D.T,T+1 T_{even} N 4 24 D

The normalized number in MQ is stored in the identical manner to storing the result of a left shift in MQ.

11. STEPS IN THE SOLUTION OF A PROBLEM



APPENDIX

Table A-1. G-15D BASIC COMMAND LIST

			4
Arithm	etic Operations	Single	Precision $(00 \le S \le 23; 00 \le D \le 23)$
$(S.T) \xrightarrow{+} AR_{C}$	TNIS	28	Clear AR and add (S.T)
$(S_T) \xrightarrow{\bullet} AR+$	TNIS	29	Add (S.T) to (AR)
$(S.T) \rightarrow AR_c$	r n 2 s	28	Clear AR and add absolute value of (S.T)
$(S.T) \longrightarrow AR+$	T N 2 S	29	Add absolute value of (S.T) to (AR)
$(S.T) \xrightarrow{-} AR_C$	T N 3 S	28	Clear AR and subtract (S.T)
$(S.T) \xrightarrow{-} AR+$	T N 3 S		Subtract (S.T) from (AR)
(AR) → D.T	T N 1 28	מ	Store sum or difference from Ak into D.T
Clear	L ₃ N 0 23	31	Clear ID, MQ, PN, sign (IP)
$(s.T) \longrightarrow \mathbb{D}_1$	$\begin{cases} T_{odd} N & 0 S \\ T_{oven} N & 6 S \end{cases}$	25)	Load multiplicand, divisor, or number to be shifted right
(-1-)	(Teven N 6 S	25 }	
$(S.T) \longrightarrow MQ_1$	Todd N O S Teven N 6 S	24 } 24 }	Load multiplier, or number to be shifted left or normalized
$(s.T) \longrightarrow MQ_1$ $(s.T) \longrightarrow PN_1$	Todd N O S	26 } 26 }	Load numerator
Multiply	56 N O 24	31	$(\mathrm{ID}_1) \times (\mathrm{MQ}_1) \longrightarrow \mathrm{PN}_{0,1} \text{ (Location } \underline{\mathrm{odd}})$
Divide	57 N 1 25	31	$(PN_1) + (ID_1) \longrightarrow MQ_0 \text{ (Location } \underline{\text{odd}})$
	$\int T_{\mathbf{p}} N = 1.26$	31	$ \begin{cases} 2^{p}(MQ_{1}) \longrightarrow MQ_{1}, & 2^{-p}(ID_{1}) \longrightarrow ID_{1}, \\ T_{p} = 2_{p} & (Location \underline{odd}) \end{cases} $
Shift	T _p N 0 26	31	$\begin{cases} 2^{p}(MQ_{1}) \longrightarrow MQ_{1}, \ 2^{-p}(D_{1}) \longrightarrow ID_{1}, \\ T_{p} = 2_{p} \text{ (Location odd)} \end{cases}$ $\begin{cases} 2^{p}(MQ_{1}) \longrightarrow MQ_{1}, \ 2^{-p}(D_{1}) \longrightarrow DD_{1}, \\ p \cdot 2^{-28} \longrightarrow AR+; \ T_{p} = 2p \text{ (Location odd)} \end{cases}$ $\begin{cases} 2^{p}(MQ_{1}) \longrightarrow MQ_{1}, \ 2^{-p}(D_{1}) \longrightarrow DD_{1}, \ 0 \le p \le 27; \\ \text{Initially, (AR)} = -p \cdot 2^{-28} \text{ (Location odd)} \end{cases}$
	54 N 0 26	31	$\begin{cases} 2^{p}(MQ_{1}) \longrightarrow MQ_{1}, \ 2^{-p}(D_{1}) \longrightarrow D_{1}, \ 0 \le p \le 27; \\ \text{Initially, (AR)} = -p \cdot 2^{-28} \text{(Location odd)} \end{cases}$
Normalize	54 N 0 27		Normalize (MQ_1) , tally $\rightarrow AR+$ (Loc. odd)
$(PN_1) \longrightarrow D.T$	T _{odd} N 0 26	ם	Store product in D.T
$(MQ_0) \longrightarrow D.T$	Teven N 0 24	מ	Store quotient in D.T
$(MQ_1) \longrightarrow D.T$	Todd N 0 24	D	Store result of left shift or normalize
$(m_1) \longrightarrow p.r$	Todd N 0 25	ם	Store result of right shift in D.T

Table A-1. G-15D BASIC COMMAND LIST (Cont.)

