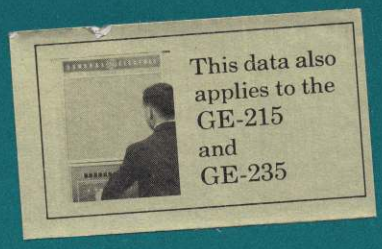
CPB-252



GE-225
PROGRAMMING REFERENCE MANUAL

GENERAL ELECTRIC

COMPUTER DEPARTMENT

GE-225
PROGRAMMING REFERENCE MANUAL

Revised October 1963

GENERAL ELECTRIC

COMPUTER DEPARTMENT

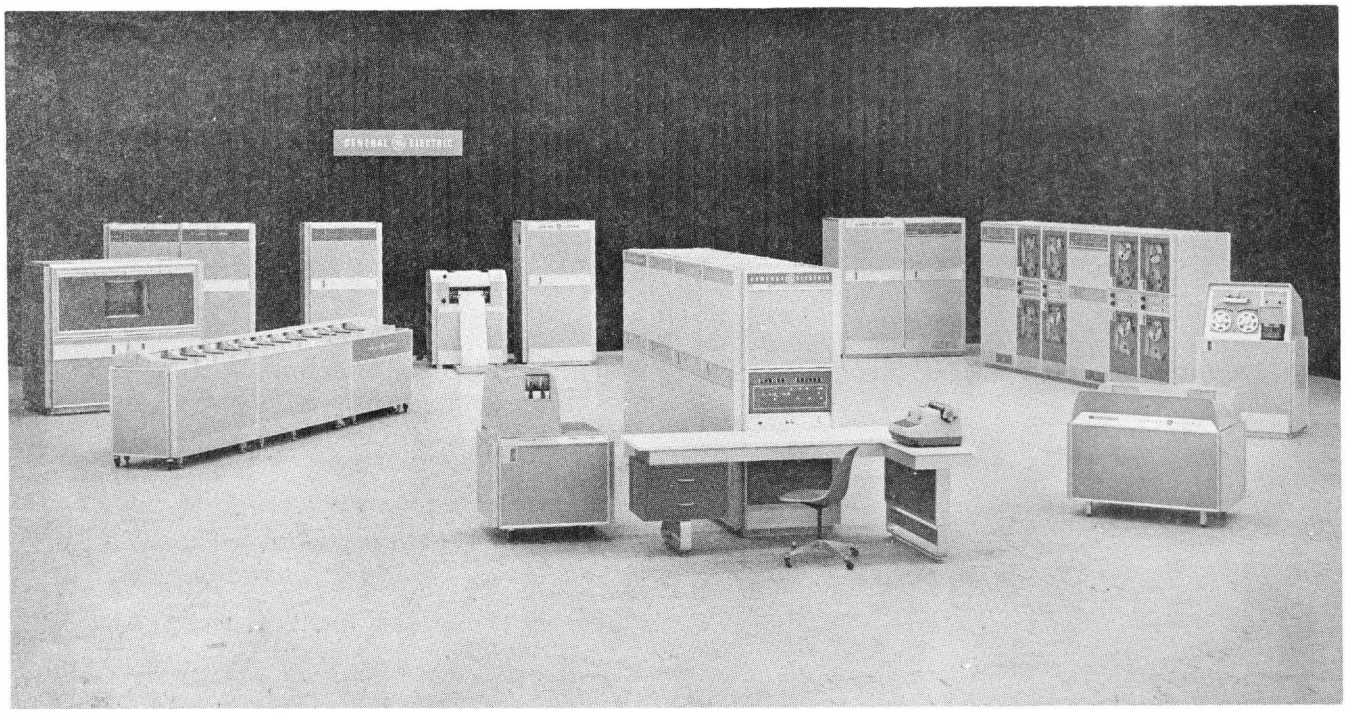
In the construction of the equipment described, General Electric reserves the right to modify the design for reasons of improved performance and operational flexibility.

PREFACE

This manual provides all information essential to basic programming for the GE-225 Information Processing System. Its primary purpose is to serve as a reference and a guide to persons actively engaged in programming or applying the GE-225 system. However, the developmental method used in presenting the subject matter makes the manual useful as an aid in training programmers and systems personnel.

CONTENTS

ILLUSTRATIONS


GENERAL ELECTRIC

SECTION I

THE GE - 225 INFORMATION PROCESSING SYSTEM

The GE-225 Information Processing System is a medium-scale, general-purpose digital computer that permits an integrated approach to the total information processing needs of business, government, and science, while providing an economical means of processing large volumes of data at high speed.

The modular design of the GE-225 system provides flexibility in meeting data processing requirements for a wide range of applications. A GE-225 system consists of reading (input) and writing (output) devices interconnected and controlled through a central processor. The number and types of input and output devices, as well as the configuration of the central processor, are determined largely by the desired applications. Input data can be from paper tape, magnetic tape, punched cards, and magnetically-encoded (MICR) paper documents. Output can be in the form of paper tape, magnetic tape, punched cards, and printed reports. Both alphabetic and numeric data can be received or produced by the computer, either locally, or over long distances from the central processor using peripheral data transmission equipment, such as the Datanet-15 and its associated terminals.

The GE-225 is a solid-state, single-address computer that operates under both stored program and operator control. Also, it is a buffered computer with an input-output priority system that permits simultaneous operations, such as reading, writing, and processing. Further flexibility is provided through the ability to operate internally in either the binary or the decimal modes.

The basic programming language for the GE-225 is provided by the General Assembly Program. It is an automatic assembly system that permits the programmer to prepare routines in meaningful symbolic language, rather than in the absolute machine language, or code, of the GE-225 and then utilize the GE-225 (and the assembly program) to assemble a computer-ready program. Extensive clerical effort is eliminated by using significant mnemonic codes that generally have a one-to-one correlation to basic machine instructions. Added flexibility is provided because addresses can be assigned using either decimal or symbolic notation. Capabilities of the General Assembly Program also include the ability to incorporate the many library routines provided by General Electric, such as input-output and mathematical packages.

SYSTEM COMPONENTS

The GE-225 system can assume various configurations, depending upon the application requirements. Brief descriptions of system components are given below. More detailed descriptions and information pertaining to their use are provided in appropriate programming sections of the manual.

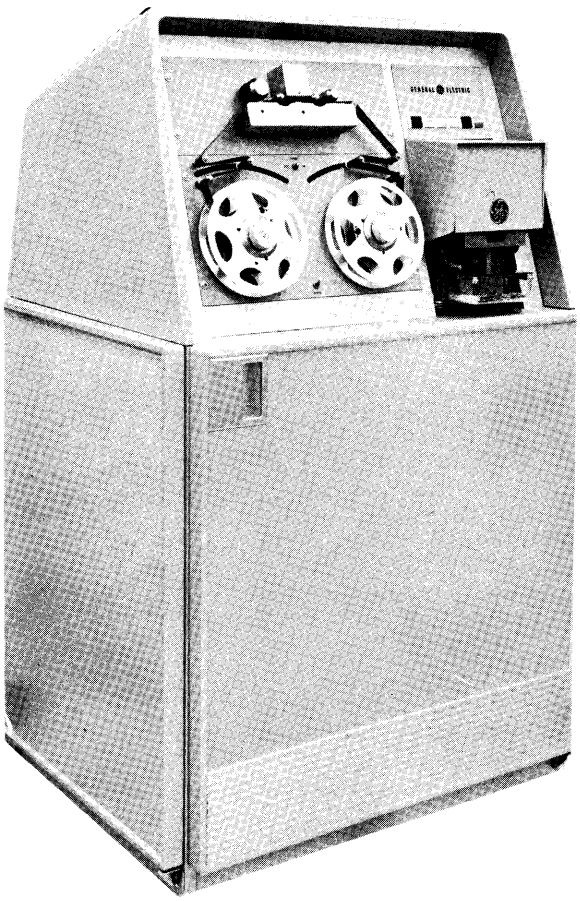
Central Processor

The GE-225 Central Processor provides arithmetic, comparison, and decision circuits and automatic control facilities for the processing system. In addition, it houses the randomly-accessed magnetic core storage (or memory).

Core storage provides the main memory element for the system, although it can be augmented by external storage in the form of magnetic tape or disks. Both data to be processed and the controlling instructions are held in core storage and called forth by the control element as required. Information in storage is retained by tiny magnetic cores, each core capable of holding one bit (binary digit) of data. The basic unit of storage is the word, each word consisting of 20 bits (plus a check bit), and each word being individually addressable. The access time associated with transferring a word into or out of memory is 18 microseconds, or one word time. Core storage can consist of 4,096, 8,192, or 16,384 locations, each of which can contain a single-address instruction, a binary data word, or three alphanumeric or binary-coded-decimal (BCD) characters.

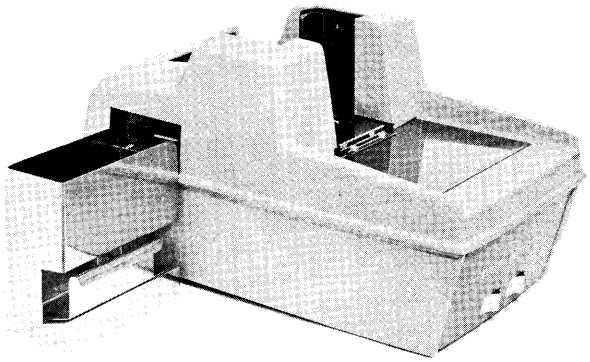
Control Console

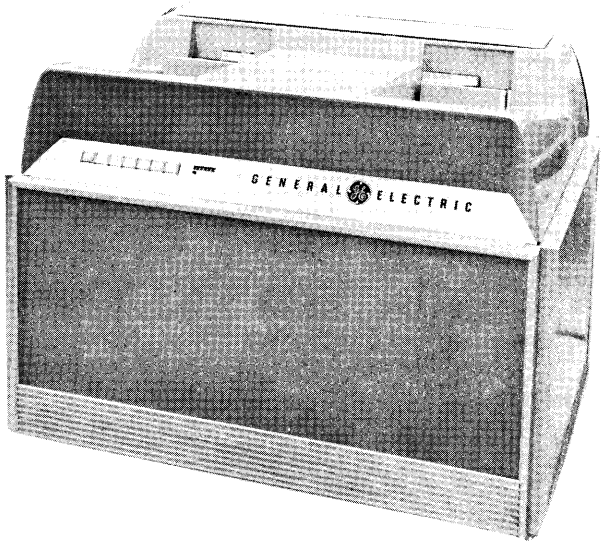
The GE-225 Control Console, attached to the central processor, provides manual control of operations, visual display of the contents of appropriate registers, program monitoring facilities for the operator, and typed output via the console typewriter, under program control. From the console, the operator controls the initial loading and starting of programs and can perform in-process modifications based upon processing results.



Paper Tape Reader-Punch

The GE-225 Paper Tape Reader-Punch is two mechanically-independent units: a mechanism for reading five-, six-, seven-, and eight-channel perforated paper tapes at 250 or 1000 characters per second, and a mechanism for punching five-, six-, seven-, and eight-channel paper tapes at 110 characters per second. Provisions are made to accommodate all common paper tape codes.



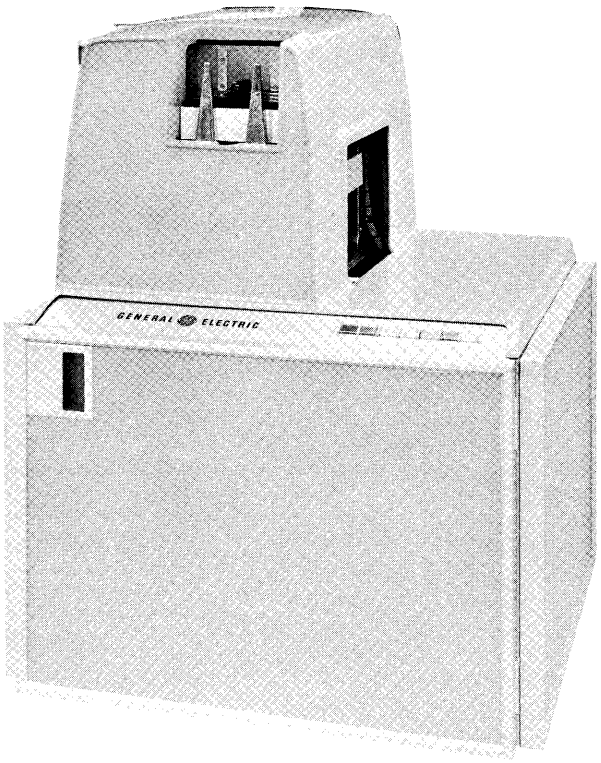


Card Reader

Either a 400 card per minute or a 1000 card per minute card reader is available with the GE-225. Both readers can read standard 80-column punched cards in one of three modes: ten-row or twelve-row binary, or standard Hollerith (alphanumeric) mode. Cards are read serially (one column at a time) in all three modes.

Either card reader can operate simultaneously with the central processor and other peripheral operations. For example, cards can be read at the same time that data is input from magnetic tape or from a 12-pocket document handler; simultaneously, previously input data can be processed within the central processor.

Standard cards are 7-3/8 by 3-1/4 inches and consist of 80 columns along the long dimension and 12 rows along the short dimension. As cards are moved through the card reader mechanism, all twelve row positions of a column are simultaneously photoelectrically sensed. Card reader logic, which is contained within the central processor, permits cards to be read on demand by the processor or continuously.



Card Punch

The card punch is an output device which punches standard 80-column cards at a rate of either 100 or 250 cards per minute, depending upon the model selected. Cards are punched in either of three modes: ten-row or twelve-row binary, or standard Hollerith mode, depending upon program control.

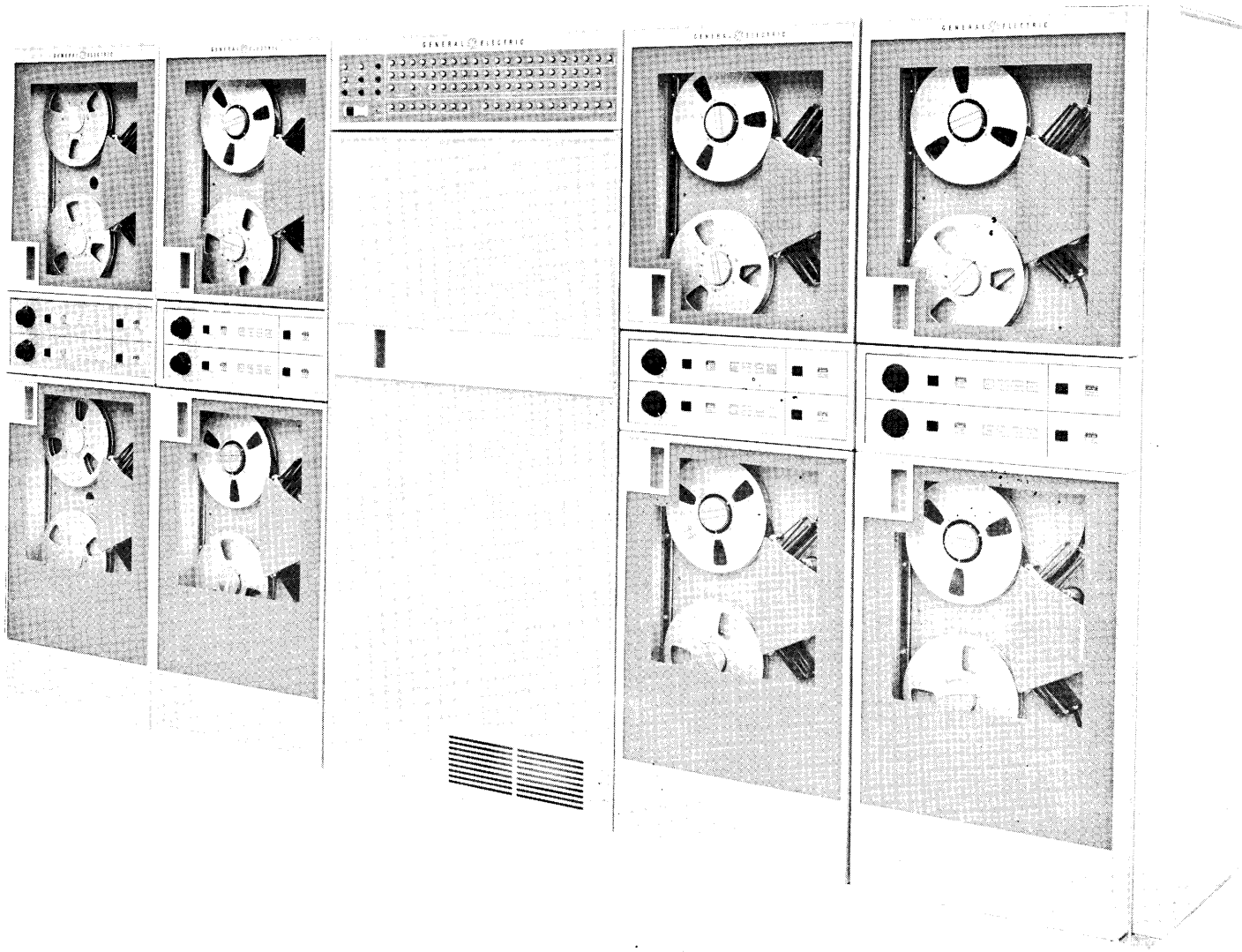
The card punch is primarily an on-line peripheral and receives basic control signals from the central processor. However, gang punching, or duplication of many cards from a master card, can be performed off-line.

As an on-line peripheral, the card punch can operate simultaneously with the central processor and other peripherals.

Controller Selector

The GE-225 Controller Selector serves as a common control and data transfer point between the central processor and the peripheral controllers for magnetic tape handlers, document handlers, high-speed printers, mass random access data storage, Datanet-15 terminals, and the auxiliary arithmetic unit. The controller selector contains eight hubs or addresses to which eight controllers can be connected. By priority assignments, which are determined by the addresses, the controller selector controls access to core storage for the attached peripheral units. This permits simultaneous operation of as many as eight peripherals on the controller selector, plus the card reader and punch, for a total of 10 concurrent input/output operations.

The logic for the controller selector is contained within the central processor. Access to the central processor and memory for peripherals and their associated controllers is provided by cables between the controller selector and the controllers.



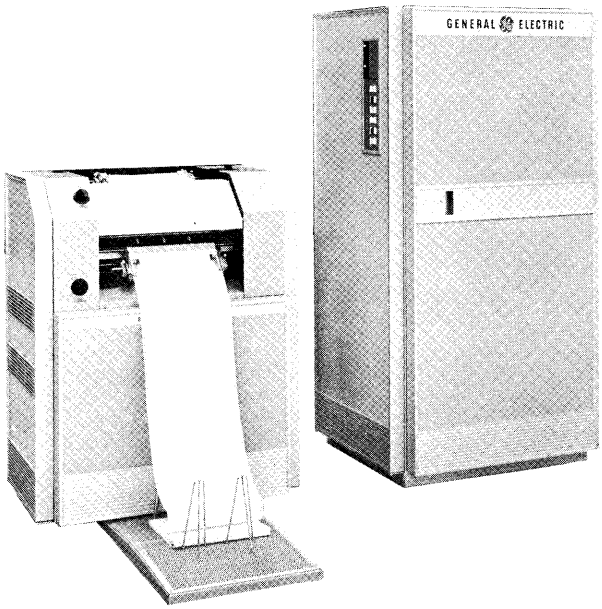
Magnetic Tape

Magnetic tape provides a fast method of transmission of data between the central processor and bulk storage. Millions of bits of data can be recorded on a single reel of tape, thus providing a compact and economical storage medium. Magnetic tape can provide in-process (on-line) or static (off-line) storage for immediate or subsequent use, yet can be erased and be re-used repeatedly.

Up to eight magnetic tape controllers can be connected to the controller selector; up to eight magnetic tape handlers can be connected to each controller, providing a maximum of 64 magnetic tape handlers for the GE-225 system. Different models of magnetic tape handlers provide two data transfer rates: 15,000 and 41,700 characters per second. Data can be read or written either in standard binary or in binary-coded-decimal (BCD) mode.

The combination of a tape controller and its associated tape handlers comprises a magnetic tape subsystem. A subsystem of one tape controller and multiple tape handlers permits reading or writing concurrently with other operations. A subsystem containing two or more tape controllers permits reading and writing simultaneously with other operations.

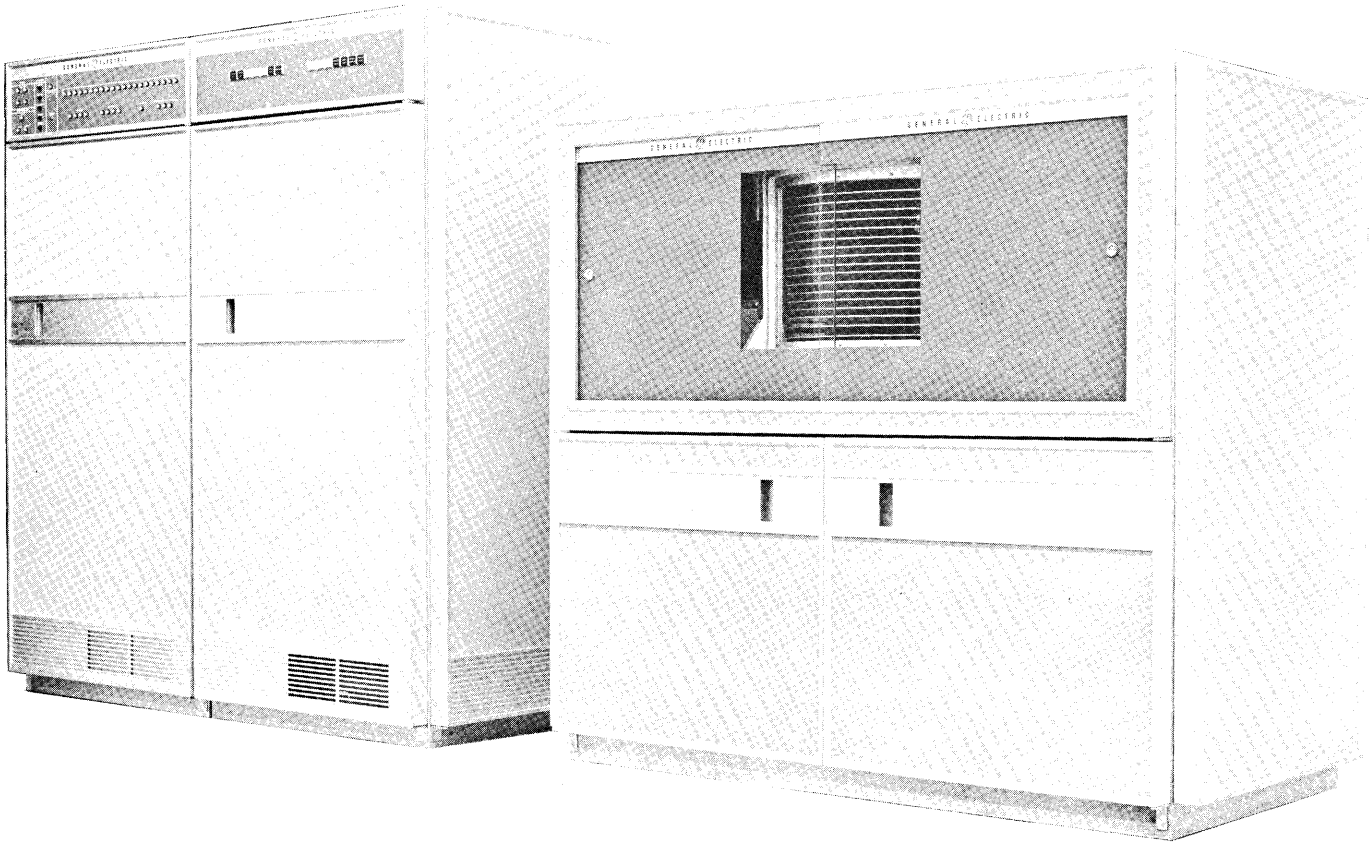
High Speed Printer

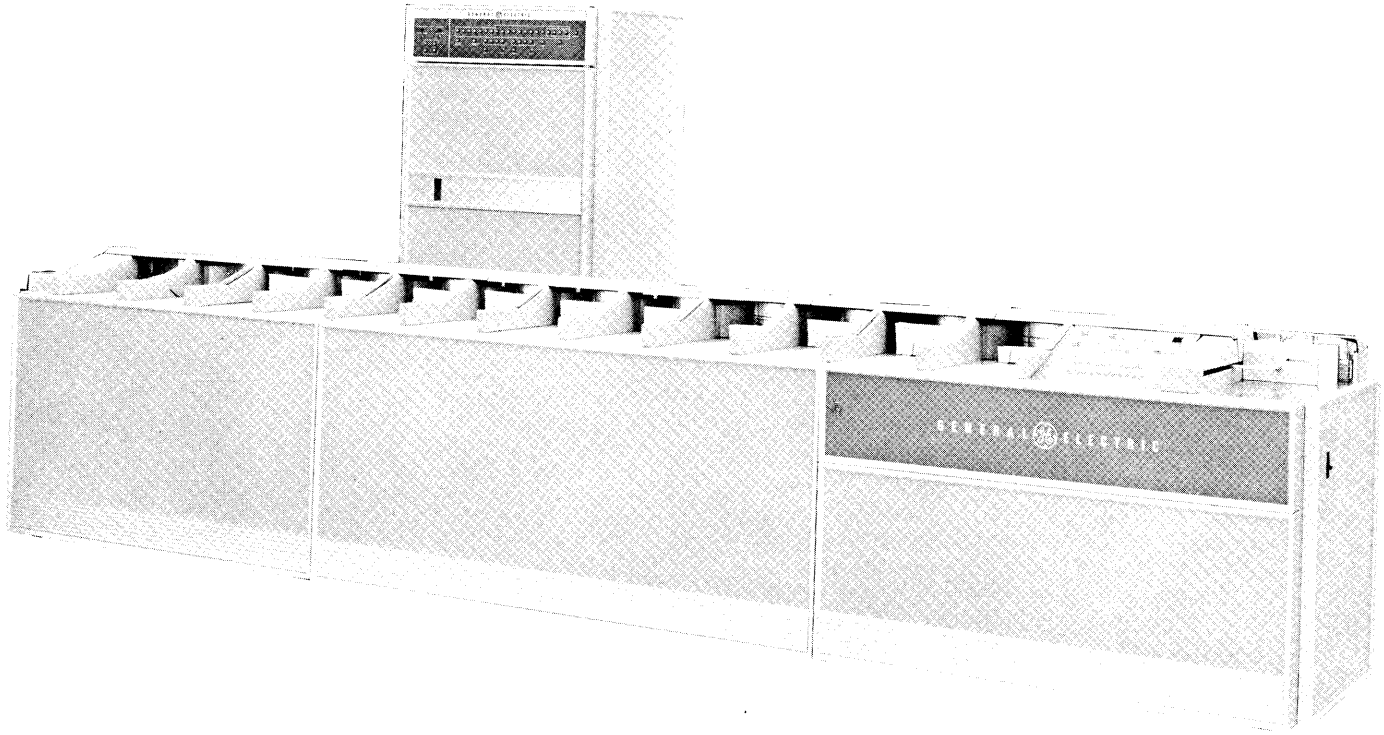


The GE-225 High Speed Printer is an output unit for applications requiring presentation of large quantities of printed information. The printer produces alphanumeric output, up to 120 characters per line, 900 lines per minute. Printing format is governed by the printer controller, which contains logic for automatically editing the print line independent of the central processor. Editing features include zero suppression, deletion of data, and insertion of special symbols, constants, and spaces. Printing can also be performed completely off-line from the system by using magnetic tape as an interim storage medium. Printing and editing can proceed simultaneously with other peripheral and central processor operations.

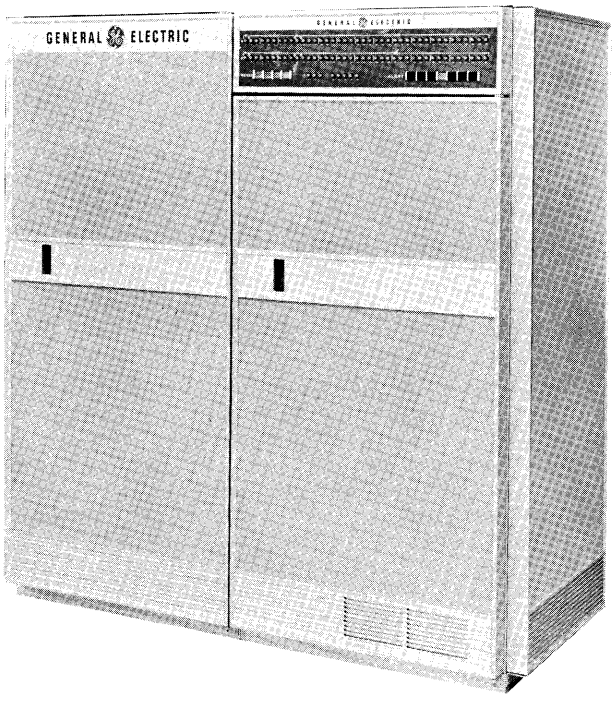
Disc Storage Unit

Disc Storage Units, each consisting of 16 vertically-mounted rotating magnetic disks, are available for non-sequential file processing. Each DSU has a total capacity of 98,304 records, or over 6 million words. This provides storage for about 19 million alphanumeric characters or 34 million numeric digits.





One or two DSU controllers can be connected to the controller selector; up to four DSU units can be connected to each controller. DSU reading and writing operations can proceed simultaneously with other peripheral and central processor operations.

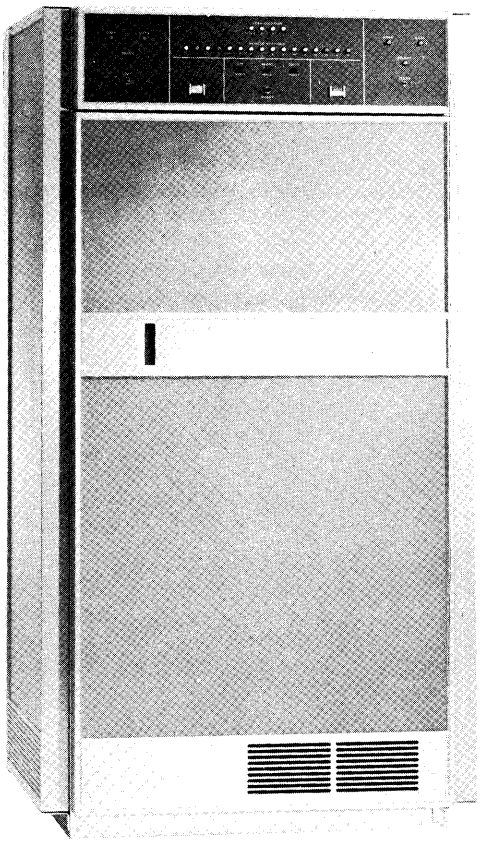


12-Pocket Document Handler

The 12-pocket document handler is an on-line or off-line peripheral that reads and sorts documents printed with magnetic ink in E13B font at a speed of 1200 documents per minute. The document handler can be used off-line as a document sorter, and it is possible to use two sorters simultaneously. The document handler adapter (controller) permits concurrent operation with other peripherals and the central processor. Two document handlers under the control of a single adapter permit an input rate to the central processor of 2400 documents per minute.

Auxiliary Arithmetic Unit (AAU)

Although the AAU is connected to the central processor through the controller selector (address 7), it is more properly considered to be an extension of the central processor, rather than a peripheral unit. The AAU provides increased facilitiy for double-length word binary arithmetic in either normalized or unnormalized floating-point modes or in fixed-point mode. The AAU can operate concurrently with normal central processor and peripheral operations.



Name	Maximun Per System
CENTRAL PROCESSOR (mandatory)	1
CONTROL CONSOLE, including Console Typewriter (mandatory)	1
DIRECT INPUT-OUTPUT UNITS	
Paper Tape Reader-Punch	1
Card Reader, 400 cpm or High Speed	1
Card Punch, 100 or 250 cpm	1
PERIPHERAL CONTROLLERS	
Controller Selector	1
Mass Random Access Data Storage Controller	1
Magnetic Tape Controller	8
High-Speed Printer Controller	8
Datanet-15	8
Document Handler Adapter	8
Auxiliary Arithmetic Unit	1
CONTROLLER SELECTOR PERIPHERALS	
Mass Random Access Data Storage Units	8
Magnetic Tape Handlers	64
High-Speed Printers	8
Datanet Terminals	120
12-Pocket Document Handlers	16

Figure 1-1. GE-225 System Components

Datanet-15

Transmission and reception of data between the GE-225 Central Processor and remote locations is made possible by the Datanet-15, which can accept serial data at speeds from 60 to 2400 bits per second. The Datanet-15 can operate with as many as 15 remote stations, one at a time, in addition to controlling a paper tape reader-punch. Terminal devices include Teletype equipment, other Datanet-15 units, or virtually any terminal device utilizing five-, six-, seven-, or eight-channel bit codes.

SIMULTANEOUS OPERATIONS

The logical design of the GE-225 permits up to eleven simultaneous input-output operations. That is, data can be transferred between core storage in the central processor and several direct and indirect peripherals at the same time that the central processor is engaged in processing data previously read in. Such operations are made feasible because of the vast differences in data transfer rates between core storage (18 microseconds per word), and peripherals, such as the 400 cpm card reader (5610 microseconds per BCD word).

To make optimum use of the high speed of core storage, the GE-225 makes provision for time sharing access to memory by buffering data transfers, assigning peripheral priorities for access to memory, and permitting simultaneous processing of two or more unrelated programs.

Buffers and Buffering

Buffering is a technique for providing optimum data transfer between two components having different data transfer rates such as core storage and the 400 cpm card reader mentioned above. Buffering involves using a temporary storage device, or buffer, that can be filled with data at a rate governed by the data source component, and subsequently unleaded into the data receiving component at a rate governed by that component. This permits both components to function at their optimum speeds when processing unrelated data without the faster component being slowed down during data transfers by the slower one.

Thus, in transfers between core storage and the 400 cpm card reader, although it takes 150,000 microseconds to read all 80 card columns, core storage

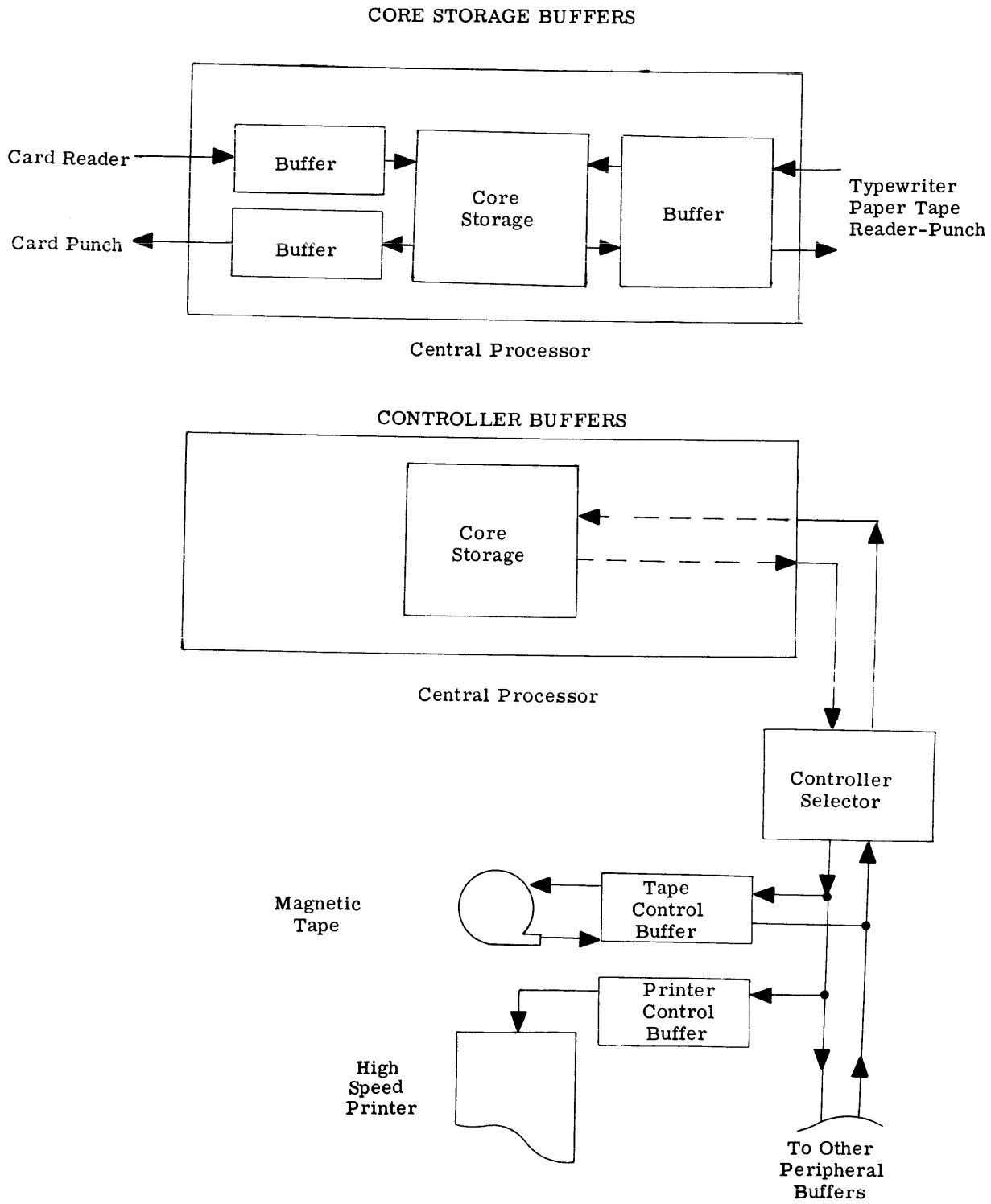


Figure 1-2. Central Processor and Controller Buffers

is occupied in receiving the data read for only 1512 microseconds (one word time per column). The balance of the time it takes to read the card (148,560 microseconds) can be used for other data processing.

Buffers in the GE-225 are of two types: direct I-O buffers and controller buffers, as illustrated in Figure 1-2. Direct I-O buffers, located within the central processor, are for use with peripherals having direct access to core storage, such as the card reader and punch, the paper tape reader-punch, and the console typewriter. Controller buffers are located in the separate controllers for high-speed peripherals, such as magnetic tape handlers, MRADS units, and high-speed printers. Buffers for these units have access to core storage indirectly through the controller selector.

The Interrupt Principle

The interrupt principle takes advantage of the significant difference in operating speeds of the central processor and the peripherals by permitting the normal 'fetch instruction, execute, fetch instruction, execute, fetch...etc.,' sequence of the central processor to be interrupted for data transfers.

Two kinds of interrupt are provided in the GE-225. One, related to normal program processing, is called priority interrupt; the other, related to multi-program processing, is called automatic program interrupt.

PRIORITY INTERRUPT

In the GE-225, buffering permits two or more operations in a program to be performed simultaneously; for example, cards or tape can be read while computing occurs in the central processor and, at the same time, cards or tape can be written. In the example, computation and access to core storage by the central processor are interrupted whenever the input or output buffers are filled or emptied and a core storage access cycle is required to transfer data.

If the central processor requests memory access while input or output peripherals are requesting access, the processor obtains access on the first free cycle. Because several requests for access to core storage might be made at the same time, provision is made to grant only one request for access during a memory cycle. The priority interrupt logic incorporated into the system analyzes these requests for access and determines which of four possible channels is to have access during that particular cycle. Refer to Figure 1-3.

All access to memory, including that by the central processor, is controlled by the priority interrupt logic, which controls four channels. The first channel has highest priority; the fourth channel has lowest priority. Normally, priority is assigned to components thusly:

Channel and Priority Assignment	Peripheral or Equipment
1	Card Reader
2	Controller Selector
3	Card Punch
4	Central Processor, including Console Typewriter and Paper Tape Reader-Punch

In general, priority is determined by the operating characteristics and buffering of system peripherals. Usually, the peripheral having a high data transfer rate will have a high priority; the peripheral with a low data transfer rate will have a low priority. Two major exceptions to this arrangement are the card reader and the central processor.

The card reader is buffered in such a way that it must have uninterrupted access to core storage while it is reading each character on a card, or data may be lost. The card reader is assigned the highest priority.

On the other hand, the central processor is assigned the lowest priority (with the console typewriter and paper tape reader-punch) because there is no danger of lost data if central processor operation is interrupted by higher-priority peripherals. Also, program-run-time is optimized if fully-buffered peripherals are permitted to operate at capacity.

The controller selector, through which all high-speed peripherals access core storage, is assigned the second-highest priority. These peripherals are fully buffered and there is little danger of data loss if their operation is interrupted. Controller selector priority is further discussed below.

The card punch which is a comparitively slow peripheral, is assigned the third priority channel because a card punch operation is initiated only when the card punch buffer is filled. The card punch buffer can maintain a partially-filled condition indefinitely; thus, interrupting card punch operations cannot cause inadvertent data loss.

Controller Selector Priority Interrupt. The controller selector is the common control and transfer point for input-output peripherals. Specifically, the controller selector: 1) provides peripheral configuration flexibility and 2) permits the establishment of user-determined priority systems.

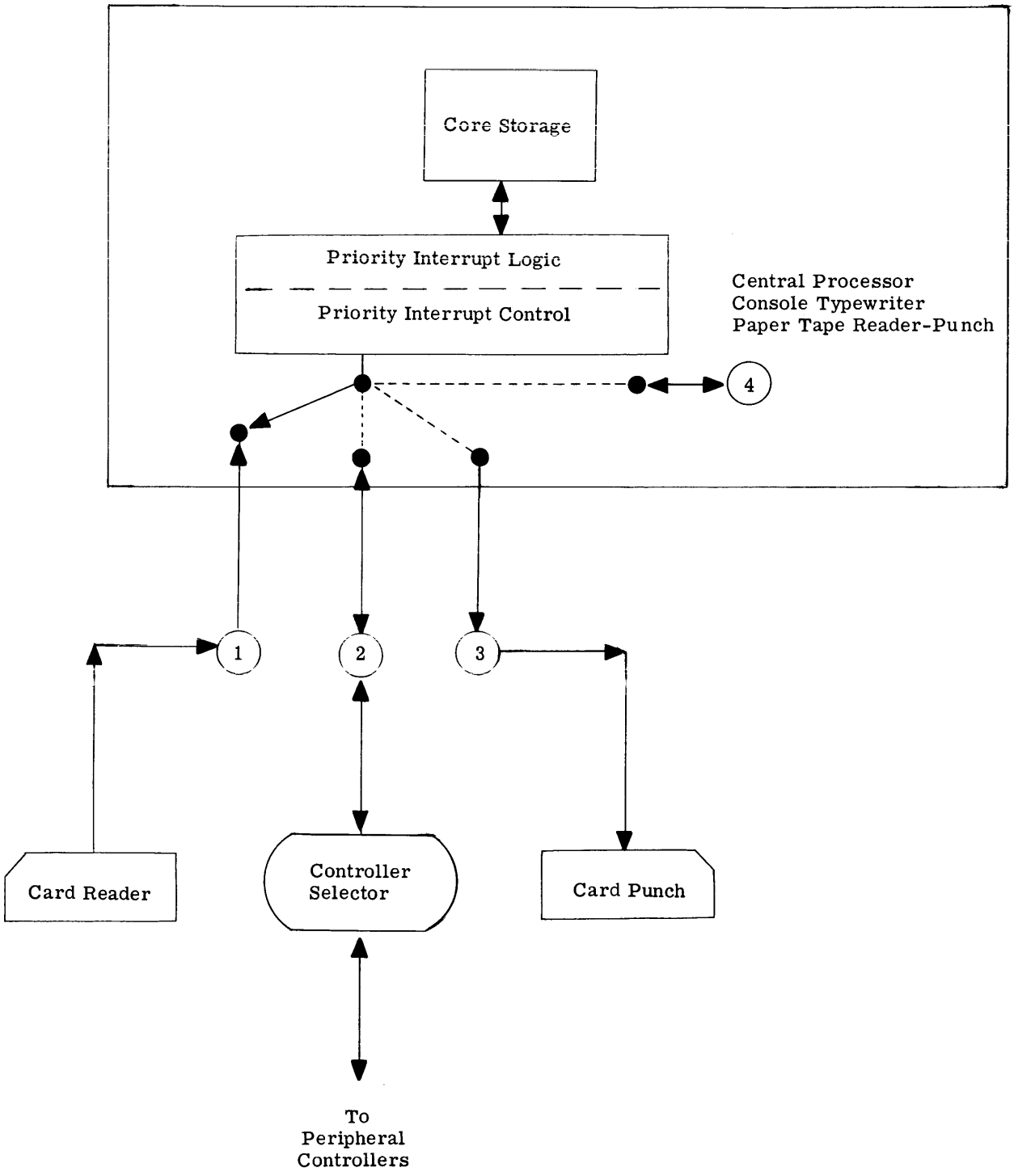


Figure 1-3. GE-225 Priority Access System

The controller selector permits the use of a wide variety of peripherals. Through plug-in connectors, peripheral controllers can be connected in many ways and changed to meet varying system requirements. This ability allows for addition of specific peripherals as the needs of an installation grow. It also allows for the addition of new or improved input-output units with little or no logic or wiring changes. Figure 1-4 illustrates one possible system configuration. Smaller or different configurations are also possible.

In Figure 1-4, the card reader, card punch, paper tape reader-punch, and console typewriter are connected directly to the central processor. The other peripherals, through their controllers, are connected to the central processor through the controller selector. As many as eight controllers can be connected to the controller selector through eight plug-in connectors, each with an individual address; these controllers can be a combination of the following:

- 1 or 2 DSU Controllers
- 1 to 8 Magnetic Tape Controllers
- 1 to 8 High-Speed Printer Controllers
- 1 to 8 Datanet-15 Controllers
- 1 to 8 Document Handler Adapters (Controllers)
- 1 Auxiliary Arithmetic Unit (includes Controller)

As shown in Figure 1-4, controllers can direct the operation of several peripherals. The following list shows the maximum possible number of peripherals each respective controller can handle:

- 1 to 4 DSU Units
- 1 to 8 Magnetic Tape Handlers
- 1 High-Speed Printer
- 1 to 15 Datanet Terminals, plus a Paper Tape Reader-Punch
- 1 to 2 12-Pocket Document Handlers
- 1 Auxiliary Arithmetic Unit

The priority interrupt system actually operates on two levels. The first level assigns priority access to core storage through one of the four priority channels, with the controller selector being assigned the second-highest priority (channel 2). The second level exists within the channel 2 priority of the controller selector and is assigned through eight address hubs, numbered 0 through 7. Once a controller selector request for access is granted, the controller selector priority system determines which of two or more requesting controllers is to receive memory access. Which controller receives access is determined by its assigned priority, as evidenced by the controller selector address hub to which it is connected. The controller connected to address hub 0 has highest priority; the controller on hub 7 has lowest priority within the controller selector priority.

Thus, any controller on the controller selector has a higher priority than the card punch (channel 3) or the central processor and its associated peripherals (channel 4).

Figure 1-5 is an expansion of the priority interrupt control system shown previously in Figure 1-3. This diagram further illustrates the relationship between overall system priority and controller selector priority.

The priority assignments for peripherals connected through the controller selector should be consistent with the data transfer rates and the relative amounts of data to be transferred by each peripheral. If requests for access are received from two units simultaneously, the one having the higher transfer rate will have the higher priority and be granted access first. The other unit, having the lower priority, must wait at least one memory cycle before attaining access. The reasoning behind this arrangement is that the slower unit can wait longer with less effect on total processing time and less danger of data loss than can the faster unit. A magnetic tape controller, for example, generally should have a higher priority (lower priority address) than does a printer controller. Once a magnetic tape controller initiates tape motion, the controller must have ready access to memory for optimum data transfer. The printer, on the other hand, does not start printing until it has received all requisite data, and can therefore afford to wait several cycles for data.

AUTOMATIC PROGRAM INTERRUPT

Because the central processor will lose no information if program processing is temporarily interrupted, it is possible to provide instruction coding in a main program for an automatic interruption of the program to process one or more 'priority' programs.

Automatic program interrupt is an optional feature to control the simultaneous processing of two or more unrelated programs. This provides for concurrent operation of peripherals while the main program is being processed. Priority programs could include those in which it is desired to transfer data from cards, tape, or core storage to the high-speed printer, or to an MRADS unit.

Automatic program interrupt in the central processor monitors the card reader, card punch, and controller selector peripherals; the interrupt feature takes effect only when a peripheral that has previously been engaged returns to the idle status. Initial engagement of the peripheral is controlled by the stored program. An

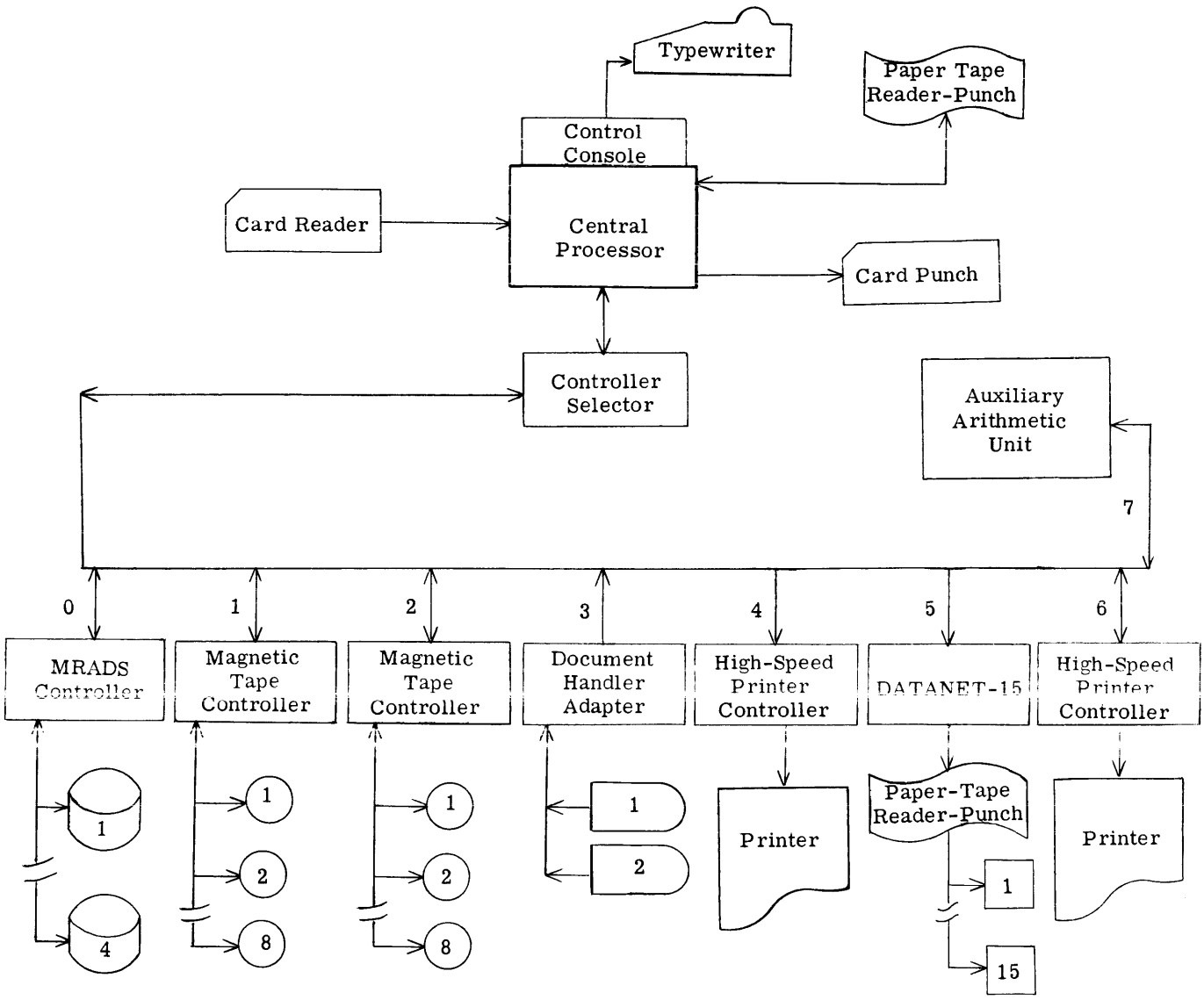


Figure 1-4. Large GE-225 System Configuration

instruction early in the main program sets the automatic program interrupt to permit exit from the program when a peripheral signals the central processor that it is idle. Note that this differs from priority interrupt, which requires that a peripheral actively request access to memory. An automatic program interrupt causes a transfer from the main program to a 'priority' routine which initiates use of a peripheral and subsequently returns control to the main program; simultaneously, the peripheral continues operation. When interruption of the main program occurs, the location of the next main program instruction to be executed is stored in a special modification word. When the 'priority' routine is completed, a branch instruction returns control to the main program.

Entry to a 'priority' routine automatically turns off the automatic program interrupt. To permit further interruptions of the main program, the 'priority' routine must reset the automatic program interrupt before returning control to the main program.

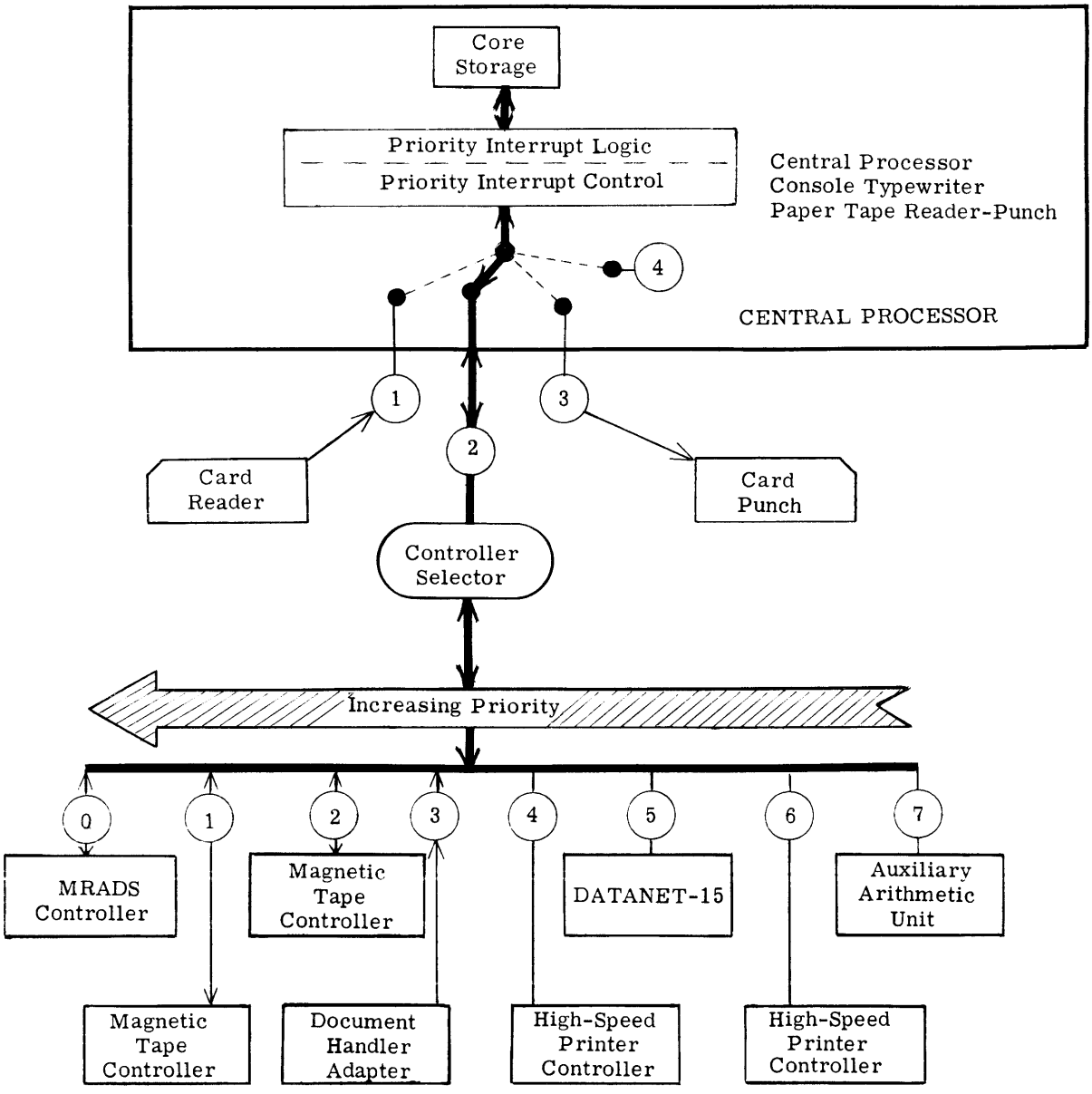


Figure 1-5. Controller Selector Priority

SECTION II

MACHINE LANGUAGE

To efficiently program the GE-225, the programmer should have a certain amount of knowledge concerning numbering systems other than the familiar decimal notation. He should also know how to convert numbers from one system to another. The reasons for this are simple: 1) the GE-225 system holds and manipulates data in binary notation, 2) the programmer generally functions most effectively when working with numbers in the decimal form, and 3) because neither decimal nor binary notation is satisfactory as a common language between programmer and computer, an intermediate numbering system (octal notation) is often useful.

NUMBER SYSTEMS

The decimal number system consists of ten digits, 0 through 9, which are used in combination to express values greater than 9. Depending upon their relative positions in a number, digits are considered to be equal to the digit times a positional factor. This factor is some exponential power of ten, the base of the decimal system. For example, the number 458 is actually an abbreviated way of expressing the following:

Digit		Positional factor		Value	
4	x	10^2	=	400	hundreds
+ 5	x	10^1	=	+ 50	tenths
+ 8	x	10^0	=	+ 8	units
			=	458	

Any value less than infinity can be expressed in the decimal system by expanding the number of positional factors as far as necessary.

	10,000's	1,000's	100's	10's	1's
Positional factor	10^n.....10^4	10^3	10^2	10^1	10^0
Digit positions	X X	X	X	X	X

Other number systems are possible, using bases other than ten. In each system, the number of digits used corresponds to the base. A number system with a base of 7 could have the digits 0 through 6, with positional values corresponding to the powers of 7. Note that, whatever the number system, the highest digit used is one less than the base of the system.

Binary Number System

The binary number system uses two digits, 0 and 1, called binary digits or bits, and has a base of 2. Positional notation is similar to that of the decimal system. Successive positions in a binary number, from right to left, have values corresponding to increasing powers of 2. Thus, the binary number 11011101 is equal to $1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$, or 221 in decimal notation.

Like the decimal system, any number less than infinity can be expressed by using enough positions.

Decimal value	etc.....256	128	64	32	16	8	4	2	1
Positional factor	2^n .....2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Digit position	X X	X	X	X	X	X	X	X	X

Counting in binary is similar to decimal, beginning with 0, then 1. Once the highest digit is reached, a carry to the left adjacent digit position is made and the count starts at zero again. Thusly:

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
etc.	etc.

Addition in binary is simpler than decimal addition, as illustrated in Figure 2-1. Other arithmetic operations are similarly easy.

+	0	1
0	0	1
1	1	10

Figure 2-1. Binary Addition Table

The table shows that 0 + 0=0, 0 + 1=1, 1 + 0=1, and 1 + 1=0 plus a 1 carry. In a two-number addition, the largest intermediate sum is never more than 1 with a 1 carry.

Example: Add the binary numbers 10110101 and 11010110

```
 1 1 1 1   1        <---carry
   1 0 1 1 0 1 0 1
 + 1 1 0 1 0 1 1 0
 ---------------------
=1 1 0 0 0 1 0 1 1
```

Octal Number System

The octal number system uses eight digits, 0 through 7, and the base 8. Again, positional notation is similar to that of the decimal and binary systems. Successive positions in an octal number, from right to left, have values corresponding to increasing powers of 8. Thus the octal number 1376 is equal to $1 \times 8^3 + 3 \times 8^2 + 7 \times 8^1 + 6 \times 8^0$, or 766 in decimal notation.

The octal system can be extended to express any size number.

Decimal value	etc.....	262,144	32,768	4096	512	64	8	1
Positional factor	8^n	8^6	8^5	8^4	8^3	8^2	8^1	8^0
Digit position	X	X	X	X	X	X	X	X

Octal counting is also similar to decimal counting. The count begins with 0, proceeds to 7 (the largest octal digit), generates a carry into the adjacent left position, and starts again at zero. Thusly:

Decimal	Octal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	10
9	11
10	12
11	13
12	14
13	15
14	16
15	17
16	20
etc.	etc.

Octal addition and other arithmetic operations are more difficult than binary or the familiar decimal operations. The most useful is octal addition, which is facilitated by tables such as that shown in Figure 2-2.

Octal Digits +	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

Octal Digits

Figure 2-2. Octal Addition Table

The table is useful in adding two octal numbers, which is the most common application the programmer will require.

Example: Add the octal numbers 642351 and 162534.

```
  1 11      carry
  642351
+ 162534
 1025105
```

Notation Convention

Wherever the possibility of confusion exists, a subscript notation is used to indicate to which system a given number belongs. For example, 1010 could be a binary representation of the decimal number 10, an octal representation of the decimal number 520, or the decimal number 1010_{10}. If a number is expressed in binary notation the subscript $_2$ is used: 1010_2. Octal numbers are shown with a subscript $_8$: 123_8. Decimal numbers are shown with a subscript $_{10}$: 876_{10}. If it is evident from the text which notation is used, the subscript is omitted.

Decimal-To-Binary Conversion

To convert a decimal number to binary, divide the decimal number repeatedly by 2. After each division, write down the remainder in sequence from right to left. The remainders will be the binary equivalent of the initial decimal number. Note that each division by two leaves either a 0 or a 1 as a remainder.

Example: Find the binary equivalent of the decimal 53.

```
    26
2 ⌈53
   52
    1  ------------------>  1      1st remainder

    13
2 ⌈26
   26
    0  ------------------>  01     1st two remainders

     6
2 ⌈13
   12
    1  ------------------>  101    1st three remainders

     3
2 ⌈ 6
    6
    0  ------------------>  0101   1st four remainders

     1
2 ⌈ 3
    2
    1  ------------------>  10101  1st five remainders

     0
2 ⌈ 1
    0
    1  ------------------>  110101 all remainders
```

Binary-to-Decimal

Binary numbers can be converted to decimal by the same method as decimal-to-binary conversion, except that the division is by 10_{10} expressed in binary (1010) and the arithmetic is in binary. After each division, the binary remainder is converted to a decimal digit.

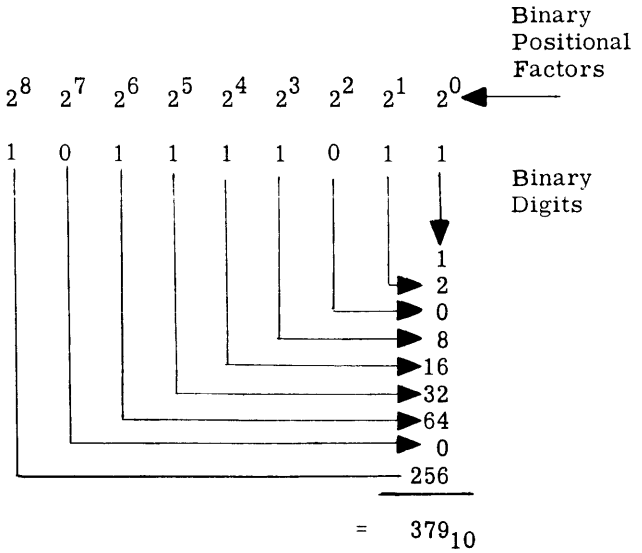
The remainders, in reverse sequence, are the decimal equivalent of the original binary number.

Example: Convert 101111011_2 to decimal notation.

```
        100101
1010 ) 101111011
       1010
        1110
        1010
         10011
         1010
          1001   = 9_10 ----> 9 units digit

          11
1010 ) 100101
       1010
        10001
        1010
          111   = 7_10 ----> 79 tens and units digits

           0
1010 )    11
          0
          11   = 3_10 ----> 379 hundreds, tens, and unit digits.
```

Another method would be simply to look up the decimal equivalents of the corresponding powers of two in the table shown in Figure 2-3 and add.

Example: Convert 101111011_2 to decimal notation.



	2^n	n
8^0=	1	0
	2	1
	4	2
8^1=	8	3
	16	4
	32	5
8^2=	64	6
	128	7
	256	8
8^3=	512	9
	1 024	10
	2 048	11
8^4=	4 096	12
	8 192	13
	16 384	14
8^5=	32 768	15
	65 536	16
	131 072	17
8^6=	262 144	18
	524 288	19
	1 048 576	20
8^7=	2 097 152	21
	4 194 304	22
	8 388 608	23
8^8=	16 777 216	24
	33 554 432	25
	67 108 864	26
8^9=	134 217 728	27
	268 435 456	28
	536 870 912	29
8^{10}=	1 073 741 824	30
	2 147 483 648	31
	4 294 967 296	32
8^{11}=	8 589 934 592	33
	17 179 869 184	34
	34 359 738 368	35
8^{12}=	68 719 476 736	36
	137 438 953 472	37
	274 877 906 944	38
8^{13}=	549 755 813 888	39
	1 099 511 627 776	40

Figure 2-3. Table of Powers of 2 and 8

Binary-To-Octal Conversion

Converting numbers from binary to octal notation is a simple mechanical procedure. Three binary digit positions are the equivalent of one octal bit position. Thus, a 15-bit number, such as $101\ 001\ 110\ 111\ 001_2$, is a 5 digit octal number when converted. To convert, the binary digits are separated into groups of three, beginning on the right. Each group of three is evaluated individually; the right-most bit has a weight of 1, the center bit is 2, and the left-most bit equals 4. Assuming 1-bits in all three positions of a group, the highest value expressible is 7, which is the largest octal digit.

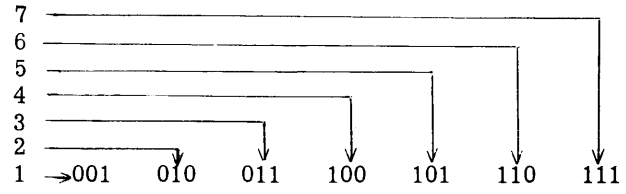
Example: Convert 1010011101110001_2 into octal notation.

$8^4\ \ 8^3\ \ 8^2\ \ 8^1\ \ 8^0$	Octal Position Factors
421 421 421 421 421	Conversion Weight
101 001 110 111 001	Binary Number
= 5 1 6 7 1	Octal Equivalent

Octal-to-Binary Conversion

By reversing the above process, conversion from octal to binary notation is simplified. Beginning with the right-most digit of the octal number, each digit is converted to its binary equivalent. Each octal digit, upon conversion, requires three bit positions.

Example: Convert 1234567_8 into binary notation.



Octal-to-Decimal Conversion

One method of converting octal numbers to their decimal equivalents is to 1) convert the octal number to binary and 2) convert the binary equivalent to decimal, by the previously described procedures.

Another method is to use a conversion table and merely look up the equivalent decimal number. For large octal numbers, such conversion tables often run to many pages. The short conversion table in Figure 2-4 is useful in converting octal numbers up to 3777777 (sufficient for GE-225 programming) directly to decimal notation. The table shows the decimal equivalents of all octal digits as a function of their position in the octal number.

To illustrate the use of the table, consider the octal number 1761354. To convert this number to its decimal equivalent, read the equivalent decimal value of each octal digit from the table and add them to find the total decimal equivalent, as shown below:

Octal Positions							Decimal Positions					
8^6	8^5	8^4	8^3	8^2	8^1	8^0	10^5	10^4	10^3	10^2	10^1	10^0
1	7	6	1	3	5	4 → =						4
						→ =					4	0
						→ =				1	9	2
						→ =				5	1	2
						→ =		2	4,	5	7	6
						→ =	2	2	9,	3	7	6
						→ =	2	6	2,	1	4	4
thus, 1761354_8						=	5	1	6,	8	4	4_{10}

OCTAL DIGIT VALUE	OCTAL DIGIT POSITION						
	8^6	8^5	8^4	8^3	8^2	8^1	8^0
1	262,144	32,768	4,096	512	64	8	1
2	524,288	65,536	8,192	1,024	128	16	2
3	786,432	98,304	12,288	1,536	192	24	3
4	-	131,072	16,384	2,048	256	32	4
5	-	163,840	20,480	2,560	320	40	5
6	-	196,608	24,576	3,072	384	48	6
7	-	229,376	28,672	3,584	448	56	7

Figure 2-4. Octal-to-Decimal Conversion Chart

Decimal-To-Octal Conversion

Decimal-to-octal conversion can be done by first converting the decimal number to its binary equivalent, then reconverting the resulting binary number to octal notation.

Another method involves the use of the two tables in Figure 2-5. The octal equivalents of the decimal digits are found in the upper table and are then added octally. The lower table assists in the required octal addition, by permitting the octal equivalents to be added in decimal, a column at a time, then converted to octal notation.

CONVERSION CHART

DECIMAL DIGIT	DECIMAL POSITION					
	10^5	10^4	10^3	10^2	10^1	10^0
1	303,240	23,420	1,750	144	12	1
2	606,500	47,040	3,720	310	24	2
3	1,111,740	72,460	5,670	454	36	3
4	1,415,200	116,100	7,640	620	50	4
5	1,720,440	141,520	11,610	764	62	5
6	2,223,700	165,140	13,560	1,130	74	6
7	2,527,140	210,560	15,530	1,274	106	7
8	3,032,400	234,200	17,500	1,440	120	10
9	3,335,640	257,620	21,450	1,604	132	11

OCTAL EQUIVALENTS OF DECIMAL NUMBERS

DECIMAL	OCTAL	DECIMAL	OCTAL	DECIMAL	OCTAL
1	1	15	17	29	35
2	2	16	20	30	36
3	3	17	21	31	37
4	4	18	22	32	40
5	5	19	23	33	41
6	6	20	24	34	42
7	7	21	25	35	43
8	10	22	26	36	44
9	11	23	27	37	45
10	12	24	30	38	46
11	13	25	31	39	47
12	14	26	32	40	50
13	15	27	33	41	51
14	16	28	34	42	52

Figure 2-5. Decimal-to-Octal Conversion Charts

Example: Convert 345978_{10} to octal notation.

Decimal Positions						Octal Positions							
10^5	10^4	10^3	10^2	10^1	10^0	8^6	8^5	8^4	8^3	8^2	8^1	8^0	
3	4	5	9	7	8 →						1	0	=
				→	=					1	0	6	=
			→	=					1	6	0	4	=
		→	=					1	1	6	1	0	=
	→	=					1	1	6	1	0	0	=
→	=					1	1	1	1	7	4	0	

thus, 345978_{10} = 1 2 4 3 5 7 2_8

Adding the 8^0 column in decimal gives 10_{10}, which is 12_8, according to the lower table in Figure 2-5. Writing the 2, carrying a 1 into the 8^1 column, and adding in decimal gives 7_8 and no carry; write the 7. Adding the 8^2 column in decimal gives 21_{10}, which is 25_8. Writing the 5, carrying a 2 into the 8^3 column, and adding gives 11_{10} or 13_8. Writing the 3 and carrying the 1 into the 8^4 column gives 4_8, no carry; write the 4. The 8^5 column gives 2 and the 8^6 column is 1.

DATA WORDS

In the GE-225, the word (or basic unit of information) consists of 20 binary digits. Words can be stored in 4096 to 16,384 core storage locations, each of which is individually addressable. Additional random access and sequential access storage is available in MRADS units and magnetic tape.

A word can be an instruction, a binary data word or number, a binary-coded-decimal word (for expressing either alphabetic or numeric characters), or any pattern of 20 bits the programmer so desires. The 20 bit positions of the GE-225 word are depicted in Figure 2-6. S (or 0) refers to the sign position, 1 indicates the high-order bit position, 2 the next highest, and so on. Bit position 19 indicates the low-order bit position.