Arithmetic Operations: Double Precision (00≤8≤23; 00≤D≤23)

(Note: T must always be even for double precision operations)

$(S.T,T+1) \xrightarrow{+} PN$	T N 5 S 26	Clear PN and add (S.T,T+1)
$(S.T,T+1) \xrightarrow{+} PN+$	T N 5 S 30	Add (S.T,T+1) to (PN)
(S.T,T+1)> PN	т и 4 з 26	Clear PN and add abs. value of (S.T,T+1)
(S.T,T+1) → PN+	T N 6 S 30	Add absolute value of (S.T,T+1) to (PN)
(S.T,T+1) PN+	T N 7 S 30	Subtract (S.T,T+1) from (PN)
(PN) → D.T,T+1	T N 5 26 D	Store sum or diff. from PN into D.T,T+1
$(S.T,T+1) \longrightarrow ID$	т и 4 з 25	Load multiplicand, divisor, or number to be shifted right
$(S.T,T+1) \longrightarrow MQ$	тиц з 21,	Load multiplier, or number to be shifted left or normalized
$(S.T,T+1) \longrightarrow PN$	T N 4 S 26	Load numerator
Multiply	₩4 N O 24 31	$(ID) \times (MQ) \longrightarrow PN \text{ (Location } \underline{\text{odd}})$
Divide	√6 N 1 25 31	$(PN)+(ID) \longrightarrow MQ (Location odd)$
	(T _p N 1 26 31	$ \begin{cases} 2^{p}(MQ) \longrightarrow MQ, & 2^{-p}(ID) \longrightarrow ID, \\ T_{p} = 2_{p} & (Location odd) \end{cases} $
Shift	$\begin{cases} T_{\mathbf{p}} & \text{N 1 26 31} \\ T_{\mathbf{p}} & \text{N 0 26 31} \end{cases}$	$\begin{cases} 2^{p}(MQ) \longrightarrow MQ, \ 2^{-p}(ID) \longrightarrow ID, \\ p \cdot 2^{-28} \longrightarrow AR+; T_{p} = 2_{p} \text{ (Location odd)} \end{cases}$
	₩2 N O 26 31	$\begin{cases} 2^{P}(MQ) \longrightarrow MQ, \ 2^{-P}(ID) \longrightarrow ID, \ 0 \le p \le 56; \\ \text{Initially, (AR)} = -p \cdot 2^{-28} \text{ (Location odd)} \end{cases}$
Normalize	√2 N O 27 31	Normalize (MQ), tally AR+ (Loc. odd)
(PN) → PN	т и 4 26 26	Prepare product for addition in PN
(PN) → D. T ₃ T+1	T N 4 26 D	Store product in D.T,T+1
(MQ) → D.T,T+1	T N 4 24 D	Store quotient or result of left shift or normalize in D.T,T+1
(ID) D.T, T+1	T N 4 25 D	Store result of right shift in D.T,701

Table A-1. G-15D BASIC COMMAND LIST (Cont.)

Arithmetic Operations: Block Commands (00 45 423)

$\sum_{i=1}^{k} (s,L_i) \xrightarrow{+} AR+$	u L _{k+1} N 1	8	29	{Add k numbers $(S.L_k)$, to (AR)
$\sum_{i=1}^{k} (S.L_i) \longrightarrow AR+$	u L _{k+1} H 2	8	29	{Add the absolute values of k numbers, $(S.L_1) + + (S.L_k)$, to (AR)
$\sum_{i=1}^{k} (S.L_i) \longrightarrow AR4$	u L _{k+1} N 3	S	29	$\begin{cases} \text{Subtract } k & \text{numbers,} \\ \left[(S.L_1) + + (S.L_k) \right], & \text{from (AR)} \end{cases}$
$\sum_{i=1}^{k} (S.L_{2i-1},L_{2i}) \xrightarrow{*} PM+$	u L _{2k+1} N 5	s	3 0	
$\sum_{i=1}^{k} \langle s. L_{2i-1}, L_{2i} \rangle \rightarrow PN+$	u L _{2k+1} N 6	s	3 0	Add the absolute values of k double precision numbers to (PN) (Loc.odd)
$\sum_{i=1}^{k} (3.L_{2i-1},L_{2i}) \rightarrow PN+$	u L _{2k+1} N 7	S	30	Subtract the sum of k double precision numbers from (PN) (Location odd)
		С	ору Оре	rations
(S.T) → D.T	TNO			$\{00 \le S \le 23, \text{ or } S = 28; 00 \le D \le 23, \text{ or } D = 28\}$ Copy a single precision number into D.T
(S.T) +→ D.T	TNl	S	D	{00∠S∠26, or S = 28, 00∠D∠26, or D = 28 Copy a positive number into D.T; enter complement of a neg. number into D.T
$ \begin{cases} (AR) \longrightarrow D.T \\ (S.T) \longrightarrow AR \end{cases} $	T N 2	s	ם	{00 ≤ S ≤ 23; 00 ≤ D ≤ 23 Interchange storage with (AR)
$\begin{cases} (AR) \longrightarrow D, T \\ (S,T) \xrightarrow{\uparrow} AR \end{cases}$	T N 3	s	D	\[\begin{aligned} \cong 00 \leq S \leq 23; & 00 \leq D \leq 23 \\ \text{Interchange storage with (AR), entering complement of (S.T) into AR if negative. \end{aligned}
$(S.T,T+1) \longrightarrow D.T,T+1$	T _{even} N 4	S	מ	{00 ≤ S ≤ 23; 00 ≤ D ≤ 23 Copy a double precision number
(S.T,T+1) → D.T,T+1	Teven N 5	S	D	{O0 ≤ S ≤ 26; O0 ≤ D ≤ 26 Copy a pos. double prec. number; enter complement of neg. double prec. number
$(S.L_1,,L_k)\rightarrow D.L_1,,L_k$				00 ≤ S ≤ 23; 00 ≤ D ≤ 23. Block copy: Copy block of k numbers from line S into corresponding positions of line D
$(S.L_1,,L_k)\rightarrow D.L_2,,L_{k+1}$ $(AR)\rightarrow D.L_1; (S.L_{k+1})\rightarrow AR$	uL _{k+2} N 2	S	D	00 ∠S ∠23; 00 ∠D ∠23. Block copy: Copy k numbers with delay of 1 word-time
N.T 18	L ₁ N 1	. 31	31	Copy number track into line 18

Table A-1. G-15D BASIC COMMAND LIST (Cont.)

Command Sequence Control

(S.T) → test	T N	o s 27	Test for non-zero (single precision) If (S.T)=0, next command from N If (S.T)=0, next command from N+1
$ \begin{cases} (AR) \longrightarrow \text{test} \\ (S,T) \longrightarrow AR \end{cases} $	T N	2 S 27	{Test (AR) for non-zero, then copy {(S.T) into AR [S < 28]
$(S.T,T+1) \longrightarrow test$	Teven N	4 S 27	Test for non-zero (double precision)
AR sign → test	L ₂ N	0 22 31	Test for (AR) negative If (AR) not negative, next command from N If (AR) negative, next command from N+1
Ready → test	n n	0 28 31	Test for end of regular input-output If not ready, next command from N If ready, next command from N+1 Set N=LO to repeat test until ready
Overflow → test	L ₂ N	0 29 31	Test for overflow If no overflow since last test, next command from N If overflow, next command from N+1
PUNCH sw. → test	T N	1 17 31	Test for PUNCH switch ON T=L1 rings bell; T=L2 does not N=L_1 repeats test if switch ON N=L_0 repeats test if switch off
Mark → C	w T N	C 21 31	Next command from C.N; Mark word T
Return → C	10 N	C 20 31 C 20 31	Next command from C.N if
N.C. from AR	•	0 31 31	$L_2 \leq N \leq \text{marked T}$; otherwise, from C.T Next command from AR.N
Halt	L ₂ N	0 16 31	Halt
		Extract Op	erations
(20·21) _T → D.T	TN	0 31 D)	
(20.51) ^T → D.1		0 30 р	
$(20\cdot21)_{T} + (\overline{20}\cdot AR)_{T} \longrightarrow$		(Single precision extraction
$\begin{cases} (02 \cdot PN)_T \longrightarrow ID \\ \text{and } (\overline{02} \cdot PN)_T \longrightarrow PN \end{cases}$	wTN	3 23 31	
$(20 \cdot 21) \longrightarrow D.T, T+1$ $(\overline{20} \cdot 21) \longrightarrow D.T, T+1$ $((02 \cdot PN)_{T,T+1} \longrightarrow ID)$ $(and (\overline{02} \cdot PN)_{T,T+1} \longrightarrow PN)$ $(20 \cdot IR) \longrightarrow D.T$	Teven ^N T _{even} N	4 31 D 4 30 D	Double precision extraction
$\begin{cases} (O2 \cdot PN)_{T,T+1} \longrightarrow ID \\ \text{and } (\overline{O2} \cdot PN)_{T,T+1} \longrightarrow PN \end{cases}$	wT _{even} N	7 23 31	
$\begin{array}{c} (20 \cdot IR) \longrightarrow D.7 \\ (20 \cdot ID) \longrightarrow OR \end{array}$	T N T N	0 29 D 0 18 31	From special input register To special output register

Table A-1. G-15D BASIC COMMAND LIST (Cont.)