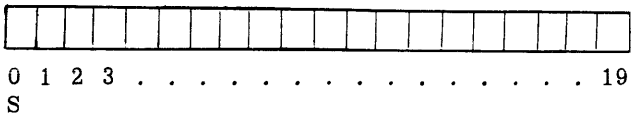


Figure 2-6. Basic GE-225 Word

Binary Data Words

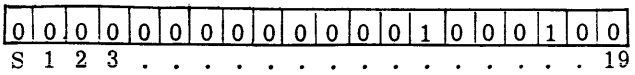
When a word is interpreted by the GE-225 as binary data, the 0 (or S) position acts as the arithmetic sign. A 0-bit in the sign position indicates that the word is positive; a 1-bit indicates that the word or number is negative. In binary words, 1-bits in positions 1 through 19 indicate values corresponding to the powers of two. A 1-bit in bit position 1 equals 2^{18} or $262{,}144_{10}$; in position 2, a 1-bit equals 2^{17} or $131{,}072_{10}$; in position 19, 2^0 or 1. The largest positive decimal number that can be expressed in the 20-bit binary word is $2^{19} - 1$, or $524{,}287_{10}$.

Negative numbers are expressed in binary form by placing a 1-bit in the sign position and the 2's complement of the desired number in bit positions 1 through 19.

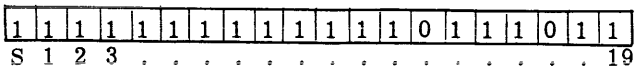
To express a given negative number:

1. Write the positive number in binary
2. Change it to the 2's complement form by
   a) converting all 1-bits to 0-bits and all 0-bits to 1-bits and
   b) adding a 1-bit to the least significant bit position.

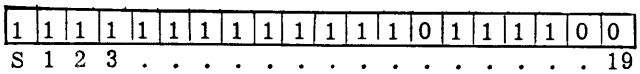
For example, to express the decimal -68_{10} in binary, write $+68_{10}$ in binary:



Inverting all bit positions gives:



Adding a 1-bit to bit position 19:



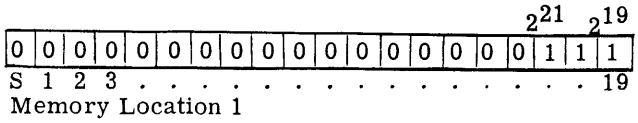
The largest negative number that can be expressed in the 20-bit binary word is 2^{19}, or $524{,}288_{10}$.

A machine instruction is provided for automatically converting a positive number to a negative number. Also, in subtract operations involving positive numbers, the required complements are automatically formed.

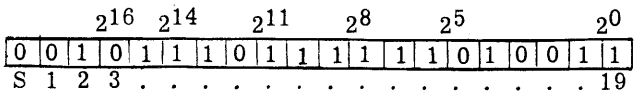
Double Length Binary Words

The GE-225 can perform double length data word operations. Double length words consist of two 20-bit words which are normally stored in adjacent memory locations. For processing, they are treated as a single word consisting of a sign bit and 38 data bits.

For illustration, consider the decimal $3{,}862{,}483_{10}$. In binary, this number would be stored in two adjacent memory locations:



Memory Location 1



Memory Location 2

The most significant half of the double word is stored in the first memory location. The adjacent (higher) location contains the least significant half of the word. Bit positions in the second memory location have values corresponding to the first nineteen powers of two (2^0 through 2^{18}), while those of the first (lower) memory location correspond to the second nineteen powers of two (2^{19} through 2^{37}). The signs of both locations are the same, 0 for plus or 1 for minus. Double length negative numbers are expressed in the 2's complement form.

Floating-Point Notation

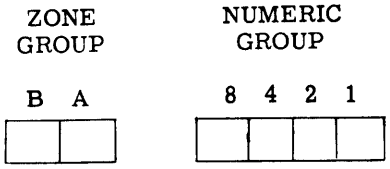
The auxiliary arithmetic unit (AAU) expands the arithmetic capability of the GE-225 to include normalized and unnormalized floating-point operations. Representation of floating-point numbers is discussed in the section, Auxiliary Arithmetic Unit Operations.

GE-225 installations, with or without the AAU, can process floating point arithmetic with utility subroutines provided by General Electric for this purpose. However, for voluminous floating point calculations, the AAU provides greater efficiency, because of its speed and capacity.

Binary-Coded-Decimal Data Words

In addition to its basic binary capability, the GE-225 can process binary-coded-decimal (BCD) or alphanumeric data. The six bit positions of the BCD code may be used to express 64 character configurations, including all alphanumeric and special characters of the GE-225 character set.

The 6-bit code consists of two groups:



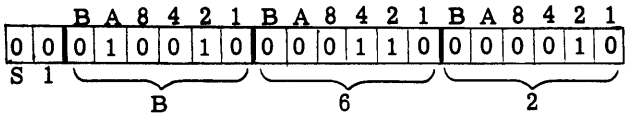
The numeric bits correspond to the first four powers of two, as they do in the binary system, and can express up to 16 numeric values, 0 through 15. The zone bits provide for coding alphabetic and special characters.

Selected characters are shown below in BCD. All GE-225 characters and their equivalent BCD codes are shown in the Appendix.

In the BCD mode, the GE-225 word can contain three characters, occupying 18 bit positions (2 through 19).

	B	A	8	4	2	1
1	0	0	0	0	0	1
5	0	0	0	1	0	1
9	0	0	1	0	0	1
A	0	1	0	0	0	1
N	1	0	0	1	0	1
R	1	0	1	0	0	1
/	1	1	0	0	0	1
Z	1	1	1	0	0	1
$	1	0	1	0	1	1

The remaining two bit positions (S and 1) do not normally contain data, but are used for program and printer control purposes discussed later. A representative GE-225 BCD word is shown:



Double length BCD words are possible to express alphanumerics consisting of as many as six characters.

Optional instructions permit variable length BCD arithmetic operations. Negative numbers must be expressed in 10's complement form with a 1-bit in the sign position. Note that, in BCD numerics, the zone bits (2, 3, 8, 9, 14, 15 bit positions) are set to zero. Although the BCD word contains only three numerics, the variable length feature permits operations with BCD numbers of any practical length.

Examples of BCD quantities:

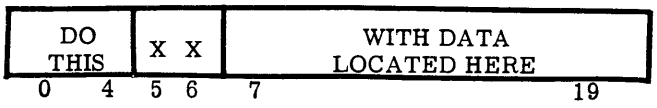
Decimal	BCD word(s)
+ 10	+ 0 1 0
+ 989	+ 9 8 9
- 10	- 9 9 0
- 989	- 0 1 1
+ 87649	+ 0 8 7 + 6 4 9
- 87649	- 9 1 2 - 3 5 1

INSTRUCTION WORDS

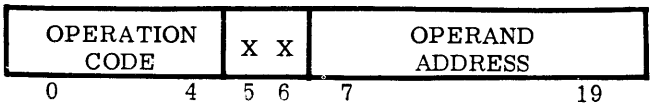
Instructions are expressed as 20-bit words. Three different formats are used.

Format I. All instructions involving reference to memory are written in Format I. Included are arithmetic, memory transfer, and certain branch instructions. Complete descriptions of these instructions are provided in subsequent sections.

The format for memory reference instructions is:



OR



The five bits (0 through 4) indicate the operation to be performed, such as add, subtract, read cards, etc.

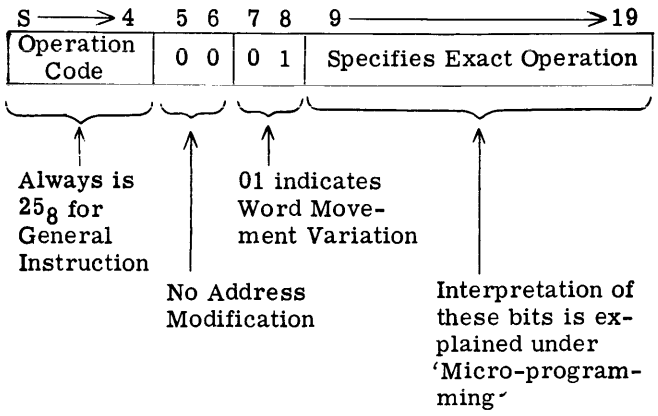
Bits 5 and 6 provide for automatic address modification by stipulating whether the contents of one of several X registers are to be used to modify the operand address. Automatic address modification is treated in Section V.

Bits 7 through 19 designate the operand address; that is, the memory location where the data to be added, subtracted, etc., is stored.

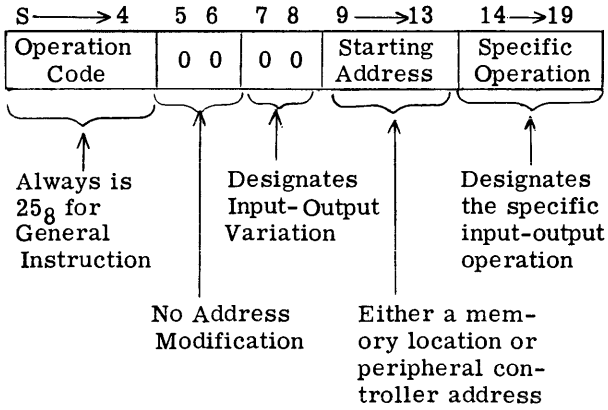
About 60 of the over 300 instructions in the GE-235 repertoire require operand addresses. Instructions without operand addresses cannot be address modified. This permits bits 5 and 6, and 7 through 19 to be used for other purposes. Instructions in this category (no operand address) are called general instructions, Format II, or shift instructions, Format III.

Format II. All instructions in data transfer (excluding memory transfer) and input-output categories and most internal test-and-branch instructions are written in Format II. Instructions in this format are commonly called general instructions and have the same operation code in bit positions S through 4 (10 101, or 25_8). Format II has three variations, corresponding to the three general categories mentioned.

The word movement variation is for instructions involving full word transfers between arithmetic registers and the arithmetic unit. They assume this format:

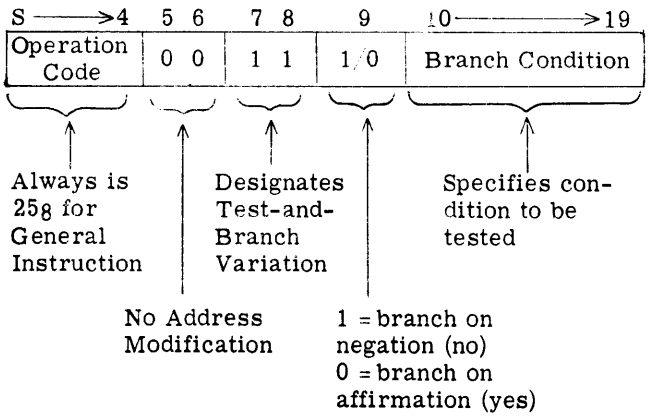


The input-output variation is used for instructions involving the central processor and peripherals. Bits S through 4 contain 25_8 (10 101) and bits 7 and 8 are 0's. The remaining bits specify the input-output operation. The format is as follows:



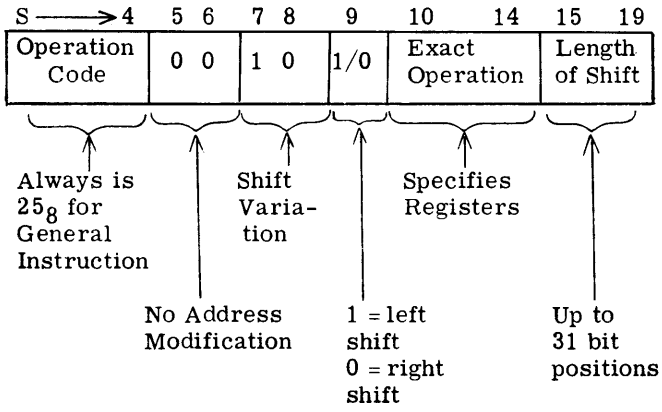
The test-and-branch variation is used for instructions that provide for breaking the normal sequence of instruction execution. These instructions are identified by 25_8 (10 101) in bit positions S through 4 and 1-bits

in positions 7 and 8. The test condition for determining a branch to another instruction is specified by bit positions 9 through 19. The format is:



The specific bit patterns for all Format II instructions can be found by converting the octal equivalent of the instructions to binary. The octal form of each instruction is included in the instruction descriptions in subsequent sections.

Format III. Only shift instructions are written in Format III. Shift instructions are used to shift one or more bits within or between arithmetic registers. Bit positions S through 4, designating the operation code, contain 25_8; bits 7 and 8 contain 1 and 0 respectively, identifying a shift operation; bit 9 indicates direction of shift (right or left); bits 10 through 14 identify the registers involved; bits 15 through 19 designate the number of bits to be shifted. The format is:



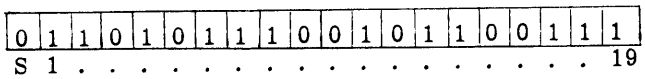
While it is possible to prepare programs for GE-225 processing directly in binary notation, it is infrequently done because such programming is tedious and subject to clerical error. However, a knowledge of binary notation and instruction structure is essential in micro-programming (the building or creating of instructions by the programmer). Micro-programming is discussed in a later section.

In program debugging and patching, octal notation is frequently used for 3 reasons: 1) octal notation provides the programmer with a more meaningful presentation than does binary, 2) the GE 225 provides printed outputs (during GAP program assembly) and memory dumps in octal notation, and 3) octal can easily be converted to binary or decimal. On the other hand, binary is difficult to read or write; also it is tedious to convert to the familiar decimal notation.

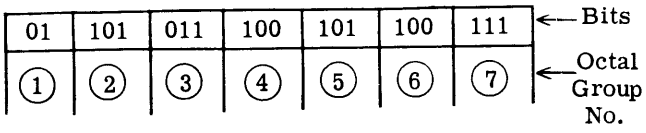
GE-225 Octal Notation

Conversion of GE-225 words from binary to octal or octal to binary is a simple mechanical procedure.

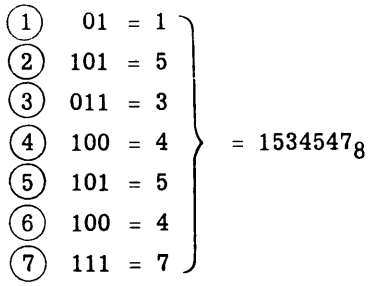
Given the GE-225 binary word:



Starting on the right, divide the word into groups of three bits (giving six groups of three, and one group of two) and assign octal values to the bit positions as shown:



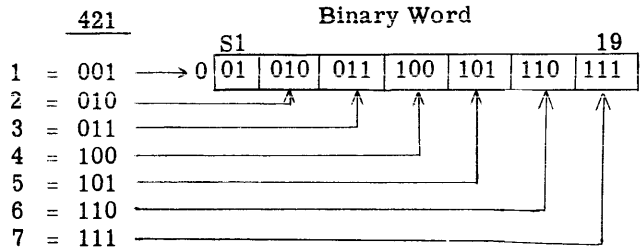
Evaluate each group and write the equivalent octal digit:



The result of the binary-to-octal conversion is a 7-digit number in place of the longer, less meaningful 20-bit binary word.

Note that any GE-225 word can be represented as a 7-digit octal number, whether it be a data word or an instruction.

The representation of the number 1234567_8 in binary is accomplished by reversing the above process:



Because of the simplicity and convenience of octal notation, it is used freely in the balance of the manual to simplify explanations and to provide familiarity.

Symbolic Programming

Most programming for the GE-225 is done with symbolic coding, made possible by the General Assembly Program (GAP). GAP provides the basic programming language and permits programming in meaningful mnemonic codes, using symbolic addressing. This relieves the programmer of much clerical and 'housekeeping' detail, thus making GAP generally preferable to binary or octal coding.

The GAP programming system is comprised of two parts: 1) the symbolic language used by the programmer in coding the source program and 2) the General Assembly Program that automatically processes the source (or symbolic) program into a ready-to-process machine-language (or object) program.

Section IV discusses all phases of the General Assembly Program, including the pseudo-instructions for controlling GAP assembly, the assembly process, and operating procedures.

The GAP symbolic programming language consists of a set of standardized mnemonic codes divided into two categories:

1. Pseudo-instructions used for memory location assignments, program constant storage, and program control during the assembly process.
2. Mnemonic operation codes corresponding to the over 300 machine instructions of the GE-225.

Pseudo-instructions, described in Section IV, have no correlation with the GE-225 machine instructions contained in the assembled object program. On the other hand, there is generally a one-to-one correlation between the mnemonic operation code prepared by the programmer and the machine instruction appearing in the object program.

In addition to convenient symbolic codes for instructions, GAP permits the programmer to reference memory with either actual numeric or symbolic locations previously defined with GAP pseudo-instructions, thereby facilitating program coding, debugging, and incremental preparation and revision.

All GE-225 instructions are described, beginning in Section V. For each instruction, the mnemonic code, its machine language equivalent (in octal), its functional description, execution time, registers affected, liberal examples of its use, and appropriate comments are shown. Instructions are grouped functionally for ready reference and are indexed both numerically and alphabetically in the Appendix.

SECTION III

CENTRAL PROCESSOR ORGANIZATION

The central processor performs all arithmetic and logical functions in the GE-225 system and acts as a central control for all internal and peripheral operations. Because the program (or instructions for data processing) is held in memory like the data to be processed, the GE-225 is known as a stored program computer.

MAGNETIC CORE STORAGE

Instructions and data are held in the primary storage unit, or memory, through the use of tiny ferrite cores. Each core is a ring, or toroid, of ferromagnetic material capable of being magnetized in one of two polarities when current is passed through wires inserted through the cores. Current through the wires generates a magnetic field which in turn magnetizes the core; when the current is stopped, the core remains magnetized. If the direction of current flow is reversed, the field about the wire is reversed and the ferrite core will be magnetized in the opposite direction. The two possible states of magnetization can be called 1 and 0, corresponding to the two binary digits.

Figure 3-1 illustrates this principle of storage. Note that two wires are used to provide the magnetizing current and current must be present in both wires to magnetize a core or switch the core from one magnetic polarity to the other. The third wire shown, the sense winding, is used to sense the change in magnetization of the core. As the core 'flips' from one magnetic polarity to the other, a pulse is induced in the sense winding by the collapsing field of original polarity and the increasing field of the new polarity.

The basic GE-225 memory module is an array or block of cores 64 cores wide, 64 cores long, and 21 cores deep. It can be visualized as 4096 vertical columns of 21 cores each. Each column of cores can contain 20 information bits plus a parity (or check) bit. When a word is stored in or read from memory, the bit pattern of the word is simultaneously set into or read from all 21 cores of the desired column or storage location. In addition to the basic 4096-word module, memory is also available with storage capacities of 8192 and 16,384 words.

Each memory word is individually addressable. Addresses are used to make data stored in memory

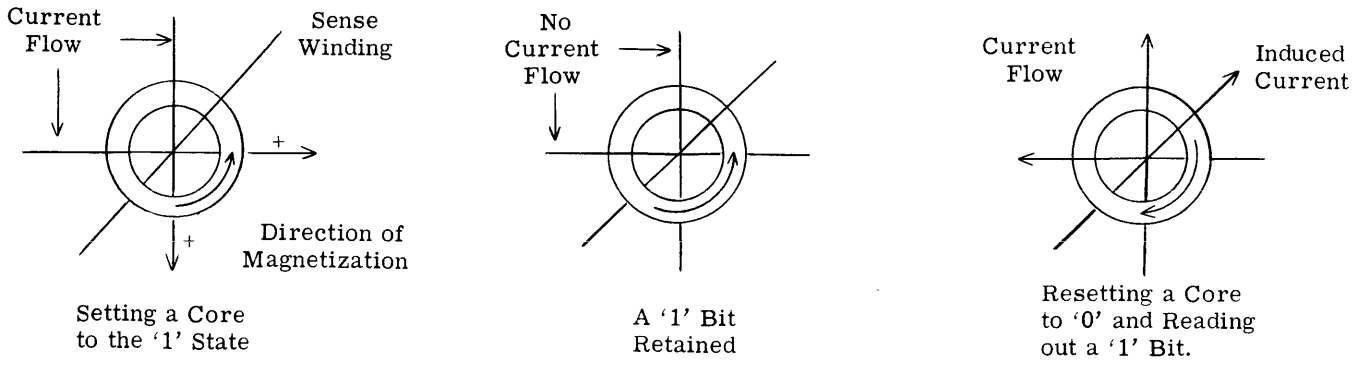


Figure 3-1. Bit Storage in a Ferrite Core

relocatable. Instructions requiring data to be moved to or from memory must specify an operand address corresponding to the memory address containing the data. Instructions held in memory are accessed by their addresses. Addresses are numbered sequentially from 0000 to 4095 (or 8191) for basic memory sizes. Addressing the additional 8192 words in a 16,384 word memory is covered in a later section.

Access time for a word stored in memory is 18 micorseconds (millionths of a second); this includes 1) reading the word from core storage, 2) storing the word in a register external to memory, and 3)restoring or replacing the word in core storage. Core storage access time is also called a memory cycle or a word time. A single data word transfer to or from memory, including access time for the instruction effecting the transfer, requires 36 microseconds (2 word times); a double length word transfer requires 54 microseconds (3 word times). When a word is read from memory, all 21 bits are transferred simultaneously. Storing a word in a given address destroys the previous contents of that address.

Stored Program

Because instructions, like data, are stored in memory, data processing can proceed automatically, performing instructions in sequence as they exist in storage, or branching to other instructions in the sequence depending upon the preceding instruction.

For the same reason, self-modifying programs are possible. Instructions can be manipulated as well as data, permitting changes to the basic program as a result of in-process decisions.

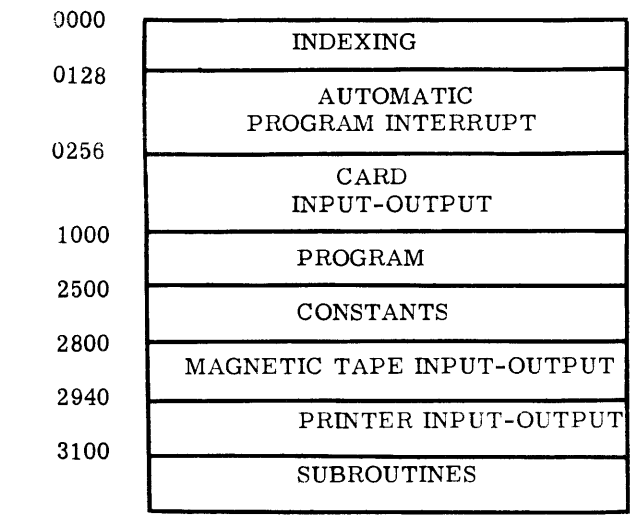


Figure 3-2. Representative Allocation of Memory

Programming efficiency is aided by good planning or the orderly use of available memory. The designation of specific areas of memory for specific purposes reduces programming time and errors. Figure 3-2 illustrates a possible allocation of memory space for input-output, constant, instruction, and subroutine storage.

X Register Operation

Memory addresses 0000 through 0003 have special properties. Instructions are provided to permit their use as program counters by making provision for incrementing their contents by a constant and testing the contents with one of two special test instructions.

In addition, locations 0001 through 0003 can be used for modification word storage and are called X registers. Bit positions 5 and 6 of the basic instruction word can be used to specify which of the three X register contents is to be used for modification, as indicated:

Bit Position 5	Bit Position 6	X Register Selected
0	0	None
0	1	0001
1	0	0002
1	1	0003

If an instruction containing an operand address also specifies an X register in bit positions 5 and 6, the contents of the specified location (0001, 0002, or 0003) are added to the operand address to give the effective address. The instruction is executed using the effective address, rather than the operand address. The original instruction in storage remains unchanged.

X registers facilitate addressing upper memory (locations above 8191), as described in the section, Addressing Upper Memory.

Additional modification words are available as part of an optional package that also provides a three-way compare instruction and decimal (BCD) arithmetic capability. The added modification words consist of 31 groups, each containing a word that can be incremented as can location 0000, and three words with the same modification properties as locations 0001 through 0003. This provides 96 modification words and 32 counter words in memory locations 0000 through 0127.

Use of the optional modification groups requires the specification of the desired modification group with a special select instruction. A group remains selected until a subsequent special select instruction is used to specify another group. Once a group is selected, the

desired modification word within the group is specified by bits 5 and 6 of the instruction. For example, if modification word group 28 were specified by a special select instruction during a normal program sequence, all subsequent instructions with X register coding of 01, 10, or 11 would be modified by the contents of locations 0113, 0114, or 0115, respectively, until another modification group was specified by another select instruction.

M Register Operation

The M register is a 21-bit register (see Figure 3-3). All information transferred to or from core storage must first pass through the M register, which is the focal point for information transfers among GE-225 system components. The 21 bits of the M register include 20 information bits, plus a parity check bit.

Parity Checking

A parity check is performed automatically as a word is read from memory into the M register. The parity check circuits count the 1-bits contained in all 21 bit positions; if the count is odd, parity is correct and operations proceed; if the count is even, then a parity error (bit drop or pick-up) has occurred and the parity alarm light on the control console is turned on. In addition, depending upon the position of the 'Stop on Parity Alarm' switch on the control console, a computer halt or a programmed branch for remedial action can occur.

Words written into memory have a parity bit generated (as required) by the parity check circuits, while the word is held in the M register. The parity check circuits count the bits and, if the count is even, generates a bit for the 21st bit position. If the count is odd, no parity bit is required. In either case, the entire 21 bit positions of the M register are stored in memory.

ARITHMETIC AND CONTROL REGISTERS

Arithmetic operations, such as addition, subtraction, multiplication, and division, require temporary storage devices external to memory for holding intermediate and final results and performing the necessary calculations. The GE-225 uses arithmetic registers for these purposes. In addition, arithmetic registers are used for shifting and other data manipulations related to decision-making and arithmetic capabilities.

Arithmetic registers include:

- B Register
- A Register
- Q Register
- N Register
- C Register (optional, not illustrated)
- Arithmetic Unit

Control registers control the sequential processing and interpretation of instructions. These registers include:

- I Register
- X Registers
- P Counter (or register)

Arithmetic Registers (Figure 3-4)

B REGISTER. The B Register is a 20-bit register which acts as a buffer register between the M register and the central processor during data transfers. The B register is also a buffer for arithmetic operation and contains:

- The addend for addition
- The subtrahend for subtraction
- The multiplicand for multiplication
- The divisor during division

Outputs from the B register are supplied to the I register and the arithmetic unit. The B register is also used in the execution of certain data transfer commands.

A REGISTER. The A Register is a 20-bit register and is used most frequently in central processor operations. It receives information from and transfers information to the arithmetic unit. It serves as the accumulator for the central processor and performs this function by holding:

- The augend during addition
- The sum after addition
- The minuend during subtraction
- The result after subtraction
- The most significant half of the product after multiplication
- The most significant half of the dividend before division
- The quotient after division
- The most significant half of a word after the execution of all double length word instructions
- A word transferred from, or to be transferred to, memory
- The word on which extraction is performed during the execution of the extract instruction (Extraction is the examination and replacement of bits in a word according to a previously-defined pattern)

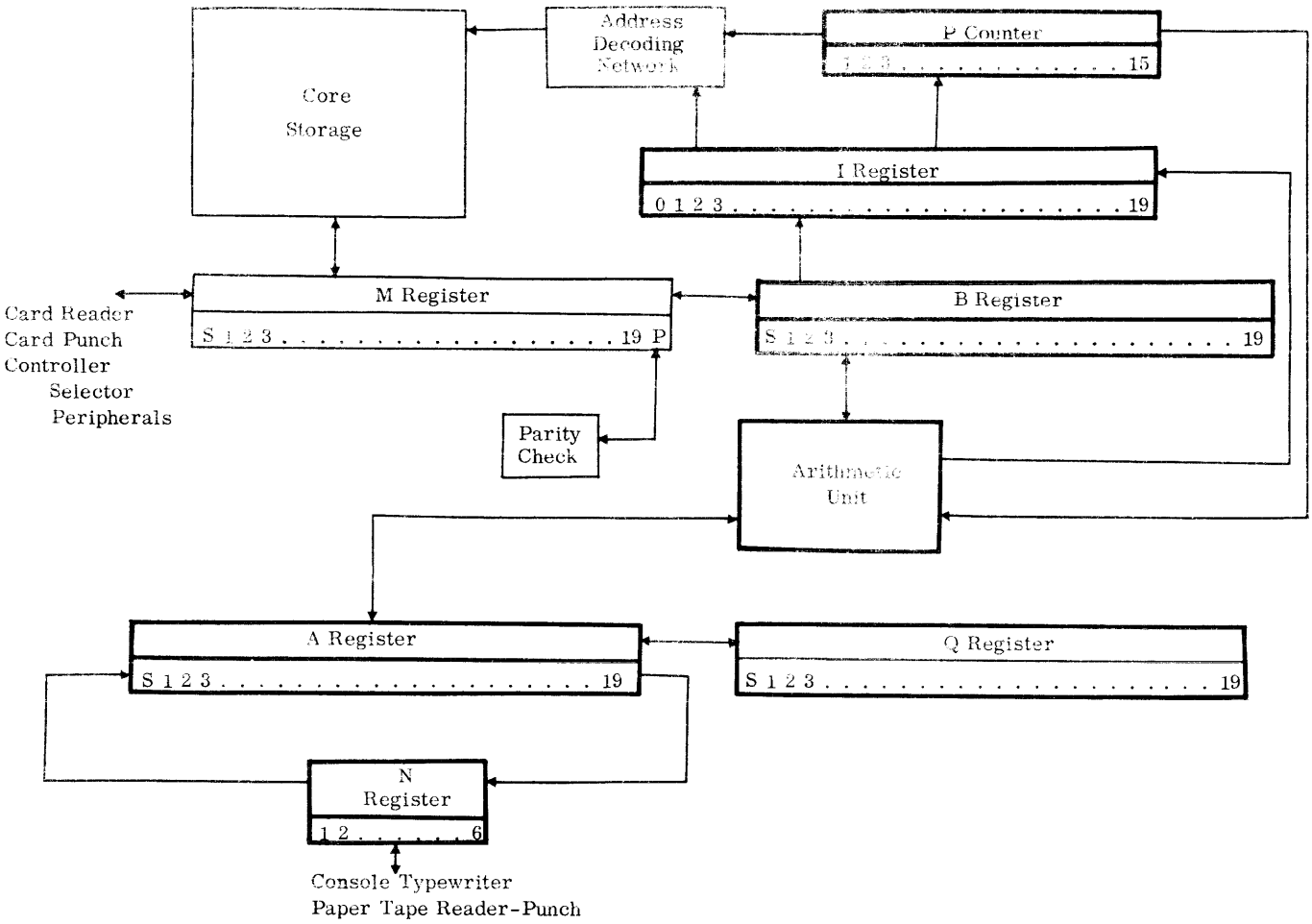


Figure 3-3. GE-225 Arithmetic and Control Register

The word to be shifted during various shift instructions

A word to be transferred to another register or to be modified in some way during the execution of various data transfer commands

The word that determines future action during the execution of branch instructions.

In addition, manual access to the A register is permitted by 20 console switches provided for this purpose.

Q REGISTER. The Q Register is a 20-bit register which acts with the A register to form a double length word accumulator (38 bits plus a sign bit) during the execution of double length word instructions. Information is not transferred directly from memory into the Q register, but is read into the A register and then shifted into the Q register. The Q register performs the following functions:

1. Holds the least significant half of the augend before double precision (double length) addition, and the least significant half of the sum after addition.
2. Holds the least significant half of the minuend before double precision subtraction, and the least significant half of the result after subtraction.
3. Holds the multiplier before multiplication.
4. Holds the least significant half of the result after multiplication.
5. Holds the least significant half of the dividend before division.
6. Holds the remainder after division.

Holds the least significant half of the double length word during the execution of double length word instructions.

Holds the least significant half of information to be shifted during double length shift instructions.

N REGISTER. The N Register is a 6-bit register which is used as a single character buffer between the central processor and 1) the console typewriter, 2) the paper tape reader, and 3) the paper tape punch. This permits input-output operations with these units to occur simultaneously with other central processor operations. Information is transferred directly between the N register and the A register by means of shift instructions.

C REGISTER. The C Register, or Real Time Clock, is an optional equipment feature that permits the timing of operations in either relative or real time. This feature is convenient where it is necessary to determine or record elapsed time of operations performed by the GE-225, or of operations external to the GE-225 system. In addition, it is possible to determine the time of an occurrence relative to actual (Greenwich or local) time or to any suitable time base.

The C register is a 19-bit binary register that can be set directly from, or read directly into, the A register. Only bits 1 through 19 of the A register are involved in such transfers.

The C register is automatically incremented by one, in binary mode, every sixth of a second while power is applied to the GE-225. When the C register count reaches the binary equivalent of 24 hours (518,400 sixths of a second), it automatically resets to zero and starts counting again. Translation of the C register contents from binary notation to clock time can be performed either manually or by a simple conversion routine. Instructions and conversion procedures are discussed in Section V.

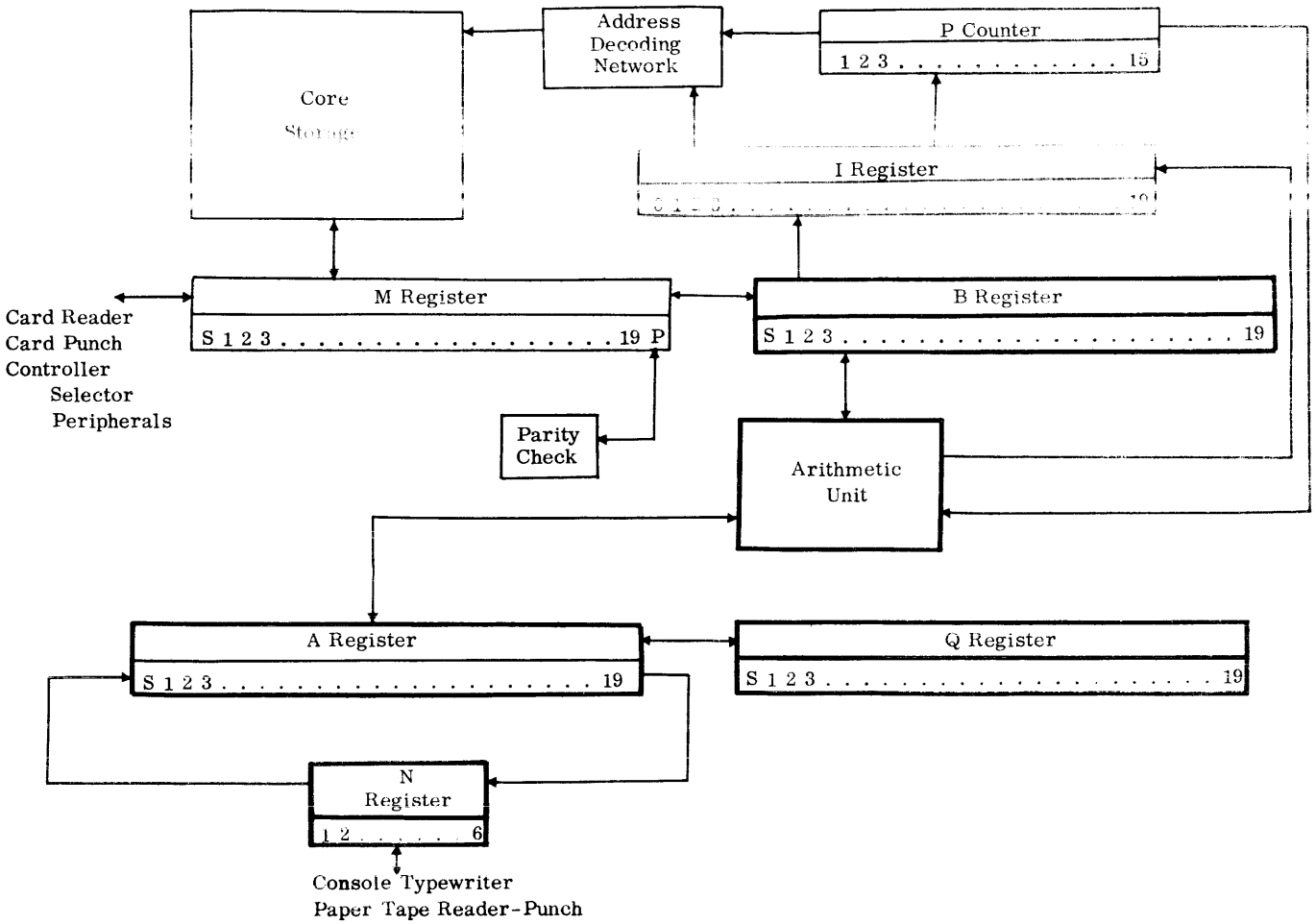


Figure 3-4. GE-225 Arithmetic Registers

ARITHMETIC UNIT. The arithmetic unit is a high-speed, parallel, binary adder network. It serves two functions. During arithmetic operations, it performs the calculations specified by the operation code in the I register. It also serves as a transfer bus for words moved between the A register and memory (via the M register), and for the operand portion of instructions moving into the I register.

Control Registers (Figure 3-5)

I REGISTER. The I Register is the instruction register. It contains all 20 bits of an instruction word during the execution of a computer instruction. While instructions are being processed, bits 0 through 4 indicate the operation to be performed, and bits 5 and 6 control the automatic address modification, if required. During the execution of instructions involving memory locations, bits 7 through 19 specify the memory address involved. Bits 5 through 19 have other meanings during the execution of general and shift instructions.

Instructions are read from memory into the M register and set into the B register. From the B register, bit positions 0 through 6, comprising the operation code and the address modification bits, are transferred directly into the I register for decoding. At the same time, bit positions 7 through 19, the operand portion of the instruction, are routed to the arithmetic unit. If bit positions 5 and 6 indicate address modification, the contents of the indicated X register are added to the instruction operand in the arithmetic unit and the modified operand is set into the I register. If no address modification is indicated, the unmodified operand is set into the I register.

X REGISTERS. X Registers, memory locations 0000 through 0003, are not actually registers, but serve some of the same functions as do control registers.

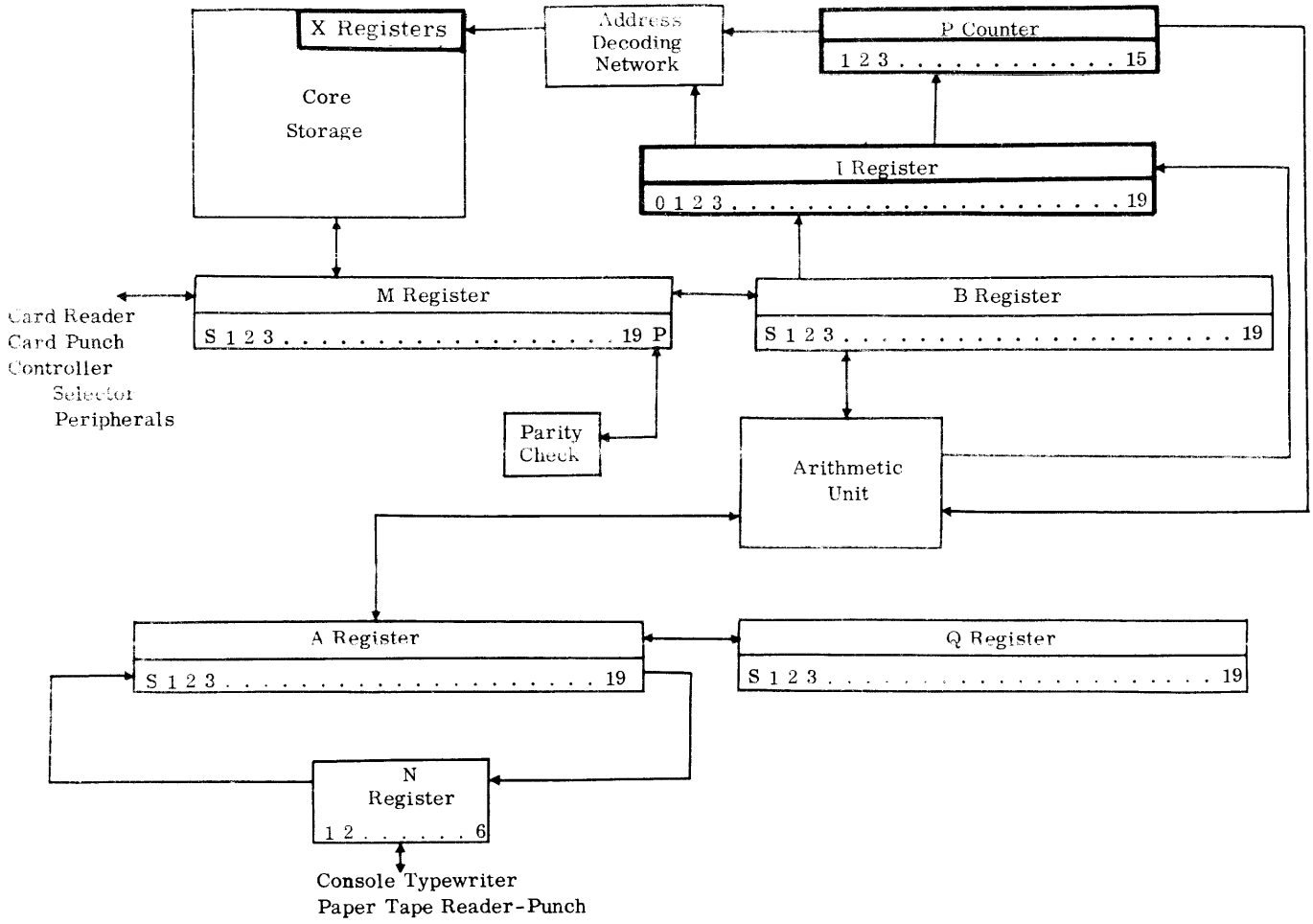


Figure 3-5. GE-225 Control Registers

These four memory locations are reserved to serve as counters and for automatic address modification.

P COUNTER. The P Counter (or register) is a 15-bit location counter that contains the memory address of the next instruction to be executed. The contents of the P counter are incremented by one before the execution of an instruction so that the P counter indicates the next instruction in sequence. The Store P and Branch instruction is an exception. The contents of the P counter can be set from the I register when unconditional branching is specified by the program. The contents of the P counter (the address of the next instruction) are displayed by 15 lights on the control console.

BASIC OPERATING CYCLE

Program execution normally proceeds with instructions executed sequentially under the control of a 450 kilocycle crystal-controlled timer. This basic timing device emits pulses every 2.25 micro-seconds. Eight sequential pulses comprise the GE-225 operating cycle of 18 microseconds, one word time. A word time is the interval required to read a word from memory, transfer it to the proper register(s), and restore the word in memory. Figure 3-6A, Word Time #1, illustrates the basic read-write cycle.

In executing a program instruction, one word time is required to fetch an instruction from memory and another (Word Time #2, Figure 3-6A) is normally required to fetch the operand specified and perform the operation - a minimum of two word times per instruction. Instructions indicating address modification require an additional word time to fetch the address modifier from the specified X register, augment the original operand with the modifier, and transfer the updated address to the appropriate register. See Figure 3-6B.

Some instructions require more than one word time for execution. Examples include double length word, multiply, divide, and shift instructions. The additional

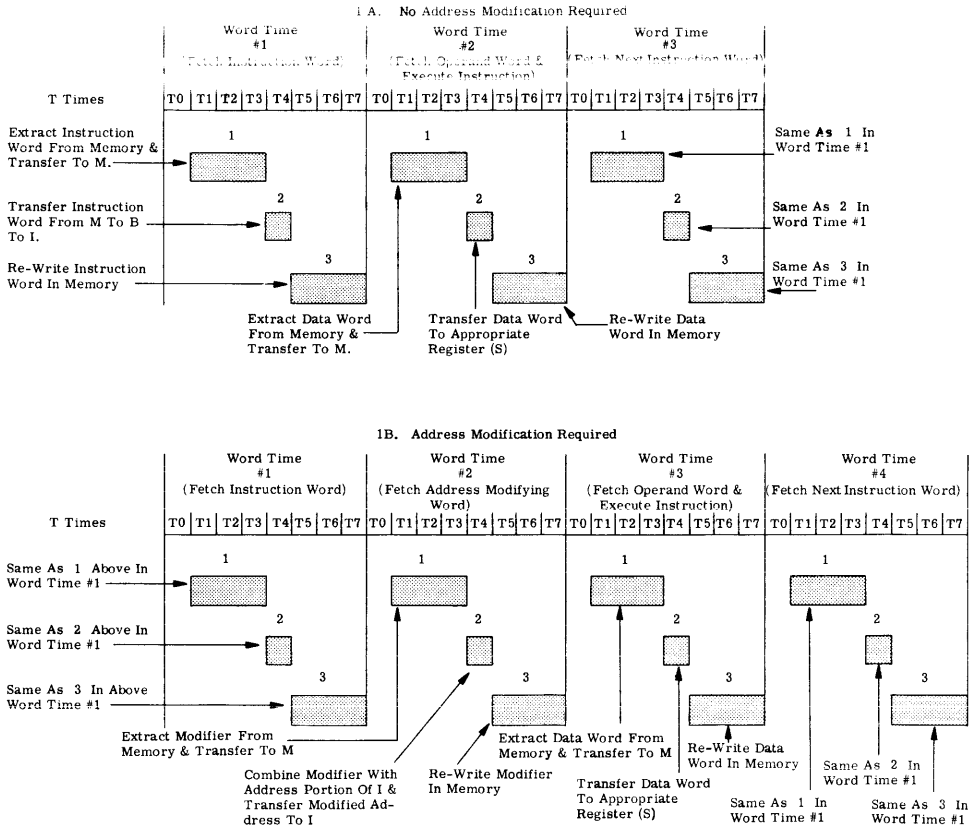


Figure 3-6. Basic Timing for Single Length Word Operations

word times required are automatically provided by the central processor sequence control logic.

Single word transfers from or to memory, including instruction access time and not involving address modification, require two word times; double length word transfers require three word times. Execution times for all instructions are included in the individual instruction descriptions.

Sequencing

Instructions are normally executed sequentially. Within each operation cycle, the control logic of the central processor provides sequence control for:

1. Fetching the instruction,
2. Modifying the operand address (if required), and
3. Executing the instruction.

The sequence control causes repetitive performance of this cycle automatically, thus permitting execution of successive program instructions. In addition, by monitoring the execution of multiple-word-time instructions, the sequence control provides appropriate control signals to make available the necessary word times for execution before the next instruction is fetched from memory.

Operation Cycle, General

Instructions are executed sequentially, except when decision instructions or priority or program interrupts break the sequence and commence processing at another point in the program. The operation cycle described briefly in Sequencing, above, consists of two phases: the instruction phase and the execution phase, thereby giving meaning to the term, instruction-execution cycle.

INSTRUCTION PHASE. The instruction phase serves three functions:

1. To locate the instruction in memory and transfer it to the I (instruction) register.
2. To locate the data in memory as specified by the instruction operand address.
3. To establish execution control circuits for the instruction.

The instruction phase is illustrated more clearly by the flow chart in Figure 3-7. During this phase, an instruction is read from memory and stored in the I register. The operation code (bits 0 through 4) of the instruction word are examined by the instruction decoding logic to determine the kind of instruction, that is, branch, shift, arithmetic, etc. If necessary, the remaining bits are also examined. This examination established the necessary controls for directing processing during the execution phase.

During the examination, the P counter is incremented by one to contain the address of the next instruction in sequence. The control circuits ask, " is the instruction in the I register to be modified?" If yes, the contents of the specified X register are read from memory and added to the operand address in the A register, then sent to the I register. If no, the instruction is executed. When the central processor is stopped manually, the P counter displays the address of the instruction currently in the I register.

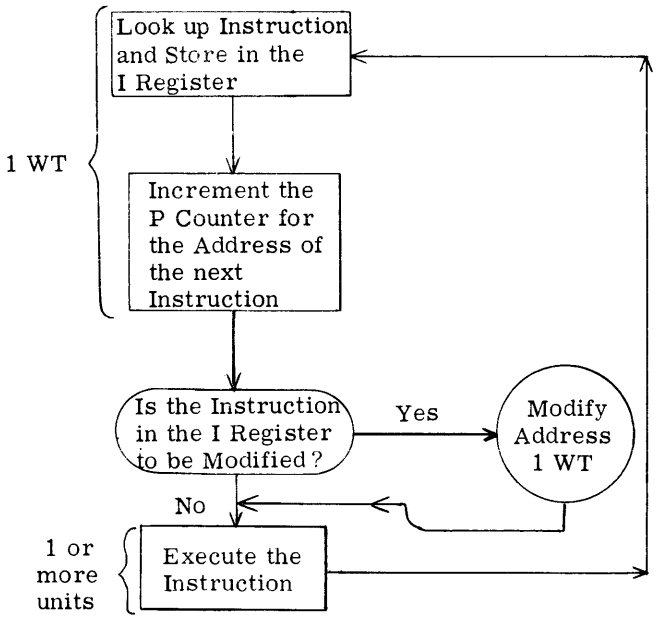


Figure 3-7. GE-225 Instruction-Execution Cycle

Normally, the instruction phase of all instructions requires the same amount of time: placing instruction in the I register and incrementing the P counter takes one word time. However, if the instruction is to be modified, an additional word time is required.

EXECUTION PHASE. During the execution phase, the central processor performs the action specified by the operation code. For example, if the instruction is LDA 3200 (load the contents of memory location 3200 into the A register), the operand address in the I register selects the proper control lines through the address decoding network to bring the contents of memory location 3200 into the M register and, through the B register and arithmetic unit, into the A register. Instruction execution can require one or several word times, depending upon the instruction.

The instruction-execution cycle is continuous in normal operation. As soon as the instruction phase is completed, the central processor enters and completes the execution phase, and another instruction phase is initiated. The cycle is automatic as long as power is applied to the system.

Operation Cycle, Detail

Three different kinds of memory access are required to execute GE-225 instructions: one requires access to memory under control of the P counter, another involves control by an X register, and the third type of access is controlled by the I register. The type of access permitted during any word time is governed by one of three flip-flop circuits as set by control logic:

1. AMP - A flip-flop in the sequence controller that is used to Address Memory from the P counter.

2. AMX - A flip-flop in the sequence controller that is used to Address Memory from one of the X registers.

3. AMI - A flip-flop in the sequence controller that is used to Address Memory from the I register.

Figure 3-8 is a flow chart depicting the operations performed by the central processor while executing a program. This diagram illustrates the nature of the operations and tests performed during one complete instruction cycle, including: 1) extraction of the instruction from memory (AMP), 2) modification of the address portion of the instruction, if required (AMX), and 3) the subsequent execution of the operation (AMI, GIS, or AMX). GIS is a flip-flop in the sequence controller that controls the execution sequence during all general instructions, hence General Instruction Sequencing, or GIS.

Program execution is accomplished by properly repeating the basic operating cycle until the program has been completely executed. Program execution can be interrupted at any time from the control console, in which event the cycle stops immediately following an AMP operation.

The symbols used in Figure 3-8 require some explanation. Each circle containing alphabetic characters represents an operation requiring one word time. The abbreviations correspond to controlling flip-flops in the instruction sequence control logic. Each smaller circle containing an X indicates that the operation involves memory access during the associated word time.

Note, for a manual start, that the first instruction is assumed already to be in the I register. Upon depression of the Start button, the first action is the stepping of the P counter by one, in preparation for the next sequential instruction.

If the instruction currently in the I register involves an X register, the next operating cycle is an AMX access cycle. Otherwise, the next cycle is either a basic AMI cycle or a general GIS cycle. Format I instructions require one or more AMI cycles for execution. After each AMI cycle, the control logic is interrogated for an end-of-execution condition, which (when detected) turns on the EOO (end of operation) signal.

If the instruction is a general instruction, the next cycles (if any) are one or more GIS cycles (to complete instruction execution) or two AMI cycles (for input-output operations involving the controller selector).

In all cases, completion of instruction execution results in the generation of the EOO signal, which initiates an AMP cycle for reading out the next instruction. Further action at this point is contingent upon the position of two switches on the control console: the Automatic-Manual switch and the Stop on Parity Error switch.

If the Automatic-Manual switch is in the Manual position, the processor halts. Otherwise, processing of the next instruction is initiated, unless the Stop on Parity Error switch is in the Stop position and a parity error has occurred during one or more of the memory access cycles of the previous instruction cycle or the just-completed AMP cycle.

If a processor halt occurs for any reason, the address in the P counter is the address of the instruction that is held in the I register upon completion of the AMP cycle preceding the halt.

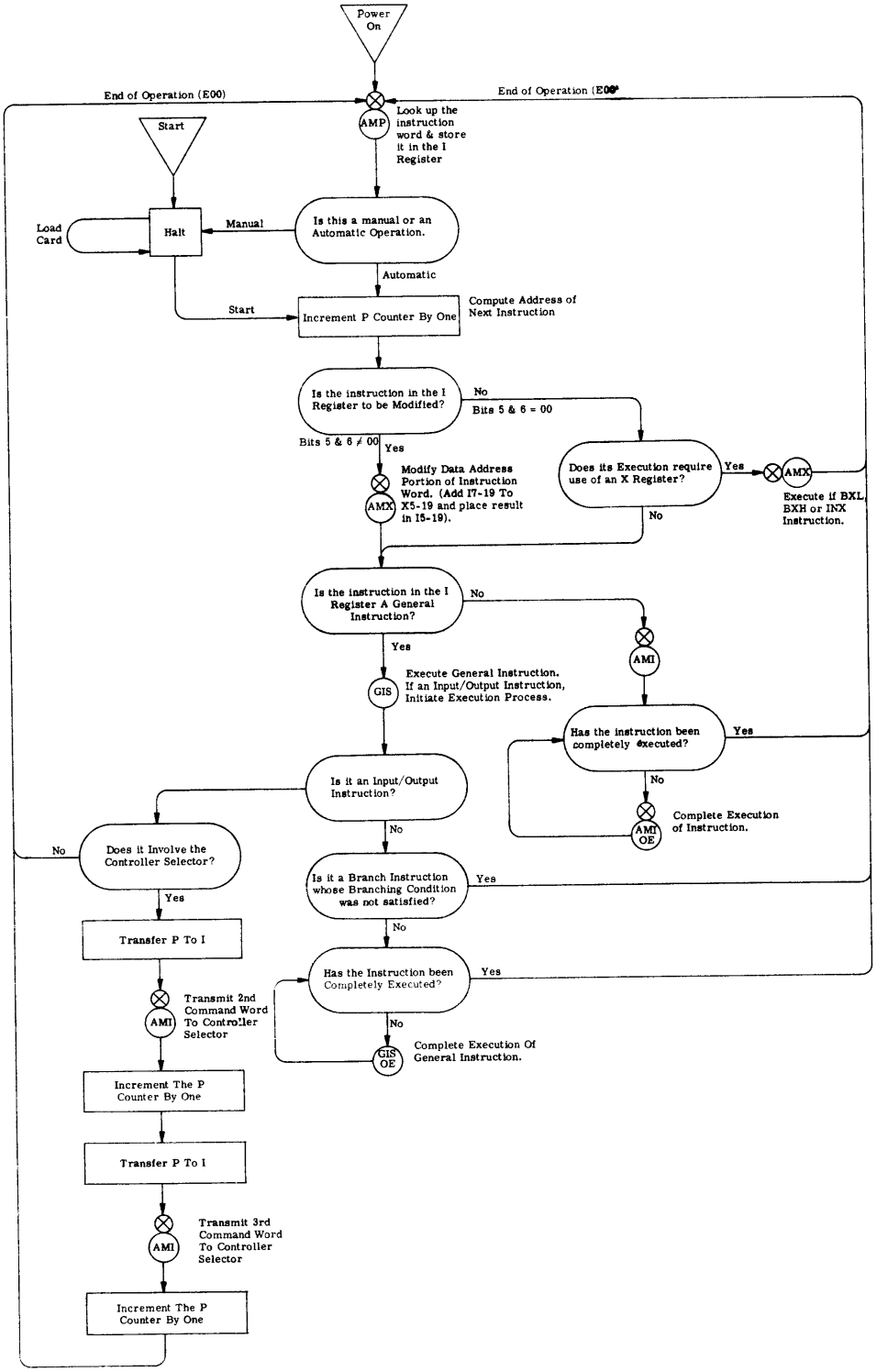


Figure 3 - 8: Flow Chart Showing Central Processor Operating Cycle

GENERAL ASSEMBLY PROGRAM II

GENERAL DESCRIPTION

General Assembly Program II is an assembly routine that provides fast, accurate and documented object programs in absolute or relocatable forms for either magnetic tape, paper tape, or punched card systems. It permits programmer use of convenient pseudo-codes and mnemonics rather than the more difficult machine codes. Additional advantages are obtained by:

1. Address representation by symbols or by decimal numbers.

2. Easily made additions or deletions in coding.

3. Ready detection of certain types of clerical or coding errors, thus saving considerable machine debugging time.

4. A listing of the assembled program plus error indications, symbol listings, and assigned addresses.

Thus, General Assembly Program II is a tool that provides its user with accurate, well-documented programs, which can be quickly debugged and placed in operation, resulting in significant monetary savings. Figure IV-1 is an illustration of a General Assembly Program coding sheet containing representative pseudo-codes, symbolic addresses, and mnemonics.

PROGRAMMER: G E Coder | PROGRAM: Manufacturing Cost - Run #1 | DATE: 1/9/63 | PAGE OF

	Symbol	Opr	Operand	X	REMARKS	Sequence
1		ORG	1000			5
2	ROUND	DEC	5.0		ROUNDING CONSTANT	10
3	RATE	DEC	1025		PROCESS COST PER ITEM	15
4	MASK	OCT	3777640		MASKING CONSTANT FOR MOD ROUTINE	20
5	FIVE	DEC	5			25
6	CON#1	DEC	-34			30
7	START	LDX	ZERO	2	ZERO INDEX REGISTERS 2 & 3	35
8		LDX	ZERO	3		40
9		SPB	CRDIN	1	CARD READ SUBROUTINE	45
10		BRU	CRDEOF		CARD END-OF-FILE RETURN	50
11		SPB	STRIP	1	BCD-BINARY CONVERSION ROUTINE	55
12		DEC	CRD		CARD IMAGE ORIGIN	60
13		DEC	1		BEGINNING OF FIELD	65
14		DEC	4		FIELD SIZE	70
15		STA	AMT#1		ITEMS PREVIOUSLY PROCESSED	75
16		SPB	STRIP	1		80
17		DEC	CRD			85
18		DEC	16			90
19		DEC	4			95
20		STA	AMT#2		ITEMS CURRENTLY PROCESSED	100
21		ADD	AMT#1			105
22		STA	SUM		TOTAL ITEMS PROCESSED	110
23		MAQ				115
24		MPY	RATE		PROCESS COST PER ITEM	120
25		DAD	ROUND		ROUND PROCESS COST	125

Figure IV-1. GE-225 General Assembly Program Coding Sheet

An assembly flow diagram is shown in Figure IV-2. Other assembly configurations are possible.

General Assembly Program II produces an object program by successively processing the source program three times; that is, the source program with the appropriate section of the assembly program is passed through the GE-225 three times. The output produced by each pass forms part of the input for the succeeding pass. General Assembly Program II can provide as its final output an object program on punched cards, binary or octal; a listing, a program tape, punched paper tape, or combinations thereof.

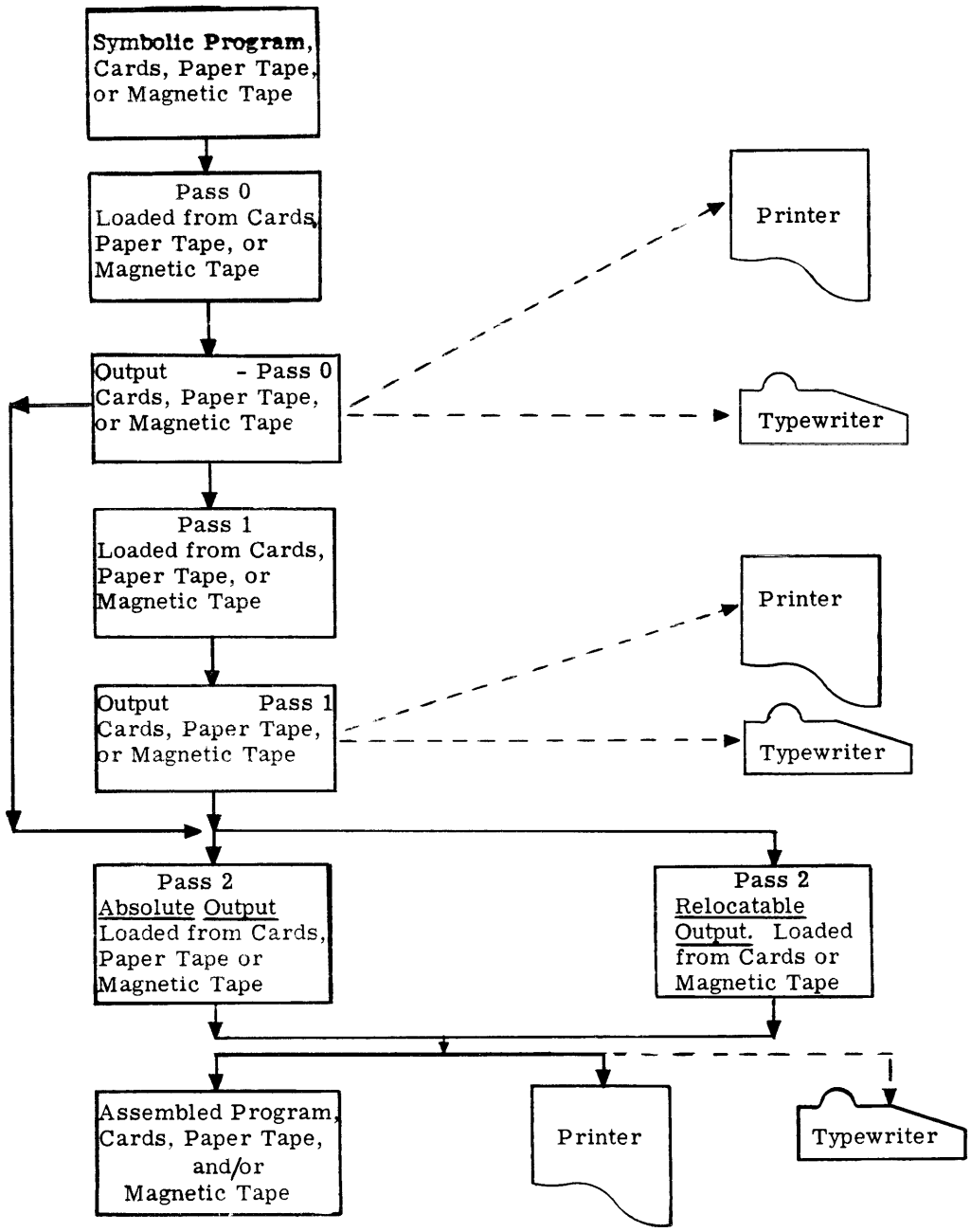


Figure IV-2. Flow Diagram of General Assembly Program II

Each of the three passes of the assembly program perform certain functions which are designed to provide the programmer with maximum information concerning the object program.

Pass 0

Pass 0 accomplishes three operations:

1. It prepares the pseudo-codes for subsequent use in other passes by packing the symbolic input program deck, four (4) instructions to one card as shown by Figure IV-3. This packed deck is produced in the form of punched cards or placed on either paper tape or magnetic tape, depending upon whether a punched card, paper tape or magnetic tape system is used.

2. A table of special symbols is formed containing all symbolic operands appearing in the card input-output, double length, floating point, or document handler instructions. This symbol table, referred to as symbol table 1 (ST1), can be on magnetic tape, paper tape, or punched cards. Figure IV-4 is a symbol table 1 card and a HSP printout of the

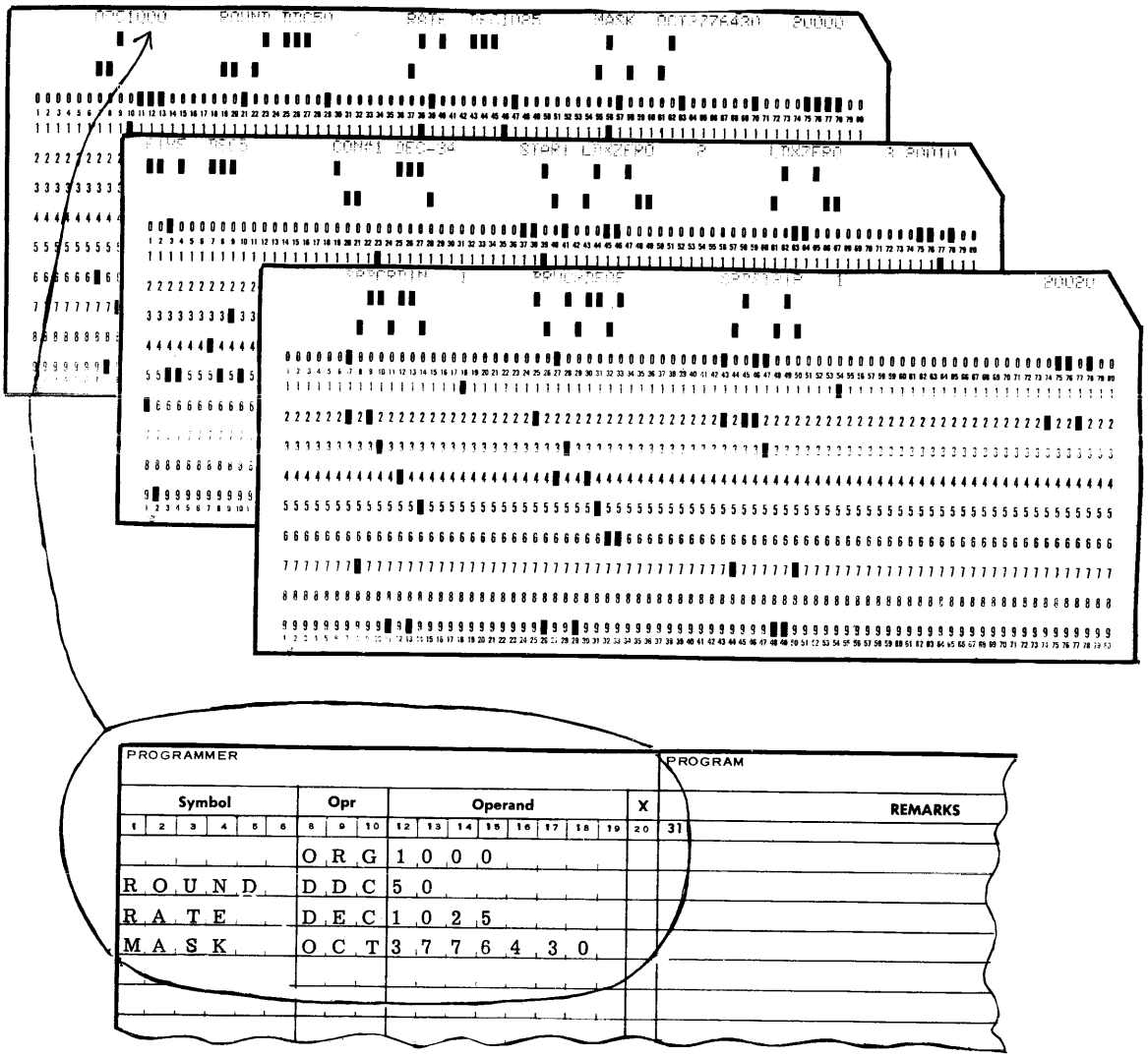
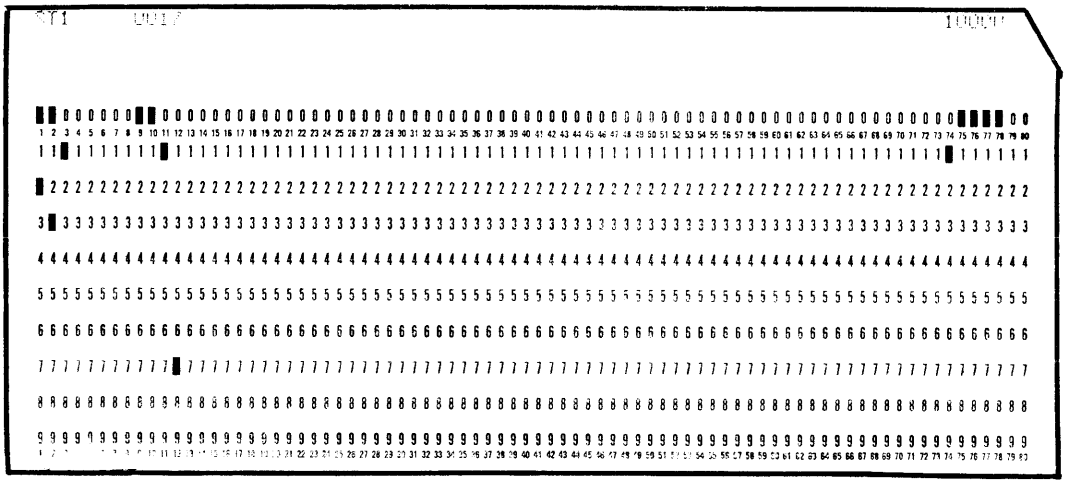


Figure IV-3. Pass 0 Packed Deck Card



ST1 Card

```
ST1     0017
ROUND DL STOR    DL CON     DL  CON1  DL  RND2  DL                    10000
                                                                      10010
```

Symbol Table 1. Listing from High-Speed Printer

Figure IV-4. Special Symbol Table as Produced by a Punched Card System and Listed on the High-Speed Printer by a Magnetic Tape System

table as it appears on tape. Columns 1 through 3 on the symbol card contain the identifying letters ST1, with columns 74 through 78 containing the sequence number of the card in the Pass 0 packed deck.

3. Pass 0 checks all symbolic names in the input deck for no reference, undefined, or multiple-defined usage. If any one of these conditions exist, Pass 0 provides a listing on the HSP or, if desired, on the console typewriter. Figure IV-5 is a sample printout by Pass 0 on the HSP.

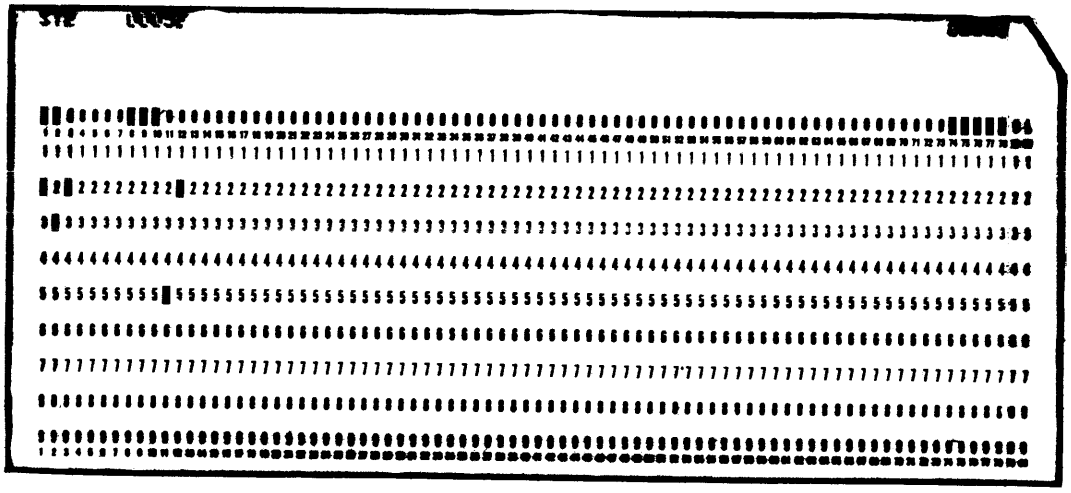
Upon completion of Pass 0, symbol table 1 and the packed instructions are used as input for Pass 1.

Pass 1

Pass 1 uses the output of Pass 0 to assign memory locations to all symbols in the input source deck. It forms a sorted table of these symbols and the numeric values assigned. Symbol Table 2 (ST2) is listed on the HSP, written on magnetic tape, punched on paper tape, or punched into cards. If no printer is available, the typewriter is used. Figure IV-6 shows a HSP listing of Symbol Table 2 and a representative ST2 card.

```
UNDEFINED SYMBOLS
AMT#1     AMT#2      CRDIN      CRDEOF      STRIP      SUM     STOR     ZERO

NO REFERENCE
BEGIN      CON#1      FIVE       MASK       START
     0   ERRORS       TAPE      3
     0   ERRORS       TAPE      4
END OF PASS 0
```

Figure IV-5. Printer Listing of Symbol Errors, Pass 0

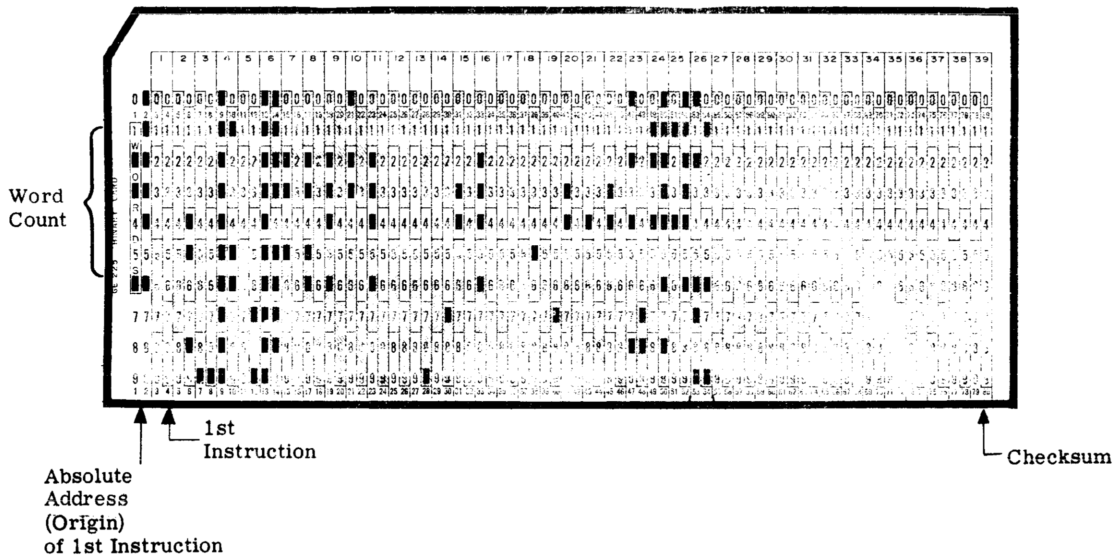


```
ST2        00052
AMT       003733BEGIN 003772CON1   103752CON#1 001755CON 103746FIVE    101754 00010
MASK      001753RATE  001752RND1   003750RND2  103754RND 003726ROUND   101750 00020
SCAL      003720START 001756                                                  00030
END OF PASS 1
```

Figure IV-6 ST2 Card and Printer Listing of Symbol Table 2 of Pass 1

Pass 2

The outputs of Pass 0 and Pass 1 are used for input to the final pass of the general assembly program. Pass 2 does the complete assembly of each instruction as specified by the symbolic input program. The output of Pass 2 is the assembly listing, with indicated errors, and the object program itself, on cards, paper tape, or magnetic tape. Figure IV-7 shows the assembly listing of the instructions shown on the coding sheet of Figure IV-1, and a typical object program binary card. Pass 2 of General Assembly Program II may be relocatable (Pass 2R) or absolute (Pass 2A).




```
            01750               ORG 1000                                               00005
01750     0000000     ROUND     DDC 50               ROUNDING CONSTANT                 00010
01751     0000062
01752     0002001     RATE      DEC 1025             PROCESS COST PER ITEM             00015
01753     3776430     MASK      OCT 3776430          MASKING CONSTANT FOR MOD ROUTINE  00020
01754     0000005     FIVE      DEC 5                                                  00025
01755     3777736     CON#1     DEC -34                                                00030
01756     0640000     START     LDX ZERO      2      ZERO INDEX REGISTERS 2+3          00035
01757     0660000               LDX ZERO      3                                        00040
01760     0720000               SPB CRDIN     1      CARD READ SUBROUTINE              00045
01761     2600000               BRU CRDEOF           CARD END-OF-FILE RETURN           00050
01762     0720000               SPB STRIP     1      BCD-BINARY CONVERSION ROUTINE     00055
01763     0000000               DEC CRD              CARD IMAGE ORIGIN                 00060
01764     0000001               DEC  1               BEGINNING OF FIELD                00065
01765     0000004               DEC  4               FIELD SIZE                        00070
01766     0300000               STA AMT#1            ITEMS PREVIOUSLY PROCESSED        00075
01767     0720000               SPB STRIP     1                                        00080
01770     0000000               DEC CRD                                                00085
01771     0000020               DEC  16                                                00090
01772     0000004               DEC  4                                                 00095
01773     0300000               STA AMT#2            ITEMS CURRENTLY PROCESSED         00100
01774     0100000               ADD AMT#1                                              00105
01775     0300000               STA SUM              TOTAL ITEMS PROCESSED             00110
01776     2504006               MAQ                                                    00115
01777     1501752               MPY RATE             PROCESS COST PER ITEM             00120
02000     1101750               DAD ROUND            ROUND PROCESS COST                00125
```

Figure IV-7. Assembly Listing and Object Program Binary Card (Absolute) from Pass 2

CODING SHEET

The GE-225 General Assembly Program Coding Sheet (Form CK-34) is divided into six fields designated left to right as symbol, operation, operand, X (index), remarks, and sequence. Each of these fields indicate to the assembly program that certain operations are to be performed. A few simple rules are provided to insure that correct results are obtained.

Symbol Field

The symbol field can be a very powerful programming aid where carefully-assigned symbols can supply helpful program information. However, certain rules apply to its proper use. These rules are:

1. Symbols used can vary from one to six characters in length and contain any combination of alphabetics and numerics.

2. Due to relative addressing (described later) the use of plus (+) or minus (-) is not allowed in the symbol field.

3. Any symbol used must contain at least one non-numeric character.

4. Symbols may start at any point within the field because leading and inserted blanks are ignored by the General Assembly Program.

The assembly program assigns any symbol field entry, along with its associated information, a specific memory location. Thus, the programmer need not know the actual computer address, but can refer to the symbolic address when the information is needed. Figure IV-8 shows both proper and improper use of symbols. The symbols shown are for illustrative purposes only.

Entries 8 and 10, CON+1 and A-B4 are improper entries in the symbol field because they violate rule two, above.

Entry 9, 110, is improper because it violates rule three.

Entry 11, A B4, violates rule one pertaining to imbedded blanks. However, the assembly program would assemble such an entry and interpret A B4 as AB4; if the symbol AB4 were defined and used elsewhere in the same program, errors would occur.

Operation Field

The operation field of the coding sheet uses a three-character mnemonic to specify the required function of the line. These mnemonics indicate to the assembly program the type of line; i.e., and instruction, assembly control, or a constant line. The latter two types of entries involve pseudo-instructions which are discussed in detail later in the section.

	Symbol	Opr	Operand	X	REMARKS
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1	CON#1	DEC	5		
2	ZERO	DEC	0		
3	TOTALS	BSS	30		
4	AB4	OCT	2176531		
5	1.10	DEC	110		
6					
7	IMPROPER USE OF SYMBOLS:				
8	CON+1	DEC	5		
9	110	DEC	110		
10	A-B4	OCT	2176531		
11	A B4	DEC	4		
12					
13					
14					

Figure IV-8. Symbol Fields

An instruction line contains a mnemonic indicating the desired computer operation. Usually this line is handled by the assembly program as one computer machine operation for each instruction mnemonic. Typical instruction lines are shown by Figure IV-9.

Symbol	Opr	Operand	X
	LDA	AMT	
	ADD	AMT 2	
	STA	SUM	
	BZE		
	BRU	CHECK	
	DLD	TOTAL	

Figure IV-9. Typical Instruction Lines

An assembly control line is interpreted and used for internal assembly operations and does not become part of the assembled program. However, a control line in certain applications can cause additional words to be reserved in the assembled program. Figure IV-10 illustrates typical control lines.

Symbol	Opr	Operand	X
	ORG	100	
SUM	BSS	20	
CRDIN	EQU	256	

Figure IV-10. Assembly Control Lines

Constant lines indicate to the assembly program the type of constant required by the user. The assembly program then assembles the constant in the correct form. Figure IV-11 contains constant lines.

Symbol	Opr	Operand	X
FIVE	DEC	5	
	DEC	3141 7	
CON#1	DDC	6258 92	
	OCT	3777 7664	
	FDC	1.0B1	

Figure IV-11. Constant Lines

Operand Field

The content of the operand field depends upon the type of line, whether it is instruction, assembly control, or constant. If an instruction line is involved, the operand may be:

1. A mnemonic specifying an operation to be performed by the computer, as shown by line 1 of Figure IV-12.

Symbol	Opr	Operand	X
	BCS	BPN	6
	BRU	*-1	
	LDA	PRINT+40	
	CHS		
	STA	PRINT+40	
	BCN		
	BRU	*-1	
	RCD	0256	
	HCR		
	LDA	SYNC	

Figure IV-12. Coding Sheet Illustrating Operand Use by Instruction Lines

2. A decimal number which is the address portion of a computer instruction. This decimal address will be converted to binary by the General Assembly Program. For an example, see line 8 of Figure IV-12.

3. A symbol representing some address or number within the program. This symbol in conjunction with plus (+) or minus (-) can be used for relative addressing. The combination of symbol and sum (+) or difference (-) must not exceed eight (8) characters. Figure IV-12 contains illustrative examples. The asterisk and relative addressing involving arithmetic expressions can be used to reduce the total number of program symbols used, if necessary.

	Symbol	Opr	Operand	X	REMARKS
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31 75
1		ORG	1000		
2	CARD	BSS	34		
3	STORE	BSS	40		
4	TOTAL	EQU	500		
5		REM			CONSTANTS
6	CONST	DEC	28		
7	CON#1	DDC	628154		
8		OCT	37776636		
9					

Figure IV-13. Examples of Operand Field in Assembly Control and Constant Lines

When an assembly control line is specified, the operand field of the line contains information required by the General Assembly Program. Figure IV-12 contains illustrative examples.

If the operand field is part of a constant line, the operand then must specify the constant.

X Field

The X field (column 20) specifies address modification before execution of the assembled computer instruction, or it may provide additional information to the assembly program, such as plug number or tape handler number. If the X field is part of an instruction line, it may be blank or contain either a number or an alphabetic. No significance is attached to this field by constant lines. Figure IV-14 shows use of the X field.

Opr	Operand	X	
8 9 10	12 13 14 15 16 17 18 19	20	31
BCS	BPN	6	
BRU	* - 1		
LDA	STORE	2	
ADD	SUM		
STA	SUM		
LAQ		A	

Figure IV-14. X Field Examples

Remarks Field

The remarks field is a helpful programming aid in that it can be used by the coder to explain or describe the actions of each program line. The remarks field is handled by the assembly program and does not require memory locations within the assembled program, nor does it affect the assembly process. The remarks field should be used extensively by programmers for adequate documentation. Refer to Figure IV-15 for examples.

Sequence

The sequence field specifies the order of the lines to be assembled. This is strictly a programmer's convenience and is checked only by General Assembly Program II when specifically requested by use of an SEQ pseudo-instruction. Cards for the source program should be sequenced to prevent accidental mixing going unnoticed. Normally, the General Assembly Program does not check the card sequence; it does show the sequence on the object program listing. Figure IV-15 shows sequenced lines.

A symbolic program written for General Assembly Program II has both pseudo-and machine instructions. Pseudo-instructions are symbols representing information needed by the assembly program for proper assembly. These instructions, along with the machine

Opr	Operand	X	REMARKS	Sequence
8 9 10	12 13 14 15 16 17 18 19	20	31 75	76 77 78 79 80
REM			SUBROUTINE TO CHECK VOID DATE	5
DLD	VOID		VOID DATE DAY/MONTH/YEAR	10
SUB	CON#1		VOID CONSTANT	15
BNZ	OK		DATA NOT VOID SO PROCESS	20
SPB	CLOSE	1	CLOSE FILE DATA VOID	25

Figure IV-15. Remarks and Sequence Entries

CONTENTS OF OPR FIELD	TYPE OF ASSEMBLED INFORMATION	NUMBER OF CHARACTERS SPECIFIED
ALF/NAL	One BCD Word	3 Alphanumeric
MAL	One to fifteen consecutive BCD Words	3 Alphanumeric per word
PAL	One to fifteen consecutive BCD Words with sign bit on in last words	3 Alphanumeric per word
DEC	One Fixed Point Binary Number	
DDC	One Double Length (2 word) Fixed Point Binary Number	
FDC	One Floating Point (2 word) Binary Number	
OCT	One Binary Word	Up to 7 Octal Characters
Z(XX)	One Binary Word	Up to 7 Octal Characters

Table 1. Pseudo-Instructions for Constant Lines

instructions, are included in the object program listing. Normally, pseudo-instructions are not executed by the computer, but are used to generate constants, control the assembly, and provide information on the program listing.

Pseudo-Instructions Used for Constant Lines

ALF — ALPHANUMERIC

This instruction causes alphanumeric constants (if three alphabetic or numeric characters) to be entered in the object program. The first three characters in the operand are converted to BCD and placed in a memory location determined by the assembly program. Blanks or spaces, where desired, must be indicated. Only columns 12, 13, and 14 of the operand field can contain the data for the instruction.

Example: The words, "PLANT CODE NOT IN TABLE", comprise a program typewriter message and may be entered in the object or assembled program by using the ALF instruction.

Symbol	Opr	Operand	X	
				Appears in Memory
TYPE	ALF	PLA		0474321
	ALF	NT		0456360
	ALF	COD		0234624
	ALF	E N		0256045
	ALF	OT		0466360
	ALF	IN		0314560
	ALF	TAB		0632122
	ALF	LE		0432560

Note that spaces are indicated by leaving the column blank, which results in an octal 60 being placed in the memory location by the assembly program. If the desired data is not left-justified, starting with column 12, incorrect constants will invariably result.

Example:

	Symbol	Opr	Operand	X	
1					Appears in
2					Memory
3		ALF	PLA		0604743
4		ALF	NT		0606045
5					

The constant in line three (3) results in the loss of the character A, and line four (4) contains two blanks (octal 60) and the character N only, losing the character T.

It should also be noted that an ALF command is required for each line containing the desired alphanumeric constant.

NAL — NEGATIVE ALPHANUMERIC

This pseudo-instruction is used to enter the 2's complement of an alphanumeric constant in the object program. The assembly program applies the same requirements to this instruction as it does to ALF.

Example: The 2's complement of the codes A14, AB2, ABF are to be placed in the object program.

Symbol	Opr	Operand	X	
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
				Appears in Memory
C O D E	N A L	A 1 4		3567674
	N A L	A B 2		3565576
	N A L	A B F		3565552

MAL MULTIPLE ALPHANUMERIC

This pseudo-instruction will enter alphanumeric data into as many as fifteen consecutive memory locations. The number of words to be filled must be specified by a numeric in columns 12 and 13 and the data in the remarks field.

Example:

Symbol	Opr	Operand	X	
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
T Y P E	M A L	8		PLANT CODE NOT IN TABLE

This data is placed in memory as follows:

Data		Memory
PLA	→	0474321
NT	→	0456360
COD	→	0234624
E N	→	0256045
OT	→	0466360
IN	→	0314560
TAB	→	0632122
LE	→	0432560

PAL MULTIPLE ALPHANUMERIC FOR PRINTER WITH PRINT LINE INDICATOR

This pseudo-instruction is similar to the MAL instruction with the exception of entering a minus sign in the last word of the alphanumeric data. (The minus sign is primarily used for control purposes during high-speed printer operation.)

Example:

Symbol	Opr	Operand	X	
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
	P A L	4		PLANT CODE

The data enters memory as shown below. Note that memory word four (4) of the data contains a one (1) in the sign bit or zero (0) position of the word.

	Data	Memory
1.	PLA	0474321
2.	NT	0456360
3.	COD	0234624
4.	E	2256060

DEC DECIMAL

This instruction places the binary equivalent of a decimal constant in the object program. The constant is assigned a memory location as determined by the assembly program. The operand portion of the constant can be symbolic or decimal. If symbolic, at least one character must be used other than 0 through 9, plus (+), minus (-), decimal point (.), B, or E. If no sign is present, the number is assumed to be plus (+). A minus sign, specifying a negative number, results in the 2's complement of the number being placed in memory.

Examples of plus numbers:

Opr	Operand	X	
8 9 10	12 13 14 15 16 17 18 19	20	31
			Appears in Memory
D E C	5		0000005
D E C	1 2 8		0000200
D E C	7 3 7 3 8		0220012
D E C	9 2 8		0001640
D E C	1 2		0000014
D E C	+ 1 7 5 0		0003326

Examples of minus numbers:

Opr	Operand	X	
8 9 10	12 13 14 15 16 17 18 19	20	31
			Appears in Memory
D E C	- 5		3777773
D E C	- 1 2 8		3777600
D E C	- 7 3 7 3 8		3557766
D E C	- 6 3 9 7 8		3603026
D E C	- 1		3777777

The character B can be used to specify a binary scale for either plus (+) or minus (-) numbers. The number following B is used to position the binary point for the decimal constant preceding the B in the operand field. If no scale is specified, the assembly program assumes a binary scale of 19.

Examples using the B character:

Opr	Operand	X	
8 9 10	12 13 14 15 16 17 18 19	20	31
			Appears in
			Memory
DEC	5B16		0000050
DEC	-5B16		3777730
DEC	7377 38B18		0440024
DEC	-4B3		3000000
DEC	+4B3		1000000

The characters . and E can be used to specify decimal scales, or decimal exponents. Normally, the . indicates a mixed number, while E specifies leading or trailing zeros. For example, E-2 indicates leading zeros and E2 indicates trailing zeros. If the character B is not used with . or E, only the integral portion of the converted number will be used. If the number of characters used to specify a decimal constant exceeds eight, the operation field of the next line is left blank and the constant is continued in the operand field, using two lines for one entry. Only one binary scale and one decimal scale can be indicated for a single decimal constant.

Examples using the characters . and E:

Opr	Operand	X	
8 9 10	12 13 14 15 16 17 18 19	20	31
			Appears in
			Memory
DEC	5.52		0000005
DEC	5.52B18		0000013
DEC	.52B18		0000001
DEC	5.5B18E2		0002114
DEC	.73738E2		0000000
	B18		
DEC	-.56251B1		
	0		3777340
DEC	-.18751B1		3777640
	0		
DEC	.32170B0		0511327

DDC DOUBLE LENGTH DECIMAL

DDC, like DEC, is used to enter a decimal constant in the object program. This constant is assigned two sequential memory locations starting with the first even-numbered location available. If no binary scale is specified, the assembly program assumes a binary scale of 38.

Example of DDC:

Opr	Operand	X	
8 9 10	12 13 14 15 16 17 18 19	20	31
			Appears in
			Memory
DDC	12		0000000
			0000014
DDC	786432		0000001
			1000000
DDC	-24		3777777
			3777750
DDC	.0796896		0024315
	7928B2		0127545
DDC	24B36		0000000
			0000140
DDC	1.570796		0622077
	3184 7B2		0651010
DDC	1018B0E-		0000000
	10		0066516
DDC	1525270B0		0000007
	E-10		1774566

FDC FLOATING POINT DECIMAL

This instruction is used to enter a floating point decimal constant in the object program. The use of FDC is the same as the DDC, i.e., two sequential memory locations, starting with an even-numbered location, are assigned by the general assembly. After conversion, the constant is in normalized form, if the specified binary scale is minimum; otherwise the constant is unnormalized. If no binary scale is specified, the assembly program determines the binary scale and a normalized floating-point number results.

Examples of FDC:

Opr	Operand	X	
8 9 10	12 13 14 15 16 17 18 19	20	31
			Appears in
			Memory
FDC	1B1		0006000
			0000000
FDC	1.44269 5		0006705
	041B1		0507312
FDC	3.3219284		0013244
	094B2		1517022
FDC	10B4		0022400
			0000000
FDC	1F40		Both give
	and		same results:
FDC	1E40B133		1027530
			0430221
			Result unnor-
			malized
FDC	1E40B134		1031654
			0214110

OCT — OCTAL

The entering of octal constants in the object program is accomplished with the OCT pseudo-instruction. The octal number specified in the operand is right-justified and assigned one memory location designated by the assembly program. Leading zeros in the operand field are ignored. A leading minus (-) in the operand sets the sign bit of the constant to one (1). Octal constants are used primarily for establishing particular bit configurations in memory.

Example of OCT:

Opr	Operand	X	
8 9 10	12 13 14 15 16 17 18 19	20	31
			Appears in
			Memory
OCT	27777766		2777766
OCT	377		0000377
OCT	-377		2000377
OCT	27600		0027600
OCT	37777746		3777746
OCT	-17777746		3777746

Z — OCTAL OPERATION CODE

The Z pseudo-instruction is used to set the operation bits of the assembled instruction to any desired configuration. The operand can be decimal or symbolic, and indexing is optional. In use, a Z is placed in column 8 with the two octal digits desired as an operation code in columns 9 and 10.

Sample Coding:

Opr	Operand	X	REMARKS
8 9 10	12 13 14 15 16 17 18 19	20	31 75
ZOO	TEMP	2	EQUIVALENT TO LDA TEMP 2
ZO4	-100	2	EQUIVALENT TO BXL 100 2
Z20	0		EQUIVALENT TO WORD 1 OF RWD

Listing:

Memory Location (Octal)	Memory Contents (Octal)	GAP
01764	0000000	TEMP DDC 0
01765	0000000	
((	((	((
0770	0041764	ZOO TEMP 2
0771	0457634	ZO4 -100 2
0772	2000000	Z20 0

Control Line Pseudo-Instructions

Assembly control instructions provide information which controls the internal operations of the general assembly program, although this information does not become part of the assembled program. In certain cases, additional words may be added to the assembled program. Table 2 is a list of control instructions.

Symbol	Opr	Operand
	ORG	Decimal or Symbolic
	LOC	Octal
Optional	EQU	Decimal or Symbolic
Optional	EQO	Octal
Optional	BSS	Decimal or Symbolic
	SBR	Symbol of Subroutine
	TCD	Decimal or Symbolic
	END	Decimal or Symbolic

Table 2. Assembly Control Pseudo-Instructions

SBR SUBROUTINE CALL

This pseudo-instruction can only be used if the assembly is called from the General Assembly Program II master tape. SBR is used to instruct Pass 0 to obtain the specified subroutine from the General Assembly Program II master tape. An error indication results if the specified routine is not present. (See section Addition of Symbolic Routines for detailed information about error indication.)

The subroutine calls are saved until the user's END card is encountered. The specified subroutines are then placed behind the user's coding. Note: the subroutines are assigned memory following the last instruction of the user's coding. If the user desires, the subroutines can be placed in reserved memory locations. See examples below.

Example:

Opr	Operand	X	REMARKS
SBR	STRIP		BINARY - BCD CONVERSION ROUTINE
END	START		

Listing:

```
                02623          SBR STRIP
                               REM
02623   2514003 STRIP  BOV
02624   0000000 =1102  LDA    0
02625   0000000        LDA    1
```

This subroutine can also be called in the following manner:

Coding:

Symbol	Opr	Operand	X
	LDA	TEMP	
	ADD	TOTAL	
	( )		
	SPBNPRIBD	1	
NPRIBDSBR			
	( )		
ZERO	DDCO		
	END	START	

In both of these examples, the STRIP and NPRIBD subroutines are assigned memory locations following the last instruction (ZERO DDCO) of the coding.

If the programmer desires to place the subroutine in a specific location other than at the end of his program, he must reserve a sufficient number of words in memory by using a BSS instruction and by placing an ORG card with the origin of the reserved area immediately preceding the END card.

Example:

Symbol	Opr	Operand	X	REMARKS
SUBR#1	BSS	76		RESERVE 76 LOCATIONS FOR ROUTINE
	LDA	TEMP		
	( )			
	SPBNPRIBD	1		
	SBRNPRIBD			
	( )			
	BRU	*		
	ORG	SUBR#1		
	END	START		

The NPRIBD subroutine is listed at the end of the object program, but the ORG SUBR#1 instruction causes General Assembly Program II to assign memory locations starting at SUBR#1 rather than locations following the BRU*.

ORG ORIGIN

This pseudo-instruction controls the memory assignments performed by the general assembly program. When an ORG instruction is encountered, the assembly program uses the contents of the operand field to reset an internal counter in the assembly program referred to as the Memory Allocation Register (MAR). If the operand is decimal, it is converted to binary before being used. If symbolic, the operand must be predefined before being used. The general assembly program ignores all but the operand field on an ORG instruction. When no ORG is used, the assembly program assigns an origin of 00000.

Examples of ORG Pseudo-Instructions:

Opr	Operand	X	REMARKS
ORG	128		THE MAR IS SET TO 010000000 (200) AND NEXT INSTRUCTION STARTS AT AN OCTAL 200

Opr	Operand	X	REMARKS
ORG	BEGIN		THE SYMBOL "BEGIN" MUST BE PREDEFINED AND THE MAR IS SET TO THE ASSIGNED VALUE IF BEGIN IS NOT PREDEFINED THE MAR IS SET TO ZERO

Example: Use an ORG to assemble an object program at memory location 1000 (decimal).

Symbol	Opr	Operand	X
	ORG	1000	
TWO	DEC	2	
TEN	DEC	10	

LOC — LOCATION IN OCTAL

This operation performs the same functions as an ORG; however, the contents of the operand field must be an octal number. The assembly program will ignore leading zeros.

Example of LOC:

Symbol	Opr	Operand	REMARKS
	LOC	1750	MAR IS SET TO 001111101000
TWO	DEC	2	
TEN	DEC	10	

EQU — EQUALS

This instruction can be used to over-rule the normal memory assignment by the assembly program. The operand (decimal or symbolic) indicates the specific memory location to be used. This instruction does not affect the memory allocation register, thus it may be used as often as necessary and at any point within the source or symbolic program without disturbing the memory assignment sequence. If the operand is symbolic, the symbol must be predefined. A decimal operand is converted to binary before being utilized.

Example of EQU:

Symbol	Opr	Operand	REMARKS
CRD	EQU	256	
AREA	EQU	CRD	CRD MUST HAVE BEEN PREDEFINED
AREA2	EQU	CRD+40	

EQO — EQUALS OCTAL

The EQO instruction is the same as the EQU except that the operand field must be in octal form. Leading zeros in the operand field are ignored.

Examples of EQO:

Symbol	Opr	Operand	REMARKS
CRD	EQO	400	
AREA	EQU	CRD	CRD MUST HAVE BEEN PREDEFINED

BSS — BLOCK STARTED BY SYMBOL

A BSS causes the assembly program to increase the Memory Allocation Register (MAR) by the number in the operand field. This instruction is used to reserve a block of memory locations in the object program. The operand field may be decimal or symbolic. If symbolic, the symbol used must be predefined; if decimal, the operand is converted to binary before use. The BSS can be used as often as needed.

Examples of BSS:

Symbol	Opr	Operand	X	REMARKS
	ORG	256		
CRD	BSS	28		MAR IS INCREASED BY 28
PRINT	BSS	40		MAR IS INCREASED BY 40
INDEX	BSS	3		MAR IS INCREASED BY 3
STORE	BSS	SAVE		SAVE MUST BE PREDEFINED

The BSS instruction of Line 2 of the example will reserve 28 consecutive memory locations starting at location 256. The other BSS commands reserve additional blocks of memory.

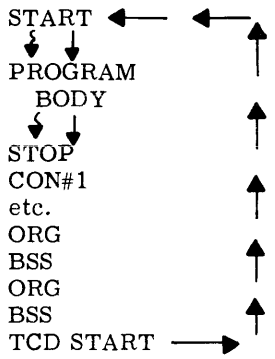
TCD — PUNCH TRANSFER CARD

A TCD generates an instruction that transfers control to the location specified by the operand field at execution. In the relocatable format this is a type 5 card. The operand may be decimal or symbolic. A TCD may be used as often as necessary in a source program because this instruction does not affect the memory allocation register. A symbol in the operand must be predefined.

Examples of TCD:

Symbol	Opr	Operand	X	REMARKS
	TCD	2900		
	TCD	START#1		START #1 MUST BE PREDEFINED

The transfer card is the last card of the object program; when the program is loaded for execution, the transfer card directs the central processor to the location of the initial program instruction:



END — END OF PROGRAM

This pseudo-instruction causes the assembly program to generate a transfer card to transfer control to the initial program location (specified in the operand field) when the object program is loaded into memory for execution. In the relocatable format, this is a type 3 card. The operand field may be decimal or symbolic; if symbolic, the symbol must be predefined. The END instruction indicates the end-of-program and terminates assembly. It must be used only once and must be the last instruction of the source program.

Failure to use the END instruction results in a typewriter message so indicating. Assembly continues, but no transfer card is generated. Thus, this instruction should appear in the program.

Examples of END Instructions:

Opr	Operand	X	REMARKS
END	1000		
END	START		START MUST BE PREDEFINED
END	CONST + 3		CONST MUST BE PREDEFINED

The END instruction of Line 1 would result in the punching of a transfer card as shown in Figure 16.

REM — REMARKS

The REM mnemonic in the operation field is used to provide additional information on the object program listing as given in the remarks and sequence fields columns 31 through 80. All other fields in the instruction line are ignored.

Example of REM:

Opr	Operand	X	REMARKS	Sequence
REM			SUBROUTINE TO CHECK VOID DATE	V.0.05

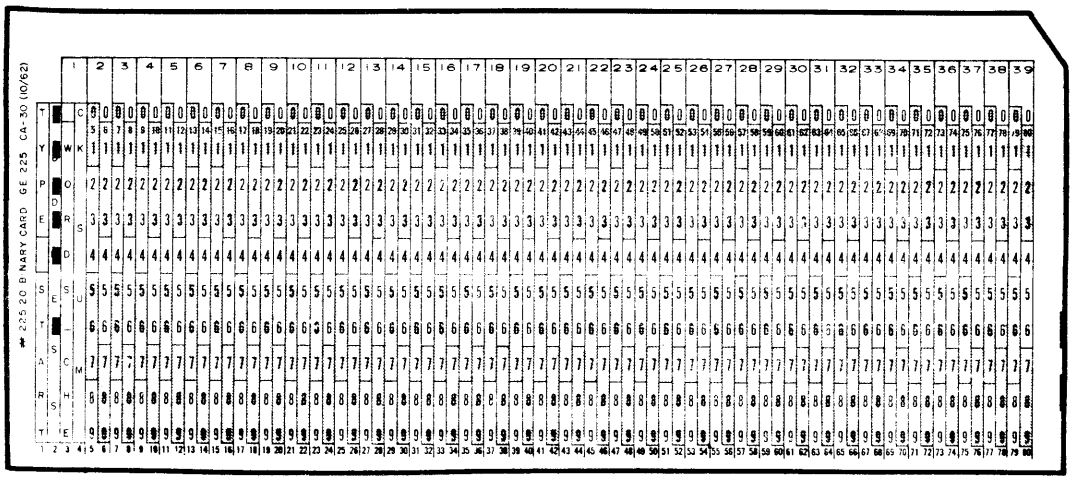


Figure IV-16. Transfer Card Generated by END Instruction

ADDITIONAL PSEUDO-INSTRUCTIONS

General Assembly Program II provides additional pseudo-instructions:

EJT — EJECT PRINTER PAPER

Normally the assembly program prints 54 lines per page and then ejects to the top of the next page. When the EJT command is encountered, the line count is reset and the paper is immediately ejected to the top of the next page.

Coding:

Opr	Operand	X	REMARKS
DEC	6315		
REM			END ADJUSTMENT CONSTANTS
EJT			CONTINUE LISTING ON NEXT PAGE

SEQ — CHECK SOURCE PROGRAM CARD SEQUENCE NUMBERS

Except for listing, sequence numbers are normally ignored. However, if the SEQ command is used, the assembly program checks to see that the card sequence numbers are in ascending order. Blanks are ignored. Numbers less than, or equal to, the preceding number are errors and are flagged with an S in the right margin of the listing.

Coding:

Opr	Operand	X	REMARKS	
SEQ			CHECK SEQUENCE NOS.	05
LDX	ZERO	2		10
DLD	TEMP	2		15

NAM — PRINT NAME OR TITLE ON EACH PAGE

A page number is printed at the top of each page of listing. When NAM is used with a name or title in the operand, this name is also printed at the top of each page. The name may be changed by issuing a new NAM instruction.

Coding:

Symbol	Opr	Operand	X	REMA
	NAMEDIT#3			ROUTINE TITLE
	ORG	1000		
ZERO	DEC	0		

NLS — NO LIST

General Assembly Program II normally lists the object program on the high-speed printer. The NLS pseudo-instruction may be used at any point in the source program to inhibit the printer listing.

LST — LIST

Where the assembled program listing has been stopped by an NLS, the listing can be resumed with an LST instruction.

Example:

Symbol	Opr	Operand	X	REMA
	NLS			ELIMINATE LISTING OF
PUN	DDC	0		NPRIBD SUBROUTINE
PUN1	BSS	4		
PUN2	BSS	2		
	⋮			
NPRIBD	EQU	PUN+12		
	LST			RESUME LISTING
TEMP	BSS	20		

RELATIVE ADDRESSING

The General Assembly Program provides the facility for assigning addresses relative to some starting point or some symbolic memory location.

This is termed relative addressing and is a useful programming aid. Relative addressing can be accomplished in several ways. Some examples are shown below.

Example 1:

Coding:

Symbol	Opr	Operand	X	REMARKS
A M T	E Q U	2 0 0		THIS INSTRUCTION ASSIGNS THE SYMBOL AMT
				TO MEMORY LOCATION 200
	L D A	A M T + 6		THIS INSTRUCTION ILLUSTRATES RELATIVE
				ADDRESSING . + 6 REFERS TO MEMORY
				LOCATION 206
	L D A	A M T - 2		AMT - 2 IS MEMORY LOCATION 198

Since the general assembly program has relative addressing capabilities, it will assign the correct memory addresses to AMT + 6 and AMT - 2.

Example 2: The starting locations of sections within a program can be determined by relative addressing ORG instructions.

Coding:

Symbol	Opr	Operand	X	REMARKS
C O N	E Q U	5 0 0		
	O R G	C O N		THIS ORG WILL START AT MEMORY LOCATION
				500 AS ESTABLISHED BY USE OF EQU FOR THE
				SYMBOL
	O R G	C O N + 5 0		STARTING MEMORY LOCATION HERE IS 550
				DUE TO RELATIVE ADDRESSING

A very convenient method of relative addressing that is widely used and which can greatly reduce the number of symbols required is the use of the asterisk (*) character. An asterisk in the operand field of an instruction is interpreted by the assembly program as the address of the instruction itself.

Example of Use of Asterisk:

Opr	Operand	X	REMARKS	Sequence
B C N			IF THESE INSTRUCTIONS START AT MEMORY	
B R U	* - 1		LOCATION 1000, THE ASTERISK IN THE OPERAND	
R C D	C R D		FIELD OF THE SECOND INSTRUCTION IS INTER-	
H C R			PRETED BY GAP AS BEING 1001. THE ADDRESS	
L D A	* + 6		IN THE OPERAND FIELD IS 1001 - 1 or 1000.	
B M I				
B R U	*		THE INSTRUCTION ON LINE 7 CAUSES THE COM-	
S U B	C O N 4		PUTER TO ENTER A CONTINUOUS LOOP SINCE	
B R U	* + 8		THE ASTERISK IS INTERPRETED BY GAP AS THE	
			ADDRESS OF THE INSTRUCTION ITSELF. THE	
			MACHINE THEN EXECUTES THE SAME INSTRUC-	
			TION CONTINUOUSLY.	

The use of the asterisk in the above example (line 7) is equivalent to writing the same symbol in both the symbol and operand fields of the same line.

DETECTED CODING ERRORS

As an aid to the programmer, the general assembly program detects certain types of coding errors and lists them on either the typewriter or high-speed printer.

Pass 0 provides a listing of undefined symbols, multiple symbols, no reference, and, if the tape assembly program is used, an indication of any tape errors. Figure 17 is a sample print-out from Pass 0.

Pass 1 provides a multiple symbol list that gives the memory locations referring to the symbol. This print-out can be done on the high-speed printer or, if no printer is available, the console typewriter. A print-out from the HSP appears as shown:

```
MULTIPLE SYMBOLS
SUM        01760      01756
END OF PASS 1
```

Pass 2 lists the assembled program along with codes which indicate an error or a suspected error in the program coding. Six symbols (O, U, M, A, T, and S) are used as error codes. These error codes are printed to the left of each line on which they occur. A brief description of these codes follows:

Code	Meaning
O	Illegal Mnemonic Operation. This becomes an LDA (00).

Example: Print-out from Pass 2

```
0    01752    0001757    LDB FIG
```

The OPR field contains an illegal operation.

Code	Meaning
U	Undefined Symbol. A symbolic name appearing in the operand field does not appear in the symbol field of any instruction or constant line. 0000 is used as an operand address.

Example: Print-out from Pass 2

```
U    0175    0000000    LDA AMT
```

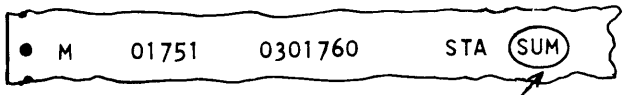
The symbol AMT is not defined in the symbol field.

```
UNDEFINED SYMBOLS
AMT       OUT       SAVE      STRIP

MULTIPLE SYMBOLS
SUM

NO REFERENCE
#3        #4        #5        #6        #7        #8        #9        #10
#11       #12       #13       #14       #15       #16       #17       #18
#19       #20       #21       #22       #23       #24       START
     0 ERRORS    TAPE    3
     0 ERRORS    TAPE    4
END OF PASS  0
```

Figure IV-17. High-Speed Printer Listing from Pass 0

Code	Meaning
M	**Multiple** -Defined Symbol. Either the **symbol** field or the operand field **contains** a symbolic name which appears in the symbol field of two different instruction lines. If the error detected was in the symbol field, assembly will continue with the present setting of the memory allocation register. If the error detected was in the operand field, the value assigned to the symbol the last time it appeared will be used as the operand address in the assembled instruction.

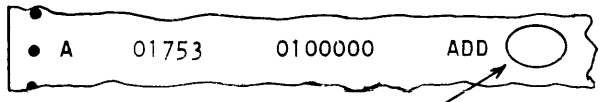
Example: From Pass 2



The symbol SUM is **used** more than once in **the symbol field.**

Code	Meaning
A	Error or Suspected Error in the Operand Address. Blank operand field in a line normally requiring an address. An entry in the operand field of a line which normally should be blank. The numeric value of the operand does not meet the requirements of the line in which it was used. The value of the operand address will be logically OR-ed into the instruction.

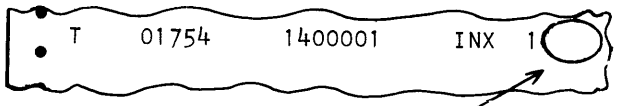
Example: From Pass 2



Address left blank. This is a possible error because the address may be added **later in the program.**

Code	Meaning
T	Error or Suspected Error in X-Field. The X-Field is blank in a line normally requiring an entry. The X-Field contains an entry in an instruction line which may be altered by address modification. The numeric value of the entry in the X-Field violates the requirements of the line in which it appears.

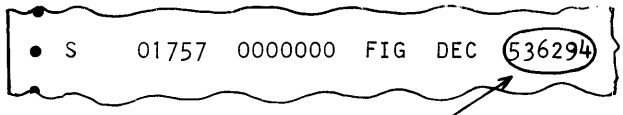
Example: From Pass 2



The X-field is blank on an instruction which requires an entry.

Code	Meaning
S	Scale Factors in DEC, DDC, FDC. The specified binary and decimal scales are incompatible. Two decimal or binary scales have been specified in the constant line.

Example: From Pass 2



The constant in the operand field is larger than the scale which is assumed to be binary 19 (B19). This constant must be double length.

ASSEMBLY OPERATION

General Assembly Program II consists of four separate programs: Pass 0, Pass 1, Pass 2A (absolute), and Pass 2R (relocatable). Operating instructions for General Assembly Program II depend upon whether a card, paper tape, or magnetic tape system is used for assembly and upon the configuration of the object system.

A flow diagram of the four programs is shown in figure 18. The specifications of the input/output media can be changed during assembly, as long as the output from one pass is acceptable as input to the subsequent passes. Thus, the operating instructions and console switch settings vary with different hardware configurations.

System Configurations

The minimum hardware requirements for the operation of the General Assembly Program II are:

A. System 1 - Punched Card

1. Card Reader
2. Card Punch
3. Typewriter
4. 4096 words of memory

B. System 2 - Punched Paper Tape

1. Card Reader
2. Paper Tape Reader
3. Paper Tape Punch
4. Typewriter
5. 4096 words of memory

C. System 3 - Magnetic Tape or Punched Cards

The hardware requirements for the operation of General Assembly Program II are:

1. Card Reader
2. Card Punch
3. Magnetic Tape Subsystem and 4 Magnetic Tape Handlers (5 Handlers for complete tape operation)
4. High-Speed Printer
5. Typewriter
6. 8192 words of memory (Minimum for Relocatable General Assembly Program II)

Regardless of the hardware configuration used during assembly of the source program General Assembly Program II will assemble object programs for any hardware configuration.

Options & Console Switches

The console switches are used to indicate the peripheral configuration available to the General Assembly Program II program as well as other modifications that may be employed while assembling. The setting of the console switches may be altered between passes, but care must be exercised to maintain peripheral compatibility between passes. For example, it is possible to specify magnetic tape, card, or paper tape input and magnetic tape output to Pass 0. At the end of Pass 0, the switches must be altered to specify magnetic tape input/output to Pass 1. At the end of Pass 1, the switches may be altered to specify magnetic tape input and paper tape output to Pass 2. Figure 18 is a flow diagram of the input/output relationships.

Console Switch Settings

Switch 1

Normal:	Absolute output.
Down:	Relocatable output.

Switch 2

Normal:	Printer is on-line.
Down:	No on-line printer. An octal deck is punched instead of binary cards.

Switch 3

Normal:	Tape 3 is used to obtain comments on the Pass 2 program listing.
Down:	Tape 3 is not available, and no comments will appear on the listing. (If switch 4 is down, switch 3 is ignored.)

Switch 4

Normal:	Tape 4 is used as output by Pass 0 and input by Pass 1 and Pass 2. Tape 5 is used as output by Pass 1 and input/output by Pass 2.
Down:	No magnetic tapes available, all input/output via cards or paper tape (switch 4 overrides switches 3 and 6).

Switch 5

Not used.

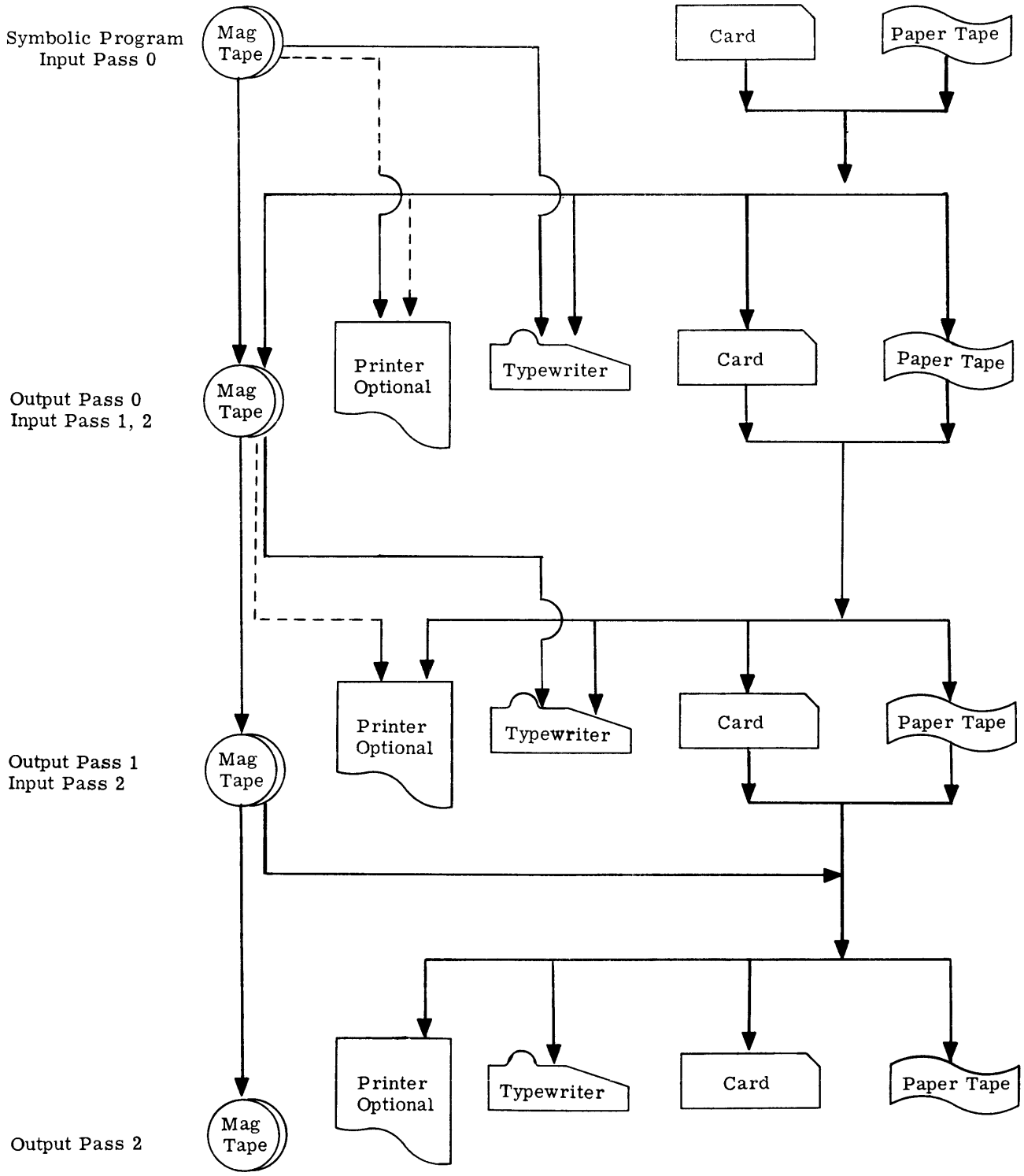


Figure IV-18. Flow Diagram of General Assembly Program II Input Media, Intermediate Storage, and Final Output Media

Switch 6

This switch is used only by Pass 2.

Normal: No tape 6 available. Punched output on cards or paper tape.
Down: Binary program output from Pass 2 is written on tape 6 (if switch 4 is down, switch 6 is ignored).

Switch 7

Not used.

Switch 8

Not Used.

Switch 9

Normal: Card punch on-line.
Down: No on-line card punch (if switch 4 is down and/or switch 6 is normal, switch 9 is ignored).

Switch 10

Not used.

Switch 11

Normal: Card or magnetic tape input.
Down: Paper tape input.

Switch 12

Normal: Card or magnetic tape output. Switch 13 is ignored.
Down: Paper tape output. Switch 13 is interrogated.

Switch 13

Switch 13 affects only Pass 2.

Normal: If switch 2 is normal, printer listing and paper tape program. If switch 2 is down, paper tape listing.
Down: Typewriter listing and paper tape program (if switch 2 is normal, switch 13 is ignored).

Switch 14

Normal: No packed symbolic listing.
Down: Print packed symbolic listing (if switch 2 is down, switch 14 is ignored).

Switch 15

This switch is used only by Pass 0.

Normal: Read input from card or paper tape and process concurrently.
Down: Read input from card or paper tape and write tape #3. Alter Pass 0 to read input from tape #3 and process.

Switch 16

This switch is used only by Pass 0.

Normal: Input to Pass 0 is on cards or paper tape. Switch 15 is interrogated.
Down: Input to Pass 0 is on magnetic tape #3. (If switch 3 or 4 is down, switch 16 is ignored, if switch 16 is down, switch 15 is ignored.)

Switch 17

Not used.

Switch 18

Normal: "No reference" symbols are typed or printed after Pass 0.
Down: Suppresses or terminates the typing or printing of the "No reference" symbols.

Switch 19

Except for machine malfunctions, the computer will stop during assembly under three circumstances only:

1. The number of special symbolic operands (symbol table 1) exceeds 250, the size allowed by the symbol table.
2. The total number of symbols (symbol table 2) exceeds the size of the table.
3. During the final phase of assembly, a name appearing in the symbol field cannot be found in the symbol table.

When these errors occur, an indicative typeout results and the computer goes into a programmed loop. Toggling switch 19 bypasses the "Symbol Table Overflow" stop during Passes 0 and 1 and the "Symbol Lost" stop during Pass 2.

Operating Instructions for the Minimum Card Configuration

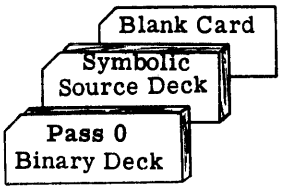
The following instructions are for loading General Assembly Program II from cards for this configuration:

Card Reader
Card Punch
Typewriter
Printer (optional)

PASS 0

When operating a minimum card system, Pass 0 must be processed first, as follows:

1. Place the symbolic program followed by one blank card behind the binary Pass 0 deck.



Pass 0 Deck Set-up

2. Load card deck into card reader.

3. Set console switches as desired.

Switch settings

Switch	Setting	Result
4	down	no magnetic tapes
2	up	on-line printer
2	down	no printer on line

All other switches normal.

4. Start Pass 0.

5. The output from Pass 0 will be a card deck consisting of packed symbolic program cards with sequence numbers starting with 20000 (columns 74-78) followed by a table of special symbolic operands (ST1) with sequence numbers starting at 10000 (columns 74-78). Figures 3 and 4 illustrate these cards. Pass 0 output is listed by the HSP, if available. A packed list of special symbols, a list of multiple symbols, a list of undefined symbols, and the symbolic names which are not referenced in the program, are printed. If no high-speed printer is available on-line, the above lists will be typed on the typewriter.

Controller selector plug assignments can be altered if necessary to meet the requirements of individual installations.

Pass 0 Messages

At the conclusion of (or during) Pass 0, messages on the typewriter or HSP indicate operating conditions. Some of these messages require immediate intervention in the form of console switch settings or rereading of input cards. These messages for Pass 0 are described in Table 3.

Card Read Error Recovery

If, during the loading of the input source deck, a card read error occurs, Pass 0 types the message CARD READ ERROR, which indicates that the last card in the output may have been mis-read or that a punch is present in column 7 or 11. If the card has a punch in column 7 or 11, correct the card. Also examine the card for off-center punching or physical damage.

Recovery from a card read error is simple. For the 400 cpm card reader:

a. Depress the MANUAL button on the GE-225 control console.

b. Remove the card deck from the input stacker; remove the card from the read platform and place it in front of the deck; load the card previously read incorrectly into the read platform; replace deck in the input stacker.

c. Press the A to I button on console.

d. Place the computer in AUTOMATIC and depress START button.

For the 1000 cpm card reader, follow essentially the same procedures, except for step b. Step b for the 1000 cpm card reader requires returning the last card in the output stacker to the front of the deck in the input hopper and then performing steps c and d.

MESSAGES	MEANING
NO END CARD	Indicates that the symbolic deck does not terminate with an END card. Assembly will continue to the normal end of job. After assembling, the transfer card may be punched manually and added at the end of the object deck.
END OF PASS 0	Signifies the end of the Pass 0 run.
SYMBOL TABLE OVERFLOW 1	Indicates that the number of special symbolic operands exceeds 250. Program goes into loop which may be overridden by setting switch 19. This causes Pass 0 to continue, but the special symbolic operands encountered after the error halt are not entered in Symbol Table 1. This may result in the improper assignment of a memory address to these symbols in the following phases.
SYMBOL TABLE OVERFLOW 2	Indicates that the total number of symbols exceeds capacity of symbol table. The program goes into a loop which may be overridden by setting switch 19. Pass 0 then continues, but the symbols following the error halt are not entered in Symbol Table 2, and are not analyzed as undefined, multiple, or no reference.

Table 3. Pass 0 Messages

Other Stops or Loops

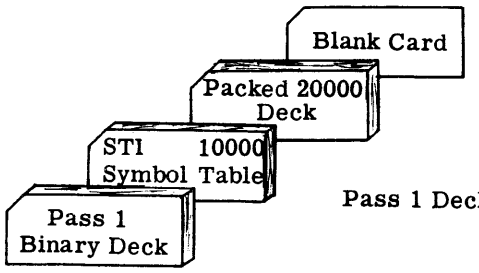
Place computer in manual mode. Check the ready status of all input/output devices. If any are in a not-ready status, take the necessary actions to ready these. Place computer in automatic and start.

If the card feed light in on, check the card deck for damaged cards. Replace such cards as necessary and reload Pass 0 from the beginning.

PASS 1

Pass 1 Follows Pass 0:

1. The output from Pass 0 is input to Pass 1, but it must be rearranged. The cards that have sequence numbers beginning with 1XXXX (columns 74 through 78) should be placed in front of cards starting with 2XXXX (columns 74 through 78). Set up the cards starting with the rearranged output from Pass 0, followed by one blank card:



Pass 1 Deck Set-up

2. Process Pass 1.

3. The output from Pass 1 is a sorted table of symbols (ST2) and equivalent locations which is punched into cards and, if the printer is on-line, listed on the high-speed printer as they are punched. In addition, a list of all

multiple-defined symbols, together with all of the equivalent values associated with each symbol, is printed (or typed, if no printer is available).

Errors or possible errors detected in the operand field of a BSS, EQU, or ORG instruction are printed or typed with the present setting of the memory allocation register, the card type, and the error code. The error codes are:

U - an undefined symbol
A - a possible error in an address

Figure 6 contains an ST2 card.

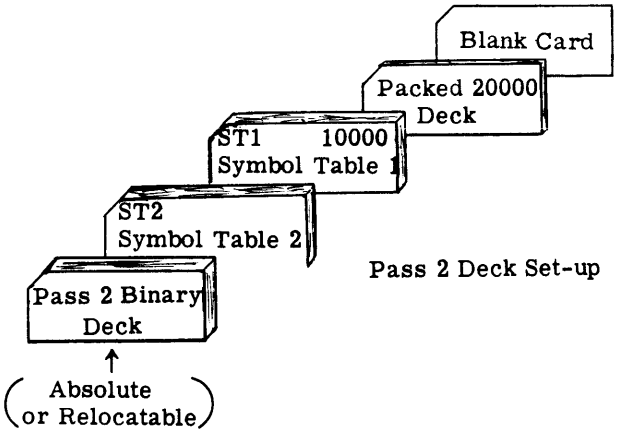
Pass 1 Messages

Table 4 shows messages that can occur during Pass 1.

PASS 2

Pass 2 Processing Follows Pass 1:

1. The input for Pass 2A or Pass 2R is the output from Pass 1 and Pass 0. Set up the input deck as follows: the output from Pass 0 and 1 followed by one blank card:



Pass 2 Deck Set-up

2. Process Pass 2.

3. The output from Pass 2 is a punched card deck (if no printer is on-line), or a punched card

MESSAGES	MEANING
NO END CARD	Indicates the symbolic deck does not terminate with an END card. Assembly continues to the normal end of job.
END OF PASS 1	Signifies the end of the Pass 1 run.
SYMBOL TABLE OVERFLOW	Messages (and action to be taken) are the same as for the Pass 0 run.
CARD READ ERROR	Card improperly read. Place GE-225 in MANUAL, backspace as described in Pass 0, press A to I, set in Automatic, and start.
ERROR IN DECK SETUP	Output cards from Pass 0 are not arranged properly. Check arrangement, correct, and reload Pass 1.

Note:
Action required for other stops or loops is the same as for Pass 0.

Table 4. Pass 1 Messages

deck and a printer listing (if a high-speed printer is available). The listing (or cards if no printer is on line) contains the octal memory location assigned to the instruction in octal, the symbolic instruction, and the codes for real or suspected errors in the instruction. These codes are:

O - illegal operation
U - undefined symbol
A - possible error in the address
M - a multiple-defined symbol in either symbol or address fields
S - errors in specified scale factors in DEC, DDC, or FDC operations
T - possible error in index

These output cards represent the assembled program deck. If the program deck is in octal (as shown in Figure 19), it can be listed off-line to obtain a program listing and loaded for execution with an octal loader; or it can be converted to a binary deck which can be loaded with a binary loader. If a printer is on-line, the output deck will be a binary deck.

Figure 7 illustrates the output from Pass 2 as listed by the HSP.

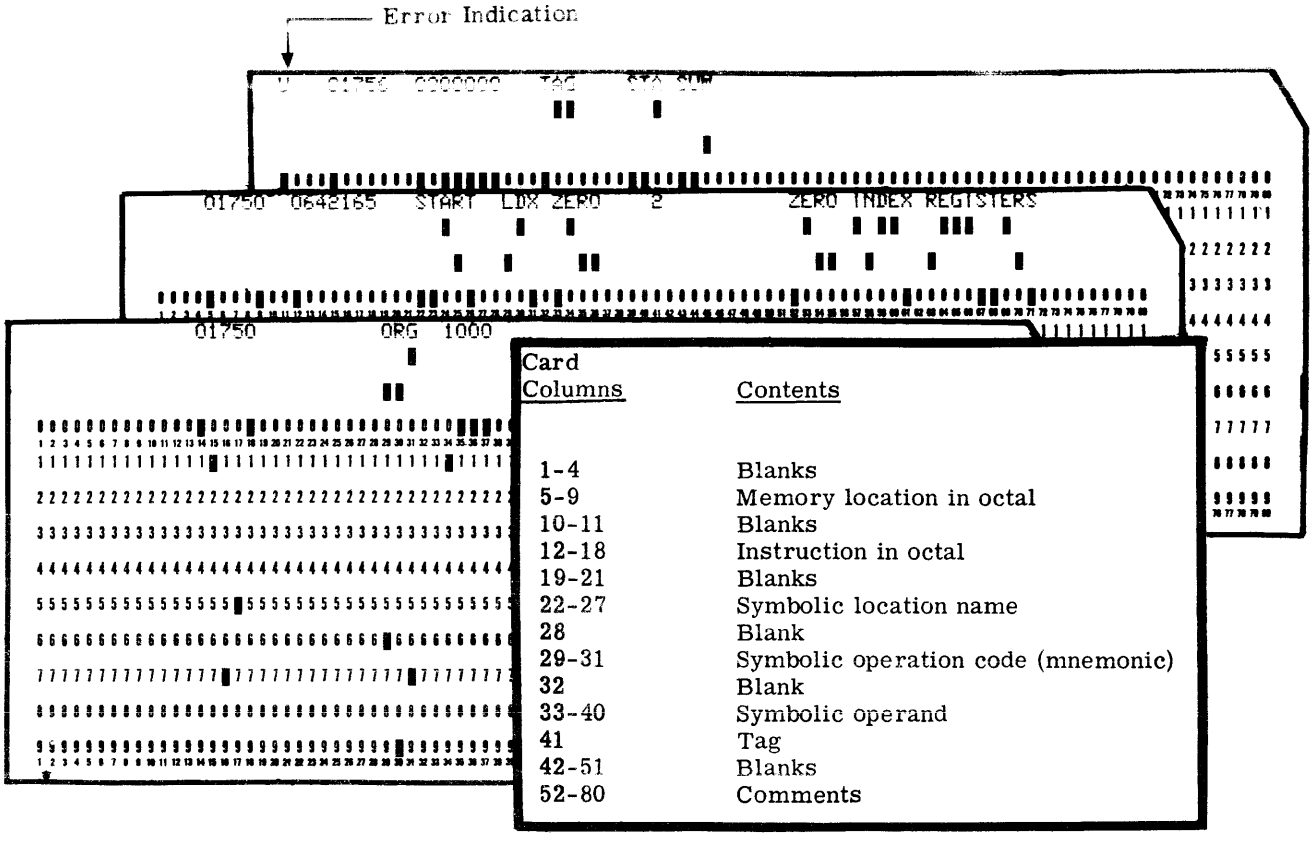
Pass 2 Messages

Table 5 illustrates Pass 2 messages.

Operating Instructions Minimum Card Configuration Plus Magnetic Tape and Printer

The following operating instructions are for loading General Assembly Program II from cards for the folllowing configuration:

Card Reader
Card Punch
2 Magnetic Tape Handlers (minimum)
Typewriter
Printer (optional)
2 additional Magnetic Tape Handlers (optional)



Card Columns	Contents
1-4	Blanks
5-9	Memory location in octal
10-11	Blanks
12-18	Instruction in octal
19-21	Blanks
22-27	Symbolic location name
28	Blank
29-31	Symbolic operation code (mnemonic)
32	Blank
33-40	Symbolic operand
41	Tag
42-51	Blanks
52-80	Comments

Figure IV-19. Octal Card Deck

MESSAGES	MEANING
ERROR	Indicates presence of a real or suspected programming error.
NO ERRORS	Indicates no real or suspected programming errors were found.
END OF PASS 2	Signifies the end of Pass 2 run.
SYMBOL LOST	Is typed with the setting of the memory allocation register and the symbol in question when a symbol appearing in the symbol field cannot be found in the symbol table. This may result from a mispunched symbol table from Pass 1, a bad read of the symbol table by Pass 2, or a bad read of the present instruction card. The program will go into a loop. Whenever a symbolic name appears in the symbol field, Pass 2 must search the symbol table formed by Pass 1 to insure that the assembled program is in phase with the memory assignment made in Pass 1. Toggle switch 19 to finish assembly or restart assembly at Pass 0. This message also results from overriding a symbol table overflow message in Pass 1. Caution: If this message is overridden, the symbol which was lost will not be found when it is used in the operand field. This may result in a number of undefined references which must be corrected by the user.
ERROR IN DECK SETUP	Indicates that the input cards to Pass 2 are not arranged properly. Correct the deck and reload Pass 2.

Note:
Action required for SYMBOL LOST.

a. Separate Pass 1 output cards from Pass 0 output cards.

b. Place Pass 0 output cards behind Pass 1 binary deck.

c. Reload Pass 1 to obtain a new symbol table.

For all other errors, restart the assembly from Pass 0.

Table 5. Pass 2 Messages

PASS 0

1. Set up cards as for card-to-card Pass 0.
2. Load cards into the card reader.
3. Set console switches as desired:

Switch settings

Switch	Setting	Result
2	Normal	Printer on-line.
2	Down	No on-line printer.
3	Normal	1 additional tape available for comments.
3	Down	Only two tapes Available for General Assembly Program II. No comments will appear on listing.
6	Normal	2 or 3 tapes available as specified by switch 3.
6	Down	1 additional tape available to write assembled binary program on tape 6. 3 or 4 tapes available as specified by switch 3.
16	Optional	

All other switches normal.

4. Start Pass 0.
5. As the symbolic cards are read by Pass 0, these will be written in card image records (27 words) on tape 3, if tape 3 is available.
6. The packed symbolic output from Pass 0 will be written on tape 4.
7. The table of special symbols (ST1) will be left in memory for Pass 1.
8. Messages, see Table 6.

   All messages and recovery procedure will be the same as for card-to-card Pass 0.
9. If switch 3 is up, and the symbolic program is on tape 3 in decimal records 27 words long, switch 16 may be set down. This will cause Pass 0 to read the symbolic program from tape 3 instead of from cards. If a read error occurs while reading from tape 3, the message, READ ERROR TAPE 3 RESTART PASS 0, will be typed and the program will halt. This means either tape 3 must be corrected before trying again, or that assembly must be restarted from the original symbolic cards.
10. Whenever Pass 0 detects a write error while writing tape 3 or tape 4, the assembly will rewrite the record until a successful write is performed.

    A count of the number of rewrites necessary during Pass 0 is typed and printed, if the printer is on-line. If these error counts are not zero, it does not necessarily mean that

MESSAGE	MEANING
XXX ERRORS TAPE 3	If Tape 3 is used for comments, this typeout signifies the number of bad spots on Tape 3.
XXX ERRORS TAPE 4	Signifies the number of bad spots on Tape 4.
READ ERROR TAPE 3 RESTART PASS 0	Read error occurred while reading from Tape 3. Correct Tape 3 or restart assembly with Pass 0.

Table 6. Pass 0 Magnetic Tape Messages

the assembly should be restarted. It is merely an indication of the number of times Pass 0 was required to rewrite the tapes, in order to get a good tape.

PASS 1

1. The input to Pass 1 is in memory and on tape 4.
2. Place two blank cards behind the Pass 1 binary deck.
3. Load cards.
4. The output from Pass 1 is the sorted table of symbols and equivalent values. This is written on tape 5 and printed, if the printer is on-line.
5. The remaining outputs are the same as for card-to-card Pass 1.
6. Messages and recovery procedures are the same as card-to-card Pass 1.

PASS 2 (ABSOLUTE OR RELOCATABLE)

1. The input to Pass 2 is the output from Pass 0 on tape 4 and on tape 3 (if present), and the output from Pass 1 on tape 5.
2. Place two blank cards behind the Pass 2 binary deck.
3. Load cards.
4. The output from Pass 2 is the same as the output from card-to-card Pass 2. In addition, the program listing is written on tape 5, which may be used to obtain multiple listings. If tape 6 is available, the binary program will be written on tape 6 as it is being punched.
5. Messages and recovery are the same as in card-to-card Pass 2.
6. ERROR TAPE 5 indicates that tape 5 was read incorrectly. Reload Pass 2. If the error message is repeated, restart assembly from Pass 0 with switch 16 down.
7. For all other errors, restart assembly from Pass 0 with switch 16 down.

Alternate Assembly Configurations

A user may wish to specify the input/output devices in a mixed fashion. For example, symbolic input from punched cards, the use of magnetic tapes as intermediate storage, and output on the printer and punched paper tape. This can be attained by properly altering the console switch settings between passes. The minimum configuration required is:

Card Reader
Two Magnetic Tape Handlers
Typewriter
Printer
Paper Tape Punch

PASS 0

Switch settings

Switch	Setting	Result
2	Normal	Printer on-line
3	Down	No magnetic tape 3
4	Normal	Tapes 4 and 5 used in assembly.

1. Load Pass 0 from cards.
2. The output from Pass 0 will be written on tape 4. The special symbol table will be left in memory for Pass 1.

PASS 1

1. No change in switches.
2. Load Pass 1 from cards.
3. The output from Pass 1 will be written on tape 5.

PASS 2

1. Set switch 12 down.
2. Load Pass 2 from cards.
3. The output from Pass 2 will be a printed listing and paper tape program.

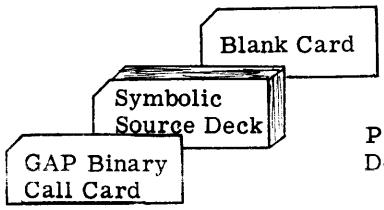
Operating Instructions for Magnetic Tape

MAGNETIC TAPE

The assembly program can be operated from a systems tape by installations having magnetic tape capability. The tape assembly program requires less assembly time than the card assembly program because less card reading and card punching is necessary. The systems tape format is described in the following section. The basic difference in operating instructions is that General Assembly Program II is to be loaded from magnetic tape 1.

1. Set console switches as desired.

2. Place symbolic input program behind the assembly program call card:



Pass 0 Tape GAP
Deck Set-up

3. Load cards.

4. The output from Pass 0 will be as specified by the switches.

5. At the end of Pass 0, Pass 1 will automatically be loaded into memory, and will go into execution phase.

6. The output from Pass 1 will be as specified by the Pass 1 switch settings previously described.

7. At the end of Pass 1, Pass 2 will automatically be loaded into memory, and will go into execution phase.

8. The output from Pass 2 will be as specified by the switches.

9. At the end of Pass 2, the systems tape will rewind and the computer will halt.

10. If a tape or check sum error is detected during the actual loading of the assembly, the computer will halt at location 44_8. Restart the assembly from the beginning. If the error halt occurs again, use the General Assembly Program II master deck to rewrite the systems tape and try again.

    Failure to load after a rewrite may be due to torn webbing in the master deck, card reader errors, or tape read errors.

11. Switch settings may be altered during the assembly, as indicated in the previous examples. Each pass reads the control switches once at the beginning of each run. After each pass begins reading the input, the switches can be altered as desired without affecting the execution of the current phase. Duplicate listings of the program can be obtained by printing tape 5.

SYSTEMS TAPE

The General Assembly Program II master deck to produce the systems tape is made up of the components listed below, including service routines for the system such as memory dumps, tape dumps, etc. The user may add service routines to the assembly program master deck as desired. It is possible for the user to insert subroutines in the master deck and up to 10 subroutines can be added to the symbolic deck at assembly time by use of the SBR pseudo-operation. The master deck is formed as follows:

1. Tape writer
2. Define controller and handler
3. PCLLDR
   Utility programs + 1 blank (additional routines may be inserted)
4. WEFPCLLDR
   Rewind tape 3 and load test program + 1 blank
5. RCDWTB
   Run ID+EOT + 1 blank
6. LDR
   Test for ID and end of tape + 1 blank
7. RCDWTB
   Run Pass 0 + 1 blank
8. LDR
   Pass 0 + 1 blank
9. WEFRCDWTD
   Subroutines + 1 blank (user option)
10.]]]]]] BSSO
11. WEFRCDWTB
    Run Pass 1 + 1 blank
12. LDR
    Pass 1 + 1 blank
13. RCDWTB
    Run Pass 2A + 1 blank
14. LDR
    Pass 2A + 1 blank
15. RCDWTB
    Run Tape 3ID + 1 blank
16. LDR
    Test tape 3 for ID record + 1 blank
17. RCDWTB
    Run Pass 2R + 1 blank
18. LDR
    Pass 2R + 1 blank
19. RCDWTB
    Run RELOAD ID TEST + 1 blank
20. LDR
    Reload ID + 1 blank
21. WEFRWD + 2 blanks

The master deck will produce a tape in the following format:

1. Tape loader
2. Memory dump program
3. Tape loader
4. Tape dump program
5. End-of-file
6. Tape loader
7. Rewind tape 3 for General Assembly Program II
8. Identification record
9. Tape loader
10. Test for program identification and end-of-tape record
11. Identification record
12. Tape loader
13. Pass 0
14. End-of-file
    (Users Subroutine Library inserted here.)
15. End-of-file
16. Identification record
17. Tape loader
18. Pass 1
19. Identification record
20. Tape loader
21. Pass 2A - Absolute
22. Identification record
23. Tape loader
24. Reposition tape 3
25. Identification record
26. Tape loader
27. Pass 2R - Relocatable
28. Identification record
29. Tape loader
30. Reload test program, blocks 9 and 10
31. End-of-file

Addition of Service Routines

The user may add service routines to the master deck by removing any card loaders from his service routine deck. The deck consisting of the binary program cards and the program transfer card is inserted in the master deck between blocks 3 and 4 in the

master deck format. Use the altered deck to write a new systems tape. During generation of the system tape, a call card is punched for each service routine in the master deck.

Addition of Symbolic Subroutines

It is also possible for symbolic subroutines to be placed onto the systems tape by modifying the assembly program master deck. The subroutines placed on the tape are normally the ones most frequently used. A typical arrangement might be:

Name	Description
CHOOSE	Least Key Finder
CRDIN	Card Read Routine
IDBNPR	Internal BCD-to-Binary
NPRIBD	Internal Binary-to-BCD
TPI/O	Tape Input and Output
SORT	Internal Memory Sort
PRINT 1	Typewriter Print Routine
DMPY	Double Precision Multiply
TRACE	Trace Routine

Although more subroutines can be stored on the tape, no more than ten can be called for during one assembly.

The advantages of having the subroutines on tape are:

1. Reduces card reading time.
2. Reduces card handling by operators and programmers.
3. Routines are easily changed and maintained.

The subroutines on the systems tape are called for as needed by the programmer using the SBR pseudo-instruction.

To prepare a subroutine for addition to the deck, perform the following steps:

1. Obtain a deck of the symbolic program cards for the desired subroutine.
2. Punch two symbolic cards to be placed at the front of the deck.

Card 1	columns	1-6	Subroutine name
		8-10	BSS
		12	0
Card 2	columns	8-10	REM

3. Punch one symbolic card to be placed at the end of the deck.

Columns	8-10	END
	12	0

4. Run this modified deck through Pass 0 with switch 4 down to obtain a deck of punched cards. This deck will look as follows:

Card 1	**columns**	**1-6**	**Subroutine name**
		7-9	BSS
		10	0
		74-78	20000
Card 2	columns	7-9	REM
		74-78	20010
Card 3	Packed Symbolic Subroutine		
	columns	74-78	20020
Card N-1			
Card N	columns	7-9	END
		10	0
Card N+1	columns	1-3	ST 1
		9-12	number of special symbols
		74-78	10000
Card N+2	Special symbol table (These will not appear if there are no special symbols.)		
Card N+M			

5. Insert cards N + 1 and N + 2 through N + M (if any) between cards 1 and 2.
6. Insert this deck in block 9 in the assembly program master deck. If two or more subroutines are in the library, they must be in ascending order according to the binary value of columns 1 through 6, card 1, which contains the symbolic name of the subroutine.
7. Use the altered deck to write a new systems tape.