Input-Output Operations

Set ready	L ₂ N 0 00 31	Set "Ready" for regular input-output
Type in	L ₂ N O 12 31	Read typewriter input
Read p. tape	L ₂ N O 15 31	Read punched tape
Read m. tape	L ₂ N C 13 31	Read magnetic tape (C = # of magnetic tape unit)
Read cards	L ₁ NO 14 31	Read punched cards
Type (AR)	L ₂ N O O8 31	Type AR and punch with
Type (19)	L ₂ N O O9 31	Type line 19 PUNCH switch ON
Punch (19)	L ₂ N O 10 31	Punch line 19 on tape
Punch cards	L ₁ NO 11 31	Punch line 19 on cards
M. tape write	woo N C 01 31	Write on magnetic tape (C = # of magnetic tape unit)
File code	L ₅ N C 30 31	Write file code on magnetic tape
P. tape back	L ₂ N O O6 31	Reverse punched tape
M. tape search	L ₁₆ N C 05 31	Search magnetic tape, forward
M. tape reverse	L ₁₆ N C O4 31	Search magnetic tape, reverse
Ring bell	L ₁ N O 17 31	Ring bell

Commands for Optional Auxiliary Equipment

Fast punch leader	L ₅ N O O2 31	Fast punch leader on punched tape
Fast punch (19)	L ₅ N O O3 31	Fast punch line 19 on punched tape
Start DA-1	00 N 0 19 31	Start Differential Analyser
Stop DA-1	00 N 1 19 31	Stop Differential Analyzer
DA-1 → test	L ₂ N 3 28 31	Test for Differential Analyzer off
Start IR	T N 2 17 31	Start special input) (Let T=L1 to ring bell
Stop IR	T N 3 17 31	Start special input { Let T=L1 to ring bell simultaneously, and T=L2 not to ring bell
IR → test	N N 1 28 31	Test "Ready" for special input register
OR → test	N N 2 28 31	Test "Ready" for special output register

^{*} Set N=L₀ to repeat test until ready.

Table A-2. COMPONENTS OF A DECIMAL COMMAND

L Location	P Prefix	T or L _k Time of Execution	N Next Command	C Code	Source	D Destination	BP Break-Point
Word Position of command: OO to u6	None, u=block command or w=deferred command	T= Word position of address. In commands for multiply, divide, shift, normalize: T= number of word times command is operative. In other block commands, the effective time of execution is: L+1 to T-1, or L+1 to L _k -1.	Word time during which next command is to be read (except for test commands when test is "set"). 00 to u6	Single precision: C = CH Double precision: C = CH + 14 CH = Characteristic CH O Copy (No change or delay) 1 Add (Preparation for addition: positive numbers unchanged, neg. numbers complemented; no delay) 2 S and D < 28: Copy via AR (No change, one-word delay) 2 S or D ≥ 28: Absolute value (Sign = O, no delay) 3 S and D < 28: Add via AR (Neg. numbers complemented, one-word delay) 3 S or D ≥ 28: Add via AR (Neg. numbers complemented, one-word delay) 3 S or D ≥ 28: Subtract (Change sign, then like CH=1)	20 21 4-word 22 lines 23 24 MQ 25 ID 26 PN 27 (20:21+ 20:AR)	19 Long 11 lines 19 Long 11 lines 19 Long 11 lines 20 Long 120 Long 130 Long 14 MQ 25 ID 26 PN 27 Test for 0 28 ARc 29 AR+ 30 PN+ 31 Special	or - = break- point