Any modifications to the General Assembly Program II must be inserted in the appropriate program (Blocks 8, 11, 13, 17) in the master deck.

Where a subroutine contains a symbol in the name field of its first instruction, it is a simple matter to adopt this symbol as the SBR call name. Most of the

programming routines have this symbol - FLIP, STRIP, etc. Where a subroutine does not contain a symbol in the name field of the first instruction, the user could insert the symbol he has chosen for his SBR call in this field. This insures that the general assembly program will link the SBR name with a symbol in the subroutine.

When a subroutine which is not on tape is called for, a print-out occurs in Pass 0. This print-out gives the subroutine call symbol followed by SBR OP. For example:

```
CRDIN   SBR   OP
```

Multiple Assemblies

When using the system tape to assemble a program and absolute output is desired, the general assembly program assembles one program.

When using the system tape to assemble and relocatable output is desired, it is possible to write several programs on tape 3 and to assemble these consecutively. However, if any one of the pseudo-instructions SBR, MAL, or PAL appears in any assembly, it will not be possible to call the source program from tape 3 for assembly. Each program on tape 3 should be preceded by an identification card containing an asterisk in column 1. Columns 2 through 80 may contain any BCD information desired. There must be an end-of-file after each symbolic program on tape 3. In addition, an end-of-tape record must be written on tape 3 after the last program to be assembled.

The format of the end of tape card is as follows:

Columns	Contents
1	Asterisk
4 through 6	END
7	Blank
8 and 9	OF
10	Blank
11 through 14	TAPE

MODIFICATIONS TO GENERAL ASSEMBLY PROGRAM II

Symbol Table Length and 8K Modifications

General Assembly Program II requires a minimum of 4096 words of core storage. Each program is packed at the beginning of memory. All of the available memory following each program may be used for working storage. If a larger memory is available, the following corrections to Pass 0 and Pass 1 serve to increase the available working storage. This allows the assembly program to form a larger symbol table, but has no effect on the internal functions.

Program	Octal Location	Correction for 8192 words
Pass 0	00054	0017776
Pass 1	05043	0017777

These alterations specify to Pass 0 and Pass 1 that the constant in the specified locations is the address of the last memory location that may be used. This may be set to any desired constant beyond the programs at the user's convenience. For example, if the user desires to reserve the last 256_{10} locations of an 8192 memory, these corrections should be 0017400. The size of the symbol table which may be held by each pass is as follows:

Symbol Table 1

1. Built in pass 0 only.
2. Consists of DL, I/O, FLP, and document handler symbols.
3. Maximum size is 250.
4. Printed out at end of pass 0.
5. Error message "Symbol Table Overflow 1" when table exceeds 250.

Symbol Table 2

1. Built in Pass 0 and Pass 1.
2. Consists of all symbols used in a program.
3. Maximum size is indicated on assembly listing in decimal form immediately following the words "GAP 0" and "GAP 1".
4. This table size will vary if the number of instructions contained in Pass 0 or Pass 1 is changed. The general assembly program itself determines the number of symbols each Pass will hold. The table size is printed out at the beginning of each assembly.
5. Printed out only at end of Pass 1.
6. Error messages:

   Pass 0 - "Symbol Table Overflow 2" when table size exceeds the number indicated on the listing at beginning of the Pass.

   Pass 1 - "Symbol Table Overflow" when table size exceeds the number indicated on the listing at beginning of the Pass. This number will not be the same in both Pass 0 and Pass 1.

Plug Assignment Modification to Pass O, Pass 1, Pass 2

Without modification, the tape controller is on Plug 1 and the printer controller is on Plug 6.

Modification of plug assignments is accomplished by inserting binary correction cards just ahead of the branch card of each deck.

The octal format of the controller number assignment word is: 000000P.

Program	Octal Location of Printer Plug #	Octal Location of Tape Plug #
Pass 0	00055	00056
Pass 1	05044	05045
Pass 2A, Absolute	06747	06750
Pass 2R, Relocatable	07132	07133

Caution: All tape plug numbers or all printer plug numbers for all four programs should be changed at one time.

An example illustrating the ease with which changes in plug assignments may be made follows: Change the tape controller from Plug 1 to Plug 2.

A. Make up octal correction cards.

	Card Columns 5-9	12-18
For Pass 0 Card 1	00056	0000002
For Pass 1 Card 2	05045	0000002

For Pass 2A Card 3	06750	0000002
For Pass 2R Card 4	07133	0000002

B. Use Utility Routine CD225C3.002, Octal to Binary Card Converter, with checksum and origin to convert the octal correction cards to binary correction cards.

C. Insert the respective binary correction cards just before the transfer cards of Passes 0, 1, 2A, and 2R.

Vacuum Pocket Retrofit Modification

Normally, General Assembly Program II assumes that magnetic tapes available have vacuum pockets. If the tapes do not have vacuum pockets, the following locations in each program should be altered by inserting binary correction cards just ahead of the transfer card.

Program	Octal Location	Octal Instruction
Pass 0	06127	0000125
Pass 1	05673	0001211
Pass 2A, Absolute	07700	0002647
Pass 2R, Relocatable	10062	0002660

Modifications to assembly program card decks are easily made by punching the octal correction cards, converting them to binary, and inserting the binary card before the respective transfer cards of each assembly program deck.

System Tape Controller Plug Modification

To change the tape controller number for the system tape, it is necessary to change the system tape definition card (block 2, General Assembly Program II Systems Tape).

Punch a card as follows:

Columns	Contents
1-3	CON
9	Controller Number
15	1

Tape loaders will be altered and written as required on the master tape. In addition, new call cards for the programs in block 3 and in block 4 will be punched for the specified tape. The call card punched for block 4 (File 2 Program 1) is the required assembly program call card for this tape. This individual pass must be modified by inserting binary correction cards just ahead of the branch card.

RELOCATABLE OBJECT PROGRAMS

General Assembly Program II can provide the user with relocatable object programs. The binary cards produced are in a format acceptable to the Multi-Capability Modular Loader (MCML II), CD225B1.006R. For additional information, refer to the MCML II documentation.

To conform to the requirements of MCML, a TCD card causes a type 5 card containing the appropriate address to be punched, and an END card causes a type 3 card containing the transfer address to be punched.

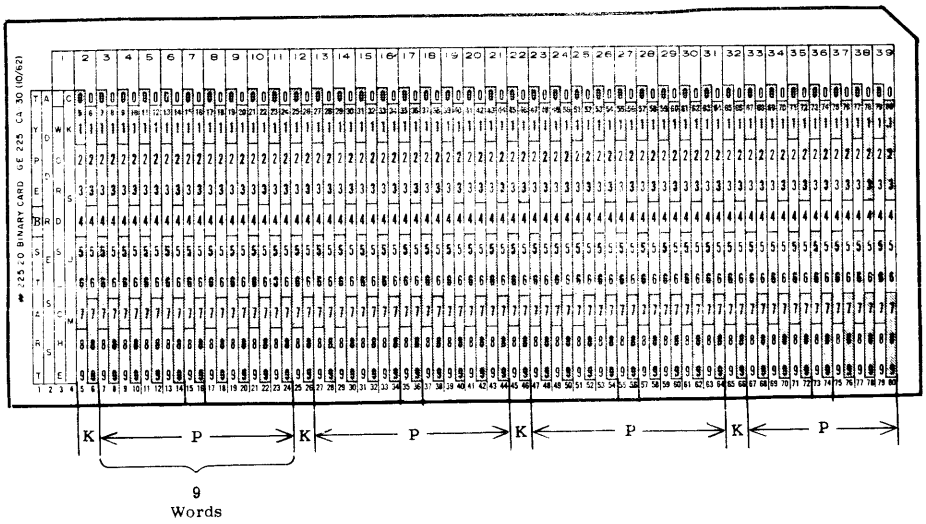
Figure 20 shows the format for relocatable instruction cards.

Calculation of Checksum

Figure 20 (page 38) illustrates the relocatable instruction card format. The checksum is punched in bit positions 7 through 9 of column 3 and bit positions 0 through 9 of column 4. The checksum is the sum of all words in the load string with the exception of the checksum itself.* Because there is the possibility of overflow there must be a test for overflow after each addition and a one must be added to the sum whenever overflow does occur. The 20-bit calculated checksum must be punched in the 13 bits allowed for it in the relocatable card format. For this reason, bits 0 through 6 of the calculated checksum must be added to bits 7 through 19 of the checksum. There is the possibility of overflow as the result of this addition and, in the case of overflow, a one must be added to the 13-bit checksum. The following coding illustrates how the checksum may be calculated for information punched on a card in the relocatable format. This example assumes that the overflow flip-flop has been cleared, index word zero contains the word count (WDCT) and index register one contains the word currently being added to the checksum.

```
        LDX     WDCT        0
        LDX     ZERO        1
        LDA     START                   } First two words
        ADD     START + 1               }
ADD     ADD     START + 2   1           }
        BØV                             }
        ADØ                             }
        INX     -1          0           }     Check A Register
        INX      1          1           }     Compute 20-bit Checksum
        BXH      1          0           }
        BRU     ADD                     } Save 13 bits of
        STO     CKSM                    } Checksum
        SRA     13                      }     Position bits 0
        EXT     MSK                     }     through 6 of Checksum
        ADD     CKSM                    }
        SLA      6                      } Test for Overflow
        SRA      6                      }
        BOV                             }
        ADØ                             }
        STO     START + 1

MSK     ØCT     3777600
CKSM    DEC     0
ZERO    DEC     0
WDCT    DEC     0                       Word computed by assembly program
```

* A load string consists of one or more sequential instructions to be contiguously loaded into memory immediately preceded by two words which contain the origin, the checksum, and other information (see Figure 20).



Explanation of Figure 20.

Field	Column	Rows	Description
TYPE	1	0-3	Type indicator: a 03 indicates a standard relocatable instruction card; a 05 indicates a mark transfer point: a 11 indicates a loader card.
B	1	4	Checksum override indicator. 0 = card checksummed during loading. 1 = checksum ignored. Usually the card will contain a checksum.
START ADDRESS	1 2	5-9 0-9	Location of the first instruction relative to the program origin.
D	3	0	Not used
WORDS	3	1-6	Number of program words on the card.
CK SUM	3 4	7-9 0-9	Checksum.
K	as shown		Relocation key or control word for the next sequence of one to nine program words. This key contains a relocation indicator for each of the following program words (P). The K field consists of nine two-bit fields each of which contains a code which applies only to the operand portion of the program word. This code is: Code (Binary) — Operand Address is 00 — Absolute 01 — Positive address relative to the relocation constant 10 — Negative address relative to relocation constant Control Word Bits 0 1 2 3 4 5 6 7 8 9 — 18 19 not used — 1st 2nd — 9th Program Word
P	as shown		Program words to be stored in consecutive memory locations relative to the location given in field.

Figure IV-20. Relocatable Instruction Card Format

PAPER TAPE ASSEMBLY

General Assembly Program II accepts paper tape as an input medium. The output can be in paper tape form, if desired.

The paper tape input is punched from the regular assembly program coding sheet using the standard Friden* Flexowriter SPD character set. Character sets are shown in Figure 21. The first character punched must be a Carriage Return (CR) followed by the 80 columns from the sheet. Each source line of coding punched from the coding sheet must be separated by a CR. However, paper tape codes prepared by off-line devices may vary widely and the user may wish to modify the paper code used by General Assembly Program II. This is easily accomplished by changing the conversion tables as described below.

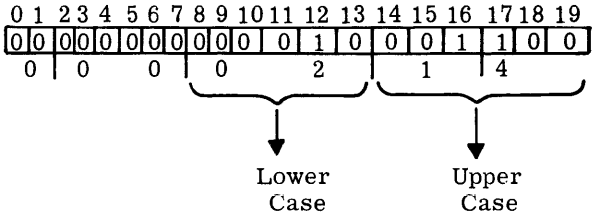
Paper Tape Input-Output Conversion Tables

Pass 0 reads and converts punched paper tape codes to internal GE-225 BCD from a conversion table. This table can be modified by correction cards to conform to the user's requirements. This table occupies octal locations 5146 to 5251:

5146	Paper tape code for lower case
5147	Paper tape code for upper case
5150	Paper tape code for carriage return
5151	Paper tape code for tab
5152	Tape codes for character set
5251	

Each entry in the table of characters carries two BCD configurations: one corresponding to an upper case paper tape code, and one to a lower case. For example, the second entry in the table contains the BCD character corresponding to a lower case paper tape 2 in bit positions 8 through 13 and the BCD character corresponding to an upper case paper tape 2 in bit positions 14 through 19.

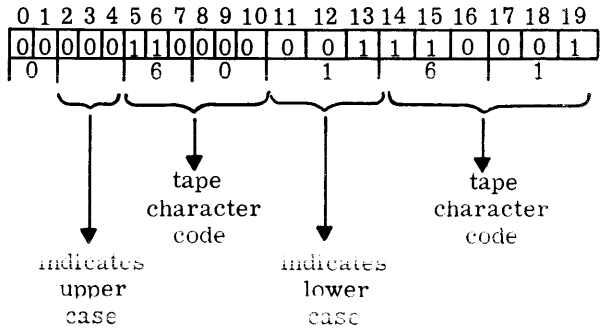
Table Memory Word



As Pass 0 and Pass 1 prepare input tapes for succeeding passes, no conversion is required. These intermediate tapes are punched in codes corresponding to internal GE-225 BCD.

The output from Pass 2 may be listed on off-line devices which require a character conversion. This is accomplished by another conversion table. A typical table memory word appears below.

Table Memory Word



In Pass 2A (absolute) this table occupies locations 6213 through 6254. Location 6213 is the lower case code, and 6214 the upper case code. Each word in the table carries the paper tape codes for two internal BCD characters, the first code in bit positions 2 through 10, the second in 11 through 19. In each 9-bit configuration, the first octal digit is a 0 if the corresponding paper tape code is in upper case and a 1 if it is in lower case. The next two octal digits contain the paper tape code itself.

Note that, in the character set shown in Figure 21, several of the Flexowriter codes are meaningless (with the two exceptions given below) to the 225 and are interpreted as spaces (Memory Octal 60). Also, any other of the Flexowriter codes that are not listed are meaningless and interpreted as spaces. The two exceptions are LOWER CASE and UPPER CASE. These are not meaningless, as they are necessary in the interpretation of two identically-punched codes that may be punched from the same Flexowriter Key, such as 3 and #.

* Trademark of Friden, Inc.

Tape Character	Lower Case	Upper Case	Paper Tape Code Channel Numbers									Memory Code (Octal)
			8	7	6	5	4	S	3	2	1	
0	x				●			.				00
1	x							.			●	01
2	x							.		●		02
3	x					●		.		●	●	03
4	x							.	●			04
5	x					●		.	●		●	05
6	x					●		.	●	●		06
7	x							.	●	●	●	07
8	x						●	.				10
9	x					●	●	.			●	11
A	x	x		●	●			.			●	21
B	x	x		●	●			.		●		22
C	x	x		●	●	●		.		●	●	23
D	x	x		●	●			.	●			24
E	x	x		●	●	●		.		●	●	25
F	x	x		●	●	●		.	●	●		26
G	x	x		●	●			.	●	●	●	27
H	x	x		●	●		●	.				30
I	x	x		●	●	●	●	.			●	31
J	x	x		●		●		.			●	41
K	x	x		●		●		.		●		42
L	x	x		●				.		●	●	43
M	x	x		●		●		.	●			44
N	x	x		●				.	●		●	45
O	x	x		●				.	●	●		46
P	x	x		●		●		.	●	●	●	47
Q	x	x		●		●	●	.				50
R	x	x		●			●	.			●	51
S	x	x			●	●		.		●		62
T	x	x			●			.		●	●	63
U	x	x			●	●		.	●			64
	x	x			●			.	●		●	65
W	x	x			●			.	●	●		66
X	x	x			●	●		.	●	●	●	67
Y	x	x			●	●	●	.				70
Z	x	x			●		●	.			●	71
SPACE	x	x				●		.				60
@		x						.		●		14
#		x				●		.		●	●	13
$		x						.	●			53
=		x				●		.	●		●	16
*		x					●	.				54
(		x				●	●	.			●	75
)		x			●			.				76
/	x				●	●		.			●	61
:		x			●	●		.			●	60
-	x			●				.				40
"		x		●				.				60
%	x			●		●	●	.		●	●	74
—		x		●		●	●	.		●	●	60
+	x			●	●	●		.				20
;		x		●	●	●		.				60
.	x	x		●	●		●	.		●	●	33
,		x						.			●	60
LOWER CASE				●	●	●	●	.		●		60
UPPER CASE				●	●	●	●	.	●			60
CARRIAGE RETURN	x	x		●				.				

Figure IV-21. Paper Tape Character Set 8 Channel Friden Flexowriter Model SPD

SECTION V

CENTRAL PROCESSOR OPERATIONS

GENERAL

Operations that occur within the central processor and do not involve either direct input-output or controller selector connected peripheral devices are classified as central processor operations. These operations are further divided into five basic categories:

1. Arithmetic
2. Data Transfer
3. Shift
4. Internal Test-and-Branch
5. Address Modification

Within each category, all instructions are discussed and presented in essentially the same format. Introducing each instruction, in GAP format, is the mnemonic operation code, the operand field (if required), and the address modification code, if the instruction can be automatically modified, thusly:

ADD	Y	X

Mnemonic Code — Memory Location — Address Modification

The Y symbol is used to indicate that, for this instruction, the operand field refers to a memory location; Y can be a symbolic or actual address. For instructions requiring an operand other than an address, the symbol K is specified in the heading. K has different meanings, depending upon the instruction, and is explained in the description of the individual instructions. The X symbol indicates that the instruction can be automatically modified. On the same heading line, the machine language form of the instruction is given in octal, followed by the required execution time of the instruction (including instruction read-out time):

ADD	Y	X	0100000	Word Times: 2

Octal Instruction — Execution Time

Following the heading is the Functional Description of the instruction, which details the effect of executing the instruction, and one or more examples of instruction usage. Included in each example are the actual GAP coding for the instruction and the contents of the affected registers before and after execution. Normally, control register contents are not shown; it can be assumed that, unless otherwise stated, the I register will contain the instruction being executed and the P counter has been stepped to the next sequential address. In other words, only the effect of the instruction is detailed.

Also, most examples are illustrated using data expressed in octal and using symbolic locations in order to provide familiarity with these forms. Octal is the form in which most GAP print-outs are made; symbolic locations are more convenient for the programmer to use than are the actual numeric locations.

ARITHMETIC INSTRUCTIONS

ADD Y X 0100000 Word Times: 2

Functional Description: ADD. The contents of memory location Y (S,1-19) are algebraically added to the contents of the A register (S,1-19). The result is placed in the A register (S,1-19). Y is unchanged. Overflow, discussed at the end of this section, is possible.

Example 1: Add a positive number 42189_{10} (0122315_8), located at GAP symbolic location AMT#2, to the positive number 52630_{10} (0146626_8), which has previously been loaded into the A register.

GAP Coding:

Symbol	Opr	Operand	X
	ADD	AMT # 2	

Register Contents in Octal

	A	Q
Before execution:	0146626	?
After execution:	0271143	?

Example 2: Add a negative number 42189_{10} (3655463_8), located at GAP symbolic location AMT#3+1, to the positive number 52630_{10} (0146626_8), which is already in the A register.

GAP Coding:

Symbol	Opr	Operand	X
	ADD	AMT # 3 + 1	

Register Contents in Octal

	A	Q
Before execution:	0146626	?
After execution:	0024311	?

Comments: Note the use of relative addressing in the operand field of Example 2. AMT#3+1 is one memory location beyond AMT#3.

SUB Y X 0200000 Word Times: 3

Functional Description: SUBTRACT. The contents of location Y (S, 1-19) are algebraically subtracted from the contents of the A register (S, 1-19). The result is placed in A (S,1-19). Y is unchanged. Overflow is possible.

GAP Coding:

Symbol	Opr	Operand	X
	SUB	AMT # 2	

Example 1: Subtract the positive number 42189_{10} (0122315_8), located at GAP symbolic location AMT#2, from the positive number 52630_{10} (0146626_8), which has been previously loaded into the A register.

Register Contents in Octal

	A	Q
Before execution:	0146626	?
After execution:	0024311	?

Example 2: Subtract the positive number 65421_{10} (0177615_8), located at GAP symbolic location AMT#3 from the smaller positive number 52630_{10} (0146626_8), which has been previously loaded into the A register.

GAP Coding:

Symbol	Opr	Operand	X
	SUB	AMT # 3	

Register Contents in Octal

	A	Q
Before execution:	0146626	?
After execution:	3747010	?

Comments: Note that, when a larger number is subtracted from a smaller number of like sign, the result is in complement form.

DAD Y X 1100000 Word Times: 3

Functional Description: DOUBLE LENGTH ADD. If the (modified) address of memory location Y is even, the contents of Y (S,1-19) and Y+1 (1-19) are algebraically added to the contents of register A (S,1-19) and Q (1-19). However, if the (modified) address Y is odd,

the contents of Y (S, 1-19) and Y (1-19) are algebraically added to the contents of A (S, 1-19) and Q (1-19). The result is placed in A (S, 1-19) and Q (1-19). The sign of the Q register is set to agree with that of the A register. Y and Y+1 are unchanged. Overflow is possible.

Example 1: Add the positive number $821{,}695_{10}$ ($0000001\ 1104677_8$), located at GAP symbolic locations AMT#7 and AMT#7+1, to the positive number 526300_{10} ($0000001\ 0003734_8$), which has been previously loaded into the A and Q registers. AMT#7 is an even-numbered memory location.

GAP Coding:

Symbol	Opr	Operand	X
	D A D	A M T # 7	

Register Contents in Octal

	A	Q
Before execution:	0000001	0003734
After execution:	0000002	1110633

Example 2: Add the positive number $821{,}695_{10}$ ($0000001\ 1104677_8$), located at GAP symbolic locations AMT#7 and AMT#7+1, to the negative number -526300_{10} ($3777776\ 3774044_8$), which has been previously loaded into the A and Q registers. AMT#7 is an even-numbered memory location.

GAP Coding:

PROGRAMMER

Symbol	Opr	Operand	X
	D A D	A M T # 7	

Register Contents in Octal

	A	Q
Before execution:	3777776	3774044
After execution:	0000000	1100743

Example 3: Add the negative number -734288_{10} ($3777776\ 3145660_8$), located at GAP symbolic locations AMT#9 and AMT#9+1, to the negative number -526300_{10} ($3777776\ 3774044_8$), which has been previously loaded into the A and Q registers. AMT#9 is an even-numbered memory location.

GAP Coding:

Symbol	Opr	Operand	X
	D A D	A M T # 9	

Register Contents in Octal

	A	Q
Before execution:	3777776	3774044
After execution:	3777775	3141724

Example 4: Add the positive number $155{,}926{,}921{,}828_{10}$ ($1104677\ 0001144_8$), located at GAP symbolic locations AMT#7+1 and AMT#7+2, to the positive number 526300_{10} ($0000001\ 0003734_8$), which has been previously loaded into the A and Q registers. If AMT#7+1 is an odd memory location, the contents of AMT#7+1 are added to the contents of both A and Q, and the contents of AMT#7+2 are ignored.

GAP Coding:

Symbol	Opr	Operand	X
	D A D	A M T # 7 + 1	

Register Contents in Octal

	A	Q
Before execution:	0000001	0003734
After execution:	1104700	1110633

DSU Y X 1200000 Word Times: 5

Functional Description: DOUBLE LENGTH SUBTRACT. If the (modified) address of memory location Y is even, the contents of Y (S, 1-19) and Y+1 (1-19) are algebraically subtracted from the contents of registers A (S, 1-19) and Q (1-19). However, if the (modified) address Y is odd, the contents of Y (S, 1-19) and Y (1-19) are algebraically subtracted from the contents of A (S, 1-19) and Q (1-19). The result is placed in A (S, 1-19) and Q (1-19). The sign of Q is set to agree with the sign of A. Y and Y+1 are unchanged. Overflow is possible.

Example 1: Subtract the positive number 526300_{10} ($0000001\ 0003734_8$), located in GAP symbolic locations AMT#6 (even) and AMT#6+1, from the positive number 821695_{10} ($0000001\ 1104677_8$) which has been previously loaded into the A and Q registers.

GAP Coding:

Symbol	Opr	Operand	X
	DSU	AMT#6	

Register Contents in Octal

	A	Q
Before execution:	0000001	1104677
After execution:	0000000	1100743

Example 2: Subtract the positive $155,929,921,828_{10}$ ($1104677\ 0001144_8$), located in GAP symbolic locations AMT#6+1 (odd) and AMT#6+2, from the positive number $155,927,218,624_{10}$ ($1104677\ 1104700_8$), which has been previously loaded into the A and Q registers.

GAP Coding:

Symbol	Opr	Operand	X
	DSU	AMT#6+1	

Register Contents in Octal

	A	Q
Before execution:	1104677	1104700
After execution:	0000000	0000001

ADO 2504032 Word Times: 3

Functional Description: ADD ONE. Plus one is added algebraically to the contents of the A register (bit position 19). If the capacity of A is exceeded, overflow occurs.

Example 1: Add one to the positive number 52630_{10} (0146626_8), which has been previously loaded into the A register.

GAP Coding:

Symbol	Opr	Operand	X
	ADO		

Register Contents in Octal

	A	Q
Before execution:	0146626	?
After execution:	0146627	?

Example 2: Add one to the negative number -42169_{10} (3655463_8), which has been previously loaded into the A register.

GAP Coding:

Symbol	Opr	Operand	X
	ADO		

Register Contents in Octal

	A	Q
Before execution:	3655463	?
After execution:	3655464	?

SBO 2504112 Word Times: 3

Functional Description: SUBTRACT ONE. Plus one is algebraically subtracted from the contents of the A register (bit position 19). If the capacity of the A register is exceeded, overflow occurs.

Example 1: Subtract one from the positive number 65421_{10} (0177615_8), which has been previously loaded into the A register.

GAP Coding:

Symbol	Opr	Operand	X
	SBO		

Register Contents in Octal

	A	Q
Before execution:	0177615	?
After execution:	0177614	?

Example 2: Subtract one from the negative number -65421_{10} (3600163_8), which has been previously loaded into the A register.

GAP Coding:

Symbol	Opr	Operand	X
	SBO		

Register Contents in Octal

	A	Q
Before execution:	3600163	?
After execution:	3600162	?

MPY Y X 1500000 Word Times: 9 to 23

Functional Description: MULTIPLY. The contents of memory location Y (S,1-19) are algebraically multiplied by the contents of the Q register (S, 1-19). The product is placed in registers A (S, 1-19) and Q (1-19). The sign of Q is the same as the sign of A after multiplication. If the contents of A are not set to zero before MPY, the contents of A are added algebraically to the least significant half of the product, thus permitting evaluation of expressions of the form AB+C. Overflow is possible.

Example 1: Multiply the positive number 52630_{10} (0146626_8) in GAP symbolic location AMT#1 by the positive number 42189_{10} (0122315_8) in the Q register. The A register contains zeros.

GAP Coding:

Symbol	Opr	Operand	X
	MPY	AMT#1	

Register Contents in Octal

	A	Q
Before execution:	0000000	0122315
After execution:	0010213	0134436

Example 2: Multiply the positive number 52630_{10} (0146626_8) in GAP symbolic location AMT#1 by the positive number $418,254_{10}$ (1460716_8) in the Q register. The A register contains the positive number 37955_{10} (0112103_8).

GAP Coding:

Symbol	Opr	Operand	X
	MPY	AMT#1	

Register Contents in Octal

	A	Q
Before execution:	0112103	1460716
After execution:	0122001	1754367

DVD Y X 1600000 Word Times: 26 to 29

Functional Description: DIVIDE. The contents of registers A (S, 1-19) and Q (1-19) are algebraically divided by the contents of location Y (S, 1-19). The quotient is placed in A (S, 1-19); the remainder is placed in Q (1-19). The sign of the remainder (Q) is the sign of the quotient (A). For proper division, the absolute magnitude of the divisor (Y) must be greater than the magnitude of the contents of A, otherwise overflow occurs.

Example 1: Divide the positive number 524220_{10} (1777674_8) in the Q register by the positive number 52630_{10} (0146626_8) in GAP symbolic location AMT#1. The A register contains zeros.

GAP Coding:

Symbol	Opr	Operand	X
	DVD	AMT#1	

Register Contents in Octal

	A	Q
Before execution:	0000000	1777674
After execution:	0000011	0142566

Decimal Arithmetic

In business applications, data to be processed is often recorded externally in the BCD format. To process such data in a binary computer requires conversion of data from BCD to binary, computation in binary mode, and subsequent reconversion to BCD format for external use.

The decimal arithmetic optional feature* provides the GE-225 with the capability of performing addition and subtraction of BCD data directly in the decimal mode, thereby eliminating the need for converting and reconverting data.

A GE-225 with the decimal arithmetic feature normally operates in the binary mode. Operation is shifted to the decimal mode only by executing a SET DECMODE instruction, and can be returned to the binary mode by executing a SET BINMODE instruction or depressing

* Part of the optional group which includes additional modification word groups and the three-way compare instruction.

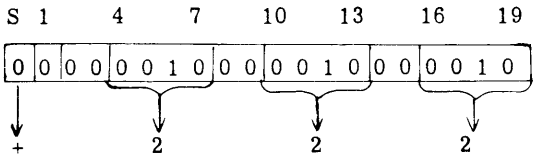
the Power On switch on the control console. The initial power on sequence automatically sets the GE-225 in the binary mode.

Rather than providing entirely new instructions and mnemonics, the decimal arithmetic feature modifies the execution of the following existing binary arithmetic instructions:

Single Add	ADD
Single Subtract	SUB
Add One	ADO
Subtract One	SBO
Double Add	DAD
Double Subtract	DSU

All other GE-225 instructions are unaffected and continue to be executed as they are in the normal binary mode. Indexing is performed in binary regardless of the mode set.

In decimal mode operations, affected GE-225 words are considered to consist of three decimal digits as shown:



Bit positions 4 through 7, 10 through 13, and 16 through 19 are used to express decimal digits in standard BCD format. Decimal quantities greater than 999 are expressed by using two or more 20-bit words.

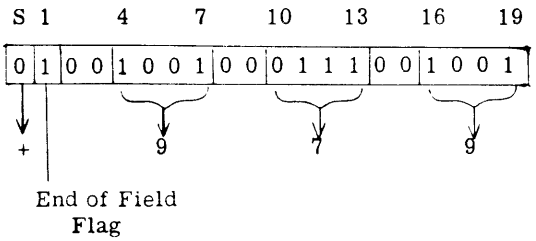
The sign of the decimal number is in the S position of the word containing the most significant decimal digit; a 0-bit designates a positive decimal number, while a 1-bit indicates a negative quantity.

Zone bits of each BCD character (2 and 3, 8 and 9, and 14 and 15) contain 0-bits and do not enter into arithmetic operations.

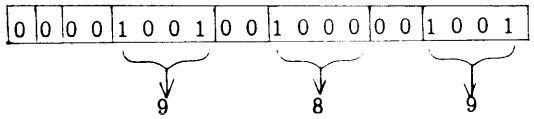
The decimal word containing the most significant (high-order) digit must be marked or flagged to define the end of the decimal field by placing a 1-bit in bit position 1.

Thus, the decimal quantity +979989 would appear in memory as two words of three digits each:

Memory Location Y



Memory Location Y+1



The programmer should flag each BCD number prior to arithmetic operations by coding which sets a 1-bit into bit position 1 of the most significant word of each quantity. Sample coding to accomplish this is shown under Program Insertion of End-of-Field Flag.

Besides defining the length of the decimal number, the end-of-field flag affects the disposition of carries generated during arithmetic operations. A carry out of the most significant digit position of a word is remembered if the word does not contain an end-of-field flag. The carry is remembered either until the next decimal instruction is executed or the Clear Alarm is depressed.

If the end-of-field marker is set (a 1-bit in position 1), then a carry out of the most significant digit position causes overflow, which turns on the overflow indicator and reverses the sign of the most significant word of the decimal number.

The end-of-field flag is not essential for both quantities involved in a decimal operation; only the high-order word of the quantity loaded into the A register must be so marked. If the field in memory is flagged and the field in the A register is not, an error condition occurs. If both fields are flagged, the effect is the same as if only the A register were flagged. A flag in the A register field automatically generates an end-of-field flag for the result field.

Negative decimal numbers must be expressed in the 10's complement form before decimal operations. The 10's complement is formed automatically by subtracting the decimal number from a decimal zero (delimited

by an end-of-field flag in bit position 1) while in the decimal mode. Negative results of decimal operations also appear in the 10's complement form. Thus, the decimal number -222222 would be converted to -777,778 (1,000,000 - 222,222) before being used in arithmetic operations.

DECIMAL ARITHMETIC INSTRUCTIONS

ADD	Y	X	0100000	Word Times: 2

Functional Description: DECIMAL ADD. The contents of Y (3 BCD digits, S, 4-7, 10-13, and 16-19) are algebraically added to the contents of the A register (bits S, 4-7, 10-13, and 16-19). The result is placed in the A register (bits S, 4-7, 10-13, and 16-19).

Example 1: Decimal add the quantity +333 in symbolic location INCR to +444 which has been previously loaded into the A register. Assume that the central processor is operating in the decimal mode, by a prior SET DECMODE instruction.

GAP Coding:

Symbol	Opr	Operand	X
	ADD	INCR	

Memory and A Register Contents in BCD

	A	INCR
Before execution:	+ 4 4 4	+ 3 3 3
After execution:	+ 7 7 7	+ 3 3 3

Example 2: Decimal add the quantity -333 in symbolic location NEGN to +444 which has been previously loaded into the A register. Assume that the central processor is operating in the decimal mode.

GAP Coding:

Symbol	Opr	Operand	X
	ADD	NEGN	

Memory and A Register Contents in BCD

	A	NEGN
Before execution:	+ 4 4 4	- 6 6 7
After execution:	+ 1 1 1	- 6 6 7

SUB	Y	X	0200000	Word Times: 3

Functional Description: DECIMAL SUBTRACT. The contents of Y (bits S, 4-7, 10-13, and 16-19) are algebraically subtracted from the contents of the A register (bits S, 4-7, 10-13, and 16-19). The result is placed in the A register (bits S, 4-7, 10-13, and 16-19).

Example 1: Decimal subtract the quantity +333 in symbolic location DECR from +444 which has been previously loaded into the A register. Assume that the central processor is operating in the decimal mode.

GAP Coding:

Symbol	Opr	Operand	X
	SUB	DECR	

Memory and A Register Contents in BCD

	A	DECR
Before execution:	+ 4 4 4	+ 3 3 3
After execution:	+ 1 1 1	+ 3 3 3

Example 2: Decimal subtract the quantity -333 in symbolic location NEGN from +444 which has been previously loaded into the A register. Assume that the central processor is operating in the decimal mode.

GAP Coding:

Symbol	Opr	Operand	X
	SUB	NEGN	

Memory and A Register Contents in BCD

	A	NEGN
Before execution:	+ 4 4 4	- 6 6 7
After execution:	+ 7 7 7	- 6 6 7

DAD Y X 1100000 Word Times: 3

Functional Description: DOUBLE DECIMAL ADD. If Y is even, the contents of Y (S, 4-7, 10-13, and 16-19) and Y+1 (4-7, 10-13, and 16-19) are algebraically added to the contents of registers A (S, 4-7, 10-13, and 16-19) and Q (4-7, 10-13, and 16-19). If Y is odd, the contents of Y (S, 4-7, 10-13, and 16-19) and Y (4-7, 10-13, and 16-19) are added to registers A (S, 4-7, 10-13, and 16-19) and Q (4-7, 10-13, and 16-19). The result is placed in registers A and Q.

Example 1: Double decimal add the quantity +123456 in symbolic locations POSN and POSN+1 to the quantity +543210 which has been previously loaded into the A and Q registers. Assume that POSN is an even memory address and that the central processor is operating in the decimal mode.

GAP Coding:

Symbol	Opr	Operand	X
	DAD	POSN	

Memory and A and Q Register Contents in BCD

Before execution:

A	Q
+ 5 4 3	2 1 0

POSN	POSN+1
+ 1 2 3	4 5 6

After execution:

A	Q
+ 6 6 6	6 6 6

POSN	POSN+1
+ 1 2 3	4 5 6

Example 2: Double decimal add the quantity +123456 in symbolic locations PREP and PREP+1 to the quantity +543210 which has been previously loaded into the A and Q registers. Assume that PREP is an odd memory address and that the central processor is operating in the decimal mode.

GAP Coding:

Symbol	Opr	Operand	X
	DAD	PREP	

Memory and A and Q Register Contents in BCD

Before execution:

A	Q
+ 5 4 3	2 1 0

PREP	PREP+1
+ 1 2 3	4 5 6

After execution:

A	Q
+ 6 6 6	3 3 3

PREP	PREP+1
+ 1 2 3	4 5 6

DSU Y X 1200000 Word Times: 5

Functional Description: DOUBLE DECIMAL SUBTRACT. If Y is even, the contents of Y (S, 4-7, 10-13, and 16-19) and Y+1 (4-7, 10-13, and 16-19) are algebraically subtracted from the contents of registers A (S, 4-7, 10-13, and 16-19) and Q (4-7, 10-13, and 16-19). If Y is odd, the contents of Y (S, 4-7, 10-13, and 16-19) and Y (4-7, 10-13, and 16-19) are subtracted from the contents of registers A (S, 4-7, 10-13, and 16-19) and Q (4-7, 10-13, and 16-19). The result is placed in the A and Q registers.

Example 1: Double decimal subtract the quantity +123456 in symbolic locations DECR and DECR+1 from the quantity +543210 which has been previously loaded into the A and Q registers. Assume that DECR is an even memory address and that the central processor is operating in the decimal mode.

GAP Coding:

Symbol	Opr	Operand	X
	DSU	DECR	

Memory and A and Q Register Contents in BCD

Before execution:

A	Q
+ 5 4 3	2 1 0

DECR	DECR+1
+ 1 2 3	4 5 6

After execution:

A	Q
+ 4 1 9	7 5 4

DECR	DECR+1
+ 1 2 3	4 5 6

Example 2: Double decimal subtract the quantity +123456 in symbolic locations NEGR and NEGR+1 from the quantity +543210 which has been previously loaded into the A and Q registers. Assume that NEGR is an odd memory address and that the central processor is operating in the decimal mode.

GAP Coding:

Symbol	Opr	Operand	X
	D S U	N E G R	

Memory and A and Q Register Contents in BCD

Before execution:

A	Q
+ 5 4 3	2 1 0

NEGR	NEGR+1
+ 1 2 3	4 5 6

After execution:

A	Q
+ 4 2 0	0 8 7

NEGR	NEGR+1
+ 1 2 3	4 5 6

ADO 2504032 Word Times: 3

Functional Description: ADD ONE DECIMAL. One is algebraically added to the contents of the A register (4-7, 10-13, and 16-19). If the capacity of A is exceeded, the overflow indicator is turned on. This instruction operates properly only on decimal words of three digits or less.

Example: Add a decimal one to the quantity +832 in the A register.

GAP Coding:

Symbol	Opr	Operand	X
	A D O		

Register Contents in BCD

	A
Before execution:	+ 8 3 2
After execution:	+ 8 3 3

SBO 2504112 Word Times: 3

Functional Description: SUBTRACT ONE DECIMAL. One is subtracted algebraically from the contents of the A register (4-7, 10-13, and 16-19). If the capacity of the A register is exceeded, the overflow indicator is turned on. This instruction operates properly only on decimal words of three digits or less.

Example: Subtract a decimal one from the quantity -763 in the A register. Assume that the 10's complement of -763 has already been formed.

GAP Coding:

Symbol	Opr	Operand	X
	S B O		

Register Contents in BCD

	A
Before execution:	- 2 3 7
After execution:	- 2 3 6

MODE CONTROL INSTRUCTIONS

SET DECMODE 2506011 Word Times: 2

Functional Description: SET DECIMAL MODE causes the arithmetic commands ADD, DAD, SUB, DSU, ADO,

and SBO to be executed in the decimal mode. No other commands are affected.

SET BINMODE 2506012 Word Times: 2

Functional Description: SET BINARY MODE causes the arithmetic commands ADD, DAD, SUB, DSU, ADO, and SBO to be executed in the binary mode. No other commands are affected.

RELATED CONSOLE CONTROLS

1. Power On Switch. Depression of this switch at any time sets the central processor into the binary mode of operation.

2. Clear Alarm Switch. Depression of this switch removes any carry resulting from uncompleted decimal operations and prepares the decimal controls for a new sequence.

PROGRAM INSERTION OF END-OF-FIELD FLAGS

To designate the beginning of a decimal field, a 1-bit is inserted into bit position 1 of the high-order word of the field. A typical method of accomplishing the bit insertion is:

GAP Coding:

Symbol	Opr	Operand	X
MILL	OCT	1000000	
	LDA	MILL	
	ORY	DECW	

DECW Contents in Binary

Before execution: 0 0 000 010 000 010 000 010

+ 2 2 2

After execution: 0 1 000 010 000 010 000 010

+ 2 2 2

End-of-Field Flag

Comments: The OCT 1000000 places the flag constant in storage; LDA MILL and ORY DECW insert a 1-bit into bit position 1 of DECW (the high-order word).

TEN'S COMPLEMENT FORMATION

Preparatory to decimal arithmetic operations, negative decimal quantities must be converted to 10's complement form. One method for so doing is:

GAP Coding:

Symbol	Opr	Operand	X
MILL	OCT	1000000	
	OCT	0000000	
	DLD	MILL	
	DSU	NEGD	
	DST	COMP	

Memory Contents in BCD

NEGD and NEGD+1: + 3 2 5 | 4 1 6

COMP and COMP+1: (after execution) - 6 7 4 | 5 8 4

PROGRAMMING DECIMAL OPERATIONS

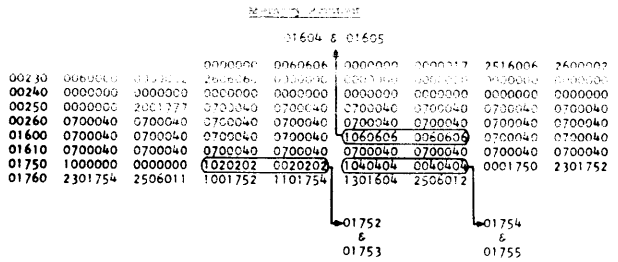
The GAP listing below illustrates the fundamentals of performing arithmetic operations in the decimal mode. Address location 01750 contains the end of field marker to be inserted in the two BCD numbers before addition. In theory, both numbers need not contain a flag; only the number in the A register must have the marker. However, it is a good practice to flag all numbers to be used in decimal arithmetic operations. Memory locations 01756, 01757 and 01760 contain the commands for flagging the BCD numbers.

Command 01761 converts the internal operation of the computer to BCD prior to the addition and command 1765 restores the computer to the binary mode.

GAP Listing

```
          01750              ORG   1000
01750     1000000    MILL    OCT   1000000
01751     0000000            OCT   0000000
01752     0020202      A1    ALF   222
01753     0020202      A2    ALF   222
01754     0040404      B1    ALF   444
01755     0040404      B2    ALF   444
01756     0001750-   START   LDA   MILL
01757     2301752            ORY   A1
01760     2301754            ORY   B1
01761     2506011            SET   DECMODE
01762     1001752            DLD   A1
01763     1101754            DAD   B1
01764     1301604            DST   0900
01765     2506012            SET   BINMODE
```

The printout of the memory addresses used in the program shows that locations 01752 and 01754 contain flags in the words containing the most significant digits. Location 01604 contains the sum which also is automatically flagged.



Overflow

During arithmetic operations, the result of the calculation can exceed the capacity of the 20-bit A register. When this happens, the register overflows (loses a bit from the high-order position). This is known as an overflow condition.

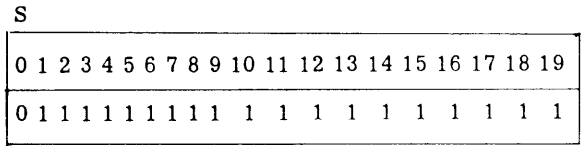
The A register can also overflow as a result of double length word calculations. For a divide instruction, register overflow can occur when the magnitude of the divisor is not greater than that portion of the dividend in the A register. An overflow condition also is possible when an attempt is made to negate (execute a NEG instruction) the largest possible negative number.

When an overflow condition arises, three things happen:

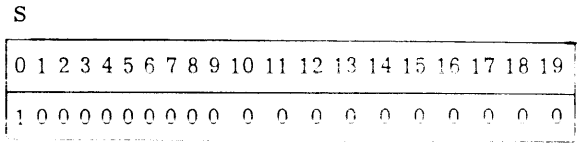
1. The sign of the result is reversed.
2. The most significant bit of the result (in bit position 1) is lost, and
3. The overflow indicator on the control console is turned ON.

The reversal of the sign bit in the A register causes the overflow indicator to turn ON, regardless of the type of instruction causing overflow.

Register Capacity. The A register can hold any number consisting of 19 numerical bits (bits 1 through 19) plus the sign bit (bit 0). Thus, it is possible to represent a maximum positive number of 524.287_{10} and a maximum negative number of -524.288_{10} before overflow could occur. These two numbers, with their binary equivalents are shown below:



Maximum Positive Number = $+524.287_{10}$



Maximum Negative Number = -524.288_{10}

The addition of any number, except 0, to the largest positive number causes an overflow of a 1-bit into the sign bit position, thereby reversing the sign.

As shown, the maximum negative number consists of a 1 bit in the sign bit position followed by all zeros. It is incorrect to consider this configuration as a 'minus zero;' it is $-524,288_{10}$. An attempt to negate the largest negative number (with the NEG instruction) results in overflow: all the bit positions are reversed, giving the 1's complement, and when one is added to form the 2's complement, a one is carried into the sign bit position. It can be seen that, although bit 0 indicates the sign of the number (0 = plus; 1 = minus), all twenty bits are involved in arithmetic operations.

The specific conditions for overflow are summed up in the following paragraphs. Overflow for each kind of arithmetic operation is illustrated by examples.

Addition Overflow. The overflow indication occurs during the addition of two positive numbers when there is a carry from the most significant bit position (bit position 1) to the sign bit position. No overflow indication is possible during the addition of numbers with unlike signs. The overflow indication occurs during the addition of two negative numbers when there is a reversal of the sign bit position.

Example 1: Add the contents of symbolic location AMT#1 (0146626_8) to 1777674_8, which has previously been loaded into the A register.

GAP Coding:

Symbol	Opr	Operand	X
	ADD	AMT#1	

Register Contents in Octal

	A	Q
Before execution:	1777674	?
After execution:	2146522	?

Example 2: Add the contents of symbolic location AMT#2 (-524288_{10} or 2000000_8) to -1, which has previously been loaded into the A register.

GAP Coding:

Symbol	Opr	Operand	X
	ADD	AMT#2	

Register Contents in Octal

	A	Q
Before execution:	3777777	?
After execution:	1777777	?

Comments: Note that, in both examples, the sign bit of the A register is reversed. In example 1, initially the sign bit position and bit position 1 contain 01; after addition, these positions contain 10. In example 2, initially the sign bit and bit position 1 contain 11; after addition, these positions contain 01.

Subtraction Overflow. In subtraction, the 2's complement of the subtrahend is added to the contents of the A register. The rules for overflow which apply to addition also apply to subtraction.

Example: Subtract the negative number in symbolic location AMT#3 (-65421_{10} or 3600163_8) from the positive number 524220_{10}, which has previously been loaded into the A register.

GAP Coding:

Symbol	Opr	Operand	X
	SUB	AMT#3	

Register Contents in Octal

	A	Q
Before execution:	1777674	?
After execution:	2177511	?

Comments: Note that this subtraction is performed by adding the 2's complement of 3600163_8 (0177615_8) to 1777674_8. Overflow occurs when the sign bit changes from 0 to 1.

Multiplication Overflow. The overflow indication occurs in multiplication only when there is an attempt to multiply the maximum negative number by the maximum negative number ($-2^{19} \times -2^{19}$). The overflow indicator on the control console is automatically turned off prior to execution of a multiply instruction.

Example: Multiply $-524,288_{10}$ in symbolic location AMT#7 by $-524,288_{10}$, which has previously been loaded into the Q register.

GAP Coding:

Symbol	Opr	Operand	X
	MPY	AMT#7	

Register Contents in Octal

	A	Q
Before execution:	0000000	2000000
After execution:	2000000	2000000

Division Overflow. For proper division, the magnitude of the divisor must be greater than the magnitude of that portion of the dividend in register A. If not, the overflow indication is turned on and control is transferred to the next instruction in sequence. The overflow indicator on the control console is automatically turned off prior to the execution of a divide instruction. Also, overflow will occur if division results in a quotient that exceeds the capacity of the A register.

Example: Divide the positive number $17,338,832,329_{10}$ ($0100457\ 0312711_8$), which has been previously double loaded into the A and Q registers, by $20,000_{10}$ (0047040_8) in symbolic location WRDS.

GAP Coding:

Symbol						Opr			Operand									X
1	2	3	4	5	6	8	9	10	12	13	14	15	16	17	18	19		20
						D	V	D	W	R	D	S						

	Register Contents in Octal	
	A	Q
Before execution:	0100457	0312711
After execution:	0201136	0625622

Scaling

The movement of the decimal point to the right or left to properly align numbers is called 'scaling' or 'decimal positioning.' Before decimal numbers can be correctly added or subtracted in the central processor, the number of places to the right of the decimal point of both numbers must be the same. For example, to add 3.0 to 4.16, 3.0 is arranged to correspond to 3.00 and then added to 4.16. If the decimal point is moved to the right in preparation for calculations, the number is 'scaled to the right;' if the decimal point is moved to the left, the number is 'scaled to the left.'

When two numbers are multiplied, the number of places to the right of the decimal point in the product is the sum of the places to the right of the decimal point in both the multiplier and the multiplicand. If it is desired to scale the product (which is expressed as a binary number) for subsequent calculations, the product must be divided by a constant that is the binary equivalent of an appropriate power of 10.

To further illustrate the concept of scaling, consider the example of adding the following two decimal numbers:

$$\begin{array}{r} 24.4 \\ +\ 13.25 \\ \hline 37.65 \end{array}$$ Desired sum

Because the central processor does not recognize decimal points in arithmetic operations, the binary equivalent of 244_{10} and 1325_{10} would appear in memory as shown in Figure 5-1.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	0	0
0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	1	1	0	1

$= 244_{10}$ and 1325_{10}

Figure 5-1. Two Numbers in Memory before Scaling

When these two numbers are added, the result would appear in the A register as 1569_{10} (Figure 5-2). This, of course, is incorrect, for the desired sum is 37.65_{10}.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	1

$= 1569_{10}$

Figure 5-2. Incorrect Sum after Addition without Scaling

To obtain the correct sum of 37.65_{10}, it is necessary to scale the augend 244_{10} to the left one decimal position by multiplying 244_{10} by 10_{10}. Through multiplication, 244_{10} becomes 2440_{10} and thus is scaled to the left so that the decimal points in the two numbers are properly aligned. After scaling, the two numbers are aligned as shown in Figure 5-3.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	1	1	0	1

$= 2440_{10}$ and 1325_{10}

Figure 5-3. Numbers in Memory after Scaling

Because the two numbers are now properly aligned, the correct sum of 37.65_{10} is achieved when the numbers are added.

Note that scaling operations can be accomplished in one of two ways: (1) by multiplying or dividing by the binary equivalent of the appropriate power of 10, or (2) by using GE-225 scaling routines available to the programmer.

Rounding

After a calculation has been completed, it is sometimes necessary to round the result to the next highest integer. 'Rounding' is accomplished by adding a '5' into the decimal position to the right of the position to receive any carry. Since all calculations, within the GE-225 are performed primarily with binary numbers, the proper rounding factor of '5' is expressed in binary and is carried as an appropriate constant within memory. For example, this constant might be programmed by using the pseudo-instruction DEC to obtain the binary equivalent of 5. The instruction would be DEC 5. See Section IV for detailed discussion of pseudo-instructions. After the rounding factor is added, the positions to the right of the digit which receives any carry can be deleted through scaling.

To illustrate further, assume that the decimal 10.75 is to be rounded to the nearest tenth. By using a rounding factor of .05 stored as a constant in memory, the desired result, 10.80, is achieved by adding the rounding factor as shown in Figure 5-4.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1.	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	1	1
2.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
3.	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	0	0	0

where $1 = 10.75_{10}$
$2 = .05_{10}$
$3 = 10.80_{10}$

Figure 5-4. Using a Rounding Factor of .05

DATA TRANSFER INSTRUCTIONS

Data transfer instructions are grouped into two major categories: memory transfers and register transfers. Although not involving a true transfer of data, register modification instructions are also included in this section.

Memory transfers involve word movement between core memory and central processor registers. In general, the previous contents of the 'receiving' unit (memory location or register) are replaced by the transferred word, while the transferred word remains unchanged in the original memory location or register.

Arithmetic register transfers involve the transfer of information between registers; the condition of the register initially holding the information is unchanged, after execution, except as noted in the discussion of each instruction.

Register modification instructions change the contents of the specified register in a predetermined manner, such as complementing, sign changing, and negating.

Data transfer instructions involve either or both the A and Q registers. In general, transfer instructions cause parallel transfers (all bits simultaneously), rather than serial transfers (a bit at a time).

Data Transfers-Memory

LDA	Y	X	0000000	Word Times: 2

Functional Description: LOAD A REGISTER. The contents of memory locations Y (S, 1-19) replace the contents of the A register (S, 1-19). Y is unchanged.

Example 1: Load the A register with the contents of GAP symbolic location AMT#1, which contains the positive number 52630_{10} (0146626_8). The A register initially contains zeros.

GAP Coding:

Symbol	Opr	Operand	X
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	L D A	A M T # 1	

	Register Contents in Octal	
	A	Q
Before execution:	0000000	?
After execution:	0146626	?

Example 2: Load the A register with the contents of GAP symbolic location AMT#5, which contains the negative number -42189_{10} (3655463_8). The A register initially contains 42189_{10} (0122315_8).

GAP Coding:

Symbol	Opr	Operand	X
	LDA	AMT#5	

Register Contents in Octal

	A	Q
Before execution:	0122315	?
After execution:	3655463	?

DLD Y X 1000000 Word Times: 3

Functional Description: DOUBLE LENGTH LOAD. If the (modified) address of location Y is even, the contents of Y (S, 1-19) and Y+1 (S, 1-19) replace the contents of the A (S, 1-19) and Q (S, 1-19) registers. If the (modified) address of Y is odd, the contents of Y (S, 1-19) replace the contents of the A (S, 1-19) and Q (S, 1-19) registers. Y and Y+1 are unchanged.

Example 1: Double length load the A and Q registers with the positive number 821695_{10} ($0000001\ 1104677_8$) in GAP symbolic locations AMT#7 (even) and AMT#7+1.

GAP Coding:

Symbol	Opr	Operand	X
	DLD	AMT#7	

Register Contents in Octal

	A	Q
Before execution:	?	?
After execution:	0000001	1104677

Example 2: Double length load the A and Q registers with the positive number 526300_{10} ($0000001\ 0003734_8$) in GAP symbolic locations AMT#6 (odd) and AMT#6+1.

GAP Coding:

Symbol	Opr	Operand	X
	DLD	AMT#6	

Register Contents in Octal

	A	Q
Before execution:	?	?
After execution:	0000001	0000001

Comments: Note that, if the specified operand address is odd, the contents of that address are loaded into both the A and Q registers and the second address is ignored.

STA Y X 0300000 Word Times: ?

Functional Description: STORE A. The contents of the A register (S, 1-19) replace the contents of memory location Y (S, 1-19). The contents of A are unchanged.

Example 1: Store the A register contents 42189_{10} (0122315_8) in GAP symbolic location RESULT.

GAP Coding:

Symbol	Opr	Operand	X
	STA	RESULT	

Register Contents in Octal

	A	Q
Before execution:	0122315	?
After execution:	0122315	?

GAP Symbolic Location, RESULT

	A
Before execution:	?
After execution:	0122315

Example 2: Store the A register contents -65421_{10} (3600163_8) in GAP symbolic location OUTPUT. OUTPUT initially contains -42189_{10} (3655463_8).

GAP Coding:

Symbol	Opr	Operand	X
	STA	OUTPUT	

Register Contents in Octal

	A	Q
Before execution:	3600163	?
After execution:	3600163	?

GAP Symbolic Location, OUTPUT

	A
Before execution:	3655463
After execution:	3600163

DST Y X 1300000 Word Times: 3

Functional Description: DOUBLE LENGTH STORE. If the (modified) address of memory location Y is even, the contents of the A (S, 1-19) and Q (S, 1-19) registers replace the contents of Y (S, 1-19) and Y+1 (S, 1-19). If the (modified) address of Y is odd, the contents of Q (S, 1-19) replace the contents of Y (S, 1-19). The contents of A and Q are unchanged.

Example 1: Double length store A and Q register contents 821695_{10} (0000001 1104677_8) in GAP symbolic locations AMT#8 (even) and AMT#8+1.

GAP Coding:

Symbol	Opr	Operand	X
	DST	AMT#8	

Register Contents in Octal

	A	Q
Before execution:	0000001	1104677
After execution:	0000001	1104677

GAP Symbolic Locations

	AMT#8	AMT#8+1
Before execution:	?	?
After execution:	0000001	1104677

Example 2: Double length store A and Q register contents 526300_{10} (0000001 0003734_8) in GAP symbolic locations AMT#7 (odd) and AMT#7+1.

Register Contents in Octal

	A	Q
Before execution:	0000001	0003734
After execution:	0000001	0003734

GAP Symbolic Locations

	AMT#7	AMT#7+1
Before execution:	?	?
After execution:	0003734	?

STO Y X 2700000 Word Times: 3

Functional Description: STORE OPERAND ADDRESS. The contents of the A register (7-19) replace the contents of memory location Y (7-19). A (S, 1-19) and Y (S, 1-6) are unchanged.

Example: Store the operand address that is in the A register, 65535_{10} (17777_8), in GAP symbolic location TAX#1, which initially contains 0001667_8, an LDA instruction.

GAP Coding:

Symbol	Opr	Operand	X
	STO	TAX#1	

Register Contents in Octal

	A	Q
Before execution:	0017777	?
After execution:	0017777	?

GAP Symbolic Location, TAX#1

Before execution:	0001667
After execution:	0017777

ORY Y X 2300000 Word Times: 3

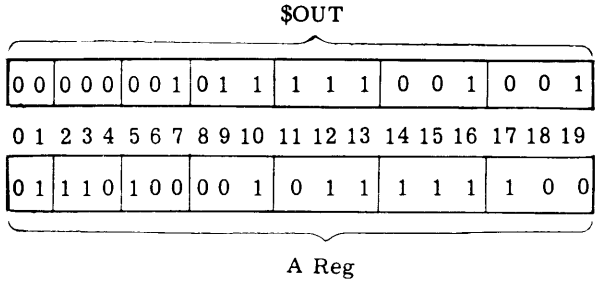
Functional Description: OR A INTO Y. Corresponding bit positions of memory location Y (S, 1-19) are set with 1-bits for every bit position of the A register (S, 1-19) containing a 1-bit. The contents of the A register and other bit positions of Y remain unchanged.

Example 1: OR A into Y with the A register containing 1641374_8 and Y is GAP symbolic location $OUT, containing 0013711_8.

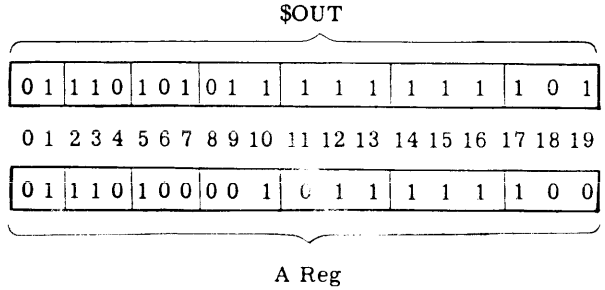
GAP Coding:

Symbol	Opr	Operand	X
	O R Y	$ O U T	

Memory and A Register before Execution (binary):



Memory and A Register after Execution (binary):



Example 2: Place a dollar sign ($), previously loaded into the A register, before the 2-digit BCD quantity 56 in GAP symbolic location PRICE.

GAP Coding:

Symbol	Opr	Operand	X
	O R Y	P R I C E	

Memory and A Register Contents (BCD)

	A Reg			PRICE		
Before execution:	$	0	0	0	5	6
After execution:	$	0	0	$	5	6

EXT Y X 2000000 Word Times: 3

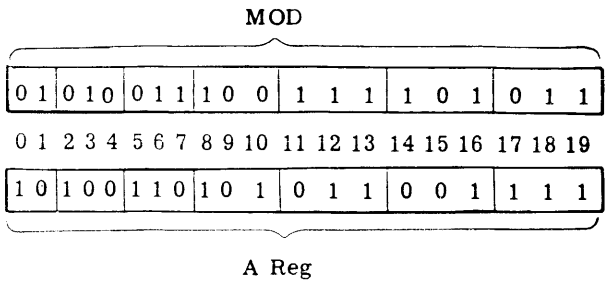
Functional Description: EXTRACT. For each 1-bit in Y (S, 1-19) a 0-bit is placed in the corresponding bit position of the A register (S, 1-19). If bit positions in Y contain 0-bits, the corresponding bit positions in the A register are unchanged. Y is not affected.

Example 1: Extract 1-bits from the A register contents 2465317_8 according to the pattern 1234753_8 contained in GAP symbolic location MOD.

GAP Coding:

Symbol	Opr	Operand	X
	E X T	M O D	

Memory and A Register before Execution (binary):



Memory and A Register after Execution (binary):

MOD

0 1	0 1 0	0 1 1	1 0 0	1 1 1	1 0 1	0 1 1

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

1 0	1 0 0	1 0 0	0 0 1	0 0 0	0 0 0	1 0 0

A Reg

Example 2: Delete the dollar sign ($) from the BCD word $89 in the A register preparatory to storing the word into memory. Assume GAP symbolic location Memory and A Register before Execution (BCD): STRIP contains the BCD word $00.

GAP Coding:

Symbol	Opr	Operand	X
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	E X T	S T R I P	

Memory and A Register Contents (BCD)

	A Reg			STRIP		
Before execution:	$	8	9	$	0	0
After execution:	0	8	9	$	0	0

* MOV Y 2400000 Word Times: 4 + 2N

Functional Description: MOVE. A block of information starting at Y is moved to another area of memory. The A register must contain the starting address of the area to which the data is to be moved, and the Q register must contain the 2's complement of the number of words to be moved. The contents of the P counter are stored automatically in index word 00 (bits 5 through 19). The time required to execute this command is 4 plus 2N word times, where N is the number of words to be moved. After execution, the A register is set to 0's and the Q register contains the 2's complement of the number of words moved. This instruction cannot be automatically modified.

* This instruction is an optional feature.

Example: Move a block of 10 words initially stored in an area starting at symbolic location START to the memory area starting at symbolic location TOTALS. Assume that GAP has assigned the symbolic location START to actual address 0177_{10} and TOTALS to actual address 1200_{10}. Assume that the number of words to be moved has previously been loaded into the Q register in 2's complement form.

Memory and Register Contents in Octal:

Before execution:

Memory		Registers	
Octal Address	Contents	A	
0261	0123456	0002260	= 1200_{10}
0262	0246531		
0263	1234567		
0264	0765432		
0265	0135764	Q	
0266	2345670	3777766	= -10_{10}
0267	1001234		
0270	0132456		
0271	2147765		
0272	1777777		
2260	?		
2261	?		
2262	?		
2263	?		
2264	?		
2265	?		
2266	?		
2267	?		
2270	?		
2271	?		

Memory and Register Contents in Octal:

After execution:

Memory		Registers	
Octal Address	Contents		A
0261	0123456		0000000
0262	0246531		
0263	1234567		
0264	0765432		
0265	0135764		Q
0266	2345670		3777766
0267	1001234		
0270	0132456		
0271	2147765		
0272	1777777		
2260	0123456		
2261	0246531		
2262	1234567		
2263	0765432		
2264	0135764		
2265	2345670		
2266	1001234		
2267	0132456		
2270	2147765		
2271	1777777		

Data Transfers-Arithmetic

LAQ 2504001 Word Times: 3

Functional Description: LOAD A FROM Q. The contents of the Q register (S, 1-19) replace the contents of the A register (S, 1-19). Q is unchanged.

Example: Load A from Q, or replace the existing contents of A 1234567_8 with the contents of Q 3654321_8.

GAP Coding:

Symbol	Opr	Operand	X
	L A Q		

Register Contents in Octal

	A	Q
Before execution:	1234567	3654321
After execution:	3654321	3654321

Comments: No operand address is required. Automatic modification will change the instruction.

LQA 2504004 Word Times: 3

Functional Description: LOAD Q FROM A. The contents of the A register (S, 1-19) replace the contents of the Q register (S, 1-19). A is unchanged.

Example: Load Q from A, or replace the existing contents of Q 2465317_8 with the contents of A 1117776_8.

GAP Coding:

Symbol	Opr	Operand	X
	L Q A		

Register Contents in Octal

	A	Q
Before execution:	1117776	2465317
After execution:	1117776	1117776

Comments: No operand address is required. Automatic modification will change the instruction.

MAQ 2504006 Word Times: 3

Functional Description: MOVE A TO Q. The contents of the A register (S, 1-19) replace the contents of the Q register (S, 1-19). Zeros replace the contents of A (S, 1-19).

Example: Move A to Q, or replace the existing contents of the Q register 3777777_8 with the contents of the A register 1333444_8 and zero the A register.

GAP Coding:

Symbol	Opr	Operand	X
	M A Q		

Register Contents in Octal

	A	Q
Before execution:	1333444	3777777
After execution:	0000000	1333444

Comments: No operand address is required. Automatic modification will change the instruction.

XAQ 2504005 Word Times: 3

Functional Description: EXCHANGE A AND Q. The contents of registers A (S, 1-19) and Q (S, 1-19) are interchanged.

Example: Exchange A and Q, or interchange the contents of A 1234567_8 and Q 1777777_8.

GAP Coding:

Symbol	Opr	Operand	X
	X A Q		

Register Contents in Octal

	A	Q
Before execution:	1234567	1777777
After execution:	1777777	1234567

Comments: No operand address is needed. Automatic modification will change the instruction.

* LAC 2504202 Word Times: 3

Functional Description: LOAD A REGISTER FROM C REGISTER. The contents of the A register (1-19) are replaced by the contents of the C register (real time clock). The sign of the A register is set to zero. The contents of the C register are unchanged.

Example: Load A register from C register. Assume that the C register contains the binary equivalent of 1 hour ($52,140_8$ sixths of a second).

GAP Coding:

Symbol	Opr	Operand	X
	L A C		

Register Contents in Octal

	A	C
Before execution:	?	0052140
After execution:	0052140	0052140

* LCA 2504210 Word Times: 3

Functional Description: LOAD C REGISTER FROM A REGISTER. The contents of the C register are replaced by the contents of the A register (1-19) The sign of the A register contents is ignored. The contents of A are unchanged.

Example: Load C register from A register. Assume that the A register contains the binary equivalent of 12 hours ($259,200_{10}$ sixths of a second).

GAP Coding;

Symbol	Opr	Operand	X
	L C A		

* This instruction is part of the real time clock optional feature.

Register Contents in Octal

	A	C
Before execution:	0772200	?
After execution:	0772200	0772200

Initial Register Contents:

C	A	Q
1205701	?	?

Comments: The C register operates as a binary counter that is incremented by one every sixth of a second. When the binary count reaches the equivalent of 24 hours (518,400 sixths of a second), it automatically resets to zero and starts counting again.

The C register contents are not directly accessible for processing or console display. However, the LAC instruction, by transferring those contents to the A register, makes the C register available to the stored program or to the console operator.

A conversion subroutine is required for program translation of the C register contents from binary notation to hours, minutes, seconds, and sixths/seconds, and for print-out of elapsed or actual time through the control console typewriter.

A simple, straightline subroutine is shown below to illustrate how conversion could be done. In actual practice, a more sophisticated approach involving X registers and controlled looping would be more efficient.

Example: Convert the C register contents 1205701_8 to decimal hours, minutes, seconds, and sixth-seconds. Assume symbolic locations CON1 through CON3 contain conversion constants as follows:

Symbolic Location	Contents	Remarks
CON 1	$52{,}140_8$	Hours Factor
CON 2	550_8	Minutes Factor
CON 3	6	Seconds Factor

Opr	Operand	X	REMARKS
LAC			TRANSFER TIME TO A REG
MAQ			TRANSFER TIME TO Q REG FOR DVD
DVD	CON 1		COMPUTE HOURS
STA	HOURS		
LDZ			CLEAR A REG
DVD	CON 2		COMPUTE MINUTES
STA	MINS		
LDZ			
DVD	CON 3		COMPUTE SECONDS
STA	SECS		
XAQ			TRANSFER SIXTH-SECONDS TO A REG
STA	SXTHS		

Registers Affected by Each Instruction:

GAP Coding	Registers A	Registers Q
LAC	1205701	?
MAQ	0000000	1205701
DVD CON 1	0000017	0015041
STA HOURS	0000017	0015041
LDZ	0000000	0015041
DVD CON 2	0000022	0000321
STA MINS	0000022	0000321
LDZ	0000000	0000321
DVD CON 3	0000042	0000005
STA SECS	0000042	0000005
XAQ	0000005	0000042
STA SXTHS	0000005	0000042

Memory Contents after Conversion:

Symbolic Location	Contents
HOURS	0000017
MINS	0000022
SECS	0000042
SXTHS	0000005

The time represented by the C register contents can also be converted manually to a chronological scale by dividing those contents by appropriate conversion factors. Perhaps the simplest method would be to convert the binary contents of the C register to octal, then decimal, and divide by decimal conversion factors. The

conversion chart in Figure 2-4 makes the octal-to-decimal conversion easy. Decimal conversion factors used for division could be:

Hours = 21,600
Minutes = 360
Seconds = 6

(Any remainder would be in sixth-seconds.)

For example, assume that the contents of the C register are 1205701_8. By keying in an LAC instruction at the control console, 1205701_8 is displayed in the A register indicators. The octal-to-decimal conversion chart in Figure 2-4 provides the decimal equivalent 330,689 (in sixth-seconds).

Dividing by the hours conversion factor:

```
          15 hours
21600 ⌈330689
       21600
       114689
       108000
         6689  sixth-seconds remainder
```

Dividing the remainder by the minutes conversion factor:

```
          18 minutes
360 ⌈6689
     360
     3089
     2880
      209  sixth-seconds remainder
```

Dividing this remainder by the seconds conversion factor:

```
          34 seconds
6 ⌈209
    18
    29
    24
     5  sixth-seconds remainder
```

Thus, the C register contents 1205701_8 represent 15 hours, 18 minutes, 34 seconds, and 5 sixth-seconds, or 15:18:34:05.

Register Modifications

LDZ 2504002 Word Times: 3

Functional Description: LOAD ZERO INTO A REGISTER. The contents of the A register (S, 1-19) are replaced by zeros.

Example: Load zero into A register, or replace the existing contents of the A register 3777777_8 with zeros.

GAP Coding:

Symbol	Opr	Operand	X
	LDZ		

Register Contents in Octal

	A	Q
Before execution:	3777777	?
After execution:	0000000	?

Comments: No operand address is needed. Automatic modification will change the instruction.

LDO 2504022 Word Times: 3

Functional Description: LOAD ONE INTO A REGISTER. A 1-bit is placed in bit position 19 of the A register; all other bit positions (S, 1-18) are set to 0-bits.

Example: Load one into A register. Assume that the A register initially contains 3777777_8.

GAP Coding:

Symbol	Opr	Operand	X
	LDO		

Register Contents in Octal

	A	Q
Before execution:	3777777	?
After execution:	0000001	?

Comments: No operand address is needed. Automatic modification will change the instruction.

LMO 2504102 Word Times: 3

Functional Description: LOAD MINUS ONE INTO A REGISTER. The contents of the A register (S, 1-19) are replaced by 1-bits, giving the octal configuration 3777777_8.

Example: Load minus one into A register. Assume that the A register initially contains 1357642_8.

GAP Coding

Symbol	Opr	Operand	X
	L M O		

Register Contents in Octal

	A	Q
Before execution:	1357642	?
After execution:	3777777	?

Comments: No operand address is needed. Automatic modification will change the instruction.

CPL 2504502 Word Times: 3

Functional Description: COMPLEMENT A. Each bit position in the A register (S, 1-19) is inverted; each 1-bit is replaced by a 0-bit and each 0-bit is replaced by a 1-bit.

Example: Complement A register. Assume that the A register contains 1234567_8.

GAP Coding:

Symbol	Opr	Operand	X
	C P L		

A Register Contents in Binary

Before execution:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
0 1 | 0 1 0 | 0 1 1 | 1 0 0 | 1 0 1 | 1 1 0 | 1 1 1

After execution:

1 0 | 1 0 1 | 1 0 0 | 0 1 1 | 0 1 0 | 0 0 1 | 0 0 0
2 5 4 3 2 1 0

Octal Equivalent

Comments: No operand address is needed. Automatic modification will change the instruction.

NEG 2504522 Word Times: 3

Functional Description: NEGATE A. The 2's complement of the contents of the A register (S, 1-19) replaces the contents of A (S, 1-19). If the capacity of A is exceeded, in an attempt to negate the maximum negative number, overflow occurs.

Example: Negate A register contents 0000101_8.

GAP Coding:

Symbol	Opr	Operand	X
	N E G		

Register Contents in Octal

	A	Q
Before execution:	0000101	?
After execution:	3777677	?

Comments: Note that, unlike the CPL instruction which forms the 1's complement, NEG forms the 2's complement of the contents of A. No operand address is needed. Automatic modification will change the instruction. Overflow occurs if an attempt is made to negate the largest negative number, $-524{,}288_{10}$.

CHS 2504040 Word Times: 2

Functional Description: CHANGE SIGN OF A REGISTER. The sign bit of the A register is changed. Bit positions 1 through 19 of A are unchanged.

Example: Change sign of A register. Assume that the A register contains 1357642_8.

GAP Coding:

Symbol	Opr	Operand	X
	C H S		

Register Contents in Octal

	A	Q
Before execution:	1357642	?
After execution:	3357642	?

Comments: No operand address is needed. Automatic modification will change the instruction.

NOP 2504012 Word Times: 3

Functional Description: NO OPERATION. Zero is added to the contents of the A register (S, 1-19).

Example: No operation, or add zero to the contents 1234567_8 of the A register.

GAP Coding:

Symbol	Opr	Operand	X
	NOP		

Register Contents in Octal

	A	Q
Before execution:	1234567	?
After execution:	1234567	?

Comments: **This instruction is useful in programming delays or reserving space in a program for later insertion of an instruction. No operand address is needed. Automatic modification will change the instruction.**

SHIFT INSTRUCTIONS

Shift instructions involve the serial (bit-by-bit) movement of data within or between registers. Shifts fall into two categories: arithmetic register shifts and input-output register shifts.

Shifting is useful in arranging data before and after transfer between direct input-output peripherals, and the central processor, scaling quantities before and after arithmetic operations, recovering from overflow conditions, and performing simple multiplications and divisions.

Shifting is limited to 31 bit positions per shift instruction because bit position 15 through 19 of the instruction word are used to indicate the length of shift. With 5 bit positions, the largest number that can be expressed is 31.

A shift instruction can require from 2 to 12 word times for execution (including instruction access time), depending upon the length of shift. A shift of one bit position or less requires two word times. Each additional 3-bit shift, or fraction thereof, requires an additional word time.

Automatic modification of shift instructions changes the instruction.

Arithmetic Register Shifts

SRA K ≤ 31 2510000 Word Times: 2 to 12

Functional Description: SHIFT RIGHT A REGISTER. The contents of the A register (1-19) are shifted right K places. If A is plus, 0-bits are inserted in the vacated positions of A; if A is minus, 1-bits are inserted in the vacated positions. Bits shifted out of bit position 19 are lost. The sign of A is not changed.

Example 1: Shift right 3 bit positions the positive number 1234567_8, previously loaded into the A register.

GAP Coding:

Symbol	Opr	Operand	X
	SRA	3	

A Register Contents in Binary

Before execution:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	1	0	1	0	0	1	1	1	0	0	1	0	1	1	1	0	1	1	1
+	1		2		3			4				5				6			7

After execution:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	0	0	1	0	1	0	0	1	1	1	0	0	1	0	1	1	1	0
+	0		1		2			3				4				5			6

Example 2: Shift right 7 bit positions the negative number 3765432_8, previously loaded into the A register.

GAP Coding:

Symbol	Opr	Operand	X
	S R A	7	

A Register Contents in Binary

Before execution:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	1	1	1	1	1	0	1	0	1		1	0	0	0	1	1	0	1	0
		3		7		6		5				4			3			2	

After execution:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	1	1	1	1	1	1	1	1	1		1	1	1	0	1	0	1	1	0
		3		7		7		7				7			2			6	

Example 3: Divide by 8 the positive number $464{,}104_{10}$ (1612350_8), previously loaded into the A register.

GAP Coding:

Symbol	Opr	Operand	X
	S R A	3	

Register Contents in Octal

	A	Q
Before execution:	1612350	?
After execution:	0161235	?

$= 58013_{10}$

SLA K 2512000 Word Times: 2 to 12

Functional Description: SHIFT LEFT A REGISTER. The contents of the A register (1-19) are shifted left K places. Vacated bit positions of A are filled with 0-bits. If a non-zero bit is shifted out of position 1, overflow occurs and the bit is lost. The sign of A is unchanged.

Example 1: Shift left 2 bit positions the positive number 123456_8, previously loaded into the A register.

GAP Coding:

Symbol	Opr	Operand	X
	S L A	2	

A Register Contents in Binary

Before execution:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	0	0	1	0	1	0	0	1		1	1	0	1	0	1	1	1	0
+			1		2		3					4			5			6	

After execution:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	1	0	1	0	0	1	1	1		0	1	0	1	1	1	0	0	0
+			5		1		6					2			7			0	

Example 2: Shift left 5 places the negative number 2036361_8, previously loaded into the A register.

GAP Coding:

Symbol	Opr	Operand	X
	S L A	5	

A Register Contents in Binary

Before execution:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	0	0	0	0	0	1	1	1	0		0	1	1	1	1	0	0	0	1
	2		0		3		6					3			6			1	

After execution:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	1	1	1	1	0	0	1	1	1		0	0	0	1	0	0	0	0	0
	3		7		1		7					0			4			0	

Example 3: Multiply by 4 the positive number 1336_{10} (2470_8), previously loaded into the A register.

GAP Coding:

Symbol	Opr	Operand	X
	S L A	2	

Register Contents in Octal

	A	Q
Before execution:	0002470	?
After execution:	0012340	?

5344_{10}

SRD K 2511000 Word Times: 2 to 12

Functional Description: SHIFT RIGHT DOUBLE. The contents of the A and Q registers (1-19) together are shifted K places to the right. Bits shifted out of A (19) shift into Q (1). Bits shifted out of Q (19) are lost.

If the sign of A is plus (0), 0-bits fill the vacated positions. If the sign of A is minus (1), 1-bits fill the vacated positions. The sign of Q is replaced by the sign of A. The sign of A is unchanged.

When the instruction is written SRD 0, only the sign of A is shifted into the sign position of Q. There is no other data transfer.

Example 1: Shift right double 2 octal positions the contents of the A and Q registers. A contains 1234567_8; Q contains 3654321_8.

GAP Coding:

Symbol	Opr	Operand	X
	S R D	6	

Register Contents in Octal

	A	Q
Before execution:	1234567	3654321
After execution:	0012345	1576543

Example 2: Shift right double 2 bit positions the contents of the A and Q registers.

GAP Coding:

Symbol	Opr	Operand	X
	S R D	2	

Register Contents in Binary

Before execution:

A Reg

00	110	101	01 1	1 0 1	1 1 0	0 0 1

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

00	011	010	11 0	1 1 0	0 1 1	0 1 0

Q Reg

After execution:

A Reg

00	001	101	01 0	1 1 1	0 1 1	1 0 0

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

00	100	110	10 1	1 0 1	1 0 0	1 1 0

Q Reg

SLD K 2512200 Word Times: 2 to 12

Functional Description: SHIFT LEFT DOUBLE. The contents of the A and Q registers (1-19) together are shifted K places to the left. Bits shifted out of Q (1) shift into A (19). The vacated positions of Q are filled with 0-bits. If a non-zero bit is shifted out of A (1), overflow occurs and the bit is lost.

The sign of Q replaces the sign of A. The sign of Q is unchanged. (SLD 0 shifts only the sign of Q to A. There is no other data transfer.)

Example: Shift left double 4 bit positions the contents of the A and Q registers.

GAP Coding:

Symbol	Opr	Operand	X
	S L D	4	

Register Contents in Binary

Before execution:

A Reg

0	0	0	0	0	1	0	1	1	0	0	1	1	1	0	1	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1

Q Reg

After execution:

A Reg

0	0	1	0	1	1	0	0	1	1	1	1	0	1	0	1	0	0	1	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	1	1	0	1	1	1	0	1	0	1	1	0	1	1	0	0	0	0

Q Reg

SCA K 2510040 Word Times: 2 to 12

Functional Description: SHIFT CIRCULAR A REGISTER. The contents of the A register (1-19) are shifted right K places in a circular fashion; that is bits shifted out of position 19 are inserted in position 1, replacing bits as they are shifted out of position 1. The sign of A is unchanged.

Example: Shift circular A register contents 8 bit positions.

GAP Coding:

Symbol	Opr	Operand	X
	S C A	8	

Register Contents in Binary

Before execution:

A Reg

0	0	0	0	0	1	0	1	1	0	0	1	1	1	0	1	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	1	1	0	1	1	1	0	1	0	1	1	0	1	1	0	0	0	0

Q Reg

After execution:

A Reg

0	0	0	1	1	0	1	1	0	0	1	1	1	0	0	0	1	0	1	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	1	1	0	1	1	1	0	1	0	1	1	0	1	1	0	0	0	0

Q Reg

SCD K 2511200 Word Times: 2 to 12

Functional Description: SHIFT CIRCULAR DOUBLE. The contents of the A and Q registers (1-19) together are shifted K places to the right in a circular fashion. Bits shifted out of A (19) shift into Q (1) and those from Q (19) shift into A (1). The sign of A replaces the sign of Q. The sign of A is unchanged.

Example: Shift circular double 4 bit positions the contents of the A and Q registers.

GAP Coding:

Symbol	Opr	Operand	X
	S C D	4	

Register Contents in Binary

Before execution:

A Reg

0	1	1	0	0	1	1	1	0	0	0	1	0	1	0	1	1	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	0	1	1	0	0	0	1	1	1	0	1	0	1	0	0	1	1	0

Q Reg

After execution:

A Reg

0	0	1	1	0	1	1	0	0	1	1	1	0	0	0	1	0	1	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	1	0	0	1	0	0	1	1	0	0	0	1	1	1	0	1	0	1	0

Q Reg

Arithmetic Register Shifts

SAN K 2510400 Word Times: 2 to 12

Functional Description: SHIFT A AND N RIGHT. The contents of the A (1-19) and N (1-6) registers together are shifted K places to the right. Bits shifted out of A (19) shift into N (1).

Bits shifted out of N (6) are lost. If the sign of A is plus, 0-bits fill the vacated positions of A. If the sign of A is minus, 1-bits fill the vacated positions of A. The sign of A is unchanged.

Example: Shift A and N right 6-bit positions (1 BCD character).

GAP Coding:

Symbol	Opr	Operand	X
	S A N	6	

Register Contents in BCD

	A	N
Before execution:	$59	?
After execution:	0$5	9

Comments: While this instruction can be modified automatically, its use in a modified form is not recommended. However, if the length of the shift is modified by the contents of an X register, then the length of the shift, plus the contents of X, cannot exceed 31 places in any one shift instruction.

SNA K 2510100 Word Times: 2 to 12

Functional Description: SHIFT N AND A RIGHT. The contents of registers N (1-6) and A (1-19) together are shifted K places to the right. Bits shifted out of N (6) shift into A (1). Vacated positions in N are filled with 0-bits. Bits shifted out of A (19) are lost. The sign of A is unchanged. The N register must be 'ready' before this instruction is executed. See BNN and BNR instructions.

Example: Shift N and A right 6 bit positions (1 BCD character).

GAP Coding:

Symbol	Opr	Operand	X
	S N A	6	

Register Contents (BCD)

	A	N
Before execution:	123	8
After execution:	2012	0

ANQ K 2511400 Word Times: 2 to 12

Functional Description: SHIFT A INTO N AND Q. The contents of the A register (1-19) are shifted K places to the right into both registers N and Q. Bits shifted out of A (19) enter both Q (1) and N (1). Bits shifted out of N (6) and Q (19) are lost. If the sign of A is plus, the vacated positions of A are filled with 0-bits; if the sign of A is minus, 1-bits fill the vacated positions of register A. The sign of A replaces the sign of Q. The sign of A is unchanged. The N register must be 'ready' before this instruction is executed. See BNN and BNR instructions.

Example: Shift A into N and Q registers 6 bit positions.

GAP Coding:

Symbol	Opr	Operand	X
	A N Q	6	

Register Contents in BCD

	A	Q	N
Before execution:	123	???	?
After execution:	012	3??	3

NAQ K 2511100 Word Times: 2 to 12

Functional Description: SHIFT N, A, AND Q RIGHT. The contents of registers N (1-6), A (1-19), and Q (1-19) together are shifted K places to the right. Bits shifted out of N (6) shift into A (1). Bits shifted out of A (19) shift into Q (1). Bits shifted out of Q (19) are lost. Vacated positions of N are filled with 0-bits. The sign of A is unchanged. The sign of Q is set to the sign of A. The N register must be 'ready' before this instruction is executed.

Example: Shift N, A, and Q right 6 bit positions (1 BCD character).

GAP Coding:

Symbol	Opr	Operand	X
	N A Q	6	

Register Contents in Octal

	A	Q
Before execution:	888	???

N: 7

	A	Q
After execution:	788	8??

N: 0

NOR K 2513000 Word Times: 3 to 12

Functional Description: NORMALIZE THE A REGISTER. The effect of this instruction depends upon the value of K, the sign of the A register contents and R (the number of leading zeros in A).

If the A register sign is plus, and the number of leading 0-bits (R) in A (1-19) is less than K, the contents of A (1-19) are shifted left R places. The difference K-R replaces the contents of memory location 0000.

If the A register sign is plus, and the number of leading 0-bits (R) in A (1-19) is greater than or equal to K, then the contents of A (1-19) are shifted left K places; 0-bits replace the contents of memory location 0000 (15-19); bit positions (S, 1-14) of 0000 are always set to zeros. The sign of A is unchanged. Vacated positions of A are filled with 0-bits.

If the A register sign is minus, the number of leading 1-bits of A (1-19) are shifted left; otherwise execution occurs as described above. If a 1-bit is shifted from A (1), overflow occurs.

Example 1: Normalize the A register which contains 0001234_8 (9 leading 0-bits) to 10 bit positions (K = 10, R = 9).

GAP Coding:

Symbol	Opr	Operand	X
	N O R	1 0	

Memory and Register Contents in Octal

	A	0000
Before execution:	0001234	00000??
After execution:	1234000	0000001

Example 2: Normalize the A register which contains 0012345_8 (6 leading 0-bits) to 5 bit positions (K = 5, R = 6).

GAP Coding:

Symbol	Opr	Operand	X
	N O R	5	

Memory and Register Contents in Octal

	A	0000
Before execution:	0012345	00000??
After execution:	1162400	0000000

Example 3: Normalize the A register which contains the negative number 3776542 (9 leading 1-bits) to 6 bit positions (K = 6, R = 9).

GAP Coding:

Symbol	Opr	Operand	X
	NOR	6	

Memory and Register Contents in Octal

	A	0000
Before execution:	3776542	00000 ??
After execution:	3654200	0000000

Comments: The NOR instruction is used primarily in normalizing the A register in normalized floating-point arithmetic operations in the AAU. See Section XII.

NOR can be automatically modified; however, the length of a shift after modification must not exceed 31 places.

DNO K 2513200 Word Times: 2 to 12

Functional Description: DOUBLE LENGTH NORMALIZE. If the sign of the A register is plus, and the number of leading 0-bits (R) of A (1-19) is less than the constant (K), then the contents of registers A (1-19) and Q (1-19) are shifted left R places. K minus R replaces the contents of location 0000 (15-19).

If R is greater than or equal to K, then the contents of registers A (1-19) and Q (1-19) are shifted left K places; 0-bits replace the contents of memory location 0000 (15-19). Bit positions S, 1-14 of location 0000 are always set to zero. Bits shifted out of Q (1) shift into A (19). Vacated positions of Q are filled with 0-bits. The sign of Q replaces the sign of A. The sign of Q is unchanged.

If the sign of A is minus, the number of 1's of A (1-19) are shifted left; all other conditions are the same as when the sign of A is plus. If a 1 bit is shifted out of bit position 1, the overflow indicator is turned ON.

Example 1: Double length normalize the A and Q registers which contain 0001234_8 0076543_8 (8 leading 0-bits) to 6 bit positions (K = 6, R = 8).

GAP Coding:

Symbol	Opr	Operand	X
	DNO	6	

Memory and Register Contents in Octal

	A	Q
Before execution:	0001234	0076543
	0000	
	00000 ??	
After execution:	1123403	1654300
	0000	
	0000000	

Example 2: Double length normalize the A and Q registers which contain 0001777_8 0000177_8 (9 leading 0-bits) to 15 bit positions (K = 15, R = 9).

GAP Coding:

Symbol	Opr	Operand	X
	DNO	15	

Memory and Register Contents in Octal

	A	Q
Before execution:	0001777	0000177
	0000	
	00000 ??	
After execution:	1777000	0177000
	0000	
	0000006	

INTERNAL BRANCH INSTRUCTIONS

Branch instructions, which provide decision-making capability in the GE-225, fall into two categories: 1) internal branch instructions (described in this section) and 2) input-output branch instructions (described in appropriate peripheral instruction sections).

Internal branch instructions can be further subdivided into two groups: 1) unconditional branch instructions and 2) test-and-branch instructions.

Unconditional Branch Instructions

These instructions, when executed, unconditionally cause transfer of program control to the instruction contained in the memory location specified by the operand address. Operands can specify actual or GAP symbolic addresses.

BRU Y X 2600000 Word Times: 1

Functional Description: BRANCH UNCONDITIONALLY. Control is transferred to the instruction at memory location Y (Y becomes the address of the next instruction). If this instruction is modified automatically, all 15 bits of the P counter are altered by the sum of bits 7-19 of the I register and by bits 5-19 of the specified X register. If no modification, then only 13 bits of the P counter are altered.

Example: Branch unconditionally to the GAP symbolic location STORE. Assume that STORE has been assigned the octal address 1766_8 by GAP and that the BRU instruction is located in memory location 00460_8.

GAP Coding:

Symbol	Opr	Operand	X
0 0 4 6 0	B R U	S T O R E	

P Counter Contents in Octal

Before execution:	00461	
After execution:	01766	

Comments: Note that, before execution, the P counter has already been stepped to the address of the next sequential instruction. BRU modifies the P counter to transfer control to the instruction located in address 01766_8. Note that automatic address modification is possible.

SPB Y X 0700000 Word Times: 2

Functional Description: STORE P AND BRANCH. The memory location of the SPB instruction (held in bits 5-19 of the P counter) replaces the contents of bit positions 5-19 of the specified modification word (of the current modification group, for systems having the additional modification group feature). Bits 0-4 of the modification word are automatically set to zero. Control transfers to the instruction held in memory location Y. The P counter is not incremented during an SPB instruction.

Example: Store P and branch. Store the location of the SPB instruction 2676_8 in X register 3 and branch to the instruction held in GAP symbolic location RERUN. Assume that GAP has assigned octal location 0500_8 to the symbol RERUN.

GAP Coding:

Symbol	Opr	Operand	X
0 2 6 7 6	S P B	R E R U N	3

P Counter and X Register Contents in Octal

	P	0003
Before execution:	02676	???????
After execution:	00500	0002676

Comments: SPB cannot be automatically modified because bit positions 5 and 6 are used to specify the X register to receive the SPB memory location.

Test-and-Branch Instructions

A test-and-branch instruction causes a check of the status or contents of a central processor indicator or register to determine if the test condition is true or false. If the test is true (condition exists), the central processor executes the next sequential instruction; if the test is false (condition does not exist), the central processor skips the next instruction and executes the second sequential instruction.

The tested registers are unchanged by the test; tested indicators may or may not change, depending upon the test and the indicator status. Test-and-branch instructions affect only the P counter. If the condition tested is true, the P counter is automatically increased by one, as in non-branch instructions; if the condition tested is false, the P counter is increased by two, thereby skipping an instruction.

Test-and-branch instructions require no operand address; they can be followed sequentially by a BRU instruction specifying the transfer address. For convenience, GAP also permits the use of relative and symbolic addressing with test-and-branch instructions, as illustrated in the examples following the instruction descriptions.

BOV	2514003	Word Times: 2

Functional Description: BRANCH ON OVERFLOW. The overflow indicator is tested for the ON condition. If ON, the indicator is automatically turned OFF and the next sequential instruction is executed. If no overflow occurred, the second sequential instruction is executed.

BNO	2516003	Word Times: 2

Functional Description: BRANCH ON NO OVERFLOW. The overflow indicator is tested for the OFF condition (if overflow occurred, the indicator is automatically turned OFF).

If no overflow occurred the next sequential instruction is executed. If overflow occurred the second sequential instruction is executed.

BPL	2516001	Word Times: 2

Functional Description: BRANCH ON PLUS. The A register is tested for a plus sign in the sign bit position. If the sign is plus, the next sequential instruction is executed. If minus, the second sequential instruction is executed.

BMI	2514001	Word Times: 2

Functional Description: BRANCH ON MINUS. The A register is tested for a minus sign in the sign bit position. If the condition tested is true, the next sequential instruction is executed. If false, the second sequential instruction is executed.

BOD	2514000	Word Times: 2

Functional Description: BRANCH ON ODD. The A register is tested for an odd value; A (19) contains a 1-bit for all odd values.

BEV	2516000	Word Times: 2

Functional Description: BRANCH ON EVEN. The A register is tested for an even value; A (19) contains a 0-bit for all even values.

BZE	2514002	Word Times: 2

Functional Description: BRANCH ON ZERO. The A register contents (S, 1-19) are tested for 0-bits in all positions.

BNZ	2516002	Word Times: 2

Functional Description: BRANCH ON NON-ZERO. The A register contents (S, 1-19) are tested for 1-bits in any positions.

BPE	2514004	Word Times: 2

Functional Description: BRANCH ON PARITY ERROR. The parity alarm indicator is tested for the ON condition. If a parity error occurred, the indicator is automatically turned OFF and the next sequential instruction is executed; if no parity error occurred, the second sequential instruction is executed. Note: If the control console parity alarm switch is in the STOP ON PARITY ALARM position and a parity error occurs, the parity alarm indicator turns on and the central processor halts. If the parity alarm switch is in the NORM position, a parity error will turn on the parity alarm indicator but processing will continue. This permits programmed interrogation of the indicator with a BPE or BPC (below) instruction and optional branching to a corrective routine.

BPC	2516004	Word Times: 2

Functional Description: BRANCH ON PARITY CORRECT. The parity alarm indicator is tested for the OFF condition. If parity is correct, the indicator remains OFF and the next sequential instruction is executed. If a parity error occurred, and the parity alarm indicator is ON, it is turned OFF automatically and the second sequential instruction is executed. See Note under BPE, above.

Example: Test the A register contents for a positive value; if negative, test for an even value; if odd, test for zero; if not zero, store A in symbolic location RESULT. Assume quantity to be tested has previously been loaded into the A register and TEST begins in location 0211_8.

GAP Coding:

Symbol	Opr	Operand	X
TEST	BPL		
	BRU	PLUS	
	BZE		
	BRU	ZERO	
	BEV		
	BRU	EVEN	
	STA	RESULT	

Comments: If the number in the A register is positive, the P counter is not stepped and the instruction at TEST+1 causes a transfer to symbolic location PLUS. If the number tested is negative, the P counter is stepped to TEST+2, which causes the number to be tested for zero. If zero, again the P counter is not stepped and control transfers to symbolic location ZERO. If not zero, the P counter steps to TEST+4 and the number is tested for an even value. If even, the P counter is not stepped and control transfers to location EVEN. If not even, the P counter is stepped +1 and the contents of the A register are stored in symbolic location RESULT. One result of the series of instructions is to store only negative odd numbers in location RESULT.

---

* CAB Y 2100000 Word Times: 2 to 4

Functional Description: COMPARE AND BRANCH. The contents of the A register are compared algebraically with the contents of location Y. If the contents of Y are greater than the contents of A, the next instruction in sequence is executed. If the contents of Y are equal to the contents of A, the next instruction is skipped and the second sequential instruction is executed. If the contents of Y are less than the contents of A, the next two instructions are skipped and the third sequential instruction is executed.

Example: Compare the contents of symbolic location TEST with the contents of the A register. If TEST is greater than A, go to symbolic location MORE for next instruction. If TEST equals A, go to symbolic location EQUALS. If TEST is less than A, go to symbolic location LESS. Assume CAB is in location 0123_8.

GAP Coding:

Symbol	Opr	Operand	X
ANS	CAB	TEST	
	BRU	MORE	
	BRU	EQUALS	
LESS	ADD	34	

Registers Affected:

P Counter in Octal

Before execution: 0000123 = ANS

After execution:

Y > A 0000124 - ANS+1

Y = A 0000125 - ANS+2

Y < A 0000126 = LESS

---

* DCB Y 2200000 Word Times: 2 to 6

Functional Description: DOUBLE COMPARE AND BRANCH. The contents of the A and Q registers are compared algebraically with the contents of memory locations Y and Y + 1. If the contents of Y and Y + 1 are greater than the contents of A and Q, the next instruction in sequence is executed. If the contents of Y and Y + 1 are equal to the contents of A and Q, the computer skips the next instruction and executes the second sequential instruction. If the contents of Y and Y + 1 are less than the contents of A and Q, the computer skips the next two instructions and executes the third sequential instruction. Y should be an even location. If Y is odd, Y and Y are compared with the contents of A and Q. The signs of Y+1 and Q are ignored.

Comments: Both the DCB and the CAB instructions provide a 'three-way compare' capability. CAB provides of single-length word comparisons, while DCB

* This instruction is an optional feature.

compares double-length words. In both instructions the effect on the P counter is similar:

Y > A (or A and Q)	P unchanged
Y = A (or A and Q)	Step P + 1
Y < A (or A and Q)	Step P + 2

MODIFICATION INSTRUCTIONS

INX K X 1400000 Word Times: 3

Functional Description: INCREMENT X. This instruction adds the number K (bit positions 7 through 19 of the I register) to the contents of the specified X register (bit positions 5 through 19). The result replaces the contents of the X register (positions 5-19); any carry from position 5 is dropped. No automatic modification is possible. X register locations are 0000 through 0003, or 0000 through 0127, if the additional modification groups are available.

Example 1: Increment X register 0002, which contains 512_{10} (1000_8), by 1.

GAP Coding:

Symbol	Opr	Operand	X
	I N X	1	2

X Register Contents in Octal

	0002
Before execution:	0001000
After execution:	0001001

Example 2: Decrement X register 0003, which contains 100_{10} (144_8), by 6 (same as incrementing by 8186_{10} or 17772_8).

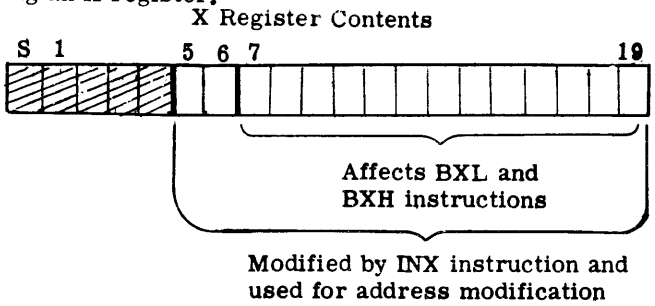
GAP Coding:

Symbol	Opr	Operand	X
	I N X	8 1 8 6	3

X Register Contents in Octal

	0003
Before execution:	0000144
After execution:	0020136

Comments: If INX is used to decrement the X register, a carry is generated into bit position 6. This 1-bit in position 6 does not affect BXH or BXL instructions (described later), because these commands compare bit positions 7 through 19 only. However, if the decremented contents of the X register are used to modify an address, the carry into position 6 will affect the modification. This is because X register bits 5 through 19 are used to modify the operand address. Also, INX should be used with caution to zero an X register; incrementing or decrementing the register by the quantity required to set it to zero actually sets the register to 8192 (1-bit in position 6). The LDA or LDX ZERO instruction is recommended for zeroing an X register.



BXH K X 0500000 Word Times: 3

Functional Description: BRANCH IF X IS HIGHER THAN OR EQUAL TO. If the contents of the X register (7-19) are greater than or equal to the constant K, the next sequential instruction is executed; if less than K, the second sequential instruction is executed. X is unchanged. No automatic modification is possible. X register locations are 0000 through 0003, or 0000 through 0127 if the additional modification groups are available.

Example 1: Branch if X is higher than or equal to 4. Assume that X register 0002, which contains 6, is to be used. Assume that BXH is in actual memory location 0163_8.

Symbol	Opr	Operand	X
	B X H	4	2
F I C A	B R U	0 7 7 7	
	S T A	T E M P	

	P Counter Contents in Octal and Symbolic	
Before execution:	0164	= FICA
After execution:	0164	= FICA

Example 2: Branch if X is higher than or equal to 4. Assume that X register 0002, which now contains a 3, is to be used. Assume that BXH is in actual memory location 0163_8.

GAP Coding:

Symbol	Opr	Operand	X
	BXH	4	2
FICA	BRU	0777	
	STA	TEMP	

	P Counter Contents in Octal and Symbolic	
Before execution:	0164	= FICA
After execution:	0165	= FICA+1

Comments: Note, in example 1, that because the tested condition is true, the P counter is not stepped to the second sequential instruction. Instead, the next instruction is the unconditional branch (BRU) which transfers control to the instruction at 0777. In example 2, the tested condition is false; that is, the X register contents are not higher than 4. Hence, the P counter, which has already been stepped once, is stepped again to 0165 and the unconditional branch is skipped.

A BXH instruction is generally, but not necessarily, followed by a BRU instruction specifying the address of the first instruction of the branch sequence.

If an optional modification word group is to be used, the BXH instruction must have been preceded by an SXG instruction, which selects the desired modification word group.

BXL K X 0400000 Word Times: 3

Functional Description: BRANCH IF X IS LESS THAN. If the contents of the X register (7-19) are less than the constant K, the next sequential instruction is executed; if greater than or equal to K, the second sequential instruction is executed. X is unchanged. No automatic modification is possible. X register locations are 0000 through 0003, or 0000 through 0127 if the additional modification word groups are available.

Example 1: Branch if X is lower than 5. Assume that X register 0003, which contains 6, is to be used. Assume that BXL is in actual location 0014_8.

GAP Coding:

Symbol	Opr	Operand	X
	BXL	5	3
MOD	BRU	1411	
	STA	TEMP	

	P Counter Contents in Octal and Symbolic	
Before execution:	0015	= MOD
After execution:	0016	= MOD+1

Example 2: Branch if X is lower than 5. Assume that X register 0003, which contains 2, is to be used. Assume that BXL is in actual location 0014_8.

GAP Coding:

Symbol	Opr	Operand	X
	BXL	5	3
MOD	BRU	1411	
	STA	TEMP	

	P Counter Contents in Octal and Symbolic	
Before execution:	0015	= MOD
After execution:	0015	= MOD

Comments: In example 1, the tested condition is false; that is, the X register contents are not lower than 5. Hence, the P counter is stepped an additional location, and the BRU instruction is skipped. In example 2, the tested condition is true; the X register contents are lower than 5. Thus, the P counter is not stepped and the next instruction executed is the BRU, which transfers control to the instruction at actual location 1411.

The BXL instruction is generally, but not necessarily, followed by a BRU instruction for the branch sequence.

If an optional modification word group is to be used, the BXL instruction must have been preceded by an SXG instruction, which selects the desired modification word group.

LDX Y X 0600000 Word Times: 3

Functional Description: LOAD X. The contents of memory location Y (S, 1-19) are loaded into register X (S, 1-19). Y is not affected.

Example: Load X with the contents of symbolic location SET1. Use X register 0003. Assume SET1 contains 0000001.

GAP Coding:

Symbol	Opr	Operand	X
	L D X	S E T 1	3

Memory and X Register Contents in Octal

	SET1	0003
Before execution:	0000001	?
After execution:	0000001	0000001

Comments: This instruction cannot be automatically address modified. X registers in optional modification word groups can be used, if LDX is preceded by an SXG instruction specifying the desired group. LDX is useful in initializing an X register.

STX Y X 1700000 Word Times: 3

Functional Description: STORE X. The contents of register X (S, 1-19) are stored in memory location Y. X is not affected.

Example: Store X register 0002 contents in symbolic location RESET. Assume 0002 contains 0135746_8.

GAP Coding:

Symbol	Opr	Operand	X
	S T X	R E S E T	2

Memory and X Register Contents in Octal

	RESET	0002
Before execution:	?	0135746
After execution:	0135746	0135746

Comments: This instruction cannot be automatically address modified. X registers in optional modification word groups can be used, if STX is preceded by an SXG instruction specifying the desired group.

---

* SXG Y 2506013 Word Times: 2

Functional Description: SELECT X REGISTER GROUP. The modification word group (00-31) specified by Y is selected and remains selected until another SXG instruction is given. After a given group is selected, all instructions referencing an X register will refer to one of the words within the selected modification group.

Example: Select X register group 27 so that subsequent instructions containing X modification coding (bit positions 5 and 6) will refer to memory locations 0108 through 0111.

GAP Coding:

Symbol	Opr	Operand	X
	S X G	2 7	

Subsequent Instruction Bit Positions 5	6	Modification Word Selected (Decimal)
0	0	0108
0	1	0109
1	0	0110
1	1	0111

* This instruction is an optional feature.

Comments: After execution of the SXG instruction, subsequent instructions containing 01, 10, or 11 in bit positions 5 and 6 will reference memory location 0109, 0110, or 0111 until another SXG instruction selects another modification word group. X register instructions (INX, BXL, BXH, LDX, and STX) containing 00, 01, 10, or 11 will reference memory locations 0108, 0109, 0110, or 0111. Note that the location specified by 00 X register coding (0108, in this case) has the same properties as location 0000.

The decimal locations of the modification words selected by the SXG are readily computed by multiplying the modification word group number by 4 and adding the X register coding of the instruction in question to the result.

For example, assume that an STA instruction specifies modification word 3 (11) and that a previous SXG instruction selected modification word group 18. To determine the actual location of the modification word, multiply 18 by 4 (giving 0072) and add 3 (giving location 0075).

PROGRAMMING 16K MEMORY SYSTEMS

The GE-225 information processing system is available with a 16k (16,384 word) memory which is regarded by programmers as being divided into two basic parts: the lower 8k memory and the upper 8k memory, referred to as the lower bank and the upper bank. The lower bank is considered to be memory locations 0000 through 8191, and the upper bank locations 8192 through 16,383. In programming 16k systems, accessing techniques and special restrictions as to instructions and software use must be considered.

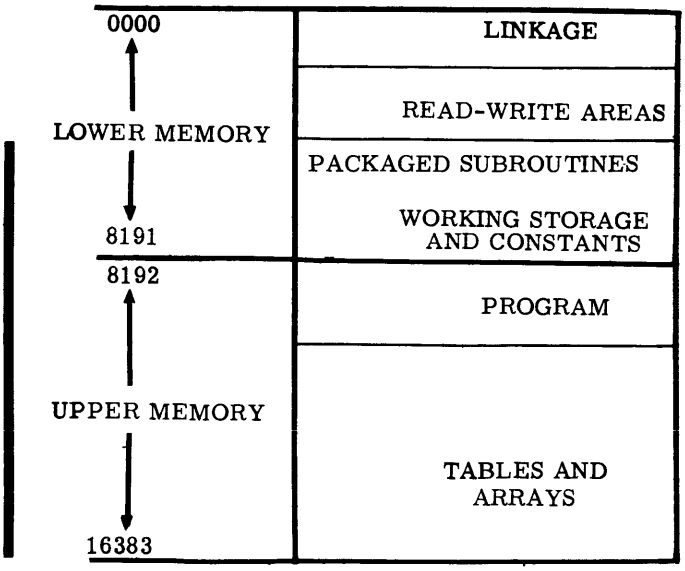


Figure 5-5. 16k Memory Layout

In addition, the proper allocation and use of memory becomes essential. Figure 5-5 illustrates an efficient and economical memory layout that allocates linkage, read-write areas, special subroutines, working storage and constants to lower memory and places the operating program and program subroutines in upper memory. Using memory in this way minimizes indexing or address modification operations.

Addressing the Upper Bank

In the 16k system, an operand address requires a fifteen-bit addressing capability, as opposed to a thirteen-bit 8k address. Thus, memory locations 00000 through 08191 in the lower bank can be addressed directly, but memory locations 08192 through 16383 must be accessed through address modification.

When modification is used, both the P and I registers which possess 15-bit address capability, are affected.

When an instruction is modified, the 15-bit constant in an index word (bits 5 through 19) is added to the 13-bit operand in the I register. After this addition, the instruction actually executed has an effective operand of 15 bits.

An example using address modification to access the upper bank is shown by the coding:

	Symbol	Opr	Operand	X	REMARKS
1	U P B N K	D E C	8 1 9 2		UPPER BANK CONSTANT
		⟨ ⟨			
2		L D X	U P B N K	2	SET INDEX TWO = 8192
		⟨ ⟨			
3		L D A	6	2	

The execution of the instruction in line two places the Constant 8192 in index word 2. The instruction of line 3 is modified by index word 2 and gives an effective address of 8192+6, or 8198, which is the desired upper bank memory location.

Index word 2 can now be used whenever access to data in an upper bank memory location is desired by the programmer. However, if the program is executing instructions in the upper bank, the P counter remains set for upper memory and is incremented in the normal manner without the need for modification.

Most GE-225 instructions access only memory locations in the lower bank when not indexed, but can access the upper bank when properly indexed. Figure 5-6 contains a brief description of the effect of GE-225 instructions when addressing 16k memories. Further explanations are given for specific commands.

Commands	Behavior
1. MOV and controller commands	1. Any memory location may be accessed with a 15-bit direct address.
2. General commands	2. The operand address is restricted or non-existent, independent of memory size.
3. Indexed BRU	3. Any memory location can be accessed through automatic address modification, and the P counter is set to obtain successive instructions from the memory bank selected by the BRU.
4. SPB and unindexed BRU	4. The 13-bit address applies only to locations in the memory bank in which the instruction is stored.
5. LDX and STX	5. The 13-bit address always applies to locations in the lower memory bank.
6. All others	6. Unindexed instructions access locations in the lower memory bank; indexed instructions may access any location via automatic address modification.

Figure 5-6. Instruction Characteristics when Addressing 16k Memories

Executing Instructions in the Upper Bank

Control can be shifted to instructions contained in memory locations in the upper bank of a 16k system by a suitably indexed BRU instruction. The effect of an indexed BRU is to set the two high-order address bits of the P counter. No other instruction may accomplish this (P automatically advances from 8191 to 8192 when no branch intervenes). Unindexed BRU instructions do not change the high-order address bits in the P counter. Also, an unindexed BRU causes subsequent instructions to be taken from the bank containing the BRU. Control remains in the upper bank until the next indexed BRU is executed, despite intervening SPB and unindexed BRU instructions.

Example 1: Change control from the lower bank to memory location 12000 in the upper bank. Assume index word 2 contains the constant 08192.

GAP Coding:

Memory Location	Opr	Operand	X
1756	BRU	3808	2

Next Instruction Location is 12000.

Subsequent instructions executed are in the upper bank.

Example 2: Upper Bank Execution. Index word 2 contains 08192.

GAP Coding:

Memory Location	Opr	Operand	X
12250	BRU	3808	2

Next Instruction Location is 12000

Execution of instructions continues in upper bank.

Example 3: Upper Bank Execution. Index word 2 contains 00000.

GAP Coding:

Memory Location	Opr	Operand	X
12250	BRU	3808	2

Next Instruction Location is 03808

Controls are changed to the lower b. memory location 03808.

In summary, it is essential that the progr ..ner remember:

1. Only a modified BRU instruction can direct the central processor to begin executing instructions in the upper bank. The BRU must be modified by the necessary increment, as illustrated in example 1, above.

2. Once operating in the upper bank, subsequent BRU instructions do not change the setting of bits 5 and 6 of the P counter unless another properly indexed BRU instruction is encountered. Also, once operating in either the lower or upper bank it is not necessary to continue indexing to keep control in that bank. Modification is only necessary when branching from one memory bank to another.

SPB Instructions

An SPB instruction can be used, at no increase in word time, in the upper bank to refer to an upper bank subroutine. However, an SPB instruction in the upper bank cannot be used to refer to a subroutine in the lower bank without first modifying a BRU instruction. The same rule exists with respect to using an SPB instruction in the lower bank to refer to a subroutine in the upper bank.

Example: Assume index word 2 contains 08192. Use an SPB and BRU in the lower bank to access a memory location in the upper bank.

GAP Coding:

Memory Location	Symbol	Opr	Operand	X
1750		SPB	UPPER	1
1751	UPPER	BRU	3808	2

Controls are changed from the lower bank to the upper bank with the instruction in memory location 12000 being executed next. The return from the upper bank routine (after execution) to lower bank memory location 01752 can be accomplished by a BRU:

GAP Coding:

Memory Location	Opr	Operand	X
12120	BRU	2	1

Next Instruction Executed is 01752

An SPB command executed in the upper bank performs exactly like an nonindexed BRU.

Example:

GAP Coding:

Memory Location	Opr	Operand	X
12225_8	SPB	2000	1

Next Instruction Executed is 10192.

The effective address of the next instruction executed (10192) is formed by bits 7 through 19 of the I register, plus bits 5 and 6 of the P counter with bits 5 through 19 of P stored in the index word.

The programmer should note that since only SPB and BRU instructions have operand addresses which relate directly to P counter contents, only the perform as described in the previous paragraphs. All other GE-225 instructions with 13-bit operands access locations in the lower bank unless they are appropriately indexed for the upper bank, regardless of where they are located.

LDX and STX Instructions

Index words are normally set and stored with LDX (Load Index) and STX (Store Index) instructions. These instructions transfer a 20-bit GE-225 word between a specified memory location, for which a 13-bit operand address is provided, and a specified index word. Since the index word selected represents a sending or receiving location in a data transfer process, automatic address modification does not occur on LDX and STX operand addresses. The 13-bit address field means that LDX and STX instructions may access only locations 00000 through 08191. Although these instructions may be stored in and executed from the upper bank, they always refer to data stored in the lower bank.

Example:

GAP Coding:

Memory Location	Opr	Operand	X
12250	LDX	6500	2
6500	DEC	0	

STO Instruction

The STO instruction is used for direct instruction address modification. Since the standard operand address field is thirteen bits, STO is designed to replace the low-order thirteen bits in the specified memory location with the low-order thirteen bits of the A register. In 8k memories, STO has virtually no special limitations. In 16k memories, STO cannot handle MOV or controller commands addressing the upper bank, nor is it adequate for direct address modification in other instructions when the address being stored is (or may be) in the other bank.

Example: **The contents of index word 2 = 08192.**

GAP Coding:

Memory Location	Opr (8 9 10)	Operand (12 13 14 15 16 17 18 19)	X (20)
12160	L D A	3 0 0 0	2
12161	S P B	* + 1	1
12162	S T O	2	1
12163	A D D	0	

Designing Subroutines for 16K Memories

Like 8k programs, subroutines and other program elements in lower 8k can access data and constants and set program switches without employing index registers. Subroutines in the upper bank must either use indexes or utilize the lower bank for data, constants, and switches. LDX and STX are essential for indexing procedures when extra index groups are employed. But LDX and STX can only access the lower bank. It is very important to remember this fact when designing subroutines for the upper bank. Therefore, constants should always be in the lower bank. Subroutines in general contain their own constants and working storage areas. If they are to be assembled into the upper bank, they must employ indexes to refer to such values, and they must do so without LDX and STX. One of two rules is necessary: either subroutines are located in the lower bank, or else subroutines are written to employ a specific index group, whose absolute core locations are used in LDA and STA instructions with LDX and STX prohibited.

16K Memories and Prior Software

Subroutines which have been written for the GE-225 with 8k memories in mind must usually be modified in order to function properly with 16k memories. There are several reasons for this:

1. Negative indexing, if used, is accomplished by simply adding the 2's complement of the desired decrement so that a carry is generated into bit position 6. This bit is effective during address modification because bits 5 through 19 are transferred during modification. Programs which use negative indexing do not perform properly when they are run on 16k systems.

2. The STO instruction can be employed extensively to set up data buffer addresses in pertinent commands in input-output subroutines. STO does not handle 14-bit addresses, so that such routines must either be modified or else be restricted to buffers in the lower 8k bank.

3. Subroutines usually contain their own constants and working storages, and do not access them with the aid of index registers. They, therefore, must be located in the lower bank.

4. Subroutines which call other subroutines have not been designed to go through a 'branch relay' process. Therefore, nested subroutines must all be placed in the same memory bank, presumably the lower bank.

5. Indirect arguments are often processed with the use of the STO instruction. Subroutines which have employed this mechanism either must be modified or else must restrict their indirect arguments to the lower bank.

6. Subroutines frequently have used the LDX and STX instructions which can only access the lower bank.

7. In general, most existing routines and even basic card formats assumed a 13-bit operand address field. The 16k memories require fourteen bits for the operand address field.

Programming for 16K Memories

The following list represents a summary of important points to be remembered when programming the GE-225 with a 16k memory:

1. Unindexed instructions, such as LDA, STA, and ADD, access the lower bank only.

2. Operand addresses of MOV and controller commands cannot be indexed but contain the full 15-bit direct addresses.

3. Some subroutines work only in the lower bank and some only in index group zero.

4. An SPB instruction does not cross the memory interface (lower-to-upper or upper-to-lower) directly.

5. Subroutines and other program elements must not straddle the memory interface; that is, they should

be located entirely in either the lower or upper bank (subject to the restriction in item 3 above).

6. Instructions LDX and STX always function as if only the lower bank were present.

7. STO stores only 13-bit operand address fields.

PROGRAMMING CENTRAL PROCESSOR OPERATIONS

Figure 5-7 illustrates a portion of the flow charting for a rejected parts cost program. GAP coding sheets corresponding to that portion of the flow chart are shown in Figures 5-8 through 5-11. The coding shown was chosen to illustrate typical usage of central processor instructions rather than to show recommended methods for programming specific problems.

In Figure 5-8, lines 2 through 10 initialize the input and cost areas by storing zeros in the affected locations. Note the use of index word 2 to loop through lines 4 through 6 until the entire block of 200 locations, starting with symbolic address APART, is filled with zeros.

In Figure 5-9, lines 2 and 3, SW#3 is interrogated. If SW#3 is OFF (contains zeros), calculation of DAREA parts follows; if SW#3 is ON, the BNZ in line three transfers control to BYPASS (line 3, Figure 5-10), DAREA calculations are skipped, and EAREA calculations are made.

Line 20 of Figure 5-9 shows a typical method for exiting from the main program to a subroutine after making provision for return to the exit point upon completion of the subroutine. The SPB NPRIBD causes an unconditional branch to a Binary-to-BCD conversion routine beginning at symbolic location NPRIBD (not shown) and causes the P counter contents (location of the SPB) to be placed in index register 1. The final instruction of the NPRIBD subroutine is a BRU 0001, modified by index register 1, which returns control to the instruction following the SPB.

Following the EAREA parts calculation in Figure 5-10 is a test for overflow. If an overflow condition exists, line 11 causes the control location to be stored in modification word 1 and control transfers to OVRFLO, line 2 of Figure 5-10. After overflow recovery the BRU 0002, modified by index word 1 returns to the main routine, line 13, Figure 5-10.

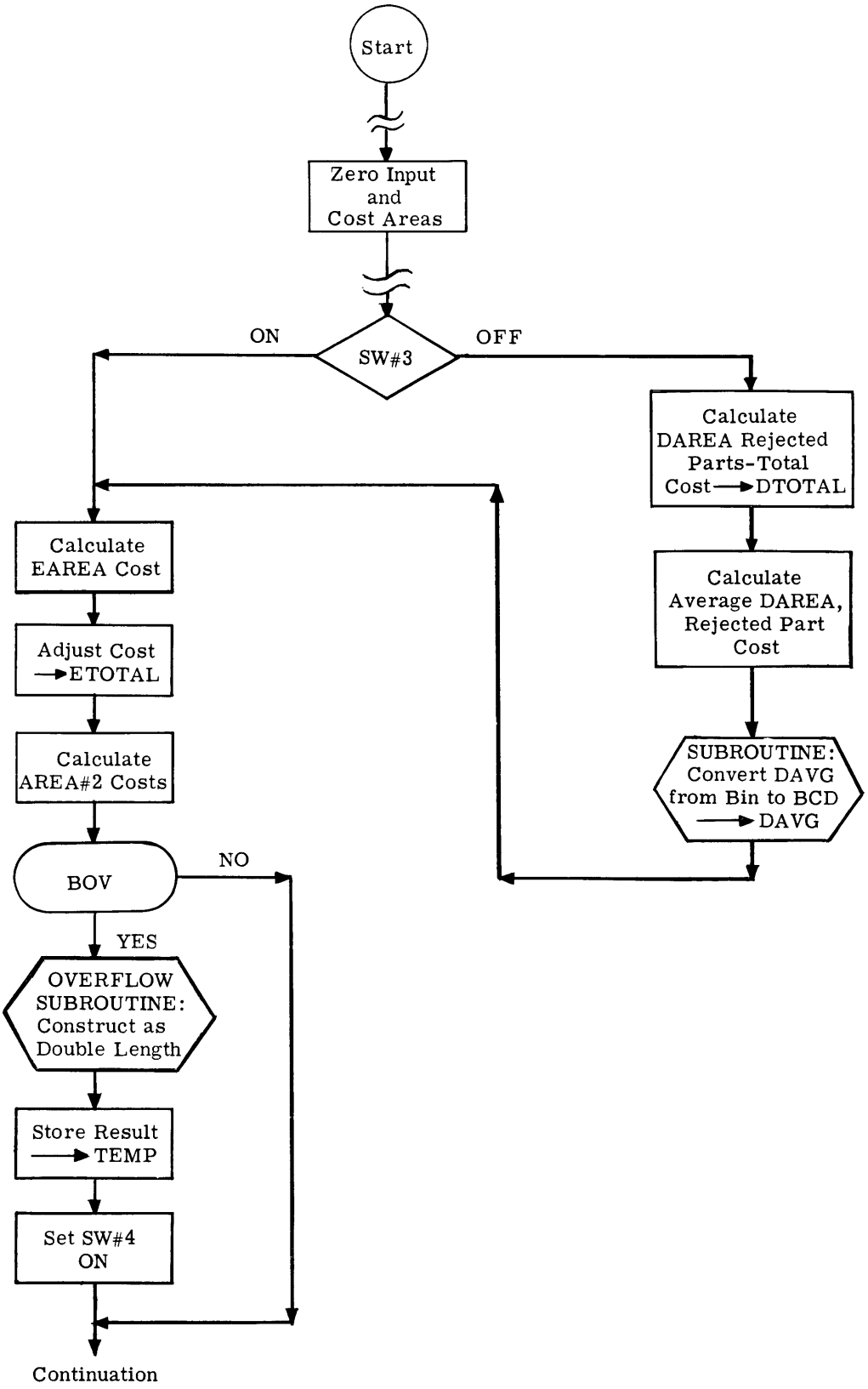


Figure 5-7. Rejected Parts Cost Flow Chart

GAP Coding:

	Symbol	Opr	Operand	X	REMARKS	Sequence
	PROGRAMMER: GE CODER		PROGRAM: Run #2 Rejected Parts Cost	DATE: 1/9/63	PAGE 4 OF 20	
1		ORG	1000		MAIN PROGRAM ORIGIN	1000
2	START	DLD	ZERO			1005
3		STA	2		ZERO INDEX WORD TWO	1010
4		DST	APART	2	ZERO INPUT AREAS	1015
5		INX	2	2		1020
6		BXL	200	2		1025
7		BRU	*-3			1030
8		LDX	ZERO	2		1035
9		DST	ACOST	2	ZERO COST AREAS	1040
10		INX	2	2		1045
11		$ $				

Figure 5-8. RPC Program - Initialization

GAP Coding:

	Symbol	Opr	Operand	X	REMARKS	Sequence
	PROGRAMMER: GE CODER		PROGRAM: Run #2 Rejected Parts Cost	DATE: 1/9/63	PAGE 6 OF 20	
1		LDX	ZERO	2	ZERO INDEX WORD TWO	1250
2		LDA	SW#3		SWITCH NO. 3	1255
3		BNZ	BYPASS		SKIP DAREA COST	1260
4		LDA	#DPART		NUMBER DAREA INDIVIDUAL PARTS	1265
5		NEG			CONVERT TO TWO'S COMPLEMENT FORM	1270
6		STO	LOOPD		SET UP NUMBER TIMES THRU LOOP	1275
7	DCALC	LDA	DPART	2	NUMBER OF EACH PART REJECTED	1280
8		MAQ				1285
9		MPY	DCOST	2	COST PER REJECTED PART	1290
10		XAQ				1295
11		ADD	DTOTAL			1300
12		STA	DTOTAL		TOTAL COST DAREA REJECTED PARTS	1305
13		INX	1	2		1310
14	LOOPD	BXL	0	2		1315
15		BRU	DCALC			1320
16		MAQ				1325
17		DVD	#DPART		CALCULATE AVERAGE DAREA COST	1330
18		DAD	ADJUST		ADJUST $	1335
19		MAQ				1340
20		SPB	NPRIBD	1	BIN-BCD CONVERSION ROUTINE	1345
21		STO	*+3			1350
22		ADO				1355
23		STO	*+3			1360
24		LDA	0		AVERAGE COST DAREA REJECTS	1365
25		STA	DAVG			1370

Figure 5-9. RPC Program - DPARTS Calculations

GAP Coding:

PROGRAMMER: GE Coder				PROGRAM: Run #2 Rejected Parts Cost	DATE: 1/9/63	PAGE: 7 OF 20
Symbol	Opr	Operand	X	REMARKS		Sequence
	LDA	0				1375
	STA	DAVG+1				1380
BYPASS	LDA	EPART		NUMBER EAREA PARTS		1385
	MAQ					1390
	MPY	ECOST		COST PER PART EAREA		1395
	XAQ					1400
	ADD	EADJ		EAREA ADJUSTMENT		1405
	STA	ETOTAL		TOTAL ADJUSTED COST EAREA REJECTS		1410
	ADD	AREA#2		CALC AREA#2 COSTS		1415
	BOV					1420
	SPB	OVRFLO	1	OVERFLOW SUBROUTINE		1425
	BRU	#2COST				1430
	DST	TEMP		TEMPORARY STORAGE		1435
	LDA	ONE				1440
	STA	SW#4		SET SWITCH 4 ON		1445
	BRU	#2COST				1450
	((					
DPART	BSS	30				
	REM			CONSTANTS AND SWITCHES		1735
ZERO	DDC	0				1740
ONE	DEC	1				1745
ADJUST	DDC	500				1750
SW#2	DEC	0				1755
SW#3	DEC	0				1760
SW#4	DEC	0				1765

Figure 5-10. RPC Program - EPARTS Calculations and Constants

GAP Coding:

PROGRAMMER: GE Coder				PROGRAM: Run #2 Rejected Parts Cost	DATE: 1/9/63	PAGE: 20 OF 20
Symbol	Opr	Operand	X	REMARKS		Sequence
	REM			OVERFLOW SUBROUTINE		2400
OVRFLO	SRD	1				2405
	CHS					2410
	SRD	18				2415
	BRU	2	1	EXIT		2420
	SBR	STRIP		BCD-BIN CONVERSION ROUTINE		2425
	SBR	NPRIBD		BIN-BCD CONVERSION ROUTINE		2430
	END	START				

Figure 5-11. RPC Program - OVRFLO Routine

SECTION VI

DIRECT INPUT-OUTPUT OPERATIONS

GE-225 peripheral units can gain access to memory either through the M and N registers or through the controller selector and then the M register, as shown in Figure 6-1. Peripherals connected to the M or N register are deemed to have direct access to memory and include the paper tape reader-punch, console typewriter, card reader, card punch, and the console switches. Operations involving these units are discussed in this section. Other peripheral operations, such as those involving the MRADS, high-speed printer, magnetic tape handlers, document handlers, and Datanet-15 terminals, are covered in the section, Controller Selector Operations.

CONTROL CONSOLE OPERATIONS

The control console is a control center from which the GE-225 operator has both manual control of processing and visual representation of the operating status of various registers and peripheral units.

Manual control includes the initial reading into memory of the program, starting program execution, and (as required) interrupting operation for checking or other purposes. Manual control is accomplished through the switches described on page VI-12. Visual

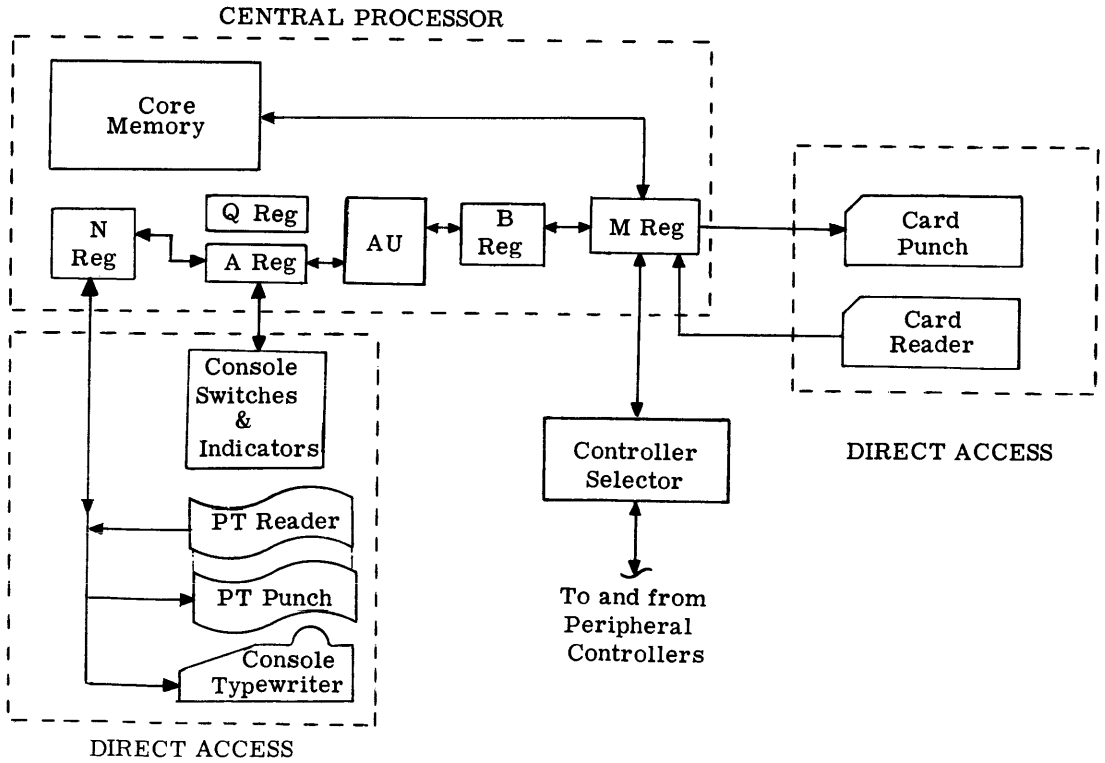


Figure 6-1. Units Directly Accessing Memory

representation of register contents and status of operational units is provided by various lensed lights, which are also described below. The control console consists essentially of a control and an indicator panel, as illustrated in Figure 6-2. The upper two-thirds of the panel contains most of the indicators, although many of the switches in the control position serve as indicators as well.

Alarm Indicators

At the top left of the console panel, Figure 6-2, are six alarm indicators. These are turned on if various error conditions are detected during program operation. All alarm indicators except the PRIORITY alarm are reset (turned off) by the RESET ALARM switch.

PRIORITY ALARM. This alarm is turned on under any of the following conditions:

1. The AUTO/MANUAL switch is in the MANUAL position.
2. The STOP ON PARITY ALARM switch is engaged and a parity error is detected.
3. The central processor does not have priority (access to memory).
4. A card punch or card reader alarm condition has occurred.

PARITY ALARM. If the STOP ON PARITY ALARM switch is on when a parity error is detected, the central processor will halt. The PARITY alarm can be turned off by pressing the RESET ALARM switch or, although not a common practice, by programmed instructions. The PARITY alarm is turned on under any of the following conditions:

1. The memory-checking circuits of the central processor detect a parity error while the AUTO/MANUAL switch is in the AUTO position.
2. The parity checking circuits associated with the paper tape reader detect a parity error.
3. A parity error is detected as information is received from a controller through the controller selector.

OVERFLOW ALARM. The central processor does not halt on an overflow alarm. The alarm may be reset automatically several times during a normal MPY instruction. The indicator can also be turned off by depressing the RESET ALARM switch or by programmed instructions. The OVERFLOW alarm is turned on under any of the following conditions:

1. The capacity of the A register is exceeded during arithmetic operations.
2. An illegal divide is attempted.

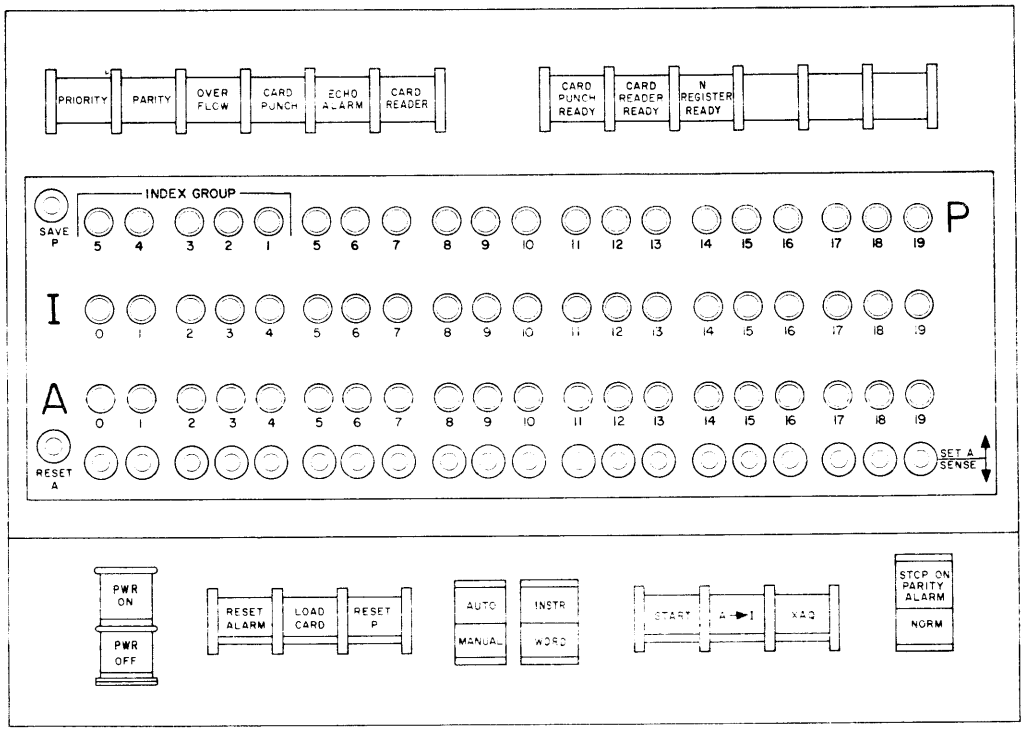


Figure 6-2. The Control Console Panel

3. A 1-bit is shifted out of bit position 1 of the A register during a shift left operation.

CARD PUNCH ALARM. This alarm is turned on any time a WCB, WCD, or WCF instruction is attempted when the card punch is not in the ready condition. As already noted, the PRIORITY alarm also comes on, and the central processor halts. The alarm can be reset only by pressing the RESET ALARM switch.

ECHO ALARM. This alarm is turned on when the central processor makes an unsuccessful attempt to select a controller through the controller selector. The ECHO alarm light can be turned off only by depressing the RESET ALARM switch. The alarm indicates any of the following conditions:

1. The selected controller is busy (delay not programmed).
2. An erroneous address was programmed, the addressed plug is not installed.
3. Controller is off line.
4. Power is off to controller.
5. Controller is malfunctioning.

CARD READER ALARM. This alarm is turned on when attempting to execute an RCB, RCD, or RCF instruction while the card reader is not in the ready condition. When the CARD READER alarm comes on, the PRIORITY alarm also comes on and the card reader and the central processor halt. The alarms in this combination are reset only by depressing the RESET ALARM switch. The reader can be 'not ready' for any of the following reasons:

1. Card reader is not turned on.
2. Input hopper is empty.
3. A card is not positioned on the sensing platform.
4. Reader is busy (already reading a card).
5. A misfeed or card jam occurs.

Ready Indicators

The upper right corner of the control console contains the ready indicators which are green. When the card punch or card reader is ready to receive information these indicators are on. If the equipment is not ready for operation, an attempt to use the equipment will set an alarm indicator and halt central processor operation. The standard ready indicators are:

CARD PUNCH READY. This light reflects the status of the card punch. If the card punch is not in an operable condition when a punch instruction is attempted, the ready light will be off and the CARD PUNCH and PRIORITY Alarms will come on. The more common conditions affecting the operating status of the card punch are:

1. An empty input hopper.
2. A full stacker.
3. A misfed card.
4. A jammed card.
5. A punch cycle.
6. An improperly seated chip box which inhibits the turn on of power.

CARD READER READY. Turn on of this indicator denotes the ready state of the card reader. Execution of a read instruction while this lamp is off causes the CARD READER and PRIORITY Alarms to light and the central processor to halt. The following conditions affect operating status:

1. An empty input hopper.
2. A read cycle.
3. A misfeed.
4. A jam.

N REGISTER READY. This lamp indicates the readiness of the N Register to receive input or transfer output data. This register is used by the typewriter, paper tape reader, or paper tape punch. If an illegal code is placed in the N Register and a TYP command is given, the N REGISTER READY light goes out and stays out until a space key is struck.

AIM (AUTOMATIC INTERRUPT MODE). If the GE-225 system configuration includes the optional Automatic Program Interrupt device, then this light (when ON) indicates that control has been transferred to an executive routine for servicing one or more peripherals in a ready condition.

8K. This is the only red lamp in the group. When lit, this lamp indicates that only an 8K memory is in use.

DECIMAL MODE. If the Decimal Mode optional feature is included, this indicator will come on when the computer operates in the decimal mode.

MODIFICATION GROUP INDICATORS

The five INDEX GROUP display lights are located below the alarm lights and to the left of the P counter display lights. The lights are numbered one through five from right to left. These five lights, read as binary digits, indicate the modification word group that has been selected by the program (Groups 0 through 31). Each group has four registers, 0 through 3. When all lights

are off, group zero is available without special selection. Only modification word group zero is standard on the GE-225 system; additional groups are optional. Any time a light is on in the index group, an index group other than zero has been selected.

P Counter Lights

The fifteen display lights for the P counter are located to the right of the INDEX GROUP indicators. They are numbered, left to right, from 5 through 19, and are arranged in groups of three to facilitate reading the binary numbers directly in octal notation. These lights show the location of the instruction which appears in the I register. The P counter is useful when debugging a program and when checking for correct operation after a manual branch command to a particular program location.

Save P Switch

This switch permits manual return to a particular position in the program after interruption to make a correction, such as to introduce an instruction manually. The SAVE P switch, in the down position, prevents the P counter from incrementing. When the SAVE P switch is returned to the up (normal) position after manual operations, the program is ready to continue from the place of interruption. When the SAVE P switch is in the down position during the automatic mode of operation, the instruction in the I register is executed repeatedly.

I Register Lights

The 20 I register display lights are located below the INDEX GROUP and P counter lights, and are numbered from 0 to 19. They display the contents of the instruction register. Like the other register display lights, they are easily read in octal notation. Following either a program halt or a change of the AUTO/MANUAL switch to the MANUAL position the I Register displays the next instruction to be executed.

A Register Lights

The 20 A register display lights are located below the I register lights. They are numbered from 0 to 19, and display the contents of the A register. These are also readable in octal. By using the XAQ switch (described later), the A register lights can be used to display the contents of the Q register. All data and instructions fed manually into the central processor go through the A register, and are entered by use of the option switches.

Option Switches

The 20 option or control switches just below the A register display lights are used to feed information into the A register. Each of these toggle switches enters information into the corresponding A register position. The numbers 0 through 19 below the A register lights also apply to the switches. When moved up, the spring-loaded switches return automatically to the center (normal) position. When moved down, they remain in the down position until manually returned to the normal position.

When the central processor is in the manual mode, moving an option switch up causes a 1-bit to be put into the corresponding position of the A register. This is indicated by an A register display light. Moving an option switch up has no effect when the central processor is in the automatic mode.

Moving an option switch down when the central processor is in the automatic mode causes a 1-bit to be put into the corresponding position of the A register at the time of a programmed RCS instruction. Specified switches are left in the down position while running certain routines and while generating GAP assemblies.

RESET A Switch

This switch is to the left of the option switches. It is effective only when the central processor is in the manual mode. Like the option switches, it is spring-loaded in the up position, but not in the down position. When moved either up or down, it clears to zero the contents of the A register, and turns off all of the A register display lights.

Control Switches

A strip of switches along the bottom of the control console, and the SAVE P and RESET A switches just described, give manual control over the central processor and certain functions of peripherals. Eight of the switches are the pushbutton type that are pressed momentarily to be activated. Three double-label switches are the rocker type with two positions. For example, the AUTO/MANUAL SWITCH is placed in the AUTO position by pressing the end that is labeled AUTO and leaving that end in the depressed position.

PWR. ON. Depressing the PWR ON pushbutton turns on DC power to the central processor, the control console, and the 400 card per minute reader. It is also used as general reset for the central processor. The pushbutton is also an indicator, for it lights when power is on.

PWR. OFF. When DC power is on, depressing this pushbutton turns it off.

RESET ALARM. This switch is effective only in the manual mode. Depressing the pushbutton clears any existing alarm condition. It turns off the alarm lights and resets flip-flops so that the central processor can continue operation. It does not clear the cause of the alarm.

LOAD CARD. This switch is effective only in the manual mode. Depressing the pushbutton initiates card reader action and causes the reader to go through one load and read cycle.

RESET P. This switch is effective only in the manual mode. Depressing the pushbutton clears the P counter.

AUTO/MANUAL. This two-position, rocker switch selects either the automatic or the manual mode of operation for the central processor. When AUTO is depressed, the central processor is placed in the automatic mode, and instructions are processed in a continuous sequence under program control. When MANUAL is depressed, the central processor is placed in the manual mode, and the program is executed one step each time that the START switch is depressed. Setting the AUTO/MANUAL switch to MANUAL during automatic operation causes the computer to halt operations at the end of the instruction or word being executed. Putting the central processor in the manual mode causes the PRIORITY alarm light to come on. The following operations can be performed only when the AUTO/MANUAL switch is set to MANUAL:

1. Clear or set information into the A register with option switches.
2. Clear alarm conditions with the RESET ALARM switch
3. Reset the P counter with the RESET P switch.
4. Load a card manually, using the LOAD CARD switch.
5. Transfer the contents of the A register to the I register using the A to I switch.
6. Exchange the contents of the A and Q registers using the XAQ switch.

INST/WORD. This is also a two-position, rocker switch which is effective only in the manual mode. It determines the length of the cycle of the central processor during manual operations. When INST is depressed, the central processor executes one complete instruction each time the START switch is engaged. When WORD is depressed, only one word time is executed each time the START switch is engaged.

START. In the automatic mode, depressing the START pushbutton initiates action. After the operation begins, the program runs automatically and depressing the START switch again has no effect. In the manual mode, depressing the START switch causes the execution of one instruction or one word time, depending upon the setting of the INSTR/WORD switch.

A→I (A to I). This switch is effective only in the manual mode. Depressing the A to I pushbutton transfers the contents of the A register, including the sign bit, to the I register. The contents of the A register remain unchanged, and can be cleared by toggling the RESET A switch. The A to I switch can be used to load an instruction manually into the I register or to correct an instruction already there.