Table A-3. COMMAND CONVERSION

Sources and Destinations in Sexadecimal Notation

	DES TINA TI ONS									
SOURCES	Line Line	Line Line 16 - 19	Line MQ 20 - 24	ID 25	PN 26	Test 27	AR 28	AR+ 29	PN+ 30	* 31
00	000 - 00z	010 - 013	014 - 018	019	Olu	Olv	Olw	0 1x	0 1y	01=
01	020 - 02z	030 - 033	034 - 038	039	03u	03 v	03w	0 3x	03 y	03 z
02	040 - 04z	050 - 053	054 - 058	059	05u	05 v	05 w	05 x	05 y	05 z
03	060 - 06z	070 - 073	074 - 078	079	07 u	07♥	07 w	07 x	0 7y	07z
OL	080 - 08z	090 - 093	094 - 098	099	09u	09 v	09w	09x	09 y	09z
05	OuO - Ouz	000 - 003	04t - 0AR	0 v 9	0vu	0 v v	0 vw	0vx	0vy	0vz
06	0w0 - 0wz	0x0 - 0x3	0x4 - 0x8	0 x 9	Oxu	oxv	Oxw	Oxxx	OxX	Oxx
07	0y0 - 0yz	0 z 0 - 0 z 3	0 z 4 - 0 z 8	0 z 9	0 2 u	0 z v	Ozw	Ozx	Ozy	022
08	100 - 102	110 - 113	114 - 118	119	llu	11v	llw	11x	11 y	11z
09	120 - 122	130 - 133	134 - 138	139	13u	13 v	13w	13x	13 y	132
10	140 - 14z	150 - 153	154 - 158	159	15u	15 v	15w	15x	15 y	15z
11	160 - 16z	170 - 173	174 - 178	179	17u	17 v	17w	17x	17 y	172
12	180 - 18z	190 - 193	194 - 198	199	19u	19 v	19w	19x	19 y	192
13	lu0 - luz	lv0 - lv3	lv4 - lv8	1 v 9	lvu	lvv	lvw	lvx	lvy	lvz
114	lw0 - lwz	1x0 - 1x3	lx4 - lx8	1x9	1xu	lxv	lxw	Ъx	lxy	lxz
15	lyO - lyz	1 z 0 - 1 z 3	1z4 - 1z8	1 z 9	lzu	lzv	lzw	lzx	lzy	lzz
16	200 - 20z	210 - 213	214 - 218	219	21u	21 v	21w	2lx	21 y	21z
17	220 - 222	230 - 233	234 - 238	239	23u	23v	23w	23 x	23y	232
18	240 - 24z	250 - 253	254 - 258	259	25u	25₹	25w	25x	25y	25z
19	260 - 26z	270 - 273	274 - 278	279	27u	27▼	27₩	27x	27у	272
20	280 - 282	290 - 293	294 - 298	299	29u	29▼	29w	29x	29у	292
21	2u0 - 2uz	2v0 - 2v3	2 vl 2 v8	2 v9	2vu	2v▼	2vw	2 vx	2 vy	2vz
22	2w0 - 2wz	2x0 - 2x3		2x9	2xu	2xv	2xw	2)(0)	2xy	2xz
23	2y0 - 2yz	2z0 - 2z3		229	2zu	2 z v	2 zw	22x	2 zy	2 zz
MQ 24	300 - 30z	310 - 313	314 - 318	319	31u	31 v	31w	31 x	31 y	31z
ID 25	320 - 32z	330 - 333		339	33u	33 v	33w	33 x	33 y	73z
PN 26	340 - 34z	350 - 353		359	35u	35▼	35w	35 x	35 y	35z
20-21+20-AR 27	360 - 36z	370 - 373		379	37u	37▼	37w	37x	37 y	37z
AR 28	380 - 38z	390 - 393	394 - 398	39 9	39u	39▼	39w	39 x	39у	392
20•IR 29	3u0 - 3uz			3 v 9		3▼▽	3vw	3vx	3vy	3vz
20.21 30	3w0 - 3wz			3x9	3xu	3xv	3xw	3xx	3 xy	3xz
20.21 31	3y0 - 3yz			329	3zu	32 v	2zw	3 z x	3 zy	322
	1,0-									ــــــــــــــــــــــــــــــــــــــ

^{*} Special Commands: See Table A-4 for operation and for values with various characteristics.

Note: The characteristic in the above commands is 0 (except 73z, Divide).

If CH = 1, add 4 to the first digit; if CH = 2, add 8, if CH = 3, add w.