XAQ. This switch is effective only in the manual mode. Depressing XAQ causes an exchange of information between the A and Q registers. That is, the contents of A go into Q and the contents of Q go into A. This permits observation/modification of the contents of the Q register. By using the RESET A switch and the option switches, the operator can clear and correct the contents of the Q register while saving the contents of the A register.

STOP ON PARITY ALARM/NORM. This is a two-position, rocker switch. It determines the response of the central processor to the detection of a parity error. When STOP ON PARITY ALARM is depressed, the central processor halts each time a parity error is detected and the PARITY and PRIORITY alarm lights come on. When NORM (normal) is depressed, the central processor continues operation, regardless of parity errors, and the only indication of a parity error is that the PARITY alarm light is turned on. The setting of the STOP ON PARITY ALARM/NORM switch is determined by the programmer. If he has included remedial action throughout the program for parity errors and provision for resetting the PARITY alarm light, he can specify the setting of the STOP ON PARITY ALARM/NORM switch to the NORM position. Otherwise, he can have the program halt at time of a parity error by specifying the setting of STOP ON PARITY ALARM.

Manual Operating Procedures

The option switches on the console permit the manual entry of instructions and data; the register indicators permit the display of the contents of memory and registers.

MANUAL LOAD AND EXECUTION OF INSTRUCTIONS. Any instruction that is meaningful to the GE-225 system can be manually loaded and executed as follows:

1. Set the INSTR/WORD switch to INSTR.
2. Set the AUTO/MANUAL switch to MANUAL.
3. Toggle the RESET A switch to clear the A register.
4. Load the octal equivalent of the instruction into the A register.
5. Depress the A to I switch.

6. Toggle the RESET A switch and load any necessary data into the A register.

7. Depress the START switch.

The central processor will execute the one instruction and halt.

LOADING DATA MANUALLY. When data is to be loaded into memory, the following procedure is useful:

1. Set the INSTR/WORD switch to INSTR.

2. Set the AUTO/MANUAL switch to MANUAL.

3. Toggle the RESET A switch.

4. Load an STA instruction in the A register (Store A is an octal 0300000) with the memory address where the data is to be stored replacing the right-hand digits of the STA instruction.

5. Depress the A to I switch.

6. Toggle the RESET A switch.

7. Load the octal equivalent of the data to be stored into the A register.

8. Depress the START switch.

Load additional words by repeating steps 3 through 8.

EXTRACTING DATA FROM MEMORY. The contents of a given memory location can be displayed by following this procedure:

1. Set the INSTR/WORD switch to INSTR.

2. Set the AUTO/MANUAL switch to MANUAL.

3. Toggle the RESET A switch, thus leaving an LDA instruction in the A register.

4. Load the memory location of the information desired into bit positions 7 through 19 of the A register.

5. Depress the A to I switch.

6. Depress the START switch.

The contents of the memory location specified in step 4 now appear in the A register.

Control Console Instruction

This instruction permits operator intervention. It can be used in programs in which alternate paths of operation are available. Job requirements may vary daily for one type of run, necessitating that the operator determine which path or leg of the program is to be followed. For example, one program path may be for card input and tape output, while the alternate path provides for both tape and printer output.

RCS	2500011	Word Times: 2

Functional Description: READ CONTROL SWITCHES. Each of the 20 console control switches for the A register is examined. If a switch is down (ON), a 1-bit is placed in the corresponding position of A; otherwise, the corresponding bit position of A will not be altered.

Example: Read the control switches and modify the A register accordingly. Assume that the A register contains a BRU 0000 instruction and the control switches are set to 0001633_8.

GAP Coding:

Symbol						Opr			Operand									X
1	2	3	4	5	6	8	9	10	12	13	14	15	16	17	18	19	20	
						R	C	S										

	Register Contents in Octal
	A
Before execution:	2600000
After execution:	2601633

Comments: RCS is used to interrogate the control switches during processing. In most applications, the A register should be cleared to zero before RCS is executed.

During AUTOMATIC operations, the A register switches on the console have no effect on the contents of the A register, except during the time that the RCS command is in the instruction register. At that time, each of the 20 console switches is examined.

CONSOLE TYPEWRITER OPERATIONS

The console typewriter, Figure 6-3, is primarily an output device, which is normally located on the control console desk. It can be used to provide brief messages to the operator during program processing, or it can serve as a more extensive output medium in lieu of a high-speed printer.

The typewriter receives and types one character at a time from the N register. The six position N register, in turn, is loaded with one character at a time from the

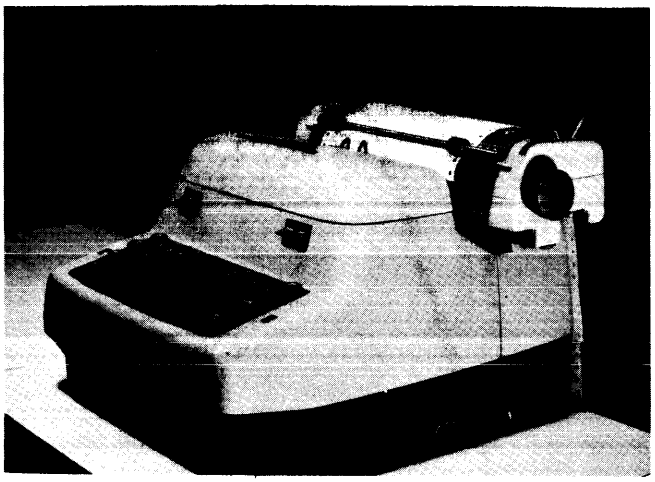


Figure 6-3. Console Typewriter

A register. The typewriter can print ten characters per second under program control. Typewriter capabilities include:

Red printout
Black printout
Print characters 0 through 9, A through Z, minus, period, slash, dollar sign, and comma
Carriage return
Space
Tabulation

Error messages are normally programmed to print in red. Figure 6-4 illustrates typewriter characters and actions and the corresponding octal codes.

Messages produced through the console typewriter can serve as a log of program performance. For this purpose, the typewriter can be programmed to record program identification, list magnetic tape labels, and provide instructions to the GE-225 operator. Operator comments can be inserted manually whenever the GE-225 is in a halt status (AUTO or MANUAL).

Required carriage returns must always be specified in the program. If returns are omitted, typing continues to the right margin stop; the carriage then halts, but typing continues, resulting in illegible messages. Typeouts involving tabulation require manual intervention. The operator must manually set required tab stops before running the program.

The typewriter shares access to memory through the N register with the paper tape reader and punch. Thus, if the N register is engaged because of a type operation, paper tape read or punch operations must be delayed until the N register is released. Also, electrical power can be on for only one of these three units at one time; if power is on for the paper tape reader, for example, then power is off for the paper tape punch and the typewriter. This permits an economy in the assignment of operation codes; the code 2500006_8 is used for type, read paper tape, and write (punch) paper tape.

Typewriter Character or Action	Octal Equivalent of BCD Codes
0	00
1	01
2	02
3	03
4	04
5	05
6	06
7	07
8	10
9	11
A	21
B	22
C	23
D	24
E	25
F	26
G	27
H	30
I	31
J	41
K	42
L	43
M	44
N	45
O	46
P	47
Q	50
R	51
S	62
T	63
U	64
V	65
W	66
X	67
Y	70
Z	71
-	40
Space	60
/	13
.	33
$	53
Carriage Return	37
Print Red	72
Print Black	75
Tab	76

Figure 6-4. Typewriter Character Set

Programmed use of the typewriter requires that the typewriter power on switch (under the right front corner of the typewriter) be turned on manually. In addition, at least 200 milliseconds before the first character is to be typed, a typewriter on instruction

must be given; the unit will remain on until a subsequent instruction (such as OFF, RON, or PON) turns off typewriter power.

Next, the N register must be tested for a ready status; if ready, then a shift to move the character to be typed into the N register may be given, followed by a TYP command. This sequence of test, shift, and type must be repeated for each character to be typed.

An optional feature enables the typewriter to be used as an input device, in addition to the described output function. The input feature enables one BCD character, as selected by a typewriter key, to be placed in the N register. The character can then be shifted into the A register for subsequent processing as desired.

The input feature is enabled by the operation code 2500016_8, which also serves as the halt paper tape (HPT) instruction. Normally, HPT has meaning only when the paper tape reader is on and is moving tape. Because typewriter and paper tape reader cannot operate concurrently, there is no disadvantage to dual use of the 2500016_8 code.

To use the optional typewriter input feature, the typewriter must be ON. Issuing a HPT instruction enables the typewriter keyboard and causes the N register to become not ready. Depressing a typewriter key places the corresponding BCD character into the N register and returns the register to the ready state.

Typewriter Instructions

TYP	2500006	Word Times: 2

Functional Description: TYPE. If typewriter power is on, one BCD (six-bit) character in the N register is typed. The contents of N are unchanged.

Example: Examples of all typewriter instructions are provided in the coding sample following the last discussed typewriter instruction.

Comments: Execution of a TYP instruction does not affect the contents of any arithmetic register.

The TYP instruction is normally preceded by a shift of data into the N register from the A register, as well as by a test-and-branch (BNR or BNN).

The N register becomes busy during the execution of TYP and remains busy until typing of the character is completed.

No typewriter keys are activated when an attempt is made to type an illegal character (that is, a character not included in the typewriter character set as shown in Figure 6-4); in addition, the N register goes busy and must be cleared by manually typing a character or depressing the space bar.

Central processor operation is not delayed by the execution of a TYP. The next sequential instruction is initiated in the following word time, although typing may not be completed for several milliseconds.

The TYP instruction is used to control typewriter action other than typing. If the N register contains one of the following codes, the indicated actions occur:

N Register Contents (Octal)	Action
60	Space
76	Tab
37	Carriage Return
72	Print Red
75	Print Black

TON	2500007	Word Times: 2

Functional Description: TYPEWRITER ON. The typewriter power is turned on (if the typewriter power on switch is on) and power for the paper tape reader-punch is turned off.

Example: Examples of all typewriter instructions are provided in the coding sample following the last discussed typewriter instruction.

Comments: To allow the typewriter motor sufficient time to attain operation speed after a TON, a delay of at least 200 milliseconds should be programmed before executing a TYP instruction. However, if the TON is given within 1 millisecond after turning off the typewriter (with a programmed OFF, RON, or PON), no delay is required.

Unless the typewriter power is already ON, failure to program a TON instruction before TYP will cause the N register to become and remain not ready.

OFF	2500005	Word Times: 2

Functional Description: POWER OFF. The power supply for the typewriter and paper tape reader and punch is turned off.

Example: Examples of all typewriter instructions are provided in the coding sample following the last discussed typewriter instruction.

Comments: After an OFF is executed, subsequent TON, RON, or PON instructions will restore power on to the

respective units. If power is on for any one of the units (typewriter, paper tape reader, or paper tape punch), it is off for the other two.

BNN **2516005** **Word Times: 2**

Functional Description: BRANCH ON N REGISTER NOT READY. If the N register is not available for input or output (that is, if a previous type, read paper tape, or write paper tape instruction has not been completely executed), the next sequential instruction is executed. If the N register is ready, the second sequential instruction is executed.

Example: Examples of all typewriter instructions are provided in the coding sample following the last discussed typewriter instruction.

Comments: The BNN instruction (or its counterpart, BNR) is used to insure that the N register is ready (not in use) before initiating a read or a punch paper tape operation, as well as before type operations.

BNR **2514005** **Word Times: 2**

Functional Description: BRANCH ON N REGISTER READY. If the N register is available for input or output (that is, if the last type, read paper tape, or write paper tape instruction has been completely executed), then the next sequential instruction is executed. If the N register is not ready, the second sequential instruction is executed.

Example: Examples of all typewriter instructions are provided in the coding sample following this instruction description.

Comments: The BNR instruction (like its counterpart, BNN) is used to insure that the N register is ready before initiating a read or a punch paper tape operation, as well as before type operations.

Typewriter Sample Coding

Prepared output routines are available to assist the programmer in preparing coding for typewriter printouts. These routines provide for single or multiple word output, red or black ribbon, punctuation, tabulation, and carriage returns.

To illustrate the use of the various instructions related to typewriter operations, a simple example is shown in Figure 6-5.

PROGRAMMER | PROGRAM | DATE | PAGE OF

	Symbol	Opr	Operand	X	REMARKS	Sequence
1	PREP	TON			TYPEWRITER ON	
2		LDZ				
3		STA	1		INITIALIZE X REGISTER 1	
4		INX	1	1		
5		BXL	1587	1	LOOP FOR 200 MS	
6		BRU	PREP+3			
7		LDZ				
8		STA	2		INITIALIZE X REGISTER 2	
9	TYPE	LDA	TAX		TYPEWRITER MESSAGE (3 CHARS.)	
10		SRD	12		SHIFT 2ND TWO CHARS. TO Q	
11		BNN				
12		BRU	*-1		TEST N REGISTER	
13		SAN	6		MOVE CHAR. TO BE TYPED TO N	
14		TYP			TYPE CHARACTER	
15		SLD	6		POSITION NEXT CHAR. IN A	
16		INX	1	2	COUNT CHARS. TYPED	
17		BXL	3	2	IF LAST CHAR., EXIT	
18		BRU	TYPE+2		LOOP TO TYPE NEXT CHAR.	
19	EXIT	LDA	RETURN		CONTAINS OCTAL 37	
20		BNN				
21		BRU	*-1		TEST N REGISTER	
22		SAN	6			
23		TYP			OCTAL 37 RETURNS CARRIAGE	
24		OFF			TURNS OFF TYPEWRITER POWER	
25						

Figure 6-5. Sample Typewriter Coding

As presented, the program assumes that a three-letter word to be typed is in symbolic location TAX and that an octal 37 (carriage return) is in location RETURN. Further, it is assumed that the manual power on switch on the typewriter has been turned on.

Line 1 of the GAP Coding Sheet turns on the typewriter. Lines 2 through 6 contain coding that sets up X register 1 to operate as a counter, then counts through the INX, BXL, BRU loop 1587 times to insure that at least 200 milliseconds (to allow the typewriter motor to reach operating speed) pass before a TYP is initiated.

Lines 7 and 8 prepare X register 2 to operate as a character counter during the following TYP operation.

The 3-character message (in BCD) is loaded into the A register (line 9) and then shifted right, 2 characters, into the Q register in order to position the first character to be typed.

Lines 11 and 12 test the N register for ready status. If it is not ready, the program loops until it is. Line 13 shifts a character into the N register and it is typed (line 14).

X register 2 is incremented to indicate that the first character has been typed (line 16), then tested to see if typing is complete. If it is not, the program loops back to line 11 and repeats the sequence until the entire word (3 characters) has been typed.

Upon completion of typing, a carriage return (octal 37) is loaded into A (line 20), the N register tested (line 20) for ready, and (if ready) receives the return code. Line 23, TYP, causes the carriage to return, and the typewriter is turned off (line 24).

PAPER TAPE OPERATIONS

The GE-225 computer system can use perforated paper tape as an on-line input and output medium. The reader and punch are housed in one cabinet and consist of a photoelectric reader, punch, and associated reader and punch control circuits. Options exist as to the combinations of paper tape equipment that can be used. These options are

1. Reader, spooler, punch
2. Reader, punch
3. Reader, spooler
4. Reader
5. Punch

The spooler has both a takeup spool and a spool for rewinding the paper tape after completion of reading or punching.

Paper Tape Reader

The paper tape reader is an on-line device that is capable of reading information from punched tape at two speeds: 1000 characters per second, at a tape speed of 100 inches per second, and 250 characters per second at a speed of 25 inches per second. At a reading speed of 1000 characters per second, the tape cannot be spooled and thus must be in strips, which normally do not exceed 10 to 15 feet in length. This speed, however, is excellent for loading programs from tape into memory. The tape operating speed is determined by a switch on the maintenance panel, Figure 6-6, at the rear of the cabinet.

The GE-225 paper tape reader allows the user to read any tape punched with standard character and sprocket hole spacing. While the GE-225 system uses primarily an 8-channel tape mode, the reader's ability to read 5, 6, 7 and 8 channel formats assures maximum compatibility with all common paper tape devices which use sprocket hole spacing for timing. A rotary switch on the reader maintenance panel, Figure 6-6, selects the channel type to be read. The paper tape reader processes standard paper tape 0.004 inch thick with the width varying with the number of channels as shown by Figure 6-7.

Paper Tape Punch

The paper tape punch is an on-line output device that punches paper tape alphanumeric information at the rate of 110 characters per second (11 inches per second). Ten characters are punched per inch of tape. The punch holds a roll of paper tape 8 inches in diameter, 800 feet long, and 0.004 inch thick.

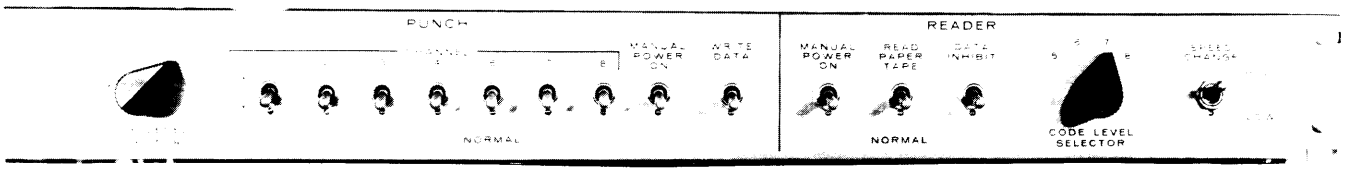


Figure 6-6. Paper Tape Reader-Punch Maintenance Panel

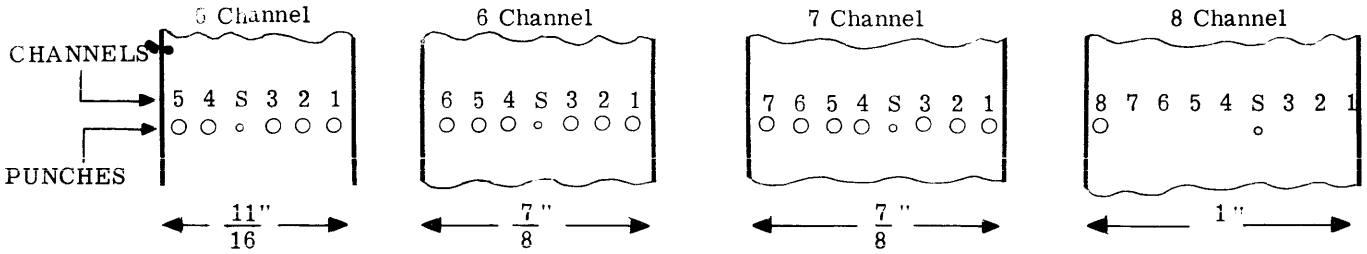


Figure 6-7. Paper Tape Modes

The GE-225 paper tape punch comes in two models:

Model GA651C punches 6, 7 and 8 channel paper tape as selected by a switch on the maintenance panel.

Model GA651D punches 5 channel tape only.

Paper Tape Reading

When a paper tape read instruction is given, tape reading is initiated under the control of the central processor paper tape reader control logic, Figure 6-8. Information from tape is detected photoelectrically when perforated tape passes between 9 photocells, or photoread heads, and a light source. Seven of the nine photoread heads monitor the tape information channels, or holes; one photocell monitors the parity channel; and one monitors the sprocket channel. How many information channels are used, and whether a parity hole is to be checked, depends on the tape mode, or format, to be read.

As each frame of information holes and the sprocket hole (1 frame per character) passes under the photoread heads, light is detected if a hole is punched in a channel. Once the leading edge of a sprocket hole is detected, the information punched is automatically placed into the N register of the central processor through the read gates and a mode selection switch. When the trailing edge of a sprocket hole is detected, this informs the central processor that a character has been loaded into the N register and is ready for transfer to the A register. At this point, an odd parity check (odd number of bits punched on tape) is made if seven or eight channel tape is being read. No parity is generated for five or six channel tape. Refer to Parity Error Detection (below) for additional data.

A read paper tape instruction initiates continuous reading of data from tape into the 6-bit N register until either a halt instruction is given or no more punched characters are detected.

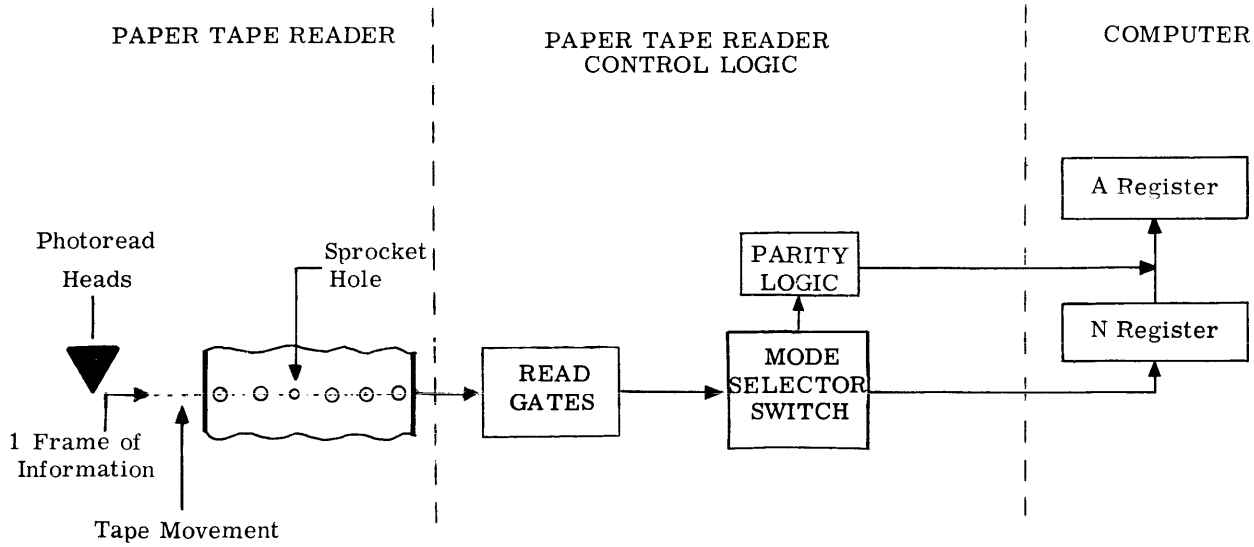


Figure 6-8. Paper Tape Reader Control Logic

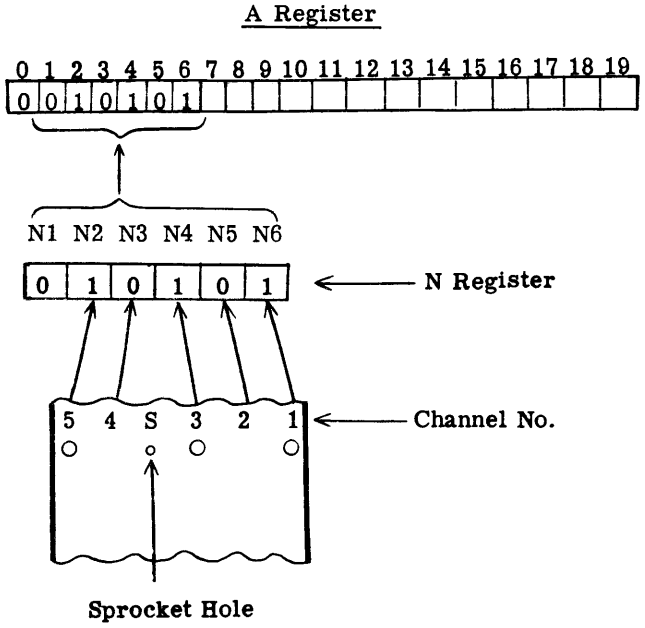
Data Transfer

During a read operation, once the leading edge of a sprocket hole is detected by the photoread heads, the N register becomes not ready, and a tape character begins filling the N register and continues filling it until the trailing edge of the sprocket hole is detected. When a character from paper tape fills the N register, it must be shifted into the A register before the next character arrives.

When reading is at the 250 cps rate, the N register is not busy for 2 milliseconds during which the N register must be emptied before the next character from paper tape enters the register. When tape is read at 1,000 cps, the N register is not busy for 500 microseconds.

In reading paper tape, the sprocket or feed hole serves as a timing source. A sprocket hole must be present with every tape character before the character can be transmitted to the N register. The relative bit positions of a character on paper tape are unchanged when the character is read from tape into the N register and then shifted into the A register. The transfer of data to the N register and then to the A register is shown below:

5 CHANNEL

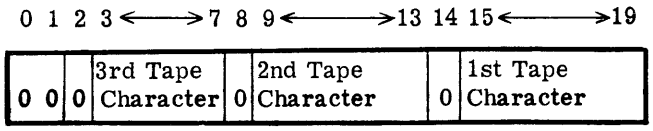


When reading paper tape in the five channel mode, only five information channels (5,4,3,2 and 1) are used in filling the N register. As shown, only N register bit positions 2 through 6 are used; bit position 1 is never filled. Note also that there is no parity bit channel on five channel tape.

VALID AND INVALID CODES. All code combinations are recognized in five channel tapes. The presence of a punch in the sprocket hole channel only, such as for tape leader or trailer strips, is recognized as a valid character and all zeros are transferred to the N register.

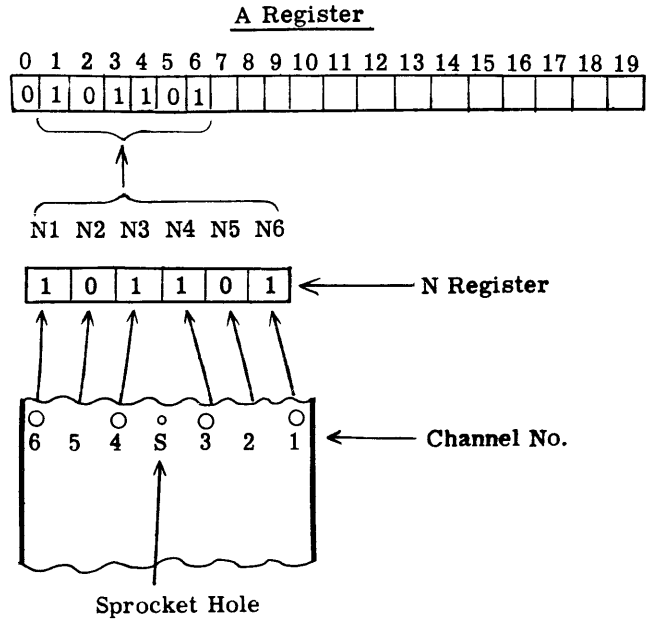
One significant difference in using five channel tape is that when three tape characters have been transferred into the N register and then shifted into the A register, bit positions 2, 8 and 14 of the A register contain zeros:

A Register



This is true because when a character is read into the N register, only bit positions 2 through 6 contain data; bit position 1 automatically is filled with a zero. As a result, when an SNA 6 instruction shifts N into A, all six bits are shifted, including the zero. When tape information is then converted into internal BCD code and stored in memory, each of the three BCD characters will occupy 6 bit positions in the normal manner.

6 CHANNEL

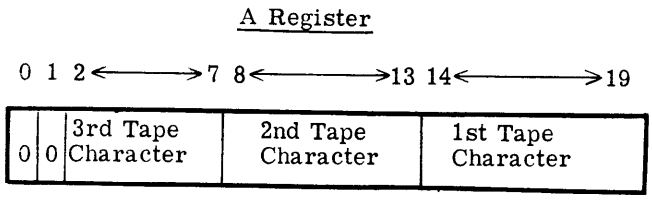


When six channel paper tape is read, all six information channels from tape enter the N register as shown above. There is no parity bit channel in six

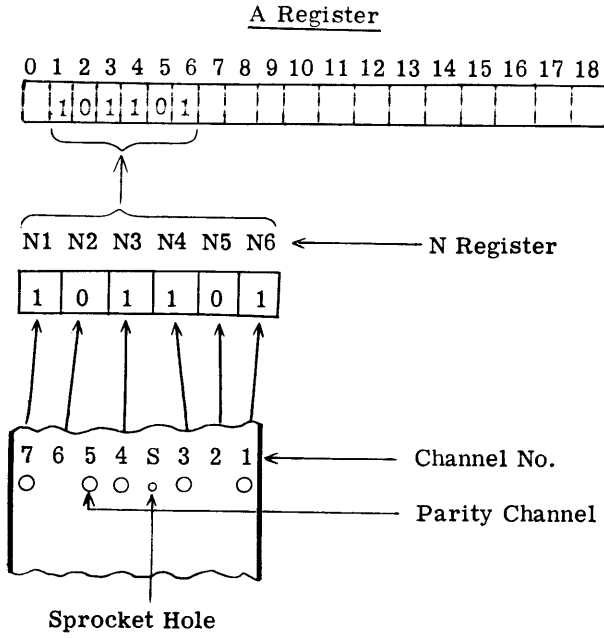
channel tape. All channels are used as information channels when data enters the N register.

VALID AND INVALID CODES. All code combinations are recognized in six channel tapes. The presence of a punch in the sprocket hole channel only, such as for tape leader or trailer strips, is recognized as a valid character and all zeros are transferred to the N register.

Shifting from the N register into the A register occurs as shown below.



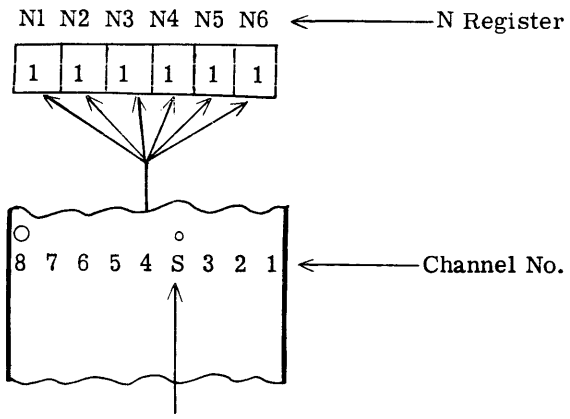
7 CHANNEL



Seven channel tape contains six information channels (1,2,3,4,6 and 7) and a parity channel (5). Tape feed and timing utilizes the sprocket holes. When a seven channel tape is read, bits are transferred into the N register and occupy the bit positions shown. Note that channel 5 is used only for a parity check and is not transferred into the N register.

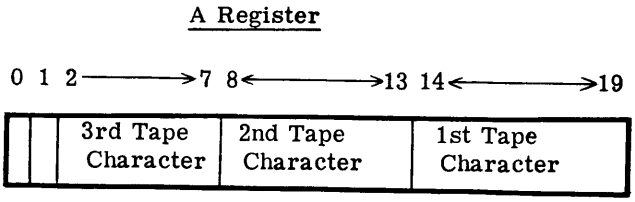
The bit patterns of information punched on seven and eight channel tapes are identical. The only difference between the two tapes is that a punch in channel 8 of eight channel tape causes all ones to be inserted into the N register.

8 CHANNEL



A punch in channel 8 inhibits the setting of the tape parity error check. This is necessary because parity is checked on each character loaded into the N register. Because the N register is loaded with six 1-bits, when a punch appears in channel 8, the parity bit is incorrect. Therefore, a parity error would occur if no provisions were made to inhibit it.

VALID AND INVALID CODES. In eight channel tape, a punch in channel 8 only is a special character and is valid. In both seven and eight channel modes, a punch in all channels (1 through 7) is a delete code and is not transmitted to the central processor as a valid character.



Seven and eight channel tape characters shifted from the N register into the A register are shown.

If the photoreader detects either a delete code or sprocket holes only when operating in either the 7 or 8 channel mode, the N register goes busy, preventing transfer of data to the A register. The N register remains busy until a legitimate character is detected.

Even though paper tape reading involves the N and A registers, other operations utilizing these registers can be performed by inserting them between N-register transfers by proper programming. However, during this time, instructions not requiring use of these registers can be performed by the central processor.

Code Conversions

Some of the code configurations punched on paper tape do not correspond to the internal GE-225 BCD configuration. Conversion of bits into BCD and the arrangement of characters (6 bits each) into GE-225

Upper Case (Letter Shift)
Lower Case (Figure Shift)

	Tape Character		Memory Character		Memory Code	
Tape Code	LC	UC	LC	UP	LC	UC
00	Tape Feed	Tape Feed	$\overline{0}$	$\overline{0}$	52	52
01	5	T	5	T	05	63
02	Car. Ret.	Car. Ret.			37	37
03	9	O	9	O	11	46
04	Space	Space	△	△	60	60
05	£	H	#	H	13	30
06	,	N	,	N	73	45
07	.	M	.	M	33	44
10	Line Feed	Line Feed	$\dot{0}$	0	32	32
11	)	L	]	L	76	43
12	4	R	4	R	04	51
13	&	G	@	G	14	27
14	8	I	8	$\dot{0}$	10	31
15	0	P	0	P	00	47
16	:	C	=	C	16	23
17	;	V		V	15	65
20	3	E	$\overline{3}$	E	03	25
21	"	Z	+	Z	20	71
22	$	D	$	D	53	24
23	?	B	–	B	74	22
24	Bell	S	EOF	S	17	62
25	6	Y	6	Y	06	70
26	!	F	/	F	61	26
27	1	X	1	X	01	67
30	-	A	-	A	40	21
31	2	W	2	W	02	66
32	'	J	*	J	54	41
33	Figure Shift	Figure Shift			36	36
34	7	U	7	U	07	64
35	□	Q	⌸	Q	34	50
36	(	K	[	K	75	42
37	Letter Shift	Letter Shift			35	35

Upper Case and Lower Case

a. The use of upper and lower case in some tape codes effectively doubles the number of tape code characters which may be used.

b. The convention employed in read is such that:

(1) Once an upper case code is read all succeeding codes will be translated as upper case codes until a lower case code is read.

(2) Once a lower case code is read all succeeding codes will be translated as lower case codes until an upper case code is read.

Figure 6-9. 5-Channel Tape Code to Memory Code.

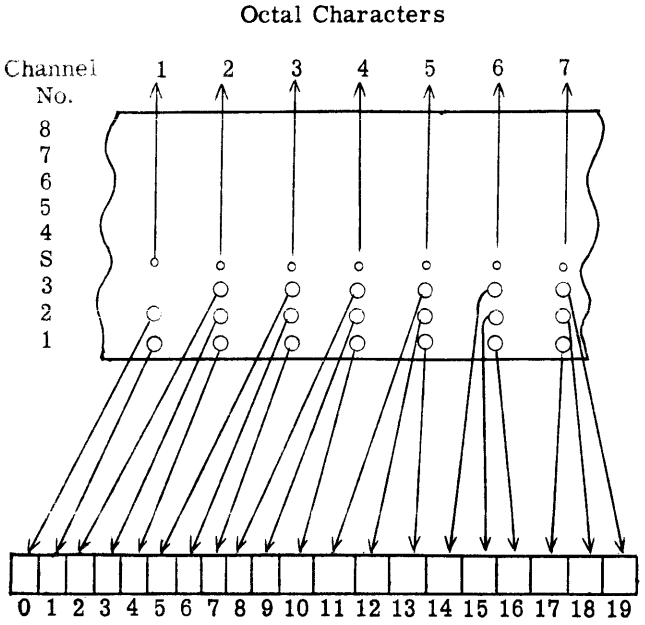
Upper Case
Lower Case

Memory Code	Memory Character	Tape Character	Tape Code
00	0	0	20
01	1	1	01
02	2	2	02
03	3	3	03
04	4	4	04
05	5	5	05
06	6	6	06
07	7	7	07
10	8	8	10
11	9	9	11
12			
13	#	#	13
14	@	@	14
15			
16			
17	EOF	&	60
20			
21	A	A	61
22	B	B	62
23	C	C	63
24	D	D	64
25	E	E	65
26	F	F	66
27	G	G	67
30	H	H	70
31	I	I	71
32			
33	.	.	73
34	⌑	□	74
35			
36			
37	Carriage Return	Delete	37
40	-	-	40
41	J	J	41
42	K	K	42
43	L	L	43
44	M	M	44
45	N	N	45
46	O	O	46
47	P	P	47
50	Q	Q	50
51	R	R	51
52			
53	$	$	53
54	*	*	54
55			
56			
57			
60	⍙	Space	00
61	/	/	21
62	B	B	22
63	C	C	23
64	D	D	24
65	E	E	25
66	F	F	26
67	G	G	27
70	H	H	30
71	I	I	31
72			
73	.	.	33
74	□	⌑	34
75			
76			
77	Delete	Carriage Return	37

Figure 6-10. Memory Code to 8-Channel Tape Code

words is done under program control. This conversion and arrangement can be accomplished by standard paper tape input-output routines. In addition, these routines can convert paper tape coded characters into internal BCD configurations so that when the information in the A register is stored in memory, it is ready for processing. Subroutines contain conversion tables that modify the characters as shown by Figures 6-9 and 6-10. Refer to the GE-225 utility routines for additional information on using these routines.

BINARY ROUTINES. Routines also are available that will transfer information from paper tape into memory in binary. Each paper tape word consists of 7 octal characters and is stored in a memory word as shown below. The tape read can be 5,6,7 or 8 channel. Refer to the paper tape binary routines for additional information as to their proper use.



GE-225 Word

Parity Error Detection

The GE-225 paper tape system uses an odd parity bit configuration for checking data on seven and eight channel tape. No parity bit is punched for five and six channel paper tape.

When data is read from seven or eight channel tape, each character is checked for parity, as indicated by channel 5. If present, the parity bit does not enter the N register. The paper tape reader detection logic senses the contents of the N register after each character has been read and an even pattern of bits in the N register indicates that an extra (odd) parity bit should have existed on the paper tape for the character read.

If the N register contains an even number of bits and there is no parity bit on tape, the PARITY light on the computer control console is turned ON. Conversely, if the N contains an odd number of bits and a parity bit is present on tape, the PARITY light on the control console is again turned ON. The test for a paper tape parity error condition is made by executing a branch on parity error (BPE) or branch on parity correct (BPC) instruction.

Paper Tape Reader Instructions

RON 2500014 **Word Times: 2**

Functional Description: PAPER TAPE READER ON. The power for the paper tape reader is turned ON. This instruction automatically turns OFF the power for the paper tape punch and typewriter, because the typewriter, paper tape reader, and paper tape punch share a common power supply and only one of the three units can be ON at any one time.

The RON instruction automatically clears the N register of any character that may have been left there from a previous punching or typing operation.

RPT 2500006 **Word Times: 2**

Functional Description: READ PAPER TAPE. The RPT instruction initiates the continuous reading of characters from paper tape. During the execution cycle of this instruction, a sprocket or feed hole must be present with every character before that character can be transmitted to the N register.

Example: Turn the paper tape reader on and a program delay of 200 milliseconds.

GAP Coding:

Opr	Operand	X	REMARKS	Sequence
RON			TURN PAPER TAPE READER ON	1 0
LDX	ZERO	2	ZERO INDEX WORD 2	2 0
INX	1	2		3 0
BXL	1588	2	DELAY 300 MS	4 0
BRU	* - 2			5 0
LDX	ZERO	3	ZERO INDEX WORD 3	6 0

Comments: When an RPT instruction is executed, assuming that the paper tape reader power is ON, then the N register is cleared and characters begin reading into the N register until a halt paper tape (HPT) instruction is given.

An RON instruction must be given and a delay of 200 milliseconds must be programmed before the first RPT instruction can be executed. This delay is necessary to allow time for the photo sensor lamp to reach full intensity. Additional RON instructions are not needed before additional RPT instructions are executed, provided that the power is not turned OFF by either an OFF instruction or by a power ON instruction for the typewriter (TON) or paper tape punch (PON).

HPT 2500016 Word Times: 2

Functional Description: HALT PAPER TAPE READER. This instruction halts the paper tape reader and terminates the read paper tape (RPT) operation.

Example: Assume reader power is on and reader ready. Read paper tape at 250 cps and move character into A register. Stop reader when A register is full.

GAP Coding:

Symbol	Opr	Operand	X	REMARKS	Sequence
	L D X	Z E R O	2	ZERO INDEX WORD 2	5 0
R D # 1	R P T			READ PAPER TAPE	6 0
	B N N	*		DELAY UNTIL CHARACTER IS READ	7 0
	S N A	6		SHIFT CHARACTER INTO A REG.	8 0
	I N X	1	2	INCREMENT INDEX WORD	9 0
	B X L	3	2	CHECK FOR 3RD CHARACTER	1 0 0
	B R U	* - 5			1 1 0
	H P T			HALT PAPER TAPE READER	1 2 0

Comments: When paper tape is halted by a halt instruction, the tape stops before the next succeeding character is read, if reading is being done at 250 characters per second. When reading at 1,000 characters per second, it is possible for two characters to pass under the read photocells after a halt instruction is given. During continuous start-stop cycling of the paper tape reader, a delay of 50 milliseconds is required between a halt and a read paper tape instruction. A delay of less than 50 milliseconds causes erratic stop distances.

Once an HPT instruction is given, a TON, PON, or OFF instruction cannot be given for at least 1 millisecond. This delay is necessary to allow the tape time to stop before turning reader power off.

In reading paper tape, it is best not to execute an RPT and immediately follow it with a halt (HPT) instruction as a parity error may be generated. Instead, delay by executing a test for N register ready allowing time for the tape character to enter N register.

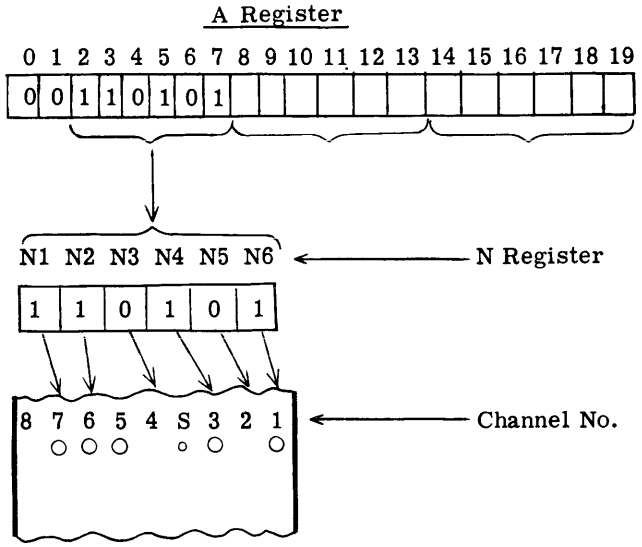
Paper Tape Punching

By executing a write paper tape instruction, one BCD character from the 6-bit N register is punched on tape in the mode previously set. Punching is executed under the control of the central processor paper tape control logic. Assuming that the power for the tape punch has been turned on, a punch instruction punches one character from the N register, advances the tape forward one frame, and halts the punch unit. A punch instruction is required for punching each tape character, regardless of what tape mode has been selected.

Before a character can be punched, it must have been previously shifted into the N register from the A register. As shown, a sprocket hole is also punched for each character punched. This hole serves as a timing source when punched information is read from tape. Actual punching of information into tape is performed by a set of 9 punching dies. The seven possible information channel punches and the parity channel punch (if required) are activated by punching magnets. The sprocket hole punch is actuated mechanically as each frame is punched.

An odd parity bit is punched in channel 5 when punching tape in either the seven or eight channel mode. A parity bit is never punched for five and six channel modes.

The paper tape punch mechanism punches any character placed in the N register by the central processor. Also, through a set of six simulator switches provided on the punch unit (reader unit), any configuration can be manually placed in the N register and punched independently of the central processor.



It takes approximately one millisecond to punch a single tape character. Therefore, other instructions not requiring the use of the N register may be executed during this time.

Paper Tape Punch Data Transfer

The punch unit punches any bit pattern placed in the N register from the A register by a shift instruction. The same holds true for any bit pattern placed in N manually by the six simulator switches. Each bit position of a character in the A register is unchanged when shifted into the N register and punched on paper tape. In each shift, the most and least significant bit positions remain the same as shown in the example above.

As shown, the relative bit positions (most and least significant) of the character remain the same when shifted from A into N and then punched into tape. Note that the character in 8 channel code has an odd number of 1-bits; therefore, when it is shifted into the N register and checked for odd parity, an extra parity bit must be punched in channel 5.

The bit patterns for information punched from the N register into tape in 5,6,7 and 8 channel modes is exactly the same as shown in the examples for the paper tape reader in Figure 6-7.

Valid and Invalid Codes

EIGHT CHANNEL TAPE. When punching eight channel tape, a character of all 1-bits in the N register is treated by the paper tape punch control logic as a special character and causes a punch in channel 8 only. Therefore, a bit pattern of punches in all channels 1 through 7 is not a valid character.

SEVEN CHANNEL TAPE. All 1-bits in the N register is an invalid code in seven channel mode and punches a sprocket hole only.

SIX CHANNEL TAPE. Because of the absence of the special character in six channel tape, all 1-bits in the N register is an invalid code and only a sprocket hole is punched.

FIVE CHANNEL TAPE. All possible bit patterns can be punched on five channel tape. No parity bit is punched. If all zeros are in the N register (no character), then only a sprocket hole is punched.

Data Conversion

The conversion of internal BCD information to the code configuration to be placed in the N register for punching is performed in the central processor under program control. This conversion can be accomplished by standard paper tape input-output routines. These routines, which also contain conversion tables, are similar to those described in the paper tape reader section.

Parity Error Detection

Error checking for the paper tape punch is an odd parity check. This means that each character punched on tape must have an odd number of holes or bits. Before the punch operation actually occurs, the 1 bits in the N register are checked for odd parity by the paper tape punch parity logic. If parity is even, a parity bit is generated according to the mode to be punched (no parity is punched for 5 or 6 channel tape). The mode is sensed by the parity logic from the mode selector switch setting. Simultaneously, the 9 punching dies are energized to reflect the bit pattern in the N register. These pulses travel through the punch gates and the mode selector switch, which activates the punching of channels according to the mode selected. After a frame is punched, the tape advances one frame and the punch mechanism halts until another punch instruction is executed.

PON 2500015 Word Times: 2

Functional Description: PUNCH ON. The power for the paper tape punch is turned ON and the power for the paper tape reader and typewriter is turned OFF.

Example: Turn on the paper tape punch and program a delay of 500 milliseconds.

GAP Coding:

```
Symbol    Opr     Operand     X   REMARKS                    Sequence
                                  31                      75
          PON                     TURN PUNCH ON              1 0
          LDX     ZERO        2   ZERO INDEX CELL 2          2 0
          INX     1           2   INCREMENT INDEX CELL       3 0
          BXL     3968            DELAY 500 MS               4 0
          BRU     *-2                                        5 0
PUTAPE    LDA     OUT
```

Comments: After a PON instruction, a delay of 500 milliseconds must be programmed before executing a write paper tape (WPT) instruction. This delay is necessary to allow the punch motor time to reach full operating speed, since the typewriter, paper tape reader and paper tape punch share a common power supply and only one of the three units can be ON at any one time.

WPT 2500006 Word Times: 2

Functional Description: WRITE PAPER TAPE. If the power for the paper tape punch is ON, whatever character is in the N register is punched. The contents of N are unchanged.

Example: Assume the punch is ready. Punch 3 characters contained in A register.

GAP Coding:

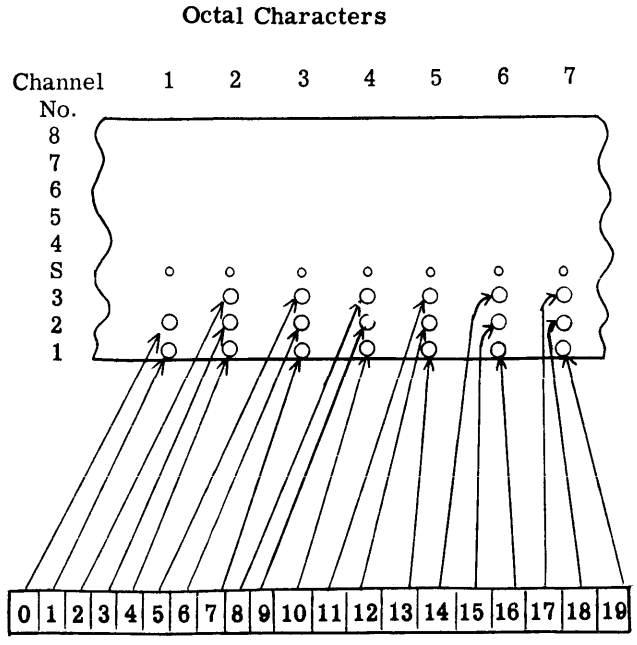
Symbol	Opr	Operand	X	REMARKS	Sequence
	LDX	ZERO	3	ZERO INDEX WORD 3	5 0
PUNTAP	BNN	*		CHECK N READY	6 0
	SAN	6		SHIFT ONE CHARACTER TO N	7 0
	WPT			PUNCH TAPE	8 0
	INX	1	3	INCREMENT INDEX CELL 3	9 0
	BXL	3	3	CHECK FOR 3RD CHARACTER	1 0 0
	BRU	* - 6			1 1 0
	LDA	DATA			

Comments: This instruction has meaning only when the power to the paper tape punch is turned ON. A punch power ON (PON) instruction must be given 500 milliseconds before a WPT instruction can be executed. Due to the nature of punch logic, it is necessary that a WPT instruction be executed for each character to be punched. A WPT instruction punches one character from the N register, feeds the tape forward one frame, and terminates the instruction. Upon the completion of a punch operation, the N register becomes 'ready', allowing the next character to be shifted from the A register into N.

Punch Paper Tape Binary

Through the use of programmed routine data can be punched from memory onto paper tape in binary form. Each word is in memory in binary and punched on tape in the form of 7 octal digits in either 5,6,7 or 8 channel.

Example:



GE-225 Word

CARD READER OPERATIONS

General

Punched cards are a commonly used media for recording data for processing because of their versatility and the availability of card processing equipment, such as key punches, verifiers, sorters, and duplicators. Cards can readily be used as the input medium in electronic information processing systems where input volume is small enough for economical operation.

Advantages of punched cards include: easy error detection and correction, rapid preparation of input data (using one or several key punches), ready manual access to records, easy separation and rearrangement (using sorters and collators), and flexibility of recording modes.

The standard punched card (Figure 6-11) is 7-3/8 by 3-1/4 inches cut from 0.009 inch paper card stock. For data representation, the card is divided into 80 vertical columns along the long dimension and 12 horizontal rows along the short dimension. Data is recorded by punching rectangular holes in these columns and rows. The interpretation of the holes into meaningful data is a function of both the card reading equipment and the recording mode or format.

Card Readers

Either of two card readers is available with the GE-225 system.

400 CPM CARD READER. This card reader is extremely compact and is designed to sit on the control console desk. Its speed is 400 cards per minute when reading continuously into alternating input areas in core storage. The maximum speed, when feeding a single card at a time under program control, is 360 cards per minute. Checks provided include input hopper empty, output hopper full, and misfeed indications; in each case, the card reader halts and the Card Reader Alarm indicator is turned on. Also, programmed tests can be made to insure that proper read synchronization was achieved, that is, each card column was read once and only once. The synchronization word, which is generated for each card read, is discussed later in the section.

Card reading is serially, by column, in one of three formats: decimal or standard (Hollerith) card code, 10-row binary, or 12-row binary.

The card reader is connected directly to the central processor, which contains the card reader control logic, through priority interrupt channel 1, thus permitting card reading to proceed simultaneously with other peripheral input-output and central processor operations.

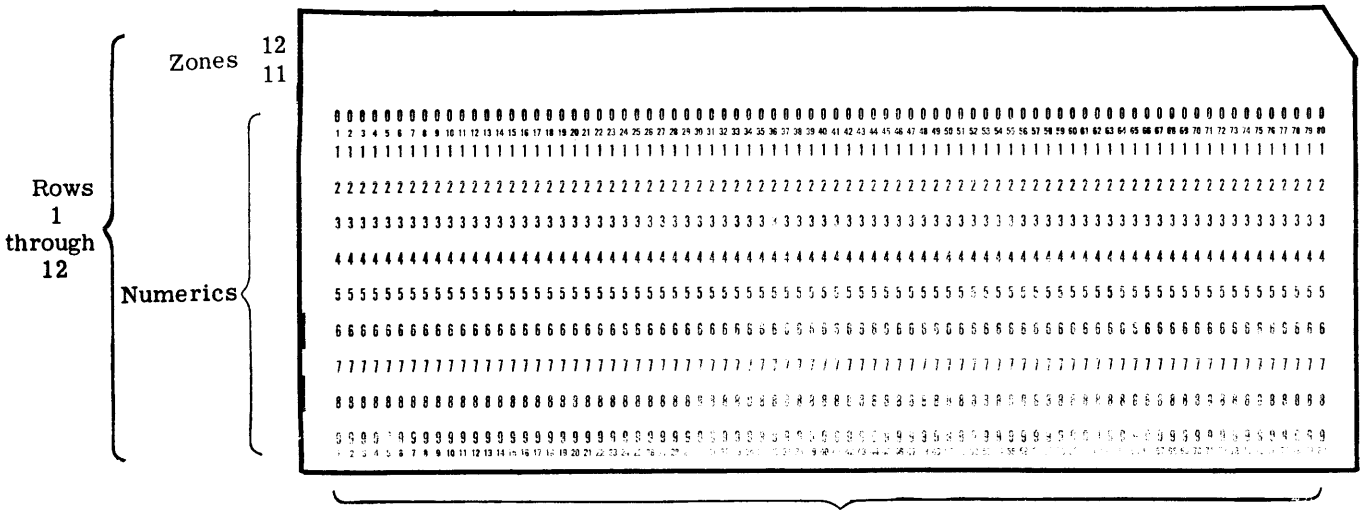


Figure 6-11. Standard Punched Card

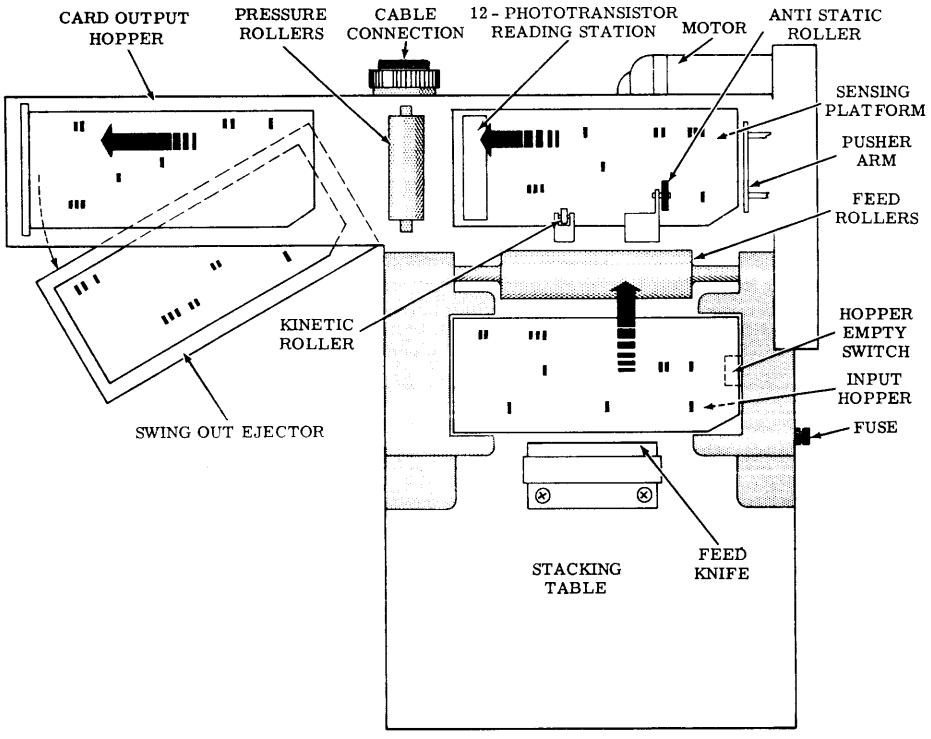


Figure 6-12. 400 CPM Card Reader Mechanism

Basic components of the card reader include an input hopper, a card feed mechanism, a reading station, and an output hopper or stacker. During normal operation, the card reader operates under control of the card reader control logic contained in the central processor to move cards from the input hopper, past a row of 12 read photosensors (the read station), and into the output hopper. Figure 6-12 shows schematically the 400 cpm card reader mechanism.

As a card passes the read station, it is read, a column at a time, into a card reader buffer and decoded (if required by the operating mode) and then transferred to core storage through the M register. The reading and transfer rates are governed by the timing circuits in the card reader rather than the central processor. After a column is read into the card reader buffer, the card reader logic demands access to core storage. The next full memory cycle is made available to transfer the card reader buffer contents to memory before the next card column reaches the read photosensors.

At 400 cards per minute, card columns are read at the rate of one column every 1875 microseconds. Thus, there is approximately 103 word times available for other processing and memory access between reading of adjacent card columns. During continuous read operations, the card reader uses only 1% of the available central processor time, leaving the other 99% available for other input-output or central processor operations.

HIGH-SPEED CARD READER. For applications involving large amounts of card input, the high-speed card reader provides card reading at 1500 cards per minute continuously and up to 850 cards per minute under program-controlled demand.

Mechanical operation is similar to that of the 400 cpm card reader. The high-speed card reader has a 2000-card capacity input hopper, a card feed mechanism, a read station, and a 2000-card capacity output stacker. Card reader control logic is contained in the central processor.

The high-speed card reader is connected to priority interrupt channel one. Card reading can occur simultaneously with other peripheral input-output and central processor operations.

At 1500 cards per minute, card columns are read at the rate of one column every 500 microseconds. Thus, there is approximately 27 word times available for other processing and memory access between readings of adjacent card columns. During continuous read operations, the high-speed card reader uses only 3-3/4% of the available central processor time.

During reading, cards are moved from the input hopper, past the reading station, and into the output stacker, under central processor control. Unlike that of the 400 cpm card reader, the reading station for the 1500 cpm reader consists of 12 photo cells, one for each card row, OR'ed together to simultaneously read each row. A punched hole is accepted as containing a punch if sensed by either or both solar cells, thereby reducing intermittent read errors. Additional checking includes: feed hopper empty, misfeed check, stacker full, end of file, and synchronization checks. Cards are read in one of three modes: decimal or standard card code, 10-row binary, and 12-row binary. An invalid card code during decimal mode operation causes bit 17 of a synchronization word to be set to zero. The synchronization word, which is generated for each card read, is discussed later in this section.

Feature	Card Reader Model	
	Standard	High-Speed
Speed - Continuous	400	1620
- Demand	320	900
Hopper Capacity - Input	600	2000
- Output	600	2000
Data Formats - Decimal	Yes	Yes
- 10-Row Binary	Yes	Yes
- 12-Row Binary	Yes	Yes
Processing Time Between Columns	103 word times	27 word times
Error Checks		
Read	None	Set sync bit 18
Invalid Code	No Invalid Codes	BCD only, set sync bit 17
Feed Empty	Stop and Alarm	Stop and Alarm
Misfeed	Stop and Alarm	Stop and Alarm
Stacker Full	Stop and Alarm	Set sync bit 16
End of File		Set sync bit 1
Synchronization	Set bit indicator in storage	Set sync bit pattern

Figure 6-13. Card Reader Feature Summary

Data Formats

Both the 400 cpm and the high-speed card readers can accept cards punched in three formats, one format at a time, depending upon the reading mode selected:

MODE	CARD FORMAT
Decimal	Standard Hollerith
Binary	10-Row Binary
Special Binary	**12-Row Binary**

STANDARD HOLLERITH. Standard format cards can be read and punched by GE-225 card equipment while operating in the decimal mode. In this mode, the alphanumeric data in each card column is converted from Hollerith code into a 6-bit character as the card is read into memory.

Hollerith code uses all twelve vertical rows of each card column to represent one numeric, alphabetic, or special character. The twelve rows are divided into two areas, zone and numeric. The zone area consists of rows 12, 11, and 0; rows 0 through 9 are the numeric area. Row 0 is common to both zone and numeric areas. Figure 6-14 shows the Hollerith code characters accepted by the GE-225 through the card readers.

Numeric characters are represented by a single punch per column. Alphabetics are represented by two punches per column, a zone punch and a numeric punch. Special characters consist of either two or three punches per column.

Because Hollerith code characters are converted to 6-bit BCD by the card reader, three characters are required to fill a GE-225 word; thus three card columns equal one word. One card can contain 27 data words. Figure 6-15 shows how card data appears when read into memory.

Note that bit positions 0 and 1 of the word are not required for data and are normally set to zero. Data punched in Hollerith can fill up to 27 consecutive GE-225 words. However, the twenty-seventh word is not completely filled because it contains card columns 79 and 80 only. This GE-225 word would appear as shown in Figure 6-16.

There is no data available for word number 27, bit positions 14 to 19, because there is no column 81 on a card. In view of this, the computer automatically generates a blank for the unused area. Note that the sample word contains a blank (octal 60) in bit positions 14 through 19.

Figure 6-17 illustrates the GE-225 character set, the equivalent Hollerith code, and the octal equivalent of each character as it is stored in memory in BCD mode.

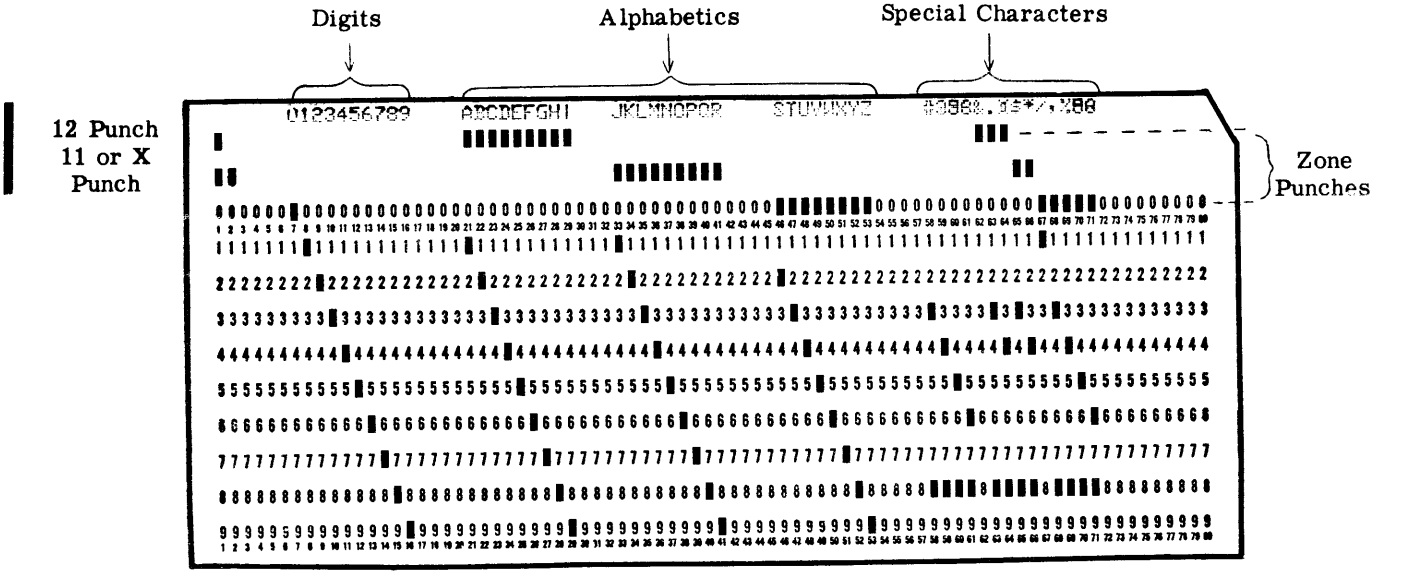


Figure 6-14. Standard or Hollerith Punched Card

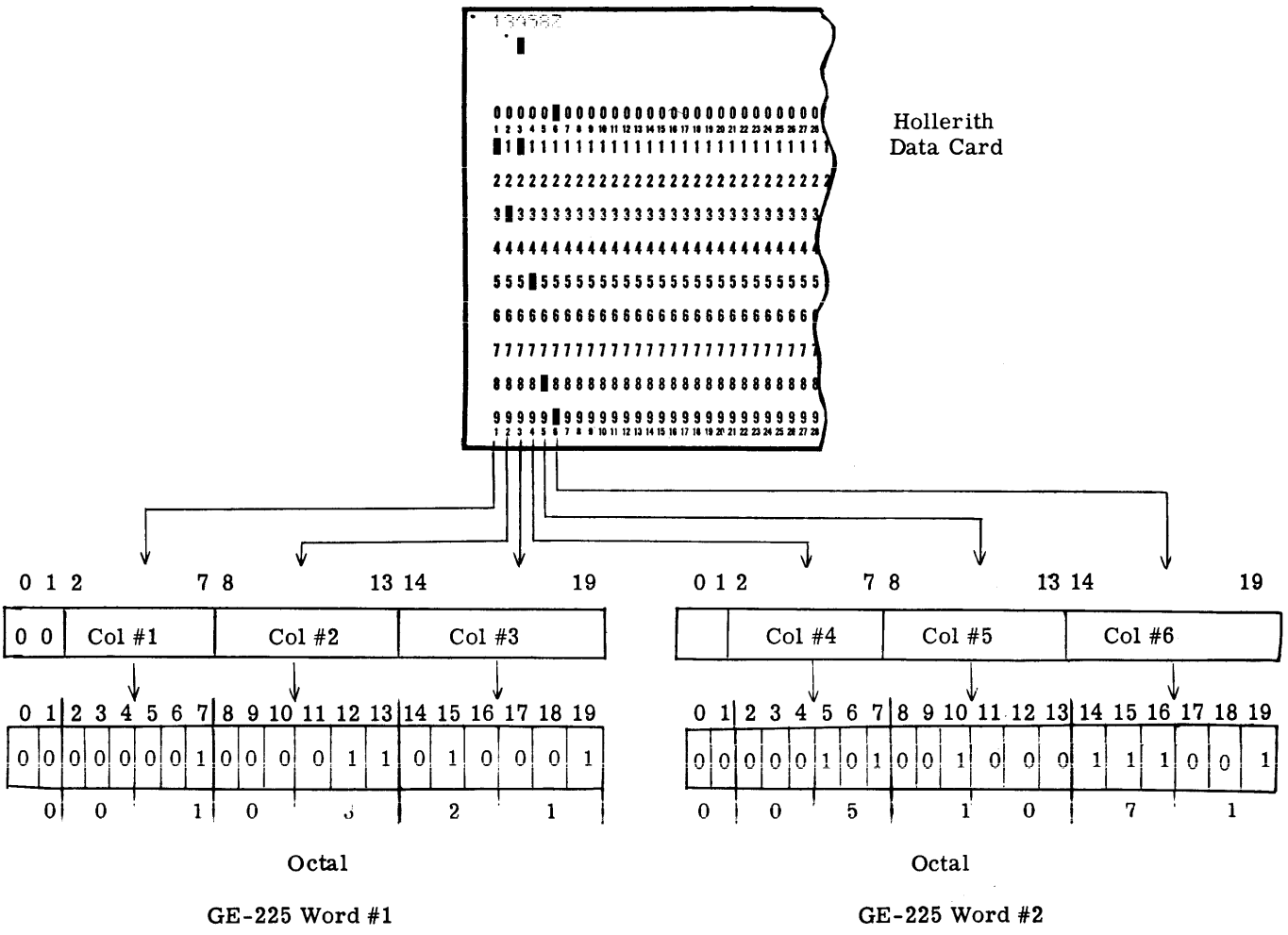


Figure 6-15. Memory Equivalent of Card Data

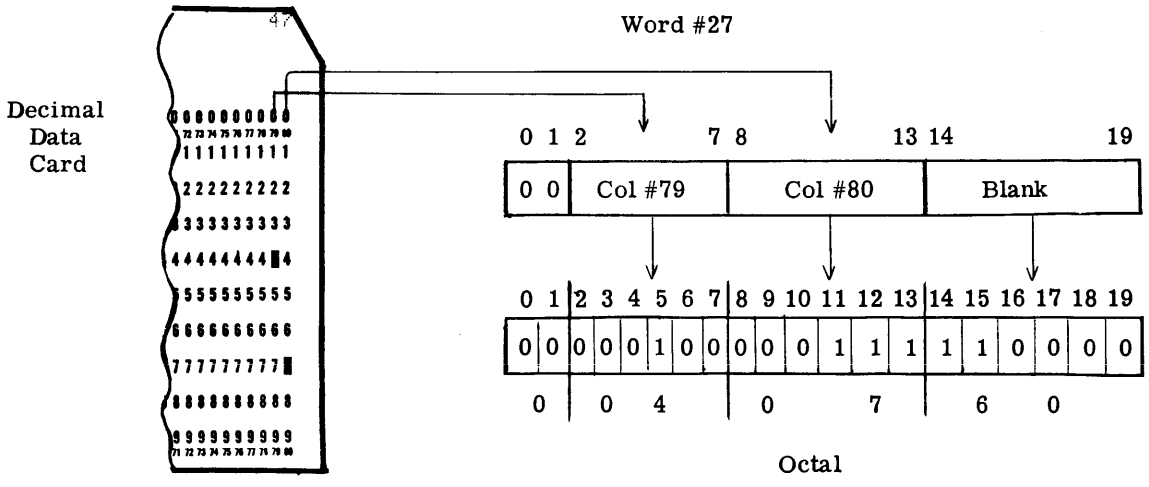


Figure 6-16. Memory Equivalent of Word 27

Character	Hollerith Code (Punch in rows)	BCD Memory	Character	Hollerith Code (Punch in rows)	BCD Memory	Character	Hollerith Code (Punch in rows)	BCD Memory
0	0	00	G	12-7	27	S	0-2	62
1	1	01	H	12-8	30	T	0-3	63
2	2	02	I	12-9	31	U	0-4	64
3	3	03	+0	12-0*	32	V	0-5	65
4	4	04	.	12-3-8	33	W	0-6	66
5	5	05	¤	12-4-8	34	X	0-7	67
6	6	06	-	11	40	Y	0-8	70
7	7	07	J	11-1	41	Z	0-9	71
8	8	10	K	11-2	42	,	0-3-8	73
9	9	11	L	11-3	43	%	0-4-8	74
#	3-8	13	M	11-4	44	(	0-5-8	75
@	4-8	14	N	11-5	45	)	0-6-8	76
(Underline)	5-8	15	O	11-6	46	←	2-8	12
=	6-8	16	P	11-7	47	Not Assigned	7-8	17
+	12	20	Q	11-8	50		12-5-8	35
A	12-1	21	R	11-9	51		12-6-8	36
B	12-2	22	-0	11-0*	52		12-7-8	37
C	12-3	23	$	11-3-8	53		11-5-8	55
D	12-4	24	*	11-4-8	54		11-6-8	56
E	12-5	25	Space	Blank	60		11-7-8	57
F	12-6	26	/	0-1	61		0-2-8	72
							0-7-8	77

* Note: The 400 cpm card reader reads all punch configurations, thus no punch configuration is illegal. For example, an 11-0 and an 11-2-8 are read as an octal 52. The configuration 12-0 and 12-2-8 are detected as an octal 32. The GE High Speed Reader treats these punch combinations 11-2-8 and 12-2-8 as invalid characters. The card punch interprets an octal 52 as an 11-0 and a 32 as a 12-0.