Examples: CH = 1, S = 28, D = 01, last three sexad. digits = 781 CH = 3, S = 21, D = 29, last three sexad. digits = yvx

Table A-4. G-15D SPECIAL COMMANDS: D=31

CODE	S CODE	HEX CODE	OPERA TION
	00	Olz	Set "Ready"
0,1,2,3	01 02 03	0 3z* 05 z 07 z	Write on Magnetic Tape Fast Punch Leader Fast Punch (19) Fast Output Operations
0,1,2,3 0,1,2,3	04 05 06	09z* 0vz* 0xz	Search Magnetic Tape, Meverse Search Magnetic Tape, Forward Reverse Punched Tape Cape Search Operations Cape Search
	08 09 10 11	11z 13z 15z 17z	Type (Art)
0,1,2,3	12 13 14 15	19z 1vz* 1xz 1zz	Read Typewriter Input Read Magnetic Tape Read Funched Cards Read Punched Tape The purchase of the control of t
0 1 2 3 0	16 17 " " 18 19	21z 23z 63z u3z y3z 25z 27z 67z	Halt Ring Bell Ring Bell and "PUNCH ON" to test Ring Bell and Start IR Ring Bell and Stop IR (20·ID) to Output Register Start DA-1 Stop DA-1
0 to 7 0 to 7 0 3 (or 7)	20 21 22 23	29z* 2 vz* 2xz 2zz yzz	Select Command Line - & Meturn Select Command Line - & Mark Sign of An to Test Clear Ma, ID, FN, and Sign (PN.02) to ID, (PN.02) to FN
1 (or 5) 1 0	24 25 26 "	31z 73z 75z 35z 37z	Multiply Divide Snift MQ Left and ID Right Shift MQ Left and ID Right Increment Normalize MQ AR Register T Relative
0 1 2 3 0,1,2,3	28 " " 29 30 31	39z 79z v9z z9z 3vz 3xz* 3zz 7zz	"Ready" "Ready IR" "Ready OR" "Ready OR" IM-1 Off Overflow Write File Code on Magnetic Pape Next Command from AR Copy Number Frack into Line 18

^{*} Hex Code applies only for C=0.
Where not specified, C is immaterial, but usually written 0.

Table A-5. FORMAT CHARACTERS

	Character	Code
D	Digit	000
E	End	001
С	Carriage Return	010
P	Period	011
s	Sign	100
R	Reload	101
т	Tab	110
W	Wait	נננ

Standard AR Format

SPDDDDDDCE

(03.03) = -8 w 00000(03.02) = 1000000

Standard Line 19 Format

S D D D D D D D T S D D D D D D T - S D D D D D D C E

(02.03) = 800000x (02.02) = 0000034 (02.01) = 00000x0 (02.00) = 0000110

Table A-6. PUNCHED TAPE CHARACTERS

Character	Code
Ο,	10000
1	10001
2	10010
3	10011
14	10100
5	10101
6	10110
7	10111
8	11000
9	11001
u	11010
v	11011
w	11100
x	11101
У	11110
Z	11111
Space	00000
Minus	00001
CR	00010
Tab	00011
Reload /	00101
Period	00110
S (end)	00100
Wait	00111

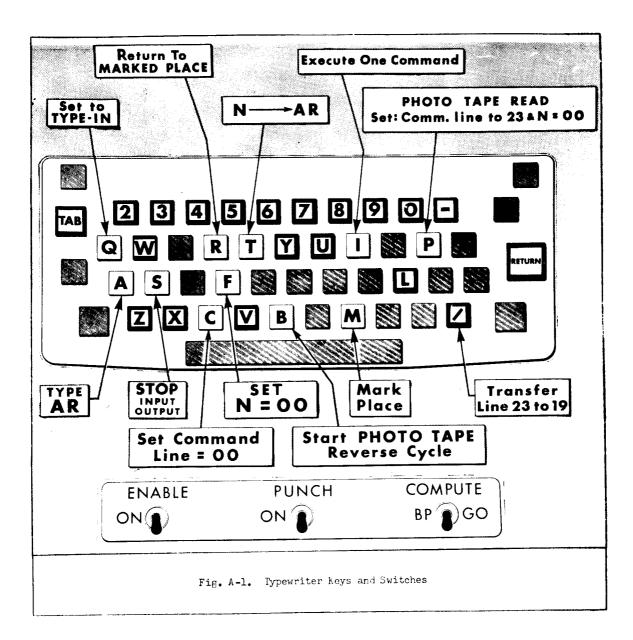


Table A-7. TYPEWRITER CONTROL KEYS AND SWITCHES

G-15D Typewriter Control Keys

Key	with ENABLE	Action	
A	ON	Type out (AR)	
В	ON	Reverse punched tape one block	
00 01 02 03 04 05	ON ON ON ON ON	Set command line to 00 Set command line to 01 Set command line to 02 Set command line to 03 Set command line to 04 Set command line to 05	
C6 C7	ON ON	Set command line to 19 Set command line to 23	
F	ON	Set N = 00	
I	ON	Execute one command (Single cycle)	
м	ON	Mark place $[N \rightarrow 00.u7; (AR) \rightarrow 01.u7]$	
P	ON	1) Read punched tape 2) Set N = 00 3) Set command line to 23	
Q	ON	Read typewriter input	
R	ON	Return to marked place	
S	ON Off	Set "Ready" (Stop input or output) Set "Ready" after reading typewriter input	
т	ON	Send $N \cdot 2^{-8}$ (in binary) to AR	

G-15D Typewriter Data Input Keys

Кеу	Description
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, u, v, w, x, y, z	Enter binary equivalent (4 bits)
Tab C.R.	Stores minus sign for later insertion Insert sign (1 bit) Insert sign (1 bit) Reload

G-15D Typewriter Control Switches

Switch	Position Left Neutral Right	Action
ENABLE PUNCH COMPUTE	ON (off) BP (off) GO	Activate control keys; Interrupt typeout Punch during typeout (no zero suppression) Go to breakpoint; Stop; Go to halt

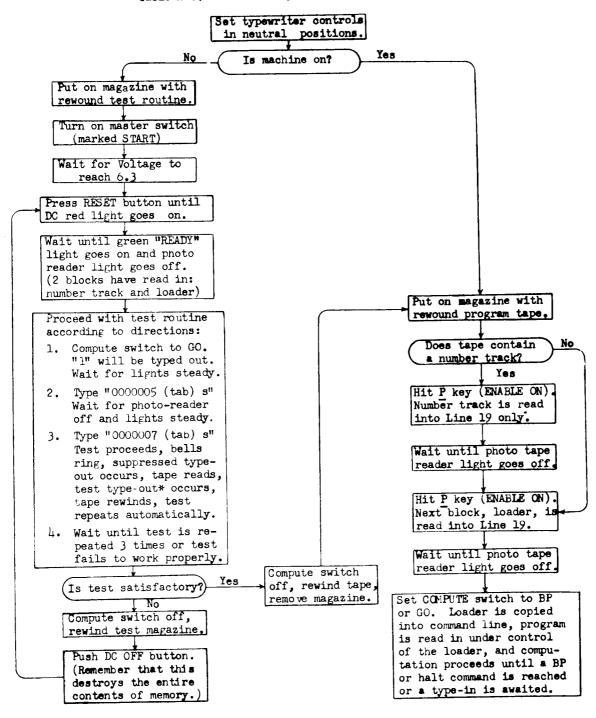


Table A-8. GENERAL G-15D OPERATING PROCEDURES

* Correct test typeout is: -1122334 445566.7 778899
-uuvvwwx xxyyzz.0 2345

Table A-9. G-15D OPERATING PROCEDURES WITH PROGRAM PREPARATION ROUTINE (PPR)

(Note: Assume machine is on and number track is in.)

Put on magazine with rewound PPR tape. Does tape contain number track? No Yes Hit P key (with ENABLE on) to read past number track. Wait until photo tape reader light goes off. Hit P key (with ENABLE on) to read loader into L.19. Wait until photo tape reader light goes off.

Set COMPUTE switch to GO. Four basic PPR blocks are read in and stored in lines 17, 16, 15 and 05.

Wait for configuration of indicator lights: 11100 Source Command Line 101 11111 Destination 01100 Input-Output

Computer is now ready to accept basic FPH instructions. These prepare for insertion of commands, storage of hex numbers, reading punched tape and various forms of output.

> Is decimal number conversion or any auxiliary service routine desired ?

> > Yes

COMPUTE switch off and hit P key (with ENABLE on), to read loader for auxiliary control into line 19.

Wait until photo tape reader light goes off.

Set COMPUTE switch to GO. Auxiliary control and decimal to binary conversion go into lines 14 and 13.

Wait for configuration of indicator lights:

101 11111 01100

Computer is now also ready to accept the PPR instructions to convert and store decimal numbers and to call in PPk's auxiliary service routines.

> Computer is in Test Ready state, awaiting type-in of a PPk instruction.