Figure 6-17. Hollerith Punched Card Code for GE-225

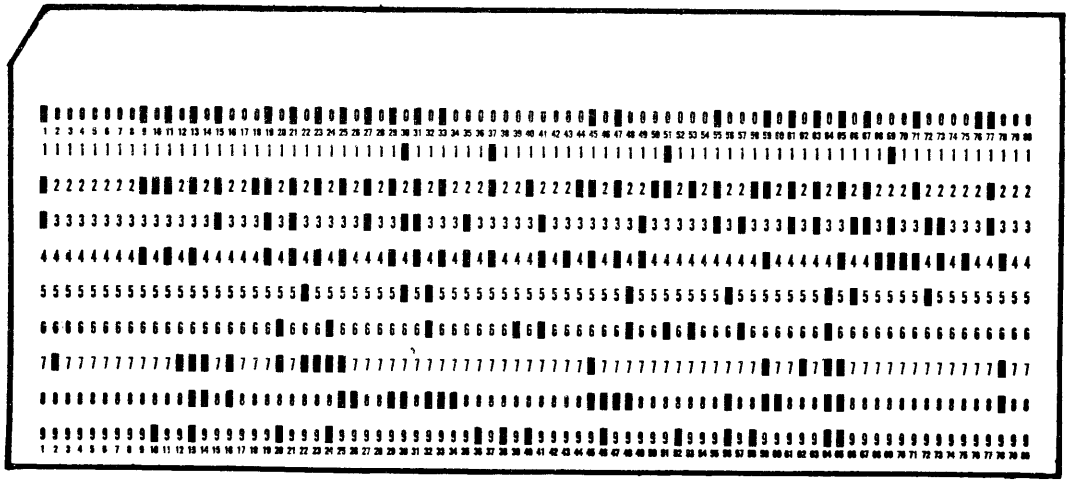
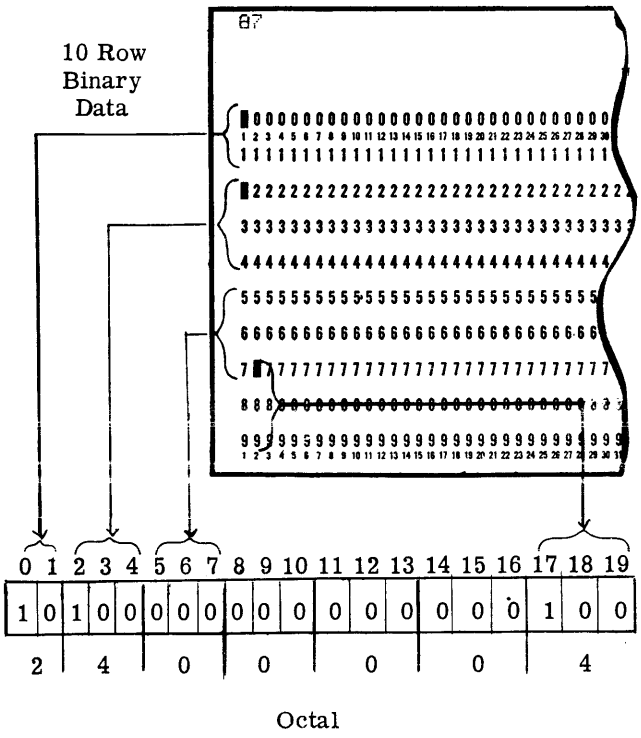


Figure 6-18. 10-Row Binary Data Representation

10-ROW BINARY. The GE-225 card readers can process information punched in the 10-row binary format. Figure 6-18 shows a 10-row binary card as produced by the GE-225 General Assembly Program. In this card, data is punched so that each row position in a column represents a binary bit. A punched hole is a binary 1 and no punch is a binary 0. Only rows 0 through 9 are used in 10-row binary. Because each row position of each column is a binary bit, two columns of a 10-row binary card (20 binary characters) fill a GE-225 word. For example, columns 1 and 2 of the card in Figure 6-18 fill a GE-225 word as follows:



Data for a maximum of 40 consecutive GE-225 words can be punched on one 10-row binary card.

12-ROW BINARY. In the 12-row binary mode, all twelve punching positions of the card columns are sensed by the card reader. As in 10-row binary, a punch indicates a binary 1 and no punch is a binary 0. Figure 6-19 is a 12-row binary card. The binary information is placed in the 12 least significant bits of a GE-225 word. The balance of the GE-225 word receives no information and remains blank. The information on a 12-row binary card can thus fill a maximum of 80 successive GE-225 words. The bit configuration of each column is placed in memory as shown in Figure 6-20.

Note that bit positions 0 through 7 do not contain data and are automatically set to zero. Note also that three GE-225 words together form the equivalent of a 36-bit binary word, a data form frequently encountered in other data processing equipment.

Card Reader Instructions

Two card reader models are available for the GE-225, with certain basic operations and instructions common to the two units.

The GE-225 card reader is an on-line device that reads punched cards by a photoelectric mechanism in a mode selected by the read instruction. Controlled by a unit in the central processor, the card reader transfers information into memory via the M register. Because the GE-225 system can simultaneously perform two or more operations, the card information is read serially, column by column into memory on a priority interrupt basis. The card reader always has the highest priority in accessing memory during a particular word time because a loss of information will result if access is

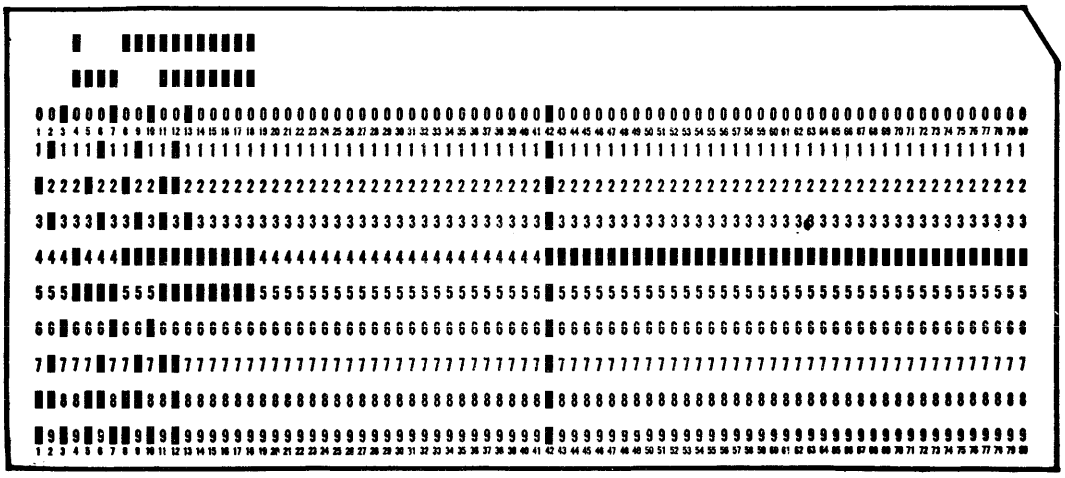


Figure 6-19. 12-Row Binary Data Card

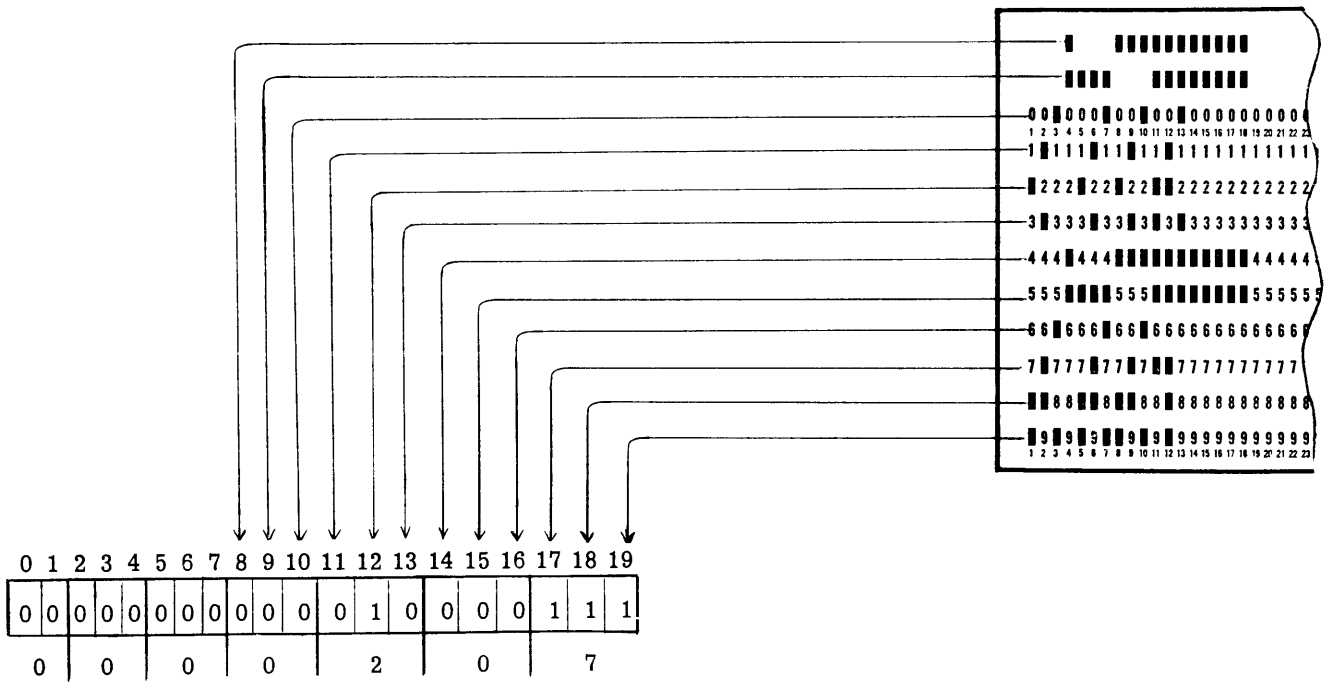


Figure 6-20. 12-Row Binary Card to Memory

not granted. The 400 cpm card reader requires access to memory once every 103 word times while the high-speed card reader accesses memory once every 27 word times.

The GE-225 reads punched cards in three different modes corresponding to the three card formats, depending on the read instruction being executed. Regardless of the card reader mode, the starting memory address into which data is read must be a multiple of 128 but less than 2048. This means, in effect, that cards can be read into memory beginning at 128, 256, 384, etc., up to 1920. It is suggested that read operations not utilize a memory address less than 256 due to the possibility of the API, Automatic Program Interrupt, being added later if not used at present. The API uses memory location 128, as well as extra modification word groups.

RCD Y 250YY00 Word Times: 2

Functional Description: READ CARDS DECIMAL initiates continuous reading of cards in the decimal mode (that is, card data is interpreted by central processor card reader logic as being in Hollerith format) into memory, starting at location Y. Y must be a multiple of 128, but less than 2048. The first card read enters locations Y through Y+26, the second enters Y+32 through Y+58, the third enters Y+64 through Y+90, the fourth enters Y+96 through Y+122. The fifth card enters Y through Y+26, etc. After each card is read in, the sign bit of the GE-225 word after the last word containing card data (Y+27, Y+59, Y+91, or Y+123) is set to minus. After the last card of the deck is read in, bit position 1 of the GE-225 word after the last word containing card data (Y+27, Y+59, Y+91, or Y+123) is set to one. If the card reader is not in ready status when the RCD is given, the central processor halts and a card reader error is indicated on the control console. Once begun, card reading is continuous until the card reader is stopped by a halt card reader (HCR) instruction, the input card hopper becomes empty, or a machine malfunction occurs.

HCR 2500004 Word Times: 2

Functional Description: HALT CARD READER. This instruction halts the card feed. If the first half of a card is being read at the time that HCR is given, the reading of this card into memory will be completed, and no further cards will be read until another read instruction is given. If the second half of the card is being read and another card has entered the feed area, both the card being read and the card in the feed area will be read before the card feed stops. This instruction does not delay the computer while the last card is being completely read; the program continues in sequence and a delay must be programmed to insure that the data is in memory before attempting to use it.

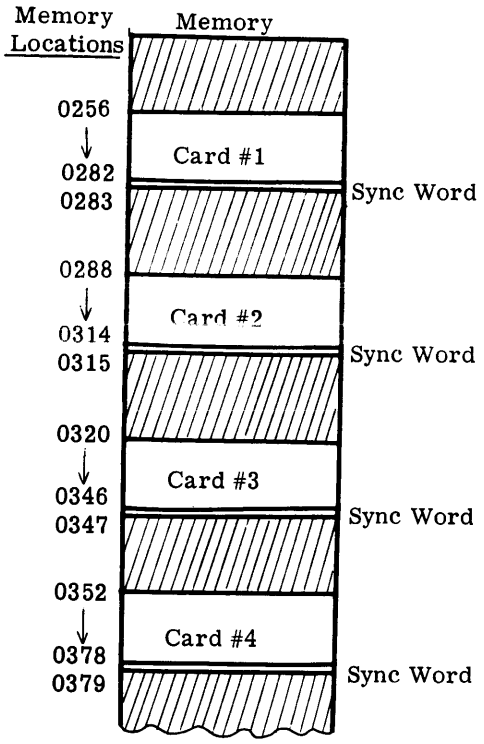
Example 1: Read cards in the decimal mode into memory, beginning at location 0256.

GAP Coding:

Symbol	Opr	Operand	X
	R C D	2 5 6	

The first four cards read will be entered into memory in the following sequence.

Note that each card occupies 27 memory locations, plus 1 for the sync word.



If the card reader contains more than four cards, and the RCD is not followed by a HCR, the fifth card is automatically read into the same memory location as card #1 (0256). The sixth card enters the same locations as did card #2 (0288); etc.

Comments: Five words of memory are automatically skipped after the areas containing card data. In the above example, between cards #1 and #2, memory locations 283 through 287 are skipped; between cards #2 and #3, locations 315 through 319 are skipped; etc. The last four locations of each group of five can be used by the programmer for storage of constants or other program data. The first word of each group (locations 283, 315, 347, and 379, in the example above) cannot be so used, because it is reserved for checking information that is automatically developed in the card reader. These locations are known as synchronization, or sync, words and are described fully under Card Synchronization.

Example 2: Read one card in the decimal mode into location 0384.

GAP Coding:

Symbol	Opr	Operand	X
	R C D	3 8 4	
	H C R		

Comments: RCD, when immediately followed by a HCR instruction, will cause a single card to be read into the specified location (0384). If the above coding were incorporated into a loop, each time the RCD, HCR sequence was executed, one card would be read (in the decimal mode) into 27 consecutive locations starting at 0384, and the automatically-generated sync word would be placed in location 0411.

RCB 250YY01 Word Times: 2

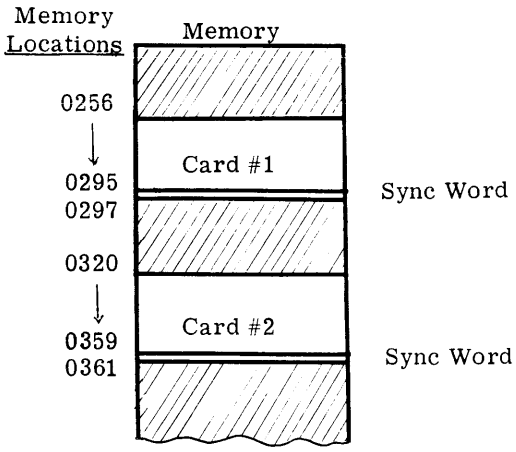
Functional Description: READ CARDS BINARY initiates continuous reading of binary cards (that is, card data is interpreted by the central processor card reader logic as being in the binary format) into memory starting at location Y. Y must be a multiple of 128 and less than 2048. The first card is read into locations Y through Y+39, the second card into Y+64 through Y+103, the third into Y through Y+39, etc. After each card is read, the sign bit of the second word following the card data, location Y+41 or Y+105, is set to minus. After the last card of a deck is read, bit position 1 of the second word following the last card image, Y+41 or Y+105, is set to 1. If the card reader is not in ready status when RCB is given, the computer halts and a card reader error is indicated on the control console.

Example: Read cards in the binary mode into location 0256.

GAP Coding:

Symbol	Opr	Operand	X
	R C B	2 5 6	

The first two cards read will be entered into memory in the following sequence:



Comments: If the card reader contains more than two cards and the RCB is not followed by a HCR, the third card would be automatically read into locations 0256 through 0295, the area previously containing card #1 data; card #4 would be read into 0320 through 0359. This sequence would be repeated until all cards were read.

Note that an area of 24 words in memory is left between the end of the first card area and the beginning of the second card area. In this example, the area consists of the locations between 0296 and 0319. All except one of these locations can be used by the programmer if desired. The second words following the last words of each card (0297 and 0361) are reserved for the synchronization word which is used for checking purposes. See Card Synchronization.

If an RCB instruction is followed by an HCR, one card is read in the 10-row binary mode into memory and the card reader halts. If additional cards are to be read, this sequence must be repeated for each card.

Example: Read a 10-row binary card into 0256 and halt the card reader.

GAP Coding:

Symbol	Opr	Operand
	R C B	2 5 6
	H C R	

The card read will be entered into memory locations 0256 through 0295, the sync word will be entered into location 0297, and the card reader will halt. If the sequence is later repeated, the next card will enter the same locations.

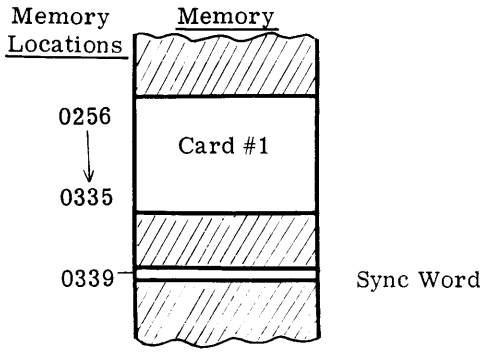
RCF Y 250YY10 Word Times: 2

Functional Description: READ CARDS FULL initiates the reading of all 12 rows of an 80-column binary card (that is, data punched into the card is interpreted by the central processor card logic as being in the special binary mode). The 12 punching positions of each column (starting with column one) are placed in the 12 least-significant bit positions of successive memory locations, starting at Y (which must be a multiple of 128 and less than 2048). The card is read into locations Y through Y+79. After the card is read, the sign bit of the fourth word following the card image is set to one. The card reader automatically halts after one card is read. If the card reader is not in ready status when RCF is given, the central processor halts and an error is indicated on the control console.

Example: Read a card in the 12-row binary mode into location 0256.

GAP Coding:

Symbol	Opr	Operand
	R C F	2 5 6



Comments: One card of 12-row binary data is read into memory starting in location 0256 through 0335. The sync word is constructed in location 0339. Only one card is read and the card reader halts automatically.

Reading Intermixed Cards High-Speed Card Reader

The following instruction is effective only with the high-speed card reader as an optional feature.

RCM Y 250YY12 Word Times: 2

Functional Description: READ CARDS MIXED. This command initiates the reading of a single card of data. The card reader will automatically halt after each card is read.

The reading of the card occurs in the RCF (READ CARD FULL) mode if column one contains a 7-9 punch. If column one does not contain a 7-9 punch the card is read in the RCD (READ CARD DECIMAL) mode.

GAP Coding:

Symbol	Opr	Operand	X
	BCN	*	
	RCM	CRDIN	
	LDA	CRDIN+83	
	BPL		
	BRU	*-2	

Comments: The read cards intermixed instruction allows random intermixed binary and decimal cards to be read into memory one at a time under program control. A binary card must be so designated by a 7 and 9 punch in column one, in which case the card will be read into memory by setting a bit in bit position 1 of the first memory word. The first word in memory would appear as 1012000_8.

If the 7 and 9 are not sensed in column one, the card is assumed to be a Hollerith decimal card and will be read into memory in the Read Cards Decimal Mode.

A typical binary card with the 7-9 punch in column one is shown at the bottom of the page.

Card Reader Synchronization and Error Checks

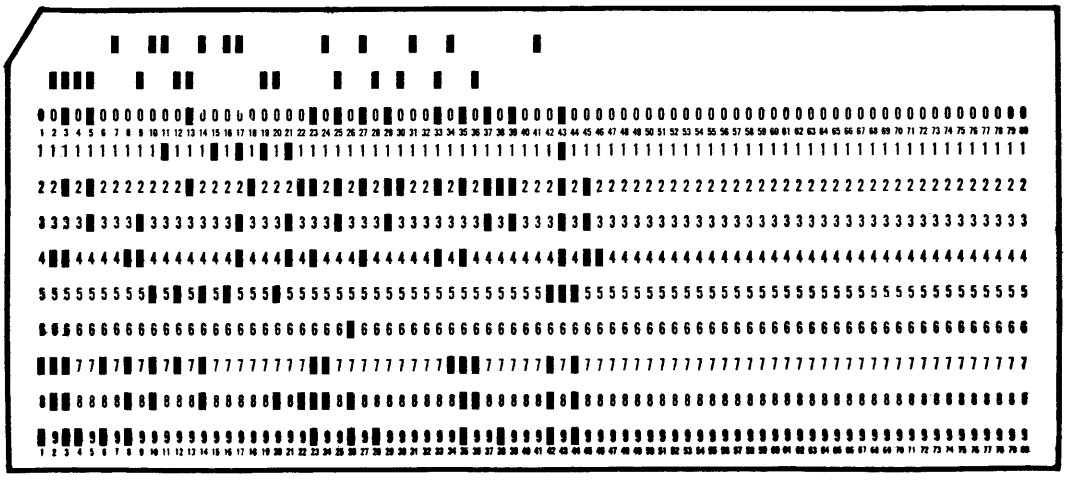
Extensive checking of the GE-225 card reader operations accompanies the reading of each punched card. The results of these checks are stored in a specific memory location by the card reader logic. This check word is referred to as the synchronization word or sync word.

The contents of the sync word provide the programmer with information essential to the successful operation of his program. Thus, the programmer must check the contents of the sync word each time a card is read into memory.

The location in memory of the sync word depends upon the read mode and whether the card reader is reading a single card and halting or reading continuously. Refer to Figures 6-21 and 6-22.

400 CPM SYNC WORD. The contents of the sync word generated by the 400 cpm card reader indicates whether:

1. reading of a card is completed and if the card was properly synchronized upon entering and leaving the read station, and
2. the last card of the deck in the input hopper has been read and the hopper is empty.



Card Read Mode	Synchronization Word Memory Location		Contents in Octal	
	Continuous	HCR	Normal	Hopper Empty
DECIMAL Decimal	Y+27 Y+59 Y+91 Y+123	Y+27	2606077	3606077
10-Row Binary	Y+41 Y+105	Y+41	200177	300177
12-Row Binary	No Continuous Read	Y+83	2007777	3007777

Figure 6-21. 400 CPM Synchronization Word

Example 1: The RCD instructions below have been executed by the computer.

GAP Coding:

Opr	Operand	X	REMARKS
R C D	2 5 6		READ A CARD IN DECIMAL MODE
H C R			HALT CARD READER

Normally, the sync word in memory location 0283 (Y+27) assumes one of these forms.

1. A normal read, no synchronization errors, input hopper not empty.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1
2		6		0		6		0			7			7					

2. The last card of the deck in the input hopper has been read and the hopper is empty.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	1	1	1	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	1
3		6		0		6		0			7			7					

Any other bit configuration indicates that an error has occurred and the card must be re-read before processing the data it contained.

Example 2: If the read card instruction above is a RCB 256, the sync word in 0297 (Y+41) for 10 row-binary under the same conditions as above is:

1. Normal Read (10-Row Binary)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
2		0		0		1		7			7			7					

2. Input hopper empty (10-Row Binary)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
3		0		0		1		7			7			7					

Any other bit configuration indicates that an error has occurred and the card must be re-read before processing its data.

Example 3: For 12-row binary, RCF 256, no HCR instruction is necessary and the sync word in 0339 (Y+83) for the same conditions of Example 1 appears as:

1. Normal Read (12-Row Binary)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
2		0		0		7			7			7			7				

2. **Input hopper empty (12-Row Binary)**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
	3		0		0		7		7			7			7			7	

Any other bit configuration in the sync word indicates that an error has occurred and the card must be re-read before processing its data.

HIGH-SPEED CARD READER SYNC WORD. The sync word generated by the high-speed card reader provides a much greater amount of pertinent information for programmer use. Figure 6-22 contains information on the high-speed card reader sync word. The sync words generated by this card reader are identical with those of the 400 cpm reader with the exception of an invalid character check and a check for the last card of the deck with the EOF (End of file) button set.

Interrogation of the bit configurations within the sync word generated by the high-speed card reader detects the following operating conditions:

Bit Position	Contents	Operating Condition
0	1	The card is in memory
1	0	The input hopper still contains cards
	1	The input hopper is empty
16	0	The output stacker is full
	1	The output stacker is not full
17	0	Malfunctioning photocell or card slippage.
	1	All photocells functioning properly and no card slippage
18	0*	Invalid character exists
	1	All characters valid
19	0	Input hopper empty and EOF switch is set
	1	Input hopper not empty and EOF switch is not set

* Valid Hollerith punch combinations are listed in Figure 6-17. In the decimal read mode, RCD, characters are checked for validity and when an invalid character is read, the sign of the word containing the character is set to minus (bit position 0 set to 1) in addition to the sync word indication (bit 17) as shown. There is no invalid punch configuration for 10-row and 12-row binary.

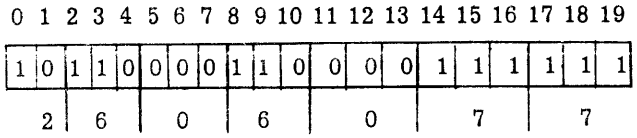
Card Read Mode	Octal Contents				
	Normal	Hopper Empty		Character Invalid	Read Error
		EOF Not Set	EOF Set		
Decimal	2606077	3606077	3606076	2606073	2606075
10-Row Binary	2001777	3001777	3001776	2001773	2001775
12-Row Binary	2007777	3007777	3007776	2007773	2007775

Figure 6-22. High-Speed Card Reader

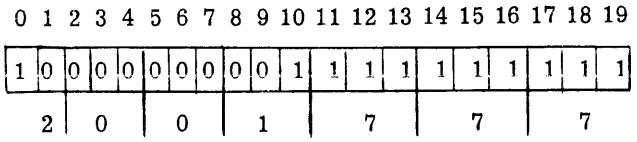
Examples of the sync word generated by the high-speed card reader are given below. Since a number of card reader operating conditions can occur with the reading of a card, the examples cover only the conditions mentioned.

Example 1: Sync word on normal card read:

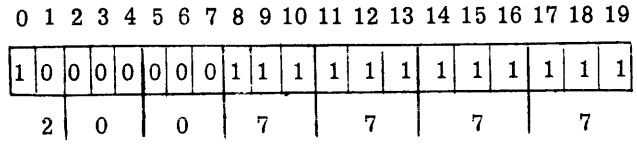
Decimal Mode



10-Row Binary

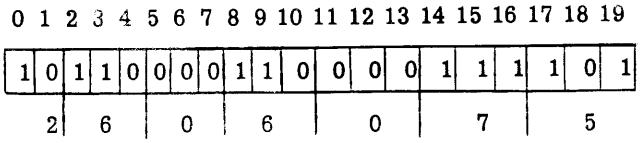


12-Row Binary

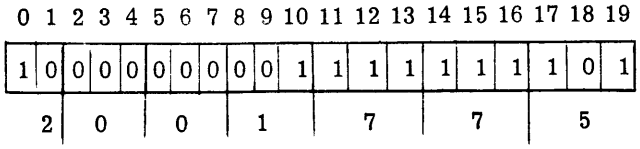


Example 2: Sync word on read error (Synchronization) Bit position 18 set to zero:

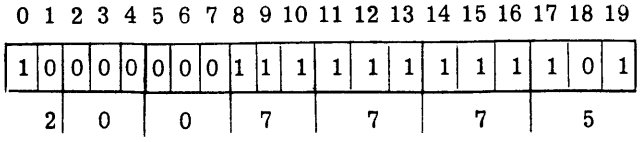
Decimal Mode



10-Row Binary



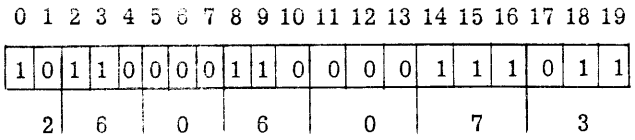
12-Row Binary



When a synchronization error occurs on a card read, the card must be re-read before processing the data.

Example 3: Sync word on invalid card character Bit position 17 set to zero:

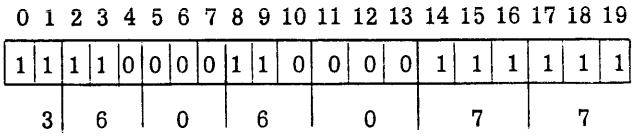
Decimal Mode



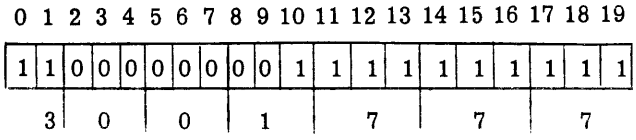
In addition, the sign of the word in memory which contains the invalid character from the card is set to minus. There are no invalid characters when reading binary cards, and bit position 17 will be set to one (1).

Example 4: Sync word for last card of a deck in the input hopper with EOF button NOT SET and HOPPER EMPTY:

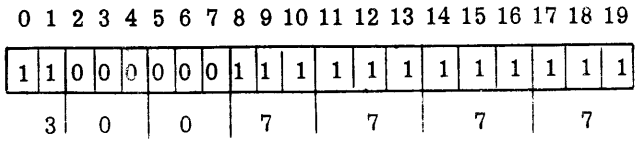
Decimal Mode



10-Row Binary



12-Row Binary



If the EOF button is set, bit position 19 of the sync word is 0:

DECIMAL	OCTAL → 3606076
10-ROW BINARY	OCTAL → 3001776
12-ROW BINARY	OCTAL → 3007776

If the EOF button is set, the circuitry generates a 1 in bit position 1 and a 0 in bit position 19 of the SYNC WORD when the last card of the deck in the input hopper is read into memory.

Card Reader Generation of Sync Word

Regardless of the mode in which a card is read, the logic of the card reader generates 84 read pulses, even though only 80 columns of information are read. Pulses 81, 82, 83, and 84 affect the bit configurations of the sync word in a manner dependent upon the read mode.

In the decimal read mode, pulse 81 places an octal 60 in bit positions 14 through 19 of the sync word. Pulses 82 and 83 place an octal 60 in bit positions 2 through 7 and 8 through 13, respectively. Since pulse 84 is timed such that it falls between cards, it generates all ones in bit positions 14 through 19 and sets a 1 in the sign bit position, signifying a complete card read. Also, if the input hopper goes empty, bit position 1 is set to 1.

If a card column contains an invalid punch configuration, a logical OR takes place and the 400 cpm reader places the results in memory. The 400 cpm reader, unlike the high-speed reader, does not indicate invalid characters. For instance, if a card column contains the invalid punch configuration of 6 and 9, the logical OR of the punches are placed in memory as shown below:

6 ⟶ 00110

9 ⟶ 01001

15 ⟶ 01111

In the 10-row binary mode, pulse 83 sets bit positions 2 through 19 of the sync word to zero and pulse 84 sets bit positions 0 and 10 through 19 to ones to indicate a complete card read. If this is the last card in the hopper, bit position 1 is set to one. Pulses 81 and 82 do not generate any sync word information.

In the 12-row binary mode, pulse 84 sets bit positions 0 and 8 through 19 of the sync word to ones upon the completion of the card read. Bit position 1 is also set to one if this is the last card of the deck in the hopper.

Compatibility of 400 CPM and High-Speed Card Readers

Programs written for the 400 cpm reader can be run on a machine using the high-speed reader. However, the reverse is not true in that, normally, programs written for the high-speed reader cannot be run on the 400 cpm reader because certain checking features on the high-speed reader are not available on it, slow machine.

Card Reader Ready Instructions

Instructions are provided that enable the programmer to determine whether or not the card reader is ready to receive a read card instruction. A programmer can thus delay or perform other tasks until the reader becomes ready.

BCR 2514006 Word Times: 2

Functional Description: BRANCH ON CARD READER READY. If the card reader is ready to read cards and the card hopper is not empty, the central processor executes the next sequential instruction; if not ready, the next instruction is skipped and the second sequential instruction is executed.

BCN 2516006 Word Times: 2

Functional Description: BRANCH ON CARD READER NOT READY. If the card reader is not ready to read cards, or if the card hopper is empty, the central processor executes the next sequential instruction; if the card reader is ready, the next instruction is skipped and the second sequential instruction is executed.

The examples below show typical use of these instructions.

Example 1: In this example, processing does not commence until the card is completely read. Where a card read operation is only an interruption of the main program this approach would not be adequate.

GAP Coding:

Opr	Operand	X	REMARKS
R C D	B 4		READ CARD DECIMAL MODE
H C R			HALT CARD READER
B C R			TEST FOR COMPLETION OF READING
B R U	* + 2		BRANCH AND PROCESS CARD
B R U	* - 2		BRANCH BACK TO TEST UNTIL CARD IS READ
L D A	B 4		PROCESSING OF CARD

Example 2: Check for card reader being ready, read a card and delay further processing until card reader becomes ready. Note the use of the asterisk.

GAP Coding:

Opr	Operand	X	REMARKS
B C N	*		CHECK FOR CARD READER READY
R C D	C R D I N		READ CARD DECIMAL
H C R			HALT CARD READER
B C N	*		DELAY FOR COMPLETE CARD READ BEFORE FURTHER PROCESSING

Programming the Card Reader

In most programs, it is easier for the programmer to read a card and halt than it is to read continuously. The operating speed of the card read decreases when the HCR is used as shown below.

Card Reader Model	Card Read Speed (CPM)	
	Non-Continuous	Continuous
400 cpm	360	400
1000 cpm	900	1620

When the high-speed reader is operating in the continuous mode, a maximum of 30 ms is available for processing between cards. If during a continuous read, an HCR command is given within 2 ms after the sync word is set on the card just read, only one more card enters memory before the card reader halts.

Since the sync word generated by the card reader contains information pertaining to the reading of a card, the programmer utilizes this information before processing the data from the card.

The brief flow chart of Figure 6-23 serves as one example of the procedure that a coder could follow in programming the card reader. The coding applying to this flow chart is shown by Figure 6-24. The 400 cpm reader is used for these illustrations.

Although the sync word contains information on each card read, it is the responsibility of the programmer to determine if the card read was normal. If not, the corrective measures required must be programmed. The coding of Figure 6-24 is one method for checking on sync error, EOF, or hopper empty conditions. The card read routine written for the high-speed card reader, in addition to checking the conditions illustrated, can check for invalid characters, stacker full, and hopper empty with EOF button set. The extent to which the various checks are carried can be determined by the needs of each installation. An example of checking for an invalid character is shown below.

Symbol	Opr	Operand	X	REMARKS
	LDA	CRD+27		SYNC WORD
	SUB	INVLID		INVALID CHARACTER CONST. OCT 2606073
	BNZ	PROCES		DATA OK PROCESS
	LDX	ZERO	2	ZERO INDEX CELL 2
	LDA	CRD	2	LOCATE WORD CONTAINING INVALID
	BMI			CHARACTER BY CHECKING FOR MINUS
	BRU	BADWD		SIGN
	INX	1	2	
	BXL	26	2	
	BRU	*-5		
BADWD	STX	MSG#3+5	2	STORE WORD # CONTAINING INVALID CHAR
	SPB	PRINTY	1	TYPEWRITER ROUTINE
	LDA	MSG#3		TYPEWRITER MESSAGE GIVING OPERATOR
	DEC	6		NECESSARY INSTRUCTIONS THEN READ
	BRU	START		NEXT CARD

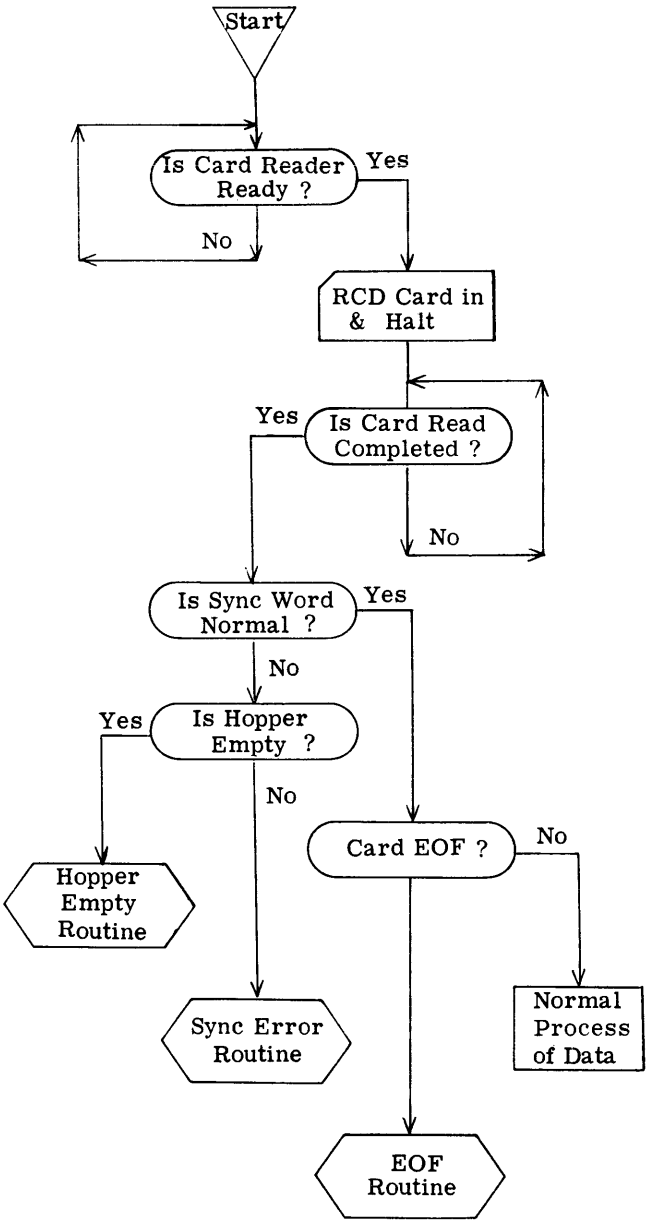


Figure 6-23 Flow Chart of Typical Card Read Procedure

Once an invalid character is detected, bit position 17 of the sync word is 0. The word containing the character is identified by the sign bit being minus.

Example of BCD Word with Invalid Character as Identified by Card Reader

1	0	0	1	1	0	1	0	0	0	1	0	0	1	0	0	0	0	1	1

Indicates Invalid Character

PROGRAMMER: G E Coder | PROGRAM: Run #3 Quality Control | DATE:

Symbol	Opr	Operand	X	REMARKS
	ORG	256		CARD READ AREA
CRD	BSS	28		
	ORG	1000		PROGRAM ORIGIN
SYNC	OCT	2606077		SYNC CONSTANT
HPEMT	OCT	1000000		HOPPER EMPTY CONSTANT
ZERO	DEC	0		
CRDEOF	ALF	END		CARD EOF CONSTANT
START	BCN	*		TEST FOR CARD READER READY
	RCD	CRD		READ CARD DECIMAL
	HCR			HALT CARD READER
	LDA	CRD+27		SYNC WORD
	BPL	*-1		DELAY UNTIL CARD READ COMPLETED
	SUB	SYNC		CHECK SYNC WORD FOR NORMAL READ
	BZE	*+6		NORMAL READ
	SUB	HPEMT		CHECK FOR HOPPER EMPTY
	BZE	EMPTY		BRANCH TO HOPPER EMPTY ROUTINE
	BRU	RDERR		BRANCH TO READ ERROR ROUTINE
	LDA	CRD+1		SECOND DATA WORD FROM CARD
	SUB	CRDEOF		CHECK FOR CARD EOF
	BZE	WINDUP		BRANCH TO EOF ROUTINE
PROCES	LDA	CRD		Q C CODE
	SBO			SUBTRACT ONE
	BZE	AREA#1		PARTS FROM AREA #1
	SBO			
	BZE	AREA#2		PARTS FROM AREA #2

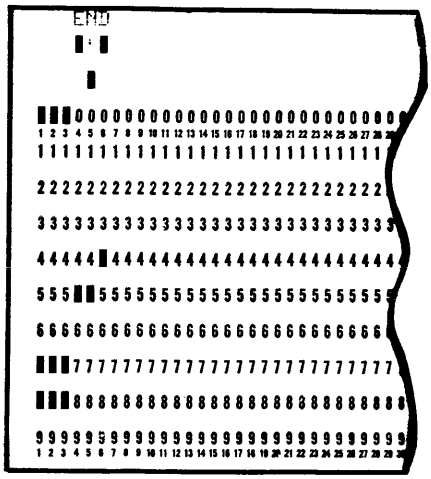
Figure 6-24. GAP Coding of Card Read

The corrective action required is determined by the individual needs of each installation. In a similar manner, other types of conditions indicated by the sync word can be detected and the necessary actions taken.

CARD END OF FILE CONVENTION. The End of File (EOF) card is used to designate the end of a deck of punched cards. This card, when detected, signifies that all cards of the deck have been read.

Recommended EOF cards for the GE-225 are shown below.

BCD File: Punches in rows 0,7, and 8 of columns 1, 2, and 3 and END punched in columns 4 through 6.



BCD
EOF Card

These columns appear in memory as:

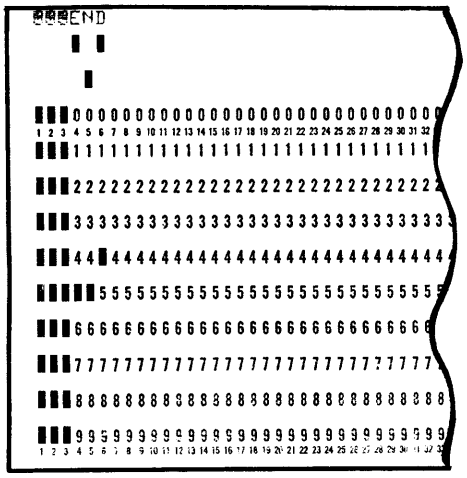
Word #1

0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0		7		7		7		7		7		7					

Word #2

0	0	0	1	0	1	0	1	1	0	0	1	0	1	0	1	0	1	0	0
0		25				45								24					
		E				N								D					

Binary File: Punches in rows 0 through 9 in columns 1, 2, and 3 and END punched in columns 4 through 6.



EOF Card

Binary

For 10-row binary this card appears in memory as:

Word #1

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3		7		7		7		7		7		7					

Word #2

1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	0
3		7		7		6				0				2				0	

Word #3

0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0		0		4		0				0				4				0	

For 12-row binary, each column is placed in one word, bit positions 8 through 19. Only columns 4, 5, and 6 are shown:

Column #4
Word #4

0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1
0		0		0		5				7				7				7	

Column # 5
Word #5

0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
0		0		0		2				0				2				0	

Column #6
Word #6

0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0
0		0		0		4				0				4				0	

Card Punch Operations

The GE-225 card punch is an on-line device used for providing output information in the form of punched cards. Available in two models with punching speeds of 100 and 300 cpm, respectively, the punches can produce punched cards in three modes, standard Hollerith, 10-row binary, and 12-row binary.

A punch instruction causes the punching of a single card regardless of the punch mode.

The memory address from which the information is punched must be a multiple of 128, but less than 2048. The card punch sets up and punches information on the card a row at a time, with the 12-row first, then the 11-row, 0-row, etc., and the 9-row last. All information punched is transferred from memory through the M register to an 80 column buffer before a set of punching dies, under control of the punch logic, is activated. Since punching is accomplished independently of the central processor, other computer operations can occur simultaneously. The punch has the lowest priority in access to memory.

An optional feature permits cards punched in Hollerith can be checked for double punches and for blank columns if desired. The 100 cpm punch checks up to 30 columns in any combination.

Punching Instructions

The GE-225 card punch operates in three modes: decimal, 10-row binary, and 12-row binary. The punching mode is determined by the punch instruction.

WCD Y 250YY02 Word Times: 2

Functional Description: WRITE CARD DECIMAL. The information in memory locations Y through Y+26 (where Y is a multiple of 128, but less than 2048) is punched into a card in decimal (alphanumeric) format. If the card punch is not in ready status when this instruction is given, the computer halts and a card punch error is indicated on the control console.

The WCD instruction results in the three 6-bit BCD characters of each word in the memory punching area being converted into the equivalent Hollerith or standard card code and punched into the card. Figure 6-14 illustrates a Hollerith card. Note that bit positions 0 and 1 of each word are not punched. Figure 6-17 contains the Hollerith code characters applicable to the punch. Twenty-seven successive memory words are needed to fill one 80 column card.

WCB Y 250YY03 Word Times: 2

Functional Description: WRITE CARD BINARY. The information in memory locations Y through Y+39 (where Y is a multiple of 128, but less than 2048) is punched into a card in binary format. If the card punch is not in ready status when this instruction is given, the computer halts and a card punch error is indicated on the control console.

One 20-bit memory word occupies two 10-row binary card columns. Thus 40 consecutive memory words fill one card. Figure 6-18 illustrates a 10-row binary card. Note that card rows 12 and 11 are not used.

WCF Y 250YY17 Word Times: 2

Functional Description: WRITE CARD FULL. The information in memory locations Y through Y+79 (where Y is a multiple of 128, but less than 2048) is punched into a card in 12-row binary format. The 12 bits punched are the least significant bits of successive memory locations. Bit position 8 is punched in row 12 and bit position 19 is punched in row 9. If the card punch is not in the ready status when this instruction is given, the computer halts and a card punch error is indicated on the control console.

In 12-row binary, 80 consecutive memory words are needed to fill one card. Figure 6-19 is an example of this type of card.

Punch Ready Instructions

Before punching a card, the punch must be in a ready status or a card punch error results and the computer halts. Instructions are available that enable the programmer to determine if the punch is ready.

BPR 2514007 Word Times: 2

Functional Description: BRANCH ON CARD PUNCH READY. If the card punch is in a ready status, the computer takes the next sequential instruction. If not ready, the computer skips the next instruction and executes the second sequential instruction.

BPN 2516007 Word Times: 2

Functional Description: BRANCH ON CARD PUNCH NOT READY. If the card punch is not in a ready status, the computer takes the next sequential instruction; if it is, the computer skips the next instruction and executes the second sequential instruction.

Programming the Card Punch

Card punching is time consuming, and it is to the programmers advantage to punch at maximum speed. When a punch instruction is given, a single card is punched. In order to punch at maximum speed, another punch instruction must be given within 10 milliseconds after completion of punching the previous card. Failure to meet this timing requirement results in maximum punching rates of 50 cpm for the 100 cpm punch and 180 cpm for the 300 cpm punch.

Example 1: Test for punch ready and punch a Hollerith card from symbolic memory address PUNCH.

GAP Coding:

Opr	Operand	X	REMARKS
6 9 10	12 13 14 15 16 17 18 19	20	31 75
B P N	*		DELAY UNTIL PUNCH READY
W C D	P U N C H		PUNCH HOLLERITH CARD

Symbolic address PUNCH must be a multiple of 128 but less than 2048. Punching from 128 is not recommended

since the Automatic Priority Interrupt (API) uses this area.

Example 2: Multiple punch areas can be used with punching proceeding from one area while the other punch area is being loaded. Normally, a subroutine is written to accomplish punching. A sample routine is shown.

GAP Coding:

```
Symbol   Opr   Operand   X   REMARKS                          Sequence
PUNCH    BPN   *             TEST FOR PUNCH READY                  5
         DLD   AREAS         PUNCH AREAS                         10
         STA   *+3           SET-UP PUNCH INSTRUCTION            15
         XAQ                 EXCHANGE PUNCH INSTRUCTIONS         20
         DST   AREAS                                             25
         NOP                 PUNCH CARD                          30
         BRU   1           1 EXIT (SPB ENTRY USED)               35
AREAS    WCD   384           PUNCH #1                            40
         WCD   512           PUNCH #2                            45
```

SECTION VII

CONTROLLER SELECTOR OPERATIONS

Certain GE-225 high-speed input-output peripherals do not access memory directly, but are buffered by means of controllers which, in turn, are granted memory access through a control and data transfer device, the controller selector. Figure 1-2 illustrates this relationship. The auxiliary arithmetic unit (AAU), although connected to the controller selector, has characteristics that distinguish it from the high-speed peripherals. While it is not an input/output unit, it is discussed in a later section like other peripherals.

CONTROLLER SELECTOR PRIORITY

Because the controller selector serves as a means of communicating between peripheral controllers and memory, each controller must have a unique address and a specified memory priority. This is accomplished with plug-in connectors which tie together the peripheral controllers and the controller selector.

The controller selector assigns each of the eight available plugs a unique memory access priority. The lower the plug number the higher is the priority, as shown in Figure 1-5. The relationship of priority to plug number means that the memory access requirements of the peripheral device must be taken into consideration before it is assigned to a specific plug. The controller selector has a data transfer rate of 55,000 20-bit words per second, which is more than sufficient for a typical GE-225 installation. A GE-225 system may have any combination of input-output controllers except for the following limitations: No more than 1 AAU, 2 41-Kc. magnetic tape controllers, 2 DSU controllers, or a combination of 2 41-Kc. magnetic tape and DSU controllers.

Devices with high memory access requirements, such as a mass random access data storage (MRADS) unit, require high priority plug numbers. Devices that can wait for access to memory without loss of information are assigned low priority. Plug assignments should be determined during the early stages of system planning and all programmers informed of the plug number of each device. Recommended plug assignments whenever possible are:

Plug Number	Peripheral Controller
0	Mass Random Access Data Storage (MRADS)
1	2nd MRADS or Magnetic Tape
2	Magnetic Tape
3	Magnetic Tape or Document Handler Adapter
4	Document Handler Adapter
5	Doc. Handler Adapter/DATANET-15
6	Printer
7	AAU

The adoption of these assignments increases compatibility of software and back-up between installations.

CONTROLLER SELECTOR INSTRUCTIONS

Input-output operations of peripherals connected to the controller selector are accomplished by a sequence of instructions.

The controller selector should first be tested to determine if it is in a ready state before issuing an instruction to perform an operation. Attempted execution by the computer of a SEL command (discussed below) when the controller selector is busy results in an alert halt condition and hangs up the computer. Interrogation of the controller selector is done by one or more BCS instructions, which are discussed in the sections on high-speed peripheral operations.

BCS XXX P 2514P2C/2516PCC Word Times: 2

Functional Description: BRANCH ON CONTROLLER SELECTOR. The peripheral connected to controller P is tested for the condition (CC) indicated by a mnemonic placed in the operand address field identified by XXX above. The BCS instructions are listed and described with the instructions for the various peripheral devices.

If the controller selector is ready, the plug containing the peripheral controller that is to be placed in operation must be selected by a Select (SEL) instruction.

SEL P X 2500P20 Word Times: 2

Functional Description: SELECT. The peripheral connected to controller P (addresses 0 through 7) is selected for the operation indicated by an associated instruction. The execution of the SEL command always sends the contents of the next two memory locations to the selected peripheral controller. Execution of the SEL instruction also resets controller error conditions.

Every peripheral connected to the controller selector requires three memory words containing instructions to perform an operation: the SEL instruction selecting the controller and two other words instructing the controller to perform a specific task. The instructions contained in the two words following the SEL command are not executed by the central processor. Therefore when the SEL is in the I register, the P register will hold the address of the third sequential instruction.

Example of SEL Coding:

Opr	Operand	X	REMARKS
8 9 10	12 13 14 15 16 17 18 19	20	31
S E L	6		SELECT PLUG NUMBER 6

The contents of the two words following the SEL instruction is governed by the operation desired and by the peripheral equipment to be used. Specific details for programming these peripheral operations are given in subsequent sections.

AUTOMATIC PROGRAM INTERRUPT (API)

A GE-225 optional feature makes it possible to program an automatic interruption of the main program to process a higher priority program. This feature, when used with the Automatic Priority Interrupt Executive Routine, controls the simultaneous operation of two or more unrelated programs. The system combines peripheral-to-peripheral runs (e.g., tape-to-printer, tape-to-punch, and card-to-tape) with a main program and can control programs associated with the remote inquiry stations.

The API feature provides for automatic interrupt of the main program whenever selected peripheral controllers change status from 'not ready' to 'ready'. This allows control to be transferred automatically from the main program to the executive routine designed to service the peripherals. Each controller on the GE-225, the card reader, and the card punch, can signal the GE-225 that it has finished an operation, and is ready for another operation. This signal may, or may not, cause a physical interrupt on the GE-225, depending upon the status of the computer. The typewriter and paper tape reader or punch cannot cause automatic program interrupt.

A switch is provided for each peripheral controller which allows only desired peripherals to cause an API thereby, in effect, masking out devices for which an interrupt is not desired.

When the switch is 'ON', the peripheral controller will be allowed to cause an automatic interrupt (under designated interrupt conditions).

When the API switch is 'OFF', the peripheral controller will not be allowed to cause automatic interrupt (under any conditions).

When a GE-225 system operates with API, the computer may be in a specific mode of operation within the program being executed. These operating modes and program are defined as:

Non-Interrupt Mode	A mode of operation in which the GE-225 is not processing a priority program; and can not be physically interrupted by a signal from a peripheral device. When power is initially applied to the GE-225, the GE-225 is in the Non-Interrupt Mode.
Interrupt Mode	A mode of operation in which the GE-225 is not processing a priority program; but can be physically interrupted as a result of a signal from a peripheral device. A set mode is required to place the GE-225 in the Interrupt Mode.
Priority Mode	A mode of operation in which the GE-225 is processing a priority program, as a result of being physically interrupted while operating on a main program in Interrupt Mode.

Definitions

Main Program - The program that is being executed at all times other than when an Automatic Program Interrupt occurs.

Priority Program - A program (peripheral-to-peripheral) that is designed to be executed in the Interrupt Mode.

Remote Inquiry Program - A program that controls the Remote Inquiry hardware and is executed in the Interrupt Mode.

Program Interrupt Instructions

SET	PST	2506015	Word Times: 2

Functional Description: SET AUTOMATIC PROGRAM INTERRUPT ON is required to cause the program interrupt feature to be effective. This instruction causes the computer to enter and remain in the interrupt mode until the priority program is completed and directions are given for return to the main program. This command must be given before the main program can be interrupted. If a programmer does not wish to use the interrupt feature, he merely avoids executing a SET PST.

SET	PBK	2506016	Word Times: 2

Functional Description: SET AUTOMATIC INTERRUPT OFF is required to disable the program interrupt hardware. This instruction causes the computer to leave the interrupt mode and remain in the normal mode until the mode is reset by a SET PST instruction.

To prevent the main routine from being interrupted after a SET PST has been executed, a SET PBK must be executed.

Because the program interrupt feature becomes effective whenever the command SET PST (Priority Set) is executed and becomes ineffective when the command, SET PBK (Priority Break) is executed, any attempted interrupt (caused by a change in status of one of the selected controllers) which occurs during the time when Automatic Interrupt is not set will be remembered and will cause an automatic interrupt immediately following the next SET PST. It then becomes the responsibliity of the Executive Routine to determine which of the selected peripheral controllers changed status and must be serviced.

Operation of API

When automatic interrupt is initiated, the following events occur:

1. Interrupt of the main program is delayed until the next instruction access time. (The P counter contains the address of the next instruction.)
2. The computer automatically selects index group 32. NOTE: Index group 32 is available only on GE-225 systems with the API feature and can be used only as prescribed for API.
3. The contents of the P counter are stored in word one of the API index group 32 (memory location 0129).
4. Control is transferred to address 0132 (the first word following index group 32) which is the start of the Executive Routine and an automatic priority break occurs.
5. During the time that control remains with group 32, the SPB command (if used) will refer to group 32 only.

The only index group available during the Executive Routine is group 32. It must be remembered that the address of the next instruction to be accessed in the main program has been stored in word 1 of this group and the contents must not be destroyed. The computer cannot be interrupted again until SET PST command has been executed as described below.

To return to the main program, the following procedure is required:

1. A SET PST command is required in all cases regardless of whether or not it is desired to continue under control of the program interrupt feature. If the programmer wishes to return to the main program with program interrupt disabled, the SET PST must be followed by a SET PBK.
2. An indexed unconditional branch (BRU) to location zero, modified by word one of index group 32, sets the P counter to the address of the next main program instruction to be accessed. This is always the final step in the sequence for returning to the main program.
3. Any peripheral controller that changed from not ready to ready status while the computer was under control of the Executive Routine will cause an interrupt after return to the main program.

It is permissible to execute any number of instructions between the SET PST and the indexed BRU which is used to transfer control back to the main program.

Also, any number of BRU instructions can be executed while in the interrupt mode.

When API is set in the program, the following occurs when a controller goes from not ready to ready status:

1. P counter + 1 is stored in location 0129_{10}.
2. Control is transferred to location 0132_{10}.
3. At this time, any or all controllers may or may not be tested and may or may not be 'put to work'. It is not necessary, however, to do any testing or to issue any commands to return to the main program.
4. The computer-generated-and-executed SPB 132_{10}, word 1, is the instruction which turns the API flip-flop off in the central processor. This generated instruction, in effect, also executes a SET PBK instruction. Any controller becoming ready while the program is interrupted will be remembered until the priority is SET and the modified branch is executed, at which time the API flip-flop will be set again if any controller went ready during the time the 'pseudo' SET PBK instruction was executed by the computer.

Once a controller causes an interrupt, it will not cause another automatic interrupt until it goes from the not ready to ready status again.

API Executive Routine

The API executive routine (CD225J4.000) is in memory with every main program or remote inquiry program. Programs with precedence or remote inquiry programs may be in memory, if desired. The API executive routine:

1. Performs functions necessary for starting and ending all programs being executed under its control.
2. Saves the A and Q registers and the overflow indication when a main program is interrupted because of a peripheral going from busy to not busy.
3. Determines which peripherals are in ready state and executes the appropriate priority programs.
4. Restores the A and Q registers and the overflow condition before returning control to the main program.

Three basic combinations of programs are designed to share memory and peripherals with the API executive at execution time. These are:

1. A main program and from one to four priority programs.
2. A main program and a remote inquiry program.
3. A main program, from one to three priority programs, and a remote inquiry program.

The Automatic Program Interrupt Executive has as its basic configuration the GE-225 with a 4K or larger memory. Any configuration of peripherals may be used in conjunction with this, excluding the document handler and paper tape reader-punch. The system must include the API feature.

The routine requires 97 memory locations and, when added to the front of a user's program, is assembled into the following areas:

1.	0128_{10}	0141_{10}	=	14 locations
2.	0143_{10}	0169_{10}	=	27 locations
3.	0552_{10}	0606_{10}	=	55 locations
4.	0606_{10}	0639_{10}	=	34 locations for future expansion

With the exception of programs for magnetic tape and MRADS controllers, programs must not refer to peripherals used by another program in the same load. When magnetic tape and MRADS controllers are both used, the same handler on MRADS must not be addressed.

Programs must not refer to memory areas used by another program, except in the use of common subroutines.

Card read-in areas are restricted to locations 0256_{10} and 0384_{10}, for programs being executed under the control of API Executive.

Card punch areas are restricted to locations 0512_{10} and 0640_{10}, for programs using API Executive.

All symbols used in the executive routine start with #API.

Locations 0142_{10} and 0144_{10} are reserved for remote inquiry and must contain zeros if remote inquiry is not used.

Restart is provided only for the main program.

All programs being executed simultaneously should used the same tape or MRADS input/output routine.

It is permissible to use two different magnetic tape I/O routines only if they refer to different tape controllers or if the read/writers are not buffered and a delay, error check, and correct is done after each.

Hardware Operation

Each controller, the card reader, and the card punch can generate a signal to the central processor that it has finished an input/output operation, and is ready for another command. Whether or not this signal is actually sent to the central processor depends upon the setting of the API switch associated with each device. The controller switches are located on the inside of the controller, usually near the controller selector plug. The card reader and card punch switches are located inside the top door on the front of the control console. With this switch off, the interrupt signal from the device is not sent to the central processor. The switch must be on for the central processor to receive the signal from the I/O device.

The action of the central processor when it receives an interrupt signal depends upon the mode of operation. Non-Interrupt Mode is established by a SET PBK command, or by resetting the computer through depression of the power on button. In the Non-Interrupt Mode, the signal merely sets a latch to remember that it received the signal for later use at such time as Interrupt Mode is set. Interrupt Mode is established by a SET PST command.

When a physical interrupt occurs, the central processor enters the Priority Mode of operation. The location of the next command to be executed in the main program (note the difference from normal SPB operation) is stored in word 1 of API index group 32 (location 201 octal). Index group 32 is set automatically; and program control is transferred to octal location 204. A SET PBK operation is executed automatically as a result of the interrupt, resetting the latch associated with the I/O devices, and dropping the Automatic Interrupt Mode. Further signals from I/O devices becoming ready during Priority Mode set the I/O latch again so that another interrupt may occur when the priority program is finished and Interrupt Mode is re-established.

When the priority program has completed its operations, control is returned to the main program by issuing a SET PST, followed by a BRU 0, index word 1. (Any modified BRU following the SET PST will cause exit from Priority Mode. Modified BRU instructions prior to issuing the SET PST have no effect, and operate normally in group 32 in Priority Mode.) Issuance of the SET PST followed by a BRU 0, word 1, will cause a return to the main program and the previous index group that the main program was operating in when the interrupt occurred. Upon return to the main program, the computer is in the Interrupt Mode. If it is desired to return to a main program from a priority program in Non-Interrupt Mode, a SET PBK should be executed between the SET PST and the BRU 0, word 1.

Interrupts can occur only at the point that an instruction has been executed completely and another instruction is about to be accessed. After a test such as BZE, an interrupt will not occur until the computer has analyzed which route it should take. Interrupts can not occur between a BRU and the location to which it goes. Hence, a program loop such as BRU * cannot be interrupted.

Programming Considerations

Each main program to be used in conjunction with the API and a priority program should be carefully scrutinized to ascertain what damage if any, could result from an interrupt at any given point. For instance, an interrupt between a RCD and an HCR might result in continuous reading of cards. (An HCR instruction at the beginning of the priority program will prevent this.) An interrupt in the middle of a type routine might result in the loss of the N register contents and a meaningless message. An interrupt just after a test-and-branch, such as BZE, has been executed might prove disastrous if the priority program should reverse the condition just after the test is made. Each of the above conditions might necessitate a SET PBK and a SET PST around the routine to prohibit interrupt during the crucial operation. Care should be exercised not to abuse the ability to prohibit interrupts in this manner, however, or the effectiveness of API will be unnecessarily reduced.

Sample API Problem

Assume that it is desirable to operate two programs concurrently within GE-225 memory. One program is a card-to-tape conversion, the other represents an independent processing function. This problem can be solved efficiently by use of the program interrupt feature, without use of the API Executive Routine.

Card-to-Tape Conversion - This should be the priority routine since it involves few program steps, requires continuous use of peripherals, and execution depends upon the card reader and the tape-controller being in a ready status.

Independent Processing Function - This should be the main program because it requires many program steps and is much less reliant upon peripheral use and readiness for processing.

Symbol	Opr	Operand	X	REMARKS
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31 75
A 1	D E C	5 1 2		First card read-in area
	D E C	6 4 0		Second card read-in area
A 2	D E C	5 1 2		First card read-in area
	D E C	6 4 0		Second card read-in area
C O N S T 1	D E C	1 4 2		Transfer location
C O N S T 2	D D C	0		Storage area for contents of the A and Q registers
	B C N			Test for card reader not ready
	B R U	A 6		Exit if card reader is not ready
A 3	R C D	0 5 1 2		Read card into memory beginning at 0512
	H C R			Halt card reader
	D S T	C O N S T 2		Store contents of A and Q registers for main program
	D L D	A 1		Load read-in area constants
	X A Q			Switch read-in areas
	D S T	A 1		Store read-in areas as constants
	S T O	A 2		Set up alternate card read-in area
A 4	B R U	A 7		Bypass writing a tape record the first time through
	B C S	B T N	2	Test for tape controller not ready
	B R U	* - 1		Delay until tape controller is ready
	S E L	2		Select controller selector address two
A 5	W T D	0 5 1 2	1	Write tape in decimal mode from memory locations beginning at 0512 onto tape 1
		2 7		Write a maximum of 27 words
	D L D	A 2		Load read-in area constants
	X A Q			Switch read-in area constants
	D S T	A 2		Store read-in area constants
	S T O	A 5		Set up memory address from which tape record is to be written
A 6	D L D	C O N S T 2		Load contents of the A and Q registers from main program
	S E T	P S T		Set priority interrupt mode on
	B R U		1	Branch to zero as modified by word one of index group 32; i.e., to the setting of the P counter when the main program was interrupted
A 7	L D A	C O N S T 1		Load binary equivalent of 142
	S T O	A 4		Will cause the writing of tape records all succeeding times through the program
	B R U	A 6		Transfer to exit

Figure 7-1. GAP Coding for API Problem

The programmer should realize that use of the API executive routine extends the usefulness of the API feature and reduces the housekeeping functions and checks necessary for efficient use.

SECTION VIII

MAGNETIC TAPE OPERATIONS

MAGNETIC TAPE

Magnetic tape is one of the most widely used computer input-output media. Light in weight, compact, and very durable, magnetic tape is excellent for storage of data which later can be quickly processed. A single reel of tape is equivalent to a minimum of nine 2-foot file drawers, or about 25,000 punched cards.

The GE-225 magnetic tape system is a fast and flexible means of efficiently processing such large files of data. Information can be permanently recorded on tape quickly and accurately, and magnetic tape data can be selectively changed or completely erased by the write operation. Since magnetic tape can be used again and again, its re-use represents a savings in the total cost of processing data.

Magnetic Tape Advantages and Disadvantages

As a recording and storage medium, magnetic tape has both advantages and disadvantages. Briefly, some of these are:

Advantages

Compact recording density permitting high volume density

Fast and accurate recording of variable lengths of data

Permanent data storage

Re-use resulting in low cost per unit of recording

Less storage bulk

Audit trail through retention of tapes

Disadvantages

Non-random access to data

Recorded data invisible to the eye

High initial equipment costs

Housekeeping routines are necessary which use memory and consume time

Physical Characteristics

Magnetic tape normally has a ferro-magnetic surface coating such as iron oxide held to a mylar or acetate base ribbon by a hard or soft binder material. Tapes for the GE-225 have a hard binding material between the coating and the base. The hard binder provides better wear characteristics, reduced tendency to shed oxide coating in the form of a fine powder, and ability to withstand wider temperature ranges. Figure 8-1 illustrates a section of magnetic tape.

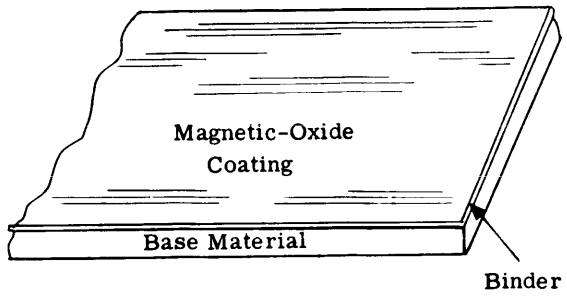


Figure 8-1. Composition of Magnetic Tape

The tape used by the GE-225 is 1/2 inch wide and up to 2400 feet long. It is wound on 10-1/2 inch diameter plastic reels. Shorter lengths of tape can of course be used; however tapes less than 50 feet long have limited applications.

Beginning and End of Tape Detection

The beginning and end of tape is detected in magnetic tape handlers by photoelectric sensing of tape markers. Tape markers are strips of aluminum foil 3/16 by 1 inch placed 10 to 15 feet from the physical beginning of the tape and 15 to 20 feet before the physical end of the tape. The strips, referred to as leader and trailer foils, indicate the beginning and the end of the portion of tape that is used when reading or writing.

To insure that sufficient tape exists beyond the trailer foil to permit memory dumps, the tape reels should be visually checked and measured. The following table shows tape lengths required for binary memory dumps at different densities and memory sizes:

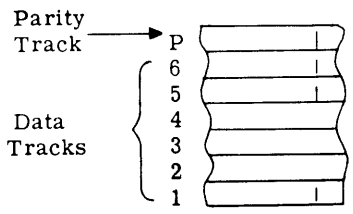
Density	8 k Dump	16 k Dump
15 kc/200 bpi	14 feet	28 feet
42 kc/555 bpi	5 feet	10 feet

Recording of Data

Information is recorded on magnetic tape as changes in the magnetic pattern in seven parallel tracks, or channels. The pattern is changed in appropriate channels as the tape moves across a multiple channel recording head consisting of seven magnetic write heads. These write heads precede a set of seven read heads; thus, as data is written, it can be immediately read to provide a check that assures that the information has been written correctly.

Magnetic Tape Characters

A magnetic tape character consists of six bits of data written laterally across the width of the tape in six of the seven parallel tracks. The seventh track contains a parity bit which is used for checking the validity of the tape character. The tracks across the tape are numbered laterally:



Magnetic Tape Records

A single collection of data written from memory onto magnetic tape is called a tape record. A record can vary in length from one word to the entire memory contents (4, 8, or 16 thousand words). Normally, the length of a record is a practical size as determined by the processing requirements.

Inter-Record Gap

Tape records are separated by a three-quarter inch gap of erased tape called an interrecord gap, or IRG. During writing, the IRG is automatically produced at the end of each record and, when reading, the record begins with the first tape character after an IRG and continues to the next gap. Thus a tape record is preceded and followed by an IRG, as shown in Figure 8-2, and the IRG is used for starting and stopping between records.

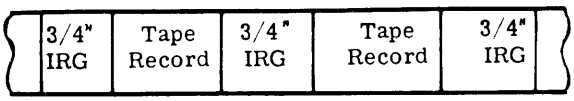
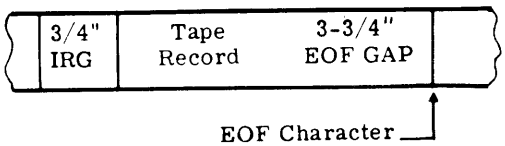


Figure 8-2. Magnetic Tape Records and Interrecord Gaps

End-of-File Gap

The end of all the records of a file is indicated by a 3-3/4 inch section of erased tape followed by a binary 001111 (octal 17) tape mark, or end-of-file (EOF) character. The EOF record is written on tape by programmed instructions.



File Protection

Because the writing operation automatically destroys any previous data on tape, care is imperative in processing files containing data that must be preserved. The GE-225 has a file protection ring that must be inserted in a groove in the tape reel before the tape can be written. If the write ring is in place, either writing or reading can occur. Without the ring, only reading can take place. The ring thus provides an added safeguard against accidentally destroying a file.

Magnetic Tape Handler

Each tape handler contains two reels, one for tape feeding called the supply reel, and the other for tape takeup.

The tape handler mechanism functions to drive the tape past separate read and write heads. Tape is threaded around tape guides, and between the capstans and their respective pinch rollers. In earlier models of tape handlers, pressure pads held the tape even and smooth as it passed the read and write heads; in the newer models, vacuum pockets perform that function. When the handler power is on, the tape-drive capstans rotate continuously in opposite directions (top one rotates clockwise), and are always ready to drive the tape when a pinch roller forces the tape against one of them. Tape moves forward when the pinch roller is against the forward capstan and backward when the pinch roller is against the reverse capstan. Two sensing cells in the photosensor are positioned to detect the beginning and end of tape markers.

Each tape handler has a power supply to move tape forward at a rate of 75 inches per second or backward (rewind) at 150 inches per second. The information transfer rate is 15,000 or 42,000 characters per second.

Each magnetic tape handler has a control and indicator panel which permits the operator to see various conditions of tape operation and permits him to perform necessary off-line operations. This panel is located on the front panel of the handler frame; all switches and indicators are labeled except the rotary address selector switch which is on the extreme left of the panel. On the panel are the following:

Address Selector Switch. This eight-position switch selects the channel from the controller (0-7) for on-line operations. The switch is completely disabled when the handler is set for local operation. Since each handler can be set to any one of eight channels, any tape reel can be mounted on any transport, and that handler can be selected by the computer program.

POWER ON switch and indicator. This pushbutton turns on power to the handler when depressed. It turns power off again when depressed a second time.

REMOTE/LOCAL switch and indicator. This pushbutton determines whether the handler will operate under local control from the handler's control and indicator panel or whether it will operate under remote control from program instructions relayed to the handler through the tape controller. The pushbutton is horizontally divided and changes from one condition to the other when depressed.

REWIND switch and indicator. This pushbutton operates only when the handler is set for local operation. When depressed, it causes tape to move in a reverse direction at a speed of 150 inches per second. This movement is caused by energizing the reverse pinch roller, and changing the speed of the capstan drive. The REWIND switch is normally used to return the tape to its load point, for the tape stops on the leader foil. The rewind motion is also stopped by depressing STOP or by placing the REMOTE/LOCAL switch in the REMOTE position.

REVERSE switch and indicator. This pushbutton operates only when the handler is set for local operation. When depressed, it energizes the reverse pinch roller and causes tape to move in a reverse direction at a speed of 75 inches per second until it is stopped by the sensing of the leader foil. The motion is also stopped by depressing STOP or by placing the REMOTE/LOCAL switch in the REMOTE position.

FORWARD switch and indicator. This pushbutton, when depressed, energizes the forward pinch roller and therefore causes forward movement of tape when the handler is in the local state. Tape will continue moving until the STOP pushbutton is depressed, the trailer foil is detected, the REMOTE/LOCAL switch is placed in the REMOTE position, or the REVERSE switch is depressed. The pushbutton is illuminated during forward movement.

STOP switch. This pushbutton operates only when the handler is set for local operation. When depressed, it stops all local movement of the tape handler; it is not an indicator.

WRITE INHIBIT indicator. This is an indicator which is illuminated when the tape supply reel does not have a write-permit ring. (Only reading can be done on the tape.)

ADDR indicator. This indicator is illuminated under program control whenever the magnetic tape controller is addressed by the central processor for a read or write operation.