Table A-10. TYPICAL STEPS IN USING PROGRAM PREPARATION ROUTINE (Note: Assume PPR tape has been read in and type-in is awaited.)

```
Type: x00 (tab) s (to clear line 18, the line used for working storage with
                      this service routine)
       yCD- (tab) s (to prepare for entry of a sequence of decimal commands
                      starting with location CD. These are stored in line 18,
                      to be copied later into line AB.)
Enter commands, each followed by: (tab) s (The next location is typed out,
                      plus its contents if not empty.)
Repeat yCD- for each new sequence or for corrections.
Type: zCD (tab) *ddddddd (tab) s (to enter each hex number, sign and 7 digits,
                      into word CD of L.18. The location is typed, plus previous
                      contents if any. Each number must be entered individually.
Repeat for each hex number or for correction.
Type: EvCD (tab) s (to prepare for entry of a decimal number into location CD
                      of line 18. E = instruction code as follows:
           E = 0 for S.P. fractions; 8 digits less preceding zeros.
           E = 1 for s.F. integers; up to 8 digits; becomes scaled to 2-28.
            E = 4 for D.P. fractions; 14 digits less preceding zeros.
           E = 5 for D.P. integers; up to 14 digits; becomes scaled to 2^{-57}.
            If no number is typed in E position, it is interpreted as E = 0.
            If double precision numbers are entered, CD must be even, and
            converted number is stored in locations CD and CD+1.
                     The location is typed out, plus previous contents if any.
                     If decimal to binary conversion is not in line 13, it is
                     called in automatically from the tape, and ".00" is typed.)
 Enter the decimal number, followed by: (tab) s (For incorrect number of digits
                      or other error, bell rings and the location is retyped.)
Repeat for each decimal number.
Type: x06 (tab) s (to punch prepared line on tape.
                                                      line 18 \rightarrow \text{line } 19 \rightarrow \text{out.}
                    L.18 is still intact. Check sum is obtained and typed first:
                    note for later reference, with line into which it will go.)
            (Note: To return to PPR without completing punch-out, hit sf.)
                        (to copy line 18 into line AE if desired for later use)
Type: ABx03 (tab) s
                        IS ADDITIONAL LINE TO BE PREPARED?
Type: BCDxOl (tab) s (to obey next command from line b word CD, to run program with
                     PPR still in memory. Computer will proceed to obey commands.
             If COMPUTE switch is set to BP instead of GO before this instruction.
             it will halt before entering program; paper may now be set; COMPUTE
             switch is then set to BP or GO depending upon problem requirements.)
```

TO RETURN TO PPR (for corrections, debugging, or next set of lines to be prepared): COMPUTE switch off; hit keys <a href="mailto:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:scottanger:sco

Table A-11. INSTRUCTIONS FOR PROGRAM PREPARATION ROUTINE

(Note: All instructions are followed by tab s.)

BASIC PPR INSTRUCTIONS

#00 #01	Read tape into line 18, type check sum, (BP), repeat for next block Read tape into line 18, type check sum, (BP), type out (line 18), repeat Read tape into line 18, type check sum, (BP), punch tape (condensed format), repeat
w03	Read tape into line 18, type check sum, (BP), type and punch (PUNCH switch ON after check sum), repeat
x 00	Clear line 18
x01	Number track into line 18, type check sum of number track, (BP), punch number track
ABCDx02 ABCDx02	(Line Ab) to line 18, type word CD as a sexadecimal number (Line Ab) to line 18, type word CD as a decimal command
ABx03	(Line 18) to line Ab
встхоц	Next command from line B (B = 0, 1, 2, 3 or 4), word CD
x 05	Type check sum of line 18, (BP), type out (line 18)
x 06	Type check sum of line 18, (BP), type and punch (line 18) (PUNCH switch must be ON after check sum typed)
ACD	Decimal command input, into word CD of line 18. Type any previous contents of CD as a sexadecimal number.
ACD-	Decimal command input, into word CD of line 18. Type any previous contents of CD as a decimal command.
zCD (tab) + d1d2d3d4d5d6d7 Sexadecimal number input, into word CD of line 18.

Note: To return to PPR after a "w" instruction or an incompleted x01, x05, x06 or x07, Compute switch off, type "sf" with enable on, compute switch to GO or BP.

DECIMAL TO BINARY NUMBER CONVERSION

EvCD	Decimal number input, into word CD of line 18:
	E = 0 for S.P. fractions; 8 digits less preceding zeros.
	E = 1 for 5.P. integers; up to 8 digits; becomes scaled to 2-28.
	E = h for D.P. fractions: 1/4 digits less preceding zeros.
	$E = 5$ for D.P. integers; up to μ digits; becomes scaled to 2^{-57} .

INSTRUCTIONS FOR AUXILIARY SERVICE ROUTINES

07x08	Lister	List a program without execution.
06x08	Repositioner	Change S and D numbers to a new line.
0kx08	Auto-Loader	Prepare and punch out a loading routine.
03x08	Forma ter	Compile an output format
02x08	Tracer	Execute and list each command and its result.
01x08	Break-Pointer	Insert or remove a breakpoint.