DENSITY select switch. This switch is available on 15/42 kc magnetic tape handlers only. In the low (15 kc) position, tape can be read or written 200 bits per inch; in the HIGH (42 kc) position, tape can be read or written 555 bits per inch. Attempts to read tape with the DENSITY switch set at other than the recording density of that tape results in parity errors.

Read/Write Modes

The GE-225 magnetic tape system can read or write data in three different modes: decimal, binary, and special binary. The operating mode is specified by the particular tape instruction given.

DECIMAL MODE

When operating in the decimal mode, the magnetic tape controller automatically alters the magnetic tape characters as data is transferred to or from memory through the controller.

The zone bits of some BCD characters are altered by the tape controller as follows:

BCD CHARACTER ZONE BITS	
MEMORY	TAPE
00	00
01	11
10	10
11	01

In addition, a BCD zero in memory (000000) is changed by the controller and written on tape as 001010.

This alteration of data during the decimal mode makes GE-225 tape format compatible with decimal tape formats used by other data processing systems.

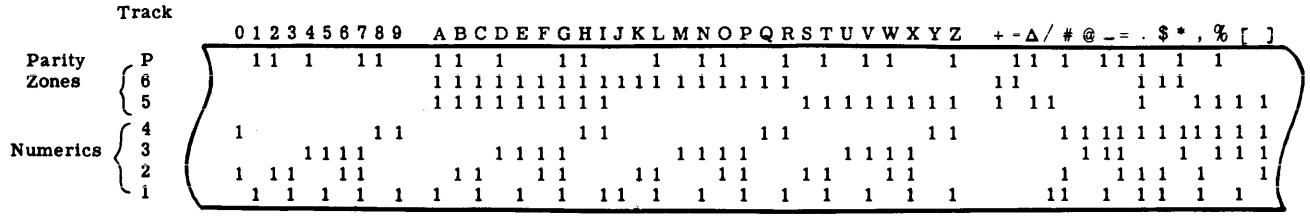
Figure 8-3 shows the octal equivalent of the BCD character set as it appears on magnetic tape, and the octal representation of each character as it appears in memory. Figure 8-4 illustrates how these characters are recorded on magnetic tape.

In the decimal mode, bit positions 2 through 19 of each memory word are written on tape as three magnetic tape characters which are referred to as a module of 3.

CHARACTER	CHARACTERS - OCTAL	
	MEMORY	TAPE
0	00	12
1	01	01
2	02	02
3	03	03
4	04	04
5	05	05
6	06	06
7	07	07
8	10	10
9	11	11
A	21	61
B	22	62
C	23	63
D	24	64
E	25	65
F	26	66
G	27	67
H	30	70
I	31	71
J	41	41
K	42	42
L	43	43
M	44	44
N	45	45
O	46	46
P	47	47

CHARACTER	CHARACTERS - OCTAL	
	MEMORY	TAPE
Q	50	50
R	51	51
S	62	22
T	63	23
U	64	24
V	65	25
W	66	26
X	67	27
Y	70	30
Z	71	31
+	20	60
-	40	40
Δ	60	20
/	61	21
#	13	13
@	14	14
_ (undrscr)	15	15
=	16	16
.	33	73
$	53	53
*	54	54
,	73	33
%	74	34
[	75	35
]	76	36

Figure 8-3. Representation of Characters in Memory and on Tape



Track P is the Parity track. Note that Parity is even for BCD and is odd for Binary mode.

Figure 8-4. BCD Characters on Magnetic Tape

Figure 8-5A shows the relationship between a BCD word in memory and the three 6-bit characters on tape. Note that bit positions 0 and 1 of the memory word are not written on tape. When reading from tape into memory, these bit positions are automatically set to zero. Figure 8-5B illustrates a word written from memory onto magnetic tape.

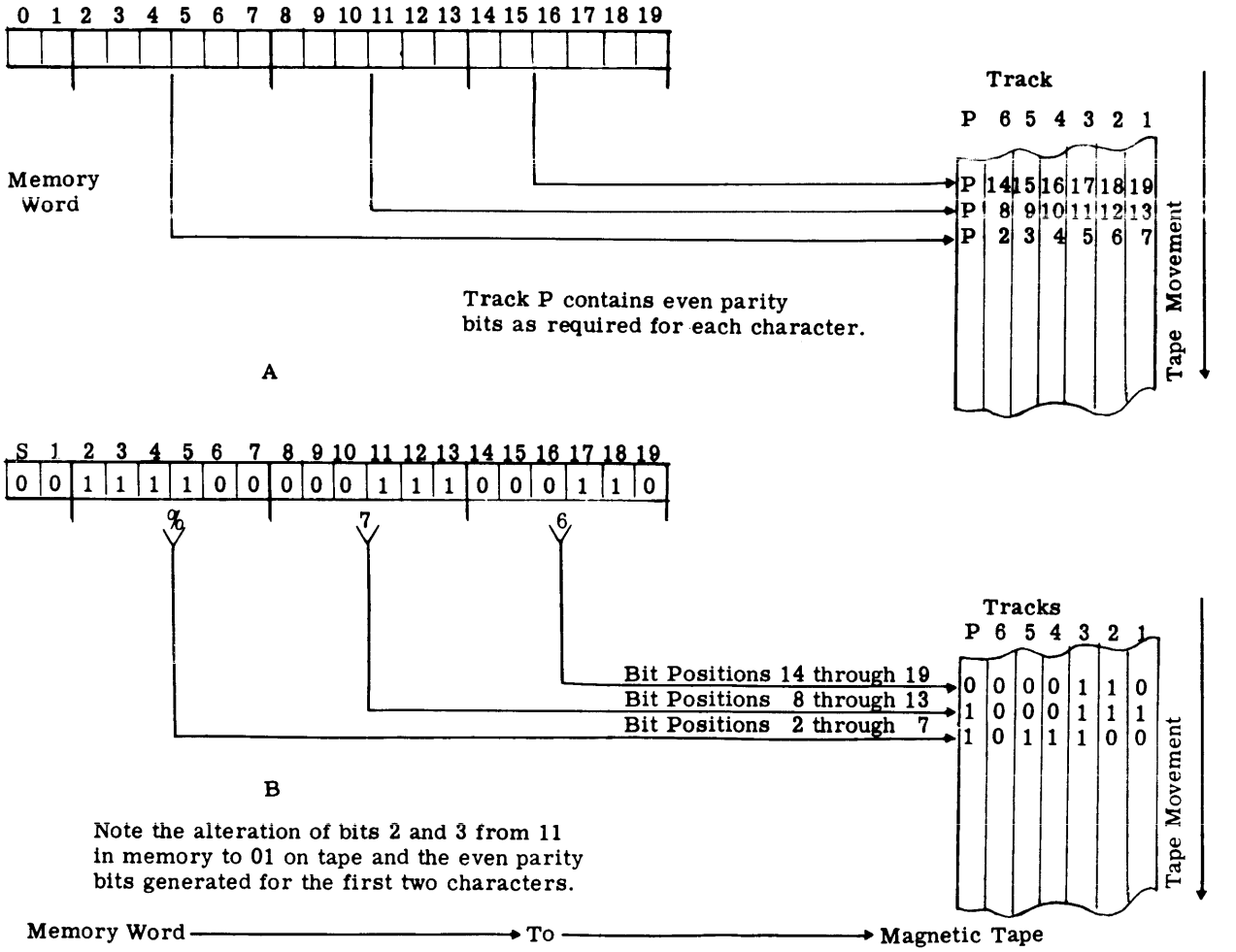


Figure 8-5. Memory to Magnetic Tape (Decimal Mode)

BINARY MODE

In the binary mode, the twenty bits of a memory word are written on magnetic tape as four 6-bit tape characters. This is referred to as a module of 4 and shown in Figure 8-6. Parity of each tape character in the binary mode is odd.

Tape tracks 1 through 4 of the first character are set to 0. When reading from tape these zeros are ignored.

SPECIAL BINARY MODE

In the special binary mode, bit positions 2 through 19 of a memory word are written on tape as three 6-bit tape characters. No alteration of data bits occurs during the transfer of data through the tape controller. Parity of each tape character is odd. When information in the special binary format is read from tape into memory, bit positions 0 and 1 of each memory word are set to 0. Figure 8-7 illustrates the transfer of a binary word from memory to magnetic tape in the special binary mode.

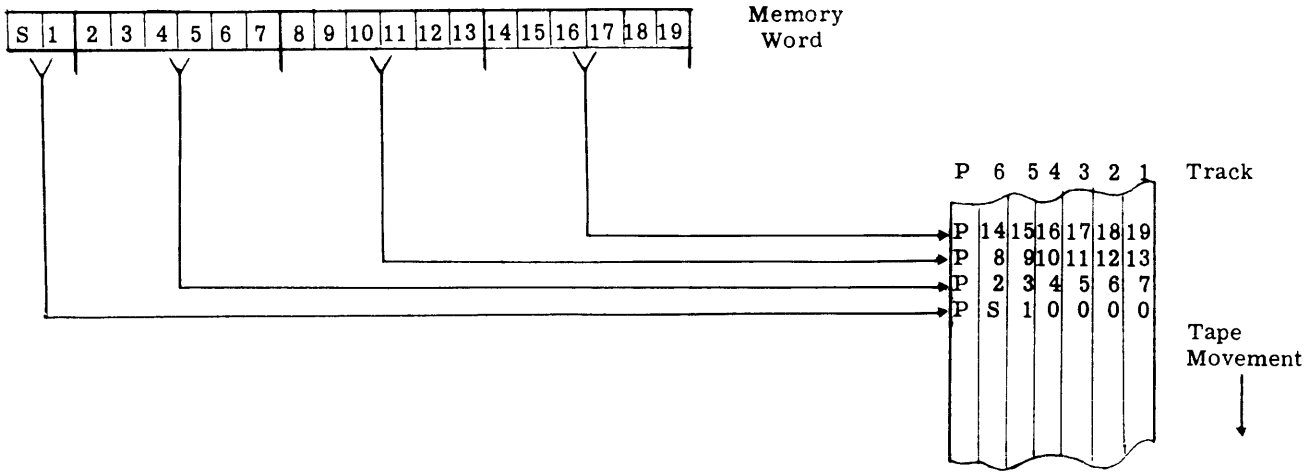
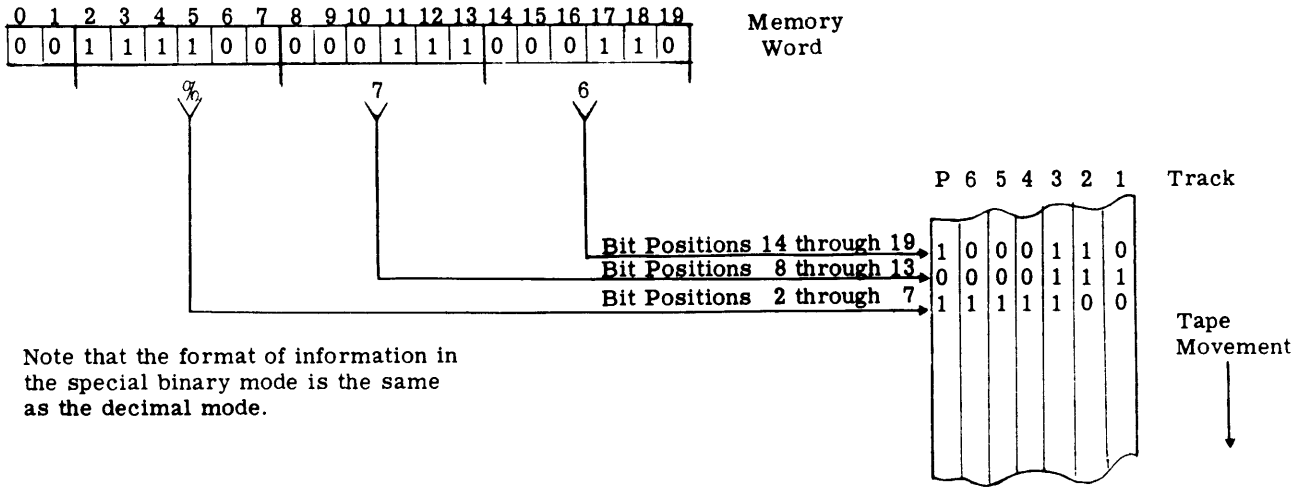


Figure 8-6. Memory to Magnetic Tape (Binary Mode)



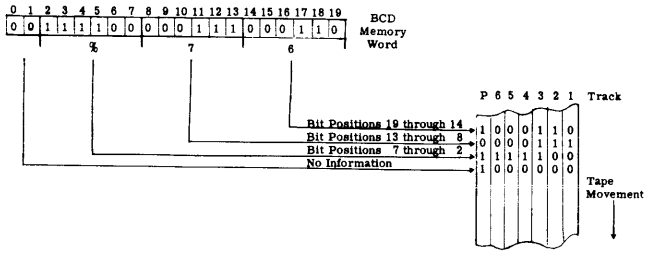
Note that the format of information in the special binary mode is the same as the decimal mode.

Figure 8-7. Memory to Magnetic Tape (Special Binary)

MIXED DATA MODES

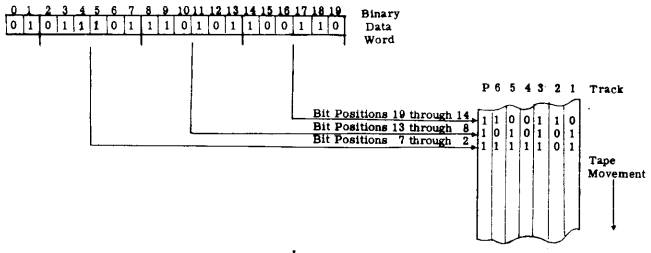
When core storage contains information in both decimal and binary form, writing and reading of this data to and from magnetic tape must be done in the binary mode so that all twenty bits of each memory word are transferred. This means, however, that each BCD memory word is placed on tape as four 6-bit characters instead of 3 characters. This fourth word on tape contains no information and is of no significant value.

Example: BCD word written on tape in binary mode.



This BCD memory word is written as four 6-bit tape characters. Parity of each character is odd.

Example: Binary data written on tape in decimal mode.



Bit positions 0 and 1 are not written on tape and thus, when read back from tape, are set to 0. However, the error detection circuits of the GE-225 will indicate an abnormal read if bits are detected in this area by the magnetic tape checking features.

MAGNETIC TAPE ALERT CONDITIONS

The GE-225 magnetic tape system automatically performs various checks to insure the accuracy of data transfer between core storage and tape. The checking circuits are also designed to prevent the execution of an illegal operation such as addressing a magnetic tape handler that is in the process of rewinding.

Magnetic tape alert conditions fall into two categories:

1. Conditions that cause an ALERT HALT of a tape handler and its associated controller and that require manual intervention to restore them to operation.

2. Conditions that do not cause an ALERT HALT, but do give an indication of abnormal tape operation.

ALERT HALT Conditions

Certain error conditions and malfunctions that may occur during magnetic tape operations cause the faulty or affected tape handler and its associated controller to halt. At the same time, the ALERT HALT indicator on the controller display panel is turned on. Dependent upon the alert condition and upon local operating procedures, some form of manual intervention is required to return controller and tape handler to operation.

In addition, an ALERT HALT may or may not cause the central processor to halt. If the alert condition occurs during the instruction process (word 1, 2, or 3), the central processor halts. If the condition occurs after the tape handler becomes busy, the central processor continues processing, but a subsequent attempt to address the halted controller and tape handler (or another handler on that controller) will halt the central processor.

Conditions that will cause an ALERT HALT are:

1. A parity error on instruction word two or three as these words are transferred from memory to the tape controller.

2. Addressing a tape handler that is rewinding.

3. Addressing an unassigned tape handler.

4. Addressing a tape handler which is one of two units on a controller having the same address.

5. A detectable malfunction of a tape handler.

6. Giving a write command to a tape handler without the write permit ring on the tape reel.

7. Giving a read backward instruction when the tape is positioned on the leader.

When an ALERT HALT occurs, the normal procedure requires that all items listed on the preceding page be checked to determine and correct the alert condition. The majority of magnetic tape ALERT HALT conditions are due to human error, such as:

1. Failure to load tape properly
2. Failure to set handler addresses correctly
3. Failure to place handler in Remote control prior to intended program use
4. Insertion of incorrect controller plug

Restart and recovery procedures after alert condition correction is a function of the individual installation.

Alerts on Last Record

Certain types of magnetic tape errors do not halt the computer. These error conditions are visually indicated on the tape controller by the ALERT ON LAST RECORD light and the specific error light.

The programmer can test for these errors by using the magnetic test and branch instructions described under the heading, Magnetic Tape Interrogation Commands. Responsibility for corrective action lies with the programmer once he has determined the specific errors.

The following errors can be tested for by the programmer.

1. PARITY. In addition to a parity check on data transfers between memory and the tape controller, data written on or read from magnetic tape is checked for validity by means of lateral and horizontal parity checks. Figure 8-8 illustrates lateral and horizontal parity for the decimal mode.

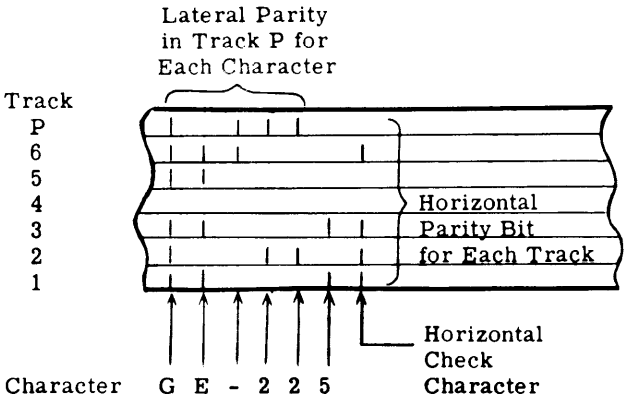


Figure 8-8. Lateral and Horizontal Magnetic Tape Parity

Lateral Parity. When data is read from tape, each character is parity checked for validity (even parity for decimal and odd for binary data). A character (or lateral) parity error turns on the LATERAL PARITY ALERT light on the tape controller. During writing, lateral parity bits are generated as required, placed on tape (track P) and checked. See Write Check, below.

Horizontal Parity. At the end of each record, an even check bit is written for each of the seven tracks. Thus, the final character of any record is the horizontal check character composed of the check bits for the seven tracks. Horizontal parity is checked when either reading or writing and any error is indicated on the tape controller by the HORIZONTAL PARITY ALERT light. See Write Check, below.

2. MOD 3 OR 4. A data word is written as three tape characters in the decimal and special binary modes and as four tape characters in the binary mode. When data is read from tape, a check is made for the correct multiple of 3 or 4, depending on the mode. A MOD error lights the MOD ALERT light on the tape controller. Thus, data written on tape in the binary mode and read in the decimal mode (or vice-versa) would give a MOD error.

3. WRITE CHECK. All information written on tape is checked for correct horizontal and lateral parity by a read head physically separate from the write head. An error turns on the LATERAL PARITY ALERT or the HORIZONTAL PARITY ALERT.

4. CONTROLLER N REGISTER MEMORY REQUEST ERROR. The input/output register (displayed as the N register on the controller display panel) in the tape controller can store only one word of information at a time. Note: this N register is a 21-bit register in the tape controller and not the 6-bit N register in the central processor.

If, during a tape read operation, a request for memory access is not granted, data in the controller N register is written over by the next word read from tape. If a request for a memory access is not granted during a write operation, a word from memory does not get to the controller in time to be written on tape so that data in the controller N register is written on tape twice. Either of these conditions turns on the MEMORY ALERT light on the controller display panel.

Note that the magnetic tape test-and-branch instructions BCS BIO (branch on input/output buffer error) and BCS BIC (branch on input/output buffer correct) test for two conditions of the tape controller input/output register. The first condition concerns the granting or not granting of memory requests; a memory request error is indicated by a MEMORY ALERT light on the controller display panel. The other condition concerns a parity error on a data word from memory; a parity error on a data word is indicated by the N REG ALERT light, as discussed, below.

5. PARITY ERROR IN THE CONTROLLER N REGISTER. When writing on tape, the parity of a data word from memory is checked. A parity error turns on the N REG ALERT light on the controller display panel (a parity error caused by instruction words 2 and 3 also turns on the N REG ALERT light, in addition to the ALERT HALT condition discussed above).

6. ALERT HALT. Any of the conditions previously discussed under ALERT HALT also turns on the ALERT ON LAST RECORD light.

Parity and other such errors are detected while reading or writing tape. These do not cause an alert halt, but do give an indication on the tape controller display panel that a specific error has occurred on the last record. These errors can be detected under program control by means of the magnetic tape test-and-branch instructions.

In addition to a parity check on data transfers between memory and the controller, instructions can be used to test for a horizontal or lateral parity error occurring on a data transfer between magnetic tape and the tape controller.

MAGNETIC TAPE INSTRUCTIONS

Each of the two basic GE-225 tape sub-systems, Figure 8-9, require three instruction words to perform an operation by a tape handler. The format of the instructions is essentially the same for either tape sub-system.

GE Tape Sub-System	Transfer Rate (KC)	Density (BPI)	Tape Speed (IPS)
TA 225	15	200	75
TB 225	15/41.6	200 555.5	75

Figure 8-9. GE Magnetic Tape Sub-Systems

Word No 1:

S-4	5-6	7-8	9	10-13	14	15	16-19
Gen	Index	I/O	Not Used	Plug	Not Used	Gen	Not Used

Gen - Word No. 1 is the general instruction SEL which is the same for all input/output commands. The SEL command is octal 2500P20.

Index - Address of index register when used, otherwise 00.

I/O - Contains the bits 00 for all input/output commands.

Plug - Contains code for the controller address to be used (0-6).

Word No 1 is the SEL command which clears any error conditions from a previous operation (except an ALERT HALT which requires manual intervention) and selects the specified tape controller.

Example of GAP Coding with Tape Controller on Plug 2:

Symbol	Opr	Operand	X
	SEL	2	

Word No 2:

S-4	5	19
T-C	M	

TC - Contains the code for the specific tape command to be performed (write, read, etc.).

M - Starting address in memory where information is to be stored or extracted.

The second instruction word specifies the desired tape operation and, in the case of read or write operations, indicates the memory address involved.

This word cannot be indexed. Also, GAP manipulates the GAP coding to form the word shown above. An example showing the GAP instruction line is given below.

P = Plug 2

M = Memory Location 1000

T = Tape Handler 4

GAP Coding:

Symbol	Opr	Operand	X
	SEL	2	
	WTD	1000	4

The information contained in line 2 specifies the tape handler that is to be used. However, the tape handler is not part of instruction word 2, as generated by GAP. Instead this is made a part of GAP word 3.

Tape instruction word 3 for the magnetic tape sub-systems, as generated by GAP, contains the following information.

Word No 3 15 kc Tape Sub-Systems:

S-4	5	19
T	N	

T - Tape handler to be used (0-7).

N - Maximum number of words read or written.

GAP Coding:

P = Plug 2

M = Memory Location 1000

T = Tape Handler 4

N = 200 Words

Symbol	Opr	Operand	X
	SEL	2	
	WTD	1000	4
		200	

The information of line 3 on the coding sheet and the tape handler number of line 2 are used by GAP to generate word 3.

In summation, the three basic instructions that the programmer must provide for each magnetic tape read or write operation will assume the following form:

Symbol	Opr	Operand	X
	SEL	P	
	CCC	M	T
		N	

The SEL P instruction clears previous error conditions (except ALERT HALT) and selects the desired magnetic tape controller P.

The CCC (line 2) represents the three-letter mnemonic code for specifying the desired operation.

The M (line 2, operand field) indicates the starting address for reading and writing operations. For non-read and non-write operations, such as backspace BKW and write end of file WEF, the operand field is blank.

The T (line 2, column 20) specifies the desired tape handler.

The N (line 3) specifies the number of words to be read or written.

Tape Movement Instructions

The GE-225 tape sub-systems can, during forward tape movement, either read or write information in the decimal, binary, or special binary modes. Intermixed reading and writing instructions should not be programmed on one tape handler.

FORWARD TAPE MOVEMENT:

The GE-225 tape sub-systems can, during forward tape movement, either read or write information in the decimal, binary, or special binary modes.

WTD	M	T	02MMMMM	Word Times: 2
(Blank)	N		TTNNNNN	

Functional Description: WRITE TAPE DECIMAL. The number of decimal words specified by N, starting from location M, are written by magnetic tape handler T.

WTB	M	T	03MMMMM	Word Times: 2
(Blank)	N		TTNNNNN	

Functional Description: WRITE TAPE BINARY. The number of binary words specified by N, starting from location M, are written by magnetic tape handler T.

WTS	M	T	23MMMMM	Word Times: 2
(Blank)	N		TTNNNNN	

Functional Description: WRITE TAPE SPECIAL BINARY MODE. The number of words specified by N, starting from location M, are written by magnetic tape handler T. Bits 2 through 19 of each word are written on tape exactly as they appear in memory (zone bits are not altered).

RTD	M	T	04MMMMM	Word Times: 2
(Blank)	N		TTNNNNN	

Functional Description: READ TAPE DECIMAL. A maximum of N decimal words is read by magnetic tape handler T and placed in memory, starting at location M.

RTB	M	T	05MMMMM	Word Times: 2
(Blank)	N		TTNNNNN	

Functional Description: READ TAPE BINARY. A maximum of N binary words is read by magnetic tape handler T and placed in memory, starting at location M.

RTS	M	T	25MMMMM	Word Times: 2
(Blank)	N		TTNNNNN	

Functional Description: READ TAPE SPECIAL BINARY MODE. A maximum of N words is read by magnetic tape handler T and placed in memory, starting at location M. A data word from tape is stored in bit positions 2 through 19 of a memory location just as it appears on tape (zone bits are unaltered).

WEF		T	0200000	Word Times: 2
			TT00000	

Functional Description: WRITE END OF FILE. The end-of-file character (0001111) and end-of-file gap are written on tape by magnetic tape handler T.

Comments: These instructions cannot be automatically modified. Also, they must always be preceded by a select (SEL) instruction.

Example 1: Write on magnetic tape, in the decimal mode, 100 BCD words from memory starting at location 1000. Use tape handler number 1 on the tape controller connected to controller selector plug 1.

GAP Coding:

Symbol	Opr	Operand
	SEL	1
	WTD	1000 1
		100

Resulting GAP Assembly:

Location in Octal	Contents in Octal	GAP Coding
02757	2500120	SEL 1
02760	0201750	WTD 1000 1
02761	0200144	100

Example 2: Read in the decimal mode from magnetic tape 100 BCD words into memory address 2000. Use tape handler 1, plug 1.

GAP Coding:

Symbol	Opr	Operand
	SEL	1
	RTD	2000 1
		100

Resulting GAP Assembly:

Location in Octal	Contents in Octal	GAP Coding
03000	2500120	SEL 1
03001	0403720	RTD 2000 1
03002	0200144	100

Example 3: Write an end-of-file record on tape using tape handler 4, plug 2.

GAP Coding:

Symbol	Opr	Operand
	SEL	2
	WEF	4

Resulting GAP Assembly:

```
02000    2500220    SEL 2
02001    0200000    WEF      4
02002    2100000
```

Note particularly that GAP has generated the third instruction word.

BACKWARD TAPE MOVEMENT:

The GE-225 tape sub-systems have instructions that result in a backward tape movement. Data can be read but not written during backward tape movements.

RBD	M	T	14MMMMM	Word Times: 2
(Blank)	N		TTNNNNN	

Functional Description: READ BACKWARD DECIMAL. Decimal information is read from tape moving backward. A maximum of N words is read into memory, the first word being placed in location M. The second word is placed in M minus 1 and so on until N words are read. The tape controller alters the zone bits of characters read so that they conform to GE-225 internal BCD characters.

RBB	M	T	15MMMMM	Word Times: 2
(Blank)			TTNNNNN	

Functional Description: READ BACKWARD BINARY. Binary information is read from tape moving backward. A maximum of N words is read into memory, the first word being placed in location M. The second word is placed in M minus 1 and so on until N words are read.

RBS	M	T	35MMMMM	Word Times: 2
(Blank)	N		TTNNNNN	

Functional Description: READ BACKWARD SPECIAL BINARY. Information is read from tape moving backwards. Bit positions 2 through 19 of each word read are placed in memory exactly as on tape (zone bits are not altered). A maximum of N words is read into memory, the first word being placed in M. The second word read is placed in M minus one and so forth until N words are read.

RWD		T	2000000 TT00000	Word Times: 2

Functional Description: REWIND. Rewind magnetic tape handler T to leader.

BKW		T	1600000 TT00000	Word Times: 2

Functional Description: BACKSPACE AND POSITION WRITE HEAD. The tape on magnetic tape handler T is back-spaced one record and the write head is positioned to write. This command must not be followed by a read command on the same tape handler until a write command has been executed.

Comments: If the last tape operation performed on the specified handler was a write operation, read tape backward commands cannot be performed on this handler.

Since a BKW command is used to move a specified tape handler backward one record and position it for rewriting a record, this command cannot be followed by a read instruction. The programmer must exercise caution in using the BKW command because, if the record being backspaced is less than 24 words in length, the record may be ignored and the tape will be backspaced over the preceding record also, regardless of its length. A subsequent write operation would then destroy this record. In addition, a write operation following a BKW command elongates the IRG by approximately 0.20 inch.

Backspace and Reread One Record. Backspacing and positioning for re-reading a record can be accomplished by using a read backward command with N set equal to zero.

Example 1: Backspace one record and position the read head.

GAP Coding:

Symbol	Opr	Operand	X
	SEL	2	
	RBD	1000	3
		0	

Tape Movement Residue Word

After a magnetic tape read operation on any GE-225 tape sub-system, a residue word is generated that contains information as to the number of words in the record just read. The location of the residue word and its contents are as described below.

Reading Tape Forward. After forward reading N words from magnetic tape in the decimal, binary or special binary modes starting at location M, memory location M + N (the residue word) contains zeros if exactly N

words were read from a tape record containing N words. For example, if M is 500 and N is 50 and the record read contains 50 words, the information is stored in locations 500 through 549 and location 550 contains all zeros.

If the number of words contained in the record is less than N, then only that number of words is stored in memory. The two's complement of the difference (N minus record length) is stored in memory cell M + N with a 1-bit in the sign position. For example, if M is 0500 and N is 50, but the record read contains only 30 words, the information is stored in memory cells 0500 through 0529 and the two's complement of the difference, 20 (N minus record length), is stored in memory cell 0550 (M + N) with a 1-bit in the sign position.

If the number of words in the record is greater than N, only N words are stored in memory, and the number of excess words (record length minus N) is stored in memory cell M + N, with a 0-bit in the sign position. For example, if M is 0500 and N is 50 but the record length is 75 words, information is stored in memory locations 0500 through 0549 and the increment 25 (record length minus N) is stored in memory cell 0550 (M + N) with a 0-bit in the sign position. The tape controller remains busy for the time required to read the record; i.e., the IRG is reached.

Example: Two hundred words of a decimal tape record are read into memory starting at location 2000. Tape handler 3 on tape controller 1 is used.

GAP Coding:

```
Symbol    Opr    Operand
                 S E L 1
                 R T D 2 0 0 0        3
                       2 0 0
```

The residue word is in memory location (M + N) which equals 2000 + 0200, or 2200. Three conditions can exist as shown by the chart below.

Condition	Record Length (R)	Octal Contents of 2200 (M + N)
R = N	R = 200	0000000
R > N	R = 250	0000062
R < N	R = 100	3777634

The programmer must allow for this residue word when reading records into memory. Otherwise, he runs the risk of destroying the contents of a memory location vital to successful operation of his program.

Reading Tape Backward. In reading tape backward, the first word read is placed in M, the second word is placed in M minus 1, the third in M minus 2, etc., until N words are read or, if the record read is less than N words, until the entire record is read.

After backward reading of N words from magnetic tape in the decimal, binary, or special binary modes starting at location N, memory location M - N contains zeros if exactly N words were read from a record containing N words. For example, if M is 0500 and N is 50 and the record read contains 50 words, the information read is stored in memory locations 0451 through 0500 and location 0450 (M - N) contains all zeros.

If the number of words contained in the record is less than N, then only that number of words is stored in memory and the two's complement of the difference (N minus record length) is stored in memory cell M - N with a 1-bit in the sign position. For example, if M is 0500 and N is 50 but the record length is only 25 words, only the record is stored in memory in locations 0476 through 0500 and the two's complement of 25 (N minus record length) is stored in location 0450 (M - N), with a 1-bit in the sign position. If the number of words in the record is greater than N, only N words are stored in memory and the number of excess words (record length minus N) is stored in memory cell M - N with a 0-bit in the sign position. For example, if M is 0500 and N is 50 but the record length is 75 words, only N words are stored in memory locations 0451 through 0500 and the difference (record length minus N) is stored in memory cell 0450 (M - N) with a 0-bit in the sign position. The tape controller remains busy for the time required to read the entire record.

Example: Two hundred words of a decimal tape record are read into memory starting at 2000 by a read backward instruction. Tape handler 3 on tape controller 1 is used.

GAP Coding:

```
                 S E L 1
                 R B D 2 0 0 0        3
                       2 0 0
```

The residue word is in memory location (M - N) which is 2000 - 200, or 1800. Three conditions can be indicated by the contents of the residue word as shown.

Condition	Record Length (R)	Octal Contents of 1800 (M - N)
R = N	R = 200	0000000
R > N	R = 250	0000062
R < N	R = 100	3777634

The residue word resulting from a tape read operation provides the programmer with a means of determining the actual number of words read into memory and thus also provides a way of determining the actual record size, if desired.

Magnetic Tape Interrogation Commands

Before attempting to perform a magnetic tape operation, it is essential that the programmer interrogate the magnetic tape controller as to its status: whether it is busy or whether a particular condition occurred during its previous operation. Test-and-branch instructions can be used by the programmer to indicate the controller status.

Test-and-Branch Instructions. Normally, the SEL instruction selects the particular controller desired, while coding lines two and three of the instruction group then tell what is to be done by the peripheral unit connected to the controller. However, before these instructions can be programmed, it is a good programming convention always to precede the SEL with a test for ready condition. These test-and-branch instructions are performed by executing a BCS command specifying the particular test condition desired.

The following instructions test to determine whether a specified magnetic tape controller condition is true or false. If the condition tested is true, the computer executes the next sequential instruction. If false, the computer executes the second sequential instruction. These instructions apply to all GE-225 magnetic tape sub-systems.

BCS BTR P 2514P20 Word Times: 2

Functional Description: BRANCH ON TAPE CONTROLLER READY. The tape controller P is tested for the ready status.

BCS BTN P 2516P20 Word Times: 2

Functional Description: BRANCH ON TAPE CONTROLLER NOT READY. The tape controller P is tested for the not ready status.

BCS BEF P 2514P21 Word Times: 2

Functional Description: BRANCH ON END OF FILE. The tape controller P is tested for end-of-file indicator ON.

BCS BNF P 2516P21 Word Times: 2

Functional Description: BRANCH ON NO END OF FILE. The controller P is tested for end-of-file indicator OFF.

BCS BET P 2514P22 Word Times: 2

Functional Description: BRANCH ON END OF TAPE. The controller P is tested for end-of-tape indicator ON.

BCS BNT P 2516P22 Word Times: 2

Functional Description: BRANCH ON NO END OF TAPE. The controller P is tested for end-of-tape indicator OFF.

BCS BPE P 2514P24 Word Times: 2

Functional Description: BRANCH ON TAPE PARITY ERROR. The controller P is tested for parity error indicator ON.

BCS BPC P 2516P24 Word Times: 2

Functional Description: BRANCH ON TAPE PARITY CORRECT. The controller P is tested for tape parity error OFF.

BCS BIO P 2514P25 Word Times: 2

Functional Description: BRANCH ON INPUT/OUTPUT BUFFER ERROR. The controller P is tested for input/output buffer error indicator ON.

BCS BIC P 2516P25 Word Times: 2

Functional Description: BRANCH ON INPUT/OUTPUT BUFFER CORRECT. The controller P is tested for input/output buffer error indicator OFF.

BCS BME P 2514P26 Word Times: 2

Functional Description: BRANCH ON MOD 3 OR 4 ERROR. The controller P is tested for modulo 3 or 4 error indicator ON.

BCS	BNM	P	2516P26	Word Times: 2

Functional Description: BRANCH ON NO MOD 3 OR 4 ERROR. The controller P is tested for modulo 3 or 4 error indicator OFF.

BCS	BER	P	2514P27	Word Times: 2

Functional Description: BRANCH ON ERROR. The controller P is tested for error indicator ON.

BCS	BNE	P	2516P27	Word Times: 2

Functional Description: BRANCH ON NO ERROR. The controller P is tested for error indicator OFF.

In tape sub-systems, the error indicator is turned on by:

1. A tape parity error
2. I/O buffer error
3. MOD 3 or 4 error

These conditions would be detected by a BCS BER instruction, although the specific error or errors would not be identified. Once an error is detected, it can be specifically identified by testing for individual error conditions. It should be noted that

End of File

End of Tape

Tape Rewinding

conditions would not be detected by the BCS BER command, but must be tested individually.

BCS	BRW	P	2514P23	Word Times: 2

Functional Description: BRANCH ON TAPE REWINDING. The controller P is tested for tape rewinding condition only.

BCS	BNR	P	2516P23	Word Times: 2

Functional Description: BRANCH ON NO TAPE REWINDING. The controller P is tested for tape not rewinding only.

The programmer must be aware that a rewind instruction for any tape handler on a controller puts that controller in the not-ready status for 250 microseconds, after which the controller returns to the ready status even though the tape handler is still rewinding. A read or write instruction can then be given to any tape unit that is not rewinding. The controller indicates when one or more tape units is rewinding for the entire rewind operation. Addressing a rewinding tape unit to read or write causes an ALERT HALT condition.

Thus, a rewind interrogation of a tape controller will indicate that a tape is rewinding without specifying which particular tape handler.

Example 1: Test tape controller 1 for ready status.

GAP Coding:

Symbol	Opr	Operand	X
	BCS	BTN	1
	BRU	* - 1	
	SEL	1	

Comments: The BCS command above interrogates the controller on plug 1. If the controller is not ready, the computer executes the next sequential command which is a branch back to the BCS command. Thus, a delay is effected until the controller becomes ready at which time the SEL command is executed and a tape operation can be performed.

When a magnetic tape controller is tested and found to be ready, or not busy, any tape handler connected to it can be addressed by a read, write, read backwards, or rewind instruction, unless the tape handler already is rewinding. Tape handlers that are rewinding should not be addressed until the completion of the rewind operation.

A read, write, or read backward instruction puts the controller in the not-ready, or busy, status until the completion of the operation. A rewind instruction puts the controller in a not-ready status for 250 microseconds, after which the controller returns to the ready status. A read or write instruction can then be given to any tape handler that is not rewinding. The controller indicates that one or more tape handlers are rewinding for the entire rewind operation. Addressing a rewinding tape handler to read or write causes an ALERT HALT condition.

Example 2: Test tape controller 1 for any error condition and branch to an error subroutine to determine the type of error, if any exists from the previous tape operation.

GAP Coding:

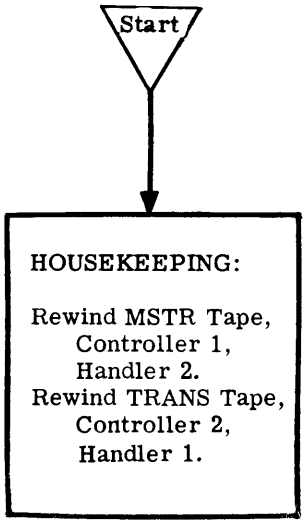
Symbol	Opr	Operand	X
	BCS	BER	1
	BRU	ERROR	

PROGRAMMING MAGNETIC TAPE OPERATIONS

General Considerations

Before processing data from magnetic tape, it is good programming practice to rewind the tapes that are to be used. This safeguards against the possibility of a tape not being at load point when processing begins. The programmer also should rewind a tape upon completion of processing and provide appropriate messages to the operator specifying what is to be done with the rewound tape. It is also necessary to test the magnetic tape controller or controllers for ready status before attempting any tape operation.

Example: Flow Chart



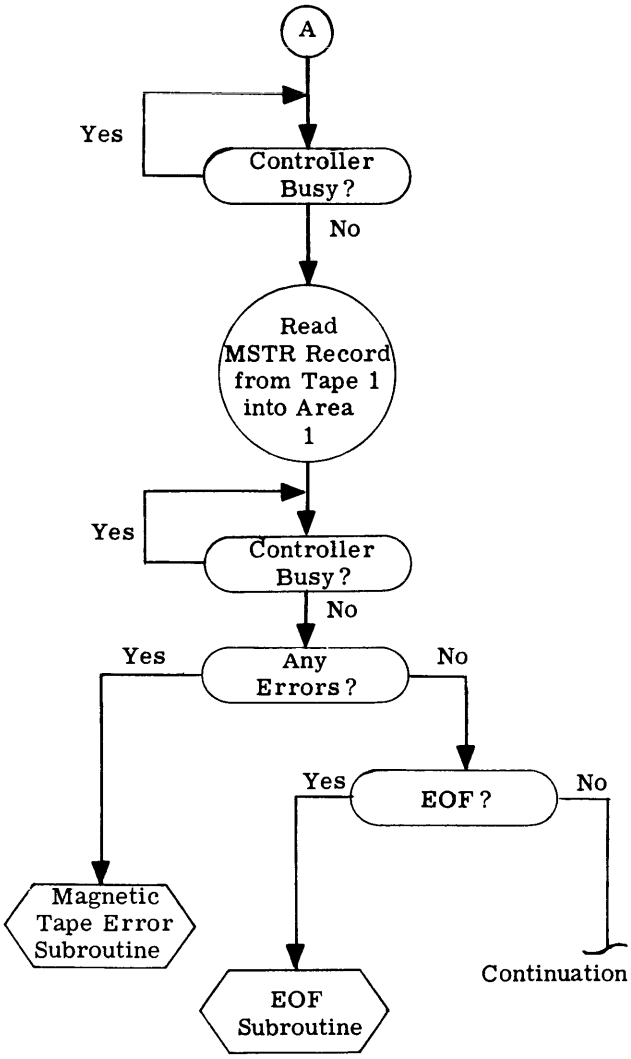
GAP Coding:

Symbol	Opr	Operand	X	REMARKS
START	BCS	BTN	1	CHECK FOR CONTROLLER 1 READY
	BRU	*-1		DELAY FOR CONTROLLER READY
	SEL	1		SELECT CONTROLLER 1
	RWD		2	REWIND TAPE HANDLER 2
	BCS	BTN	2	CHECK CONTROLLER 2 READY
	BRU	*-1		DELAY FOR CONTROLLER READY
	SEL	2		SELECT CONTROLLER 2
	RWD		1	REWIND TAPE HANDLER 1

After performing a tape operation on a controller and before using the controller for another operation, tests must be made to determine if the operation was successful or if any condition such as end-of-file, end-of-tape, etc., occurred. It is essential that these tests be made before another SEL command is executed since this instruction clears the controller.

Example: Assume that a controller on Plug 1 and tape handlers 2 and 3 are being used.

Flow Chart



GAP Coding:

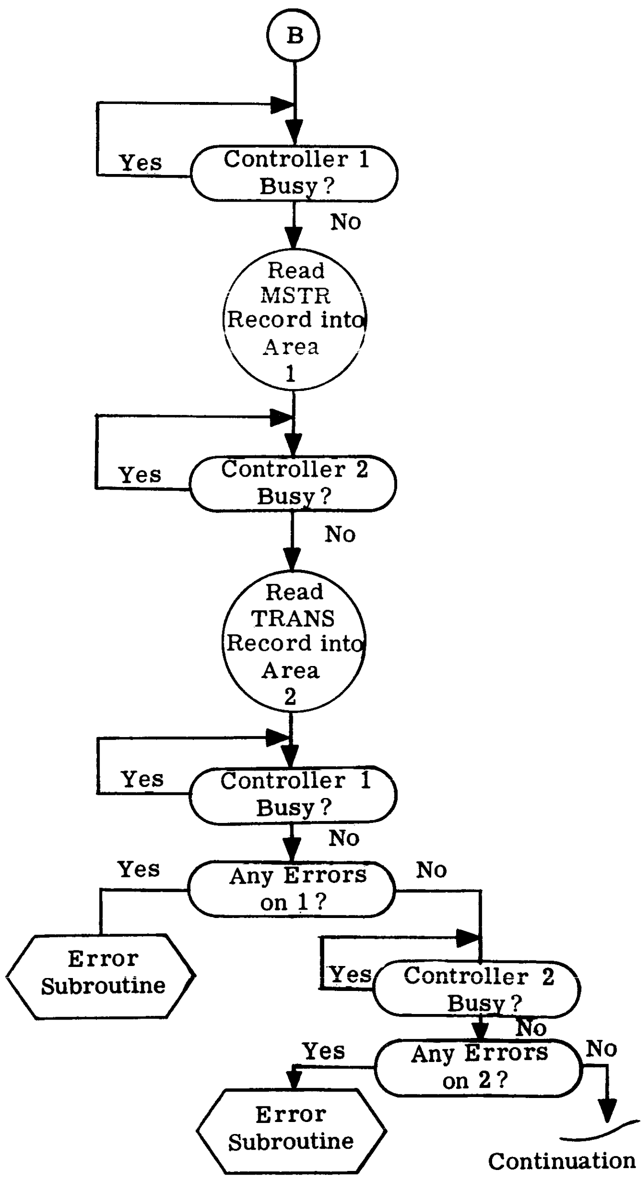
Symbol	Opr	Operand	X	REMARKS	Sequence
A	BCS	BTN	1	CHECK FOR CONTROLLER BUSY	200
	BRU	*-1			210
	SEL	1		SELECT CONTROLLER #1	220
	RTD	AREA#1		READ MSTR RECORD	230
		250		NO. WORDS IN RECORD	240
	BCS	BTN	1	DELAY UNTIL READ COMPLETED	250
	BRU	*-1			260
	BCS	BER	1	CHECK FOR READ ERROR	270
	BRU	ERROR		GO TO ERROR SUBROUTINE	280
	BCS	BEF	1	CHECK FOR END-OF-FILE	290
	BRU	EOF		GO TO END-OF-FILE SUBROUTINE	300

Any error occurring during the read operation in the example would cause a branch to a sub-routine to determine the type of error and the corrective action necessary.

Reading and writing operations can be performed simultaneously only when two or more tape controllers are available. Again, each operation must be tested by the programmer for any errors or other conditions.

Example 1: Assume MSTR tape on controller 1, tape handler 1; TRANS tape on controller 2, tape handler 1.

Flow Chart



GAP Coding:

Symbol	Opr	Operand	X	Remarks	Sequence
B	BCS	BTN	1	CHECK CONTROLLER #1 BUSY	300
	BRU	*-1			310
	SEL	1		SELECT CONTROLLER #1	320
	RTD	AREA#1		READ MSTR RECORD	330
		200		NO.WORDS	340
	BCS	BTN	2	CHECK CONTROLLER #2 BUSY	350
	BRU	*-1			360
	SEL	2		SELECT CONTROLLER #2	370
	RTD	AREA#2		READ TRANS RECORD	380
		200		NO. WORDS	390
	BCS	BTN	1	CHECK CONTROLLER #1 BUSY	400
	BRU	*-1			410
	BCS	BER	1	CHECK FOR ANY ERROR	420
	BRU	ERROR		GO TO ERROR SUBROUTINE	430
	BCS	BTN	2	CHECK CONTROLLER #2 BUSY	440
	BRU	*-1			450
	BCS	BER	2	CHECK FOR ANY ERROR	460
	BRU	ERROR		GO TO ERROR SUBROUTINE	470
	PROCESS				

Example 2: At the conclusion of processing an old master file on controller 1, handler 1 and creating a new master file on controller 2, handler 1, write an end-of-file record on the new master and rewind both tapes.

GAP Coding:

Symbol	Opr	Operand	X	Remarks	Sequence
	BCS	BTN	1	CHECK CONTROLLER #1 BUSY	520
	BRU	*-1			530
	SEL	1		SELECT CONTROLLER #1	540
	RWD		1	REWIND TU #1	550
	BCS	BTN	2	CHECK CONTROLLER #2 BUSY	560
	BRU	*-1			570
	SEL	2		SELECT CONTROLLER #2	580
	WEF		1	WRITE EOF ON TU #1	590
	BCS	BTN	2	DELAY FOR EOF COMPLETION	600
	BRU	*-1			610
	SEL	2			620
	RWD		1	REWIND TU #1	630

Multi-Reel Flip-Flops

The use of magnetic tapes implies files of great size, and in many applications a file consists of more than one reel of tape. Thus, whenever the end of a tape or the end of a file is reached, the tape reel must be rewound, removed, and replaced with the next tape to be processed. To maintain the continuity of the running program, it is worthwhile to program an immediate switch, or alternation, from the just-completed tape to a succeeding tape (already mounted on another tape handler). This is accomplished by a technique called flip-flopping of multireel tape files.

This technique permits the mechanical rewind and manual removal of the completed tape to proceed entirely independent of further processing. If sufficient tape handlers are not available to permit such convenient switching of all files, then the most extensive files should be given priority in the allotment of tape handlers.

As explained earlier, the actual number of the tape handler (T) to perform a specified operation appears in bit position 0 through 4 of the third coding line. The tape handler is selected according to the following formats:

Handler Number	Bits 0-4	Octal Code
0	00001	01
1	00010	02
2	00100	04
3	01000	10
4	10001	21
5	10010	22
6	10100	24
7	11000	30

These configurations make flip-flopping of multireel tape files relatively easy. It is only necessary to load the third instruction word of a magnetic tape instruction into the A register, change the sign and store the word back into memory.

For example, assume a multireel file is placed on tape handler 1 and the second reel on tape handler 5. When the end-of-file indication is reached on the tape on tape handler 1, the third word of the instruction is loaded into the A register, the sign changed, and then stored in memory as the third word of the tape instruction. The same technique can be used for the following tape handler combinations:

0 ⟷ 4

1 ⟷ 5

2 ⟷ 6

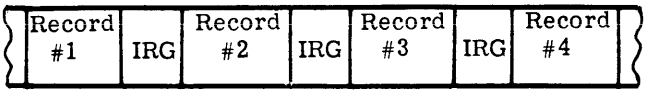
3 ⟷ 7

Thus, if a spare tape handler is available during a given program run, the flip-flopping of handler numbers of all tape instructions related to a given multireel file, after an end-of-file condition, allows the program to continue uninterrupted while the 'used up' reel is rewinding.

Processing Magnetic Tape Records

Since the GE-225 is a buffered computer, it is capable of performing simultaneous peripheral operations. This is of particular importance in processing magnetic tape data in that reading and writing can be done simultaneously with two tape controllers. However, the time consumed processing tape data is directly affected by the arrangement of the data records on the tape. For example:

Tape A - 24 Word BCD Records, 15 kc Tapes



The tape time involved in reading these records is calculated by the following timing formula.

$$\frac{3RW}{K} + (R-1)\,S \qquad \text{Where}$$

R = No. Records
W = Words per Record
K = Tape transfer rate in kc
S = Start-stop time in IRG

This equation applies only for tapes with BCD data. For binary tapes the expression (3RW) should be 4 RW.

For tape A, the tape time for reading the data is:

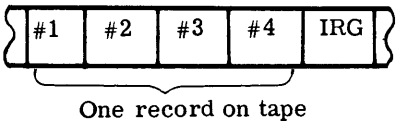
$$\frac{3 \times 4 \times 24}{15000} + 3\,(.012) = .0552 \text{ Sec.}$$

This time is based on a start-stop time in the IRG of 12 ms for both the high-density and the low density magnetic tape sub-systems.

If the data on tape A were written on tape B as a blocked record as shown below the time to read the data is:

$$\frac{3 \times 1 \times 96}{15000} + .012 = .0312 \text{ Sec.}$$

Tape B



The time saved in reading the data from Tape B as compared to Tape A is due to elimination of three inter-record-gaps due to the blocking of four records into one. Blocking of records can result in considerable reduction in tape processing time.

Record Documentation

Adequate record documentation on the part of the systems designer or the programmer helps reduce programming costs, machine time needed for debugging, and file maintenance costs. Figure 8-10 illustrates a record layout sheet for systems designer or programmer use. Documentation of this type provides a clear description as to the contents of a record, the size of each individual data field, and the size of the record itself.

RUN: #2 Inventory Control GENERAL ELECTRIC Computer Department Phoenix, Arizona DATE: 1/30/63

FILE: INV MSTR PROGRAMMER: GE Coder

RECORD TYPE: BCD **GE 225 MAGNETIC TAPE RECORD LAYOUT SHEET** PAGE: 1 OF: 20

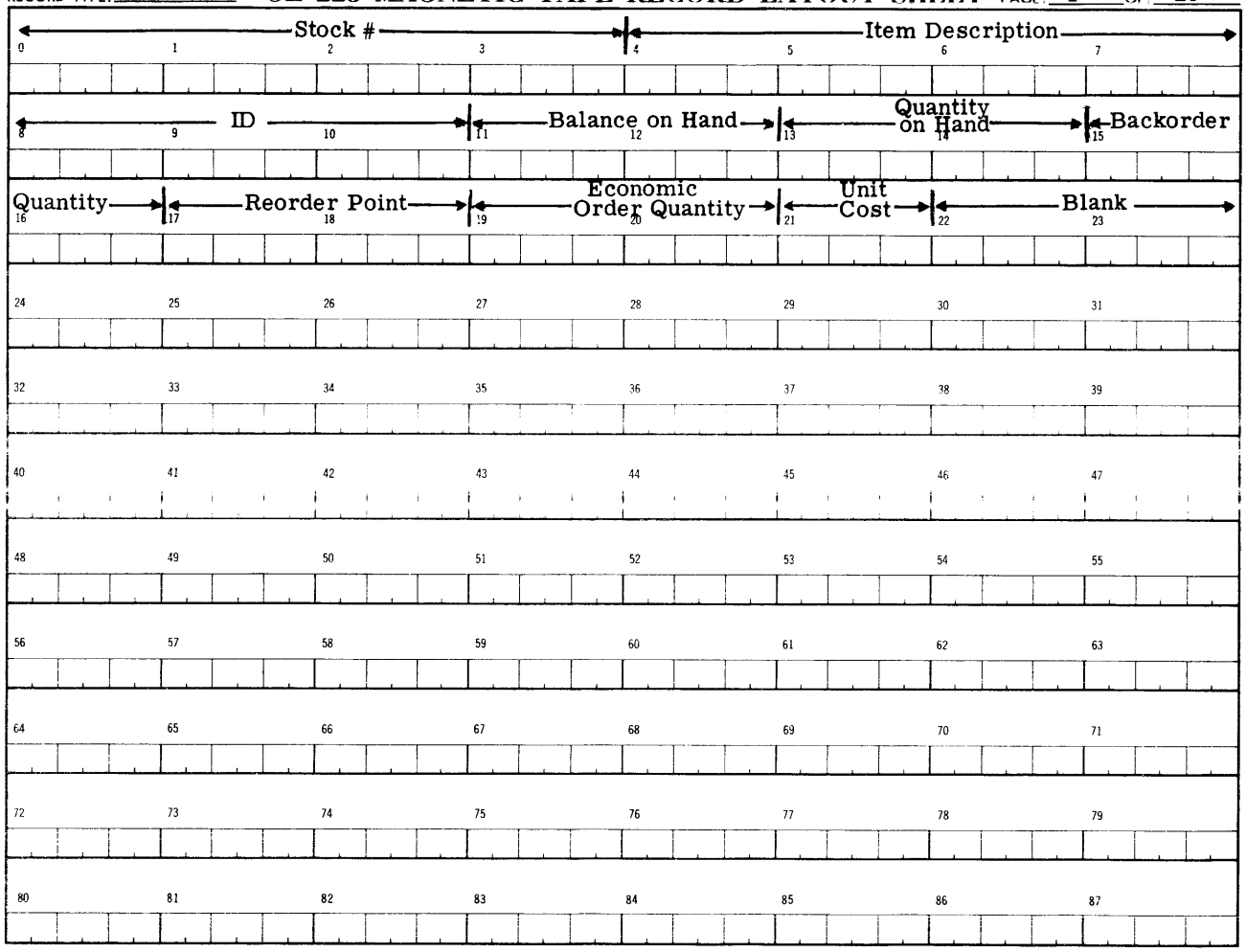


Figure 8-10. Sample Magnetic Tape Record Layout Sheet

SECTION IX

HIGH SPEED PRINTER OPERATIONS

Two high-speed printers are available for use with the GE-225 system. One printer (Model 4WP225A) is for on-line operations only and accesses data from core memory in the central processor. The other printer (Model 4WPA690) operates both on-line and off-line. During off-line operation, a separate core memory within a special magnetic tape controller is used. Both printers operate similarly when on-line, except that Model 4WPA690 has more powerful field-spreading ability. With the exception of some indicator lights, the physical characteristics of the two high-speed printer models are basically the same with respect to printing speed, slewing speed, printable characters and paper specifications. The off-line operation of Model 4WPA690 is discussed later in the section.

Paper Forms. Minimum paper width for a printed document is 3-1/2 inches; maximum width is 16-1/2 inches. Maximum form length is 22 inches. During any printing operation, up to 5 copies can be made, depending upon the weight paper used.

ON-LINE PRINTER OPERATIONS

The high-speed on-line printer is an output device operating through the controller selector. It is capable of single-line printing at a rate of 900 lines per minute for either numeric or alphanumeric data. By means of printer controllers operating through the controller selector, high-speed printer operations can occur simultaneously with the processing of other information in the central processor.

A print line can contain up to 120 alphanumeric characters which may occupy more than 40 consecutive BCD words in memory. A lesser number of printline characters require fewer memory locations.

Printing is executed by a revolving drum (120 print wheels locked together) and hammer-firing mechanism which can print up to 50 different characters. These include 26 alphabetic, 10 numeric, and 14 special characters as shown in Figure 9-1. All printing is of the open Gothic type font at 10 horizontal characters and 6 lines per inch.

HIGH SPEED PRINTER SYMBOLS	BCD MEMORY (OCTAL)	HIGH SPEED PRINTER SYMBOLS	BCD MEMORY (OCTAL)
A	21	0	00
B	22	1	01
C	23	2	02
D	24	3	03
E	25	4	04
F	26	5	05
G	27	6	06
H	30	7	07
I	31	8	10
J	41	9	11
K	42	+	20
L	43	-	40
M	44	blank	60
N	45	/	61
O	46	#	13
P	47	@	14
Q	50		15
R	51	=	16
S	62	.	33
T	63	$	53
U	64	*	54
V	65	,	73
W	66	%	74
X	67	[	75
Y	70	]	76
Z	71	End-of-Line	37

Figure 9-1. High-Speed Printer Character Set

The high-speed printer provides horizontal and vertical format control and, through programmed format instructions, the printer controller automatically edits each horizontal print line. After each line is printed, the paper is advanced one line under program control. However, if more than one line is to be spaced, or slewed, this can be programmed separately or performed automatically as part of the print instruction. Vertical slewing can be accomplished by either the countdown method or by slewing to a particular channel of the printer punched tape format loop. Slewing speed is 25 inches per second.

Printer Controllers. All printing operations are conducted by printer controllers. Each controller communicates with the central processor core memory through the controller selector. A GE-225 system can have up to eight printer controllers connected to the controller selector. However, unlike the magnetic tape controllers, only one high-speed printer can be connected to any one printer controller, Figure 9-2. The controller allows printing operations to occur simultaneously with central processor computations.

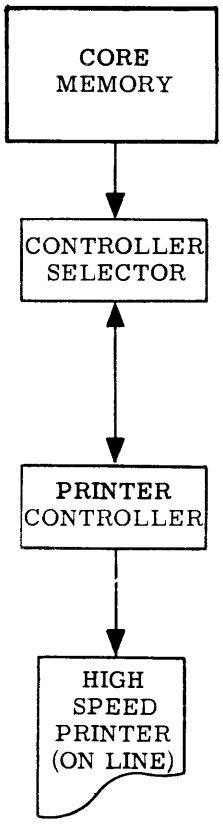


Figure 9-2. High Speed Printer, Controller, and Controller Selector

The controller serves as the link between the core memory of the central processor and the high-speed printer. It monitors the flow of information between memory and the printer, receives and interprets printer instructions from the central processor core memory, and accumulates and stores printable characters in the actual operation of the high-speed printer mechanism.

An operator panel on the printer controller, Figure 9-3, provides for manual operation of the printer. By using this panel, an operator can slew paper or print out a central processor memory dump in octal form.

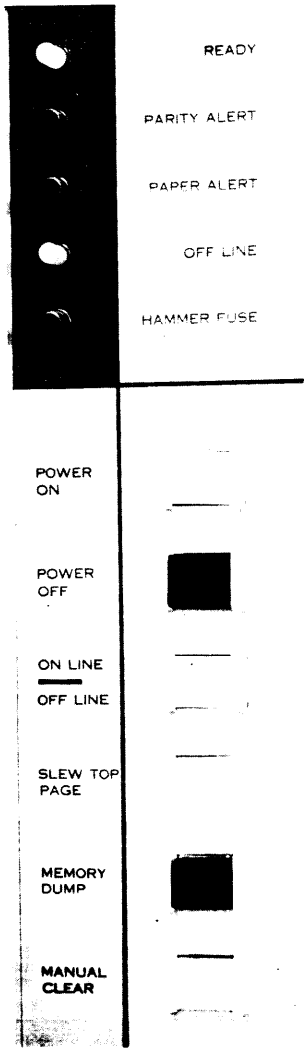


Figure 9-3. Operator Control Panel - High Speed Printer Controller

Priority. When operating on-line with other GE-225 peripheral units, a high-speed printer is granted access to memory according to its priority level in the controller selector priority interrupt system. This level is pre-established within a GE-225 system by plug-in connectors to the controller selector. Through this priority interrupt system of time sharing, each printer controls itself (through the respective printer controller), executes its own instructions and thus permits printing operations to occur simultaneously with central processor computations. Normally the printer controller occupies plug 6 (one of the lower priority plugs) because the printer can afford to wait for priority without a loss of information.

Controller Data Flow

During all on-line high-speed print operations, data is sent to the printer controller from the central processor M register. This register is used for checking parity of each 20-bit word (three BCD characters) as it leaves memory.

Parity is also checked in the printer controller before the word is stored in the controller. A controller has two 20-bit (plus a parity bit) buffers; one buffer holds format line information and the other holds printable data as it comes from memory. As shown in Figure 9-4, these are the format buffer and the data buffer.

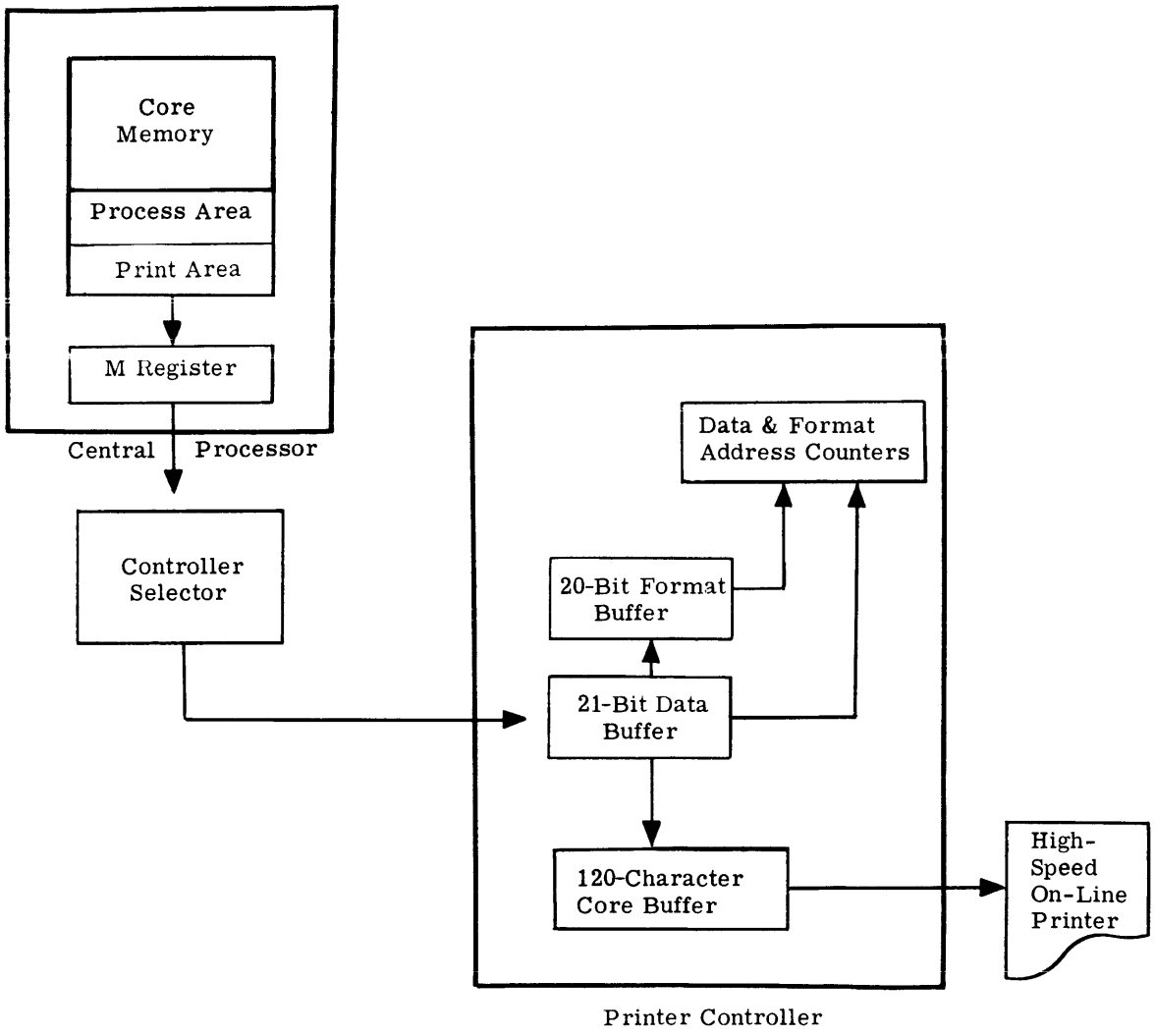


Figure 9-4. Printer Controller Buffering Diagram

The data buffer actually holds 21 bits because data and format words entering it are checked for parity before being transferred to the printer controller 120-character core buffer.

By using these buffers and the controller selector priority interrupt system, it is possible to time share access to memory, fill the printer buffers, and print, at the same time allowing central processor computations. This simultaneous operation of processing and printing may require alternate input and output areas of memory.

Checking Features. The accuracy of all information transferred between memory and the printer controller is checked in the following ways: a parity check for each word received from memory, end of paper detection, and inoperable hammer drive detection. Input and buffer errors also are checked by the printer controller check circuits and indicated on the controller display panels. Program instructions can be used to check certain of these conditions. These are described under Printer Test-and-Branch Instructions.

Printer Instruction Format

Printer instructions always require three lines of coding before print operations can begin, although at times, only two instruction words are coded by the programmer, such as in slewing paper or printing without automatic format control. In such cases, the General Assembly Program generates the third line of coding.

Figure 9-5 illustrates the bit configurations for each of the three required printer instructions.

WORD 1 (SEL)

0 1 2 3 4	5 6	7 8	9	10 11 12 13	14	15	16 17 18 19
1 0 1 0 1	X X	0 0	0	P P P P	0	1	0 0 0 0
General	Index	I/O	N/U	Controller Address	Not Used	Gen	Not Used

Bit positions 7 and 8 are zeros for all Input/Output (I/O) instructions

WORD 2

←octal group→		←octal group→ 32				
0	1	2	3	4	5	6-19
1=Print & Slew 0=Slew only	1=Format 0=No Format	VFU 8	VFU 7	VFU 6	1=Numeric 0=Alpha-numeric	Format Address

WORD 3

←octal group→		←octal group→			
16	8	4	2	1	
0	1	2	3	4	5 - 19
VFU 5	VFU 4	VFU 3	VFU 2	VFU 1	Data Address

Figure 9-5. Printer Instructions

WORD ONE. The SEL instruction informs a printer controller that it is to be brought 'on-line' to perform a print operation. Word one prepares the controller to receive instruction words two and three. Bits 10 through 13 of the word indicate the address (plug) of the desired printer controller.

WORD TWO. This word determines the operation to be performed. However, the information contained is not complete enough to begin the operation indicated. Note: In Figure 9-5, VFU means vertical format unit and is referred to as VFU in the descriptions of slewing by countdown and slewing to a channel.

Bit Position 0

If this bit position contains a 1-bit, it indicates a write print line (WPL) instruction in which slew operations are possible. The slew operation is defined by bits 2 through 4 of this word and bits 0 through 4 of instruction word three (see Figure 9-5). If bit position 0 contains a 0-bit, a slew only operation is indicated, with the slew operation defined by the bits mentioned.

Bit Position 1

If this bit position contains a 1-bit, the print operation is to be under automatic horizontal format control. The location in memory in which the format information is stored is indicated by bits 6 through 19 within this word. If bit position one is a 0-bit, printing is not done under automatic format control. The address of format data, if it were included in this word, would be disregarded.

Bit Positions 2 and 3

If these bits are both 1-bits, slewing is to be a definite number of lines (countdown method) with that number indicated by the slew bits that follow (bit position 4 of word two and 0 through 4 of word three). If both bits are not on, slew is to a hole in the channel indicated by a bit in one of the VFU positions.

Bit Position 4

This is the most-significant bit in a six-bit binary number, which is used to indicate the number of lines that are to be slewed. This bit has a binary value of 32. Thus, if a one were placed in this bit position, paper would be slewed 32 lines. The five least-significant bits are contained in instruction word three.

Bit Positions 6 through 19

These bits contain the fourteen least-significant address bits of the location in memory in which format data is contained. The most-significant bit of the memory address is the same as the most-significant address bit of the location in memory in which the information to be printed is stored (bit 5, word 3). The programmer must insure that the information to be printed and the format data are stored in the same half of memory.

WORD THREE. Instruction word three contains further information pertaining to the operation that is to be performed. This information, in conjunction with the second instruction word, is sufficient for the indicated operation to begin.

Bit Positions 0 through 4

Bit positions 0 through 4 contain the five least-significant bits of the six-bit binary number indicating the number of lines to be slewed. The binary values for these bits are shown in Figure 9-5.

Bit Positions 5 through 19

These bits indicate the starting address in memory from which information is to be extracted for printing. The information contained in any such memory location is defined as a data word.

High Speed Printer Instructions

WPL	Y	N	2000000 01YYYYY	Word Times: 2

Functional Description: WRITE PRINT LINE. One line of BCD information, 1 to 120 characters long, is printed, starting at memory location Y. The N indicates that information is numeric BCD only (if N is blank, information is alphanumeric). This instruction cannot be modified and must be preceded by an SEL P instruction which, when it enters the I register, sends the two words of the WPL instruction to the selected controller P automatically. Since only one word of the WPL instruction is shown by the programmer in his coding, the General Assembly Program automatically generates the third instruction word necessary.

Example: Print one line of BCD information from memory location PRT. Assume that data is in memory as required for printing. Printer is on controller selector plug 6. Symbolic location PRT is memory cell 1000 (1750 octal).

GAP Coding:

Symbol	Opr	Operand	X
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	SEL	6	
	WPL	PRT	

GAP Assembly:

Octal Memory	Contents	Coding
01115	2500620	SEL 6
01116	2600000	WPL PRT
01117	0101750	

Comments: Print line is terminated by a minus sign placed in the sign bit of the last word to be printed.

WFL Y X N	30YYYYY	Word Times: 2
(WPL)	01XXXXX	

Functional Description: WRITE FORMAT LINE. One line of BCD information is printed under format control. Y is the starting location in memory of the format control words. This instruction should always be followed by a WPL instruction to specify the location (X) of the first word of information to be printed and whether the information is alphanumeric or numeric (N). If N is blank, the information is alphanumeric.

A SEL command must precede this instruction. No modification is permissible.

Example: Print one line of BCD data under format control.

GAP Coding:

Symbol	Opr	Operand	X
	SEL	6	
	WFL	FORMAT	
	WPL	DATA	

GAP Assembly:

Octal Memory	Contents	Coding
02120	2500620	SEL 6
02121	3602304	WFL FORMAT
02122	0102336	WPL DATA

SLW N	0600000	Word Times: 2
	NN00000	

Functional Description: SLEW PAPER N LINES. The printer paper is spaced N (0-63) number of lines. If the number of lines is greater than 31, the high-order bits will be in instruction word 1, the low order bits in instruction word 2.

This instruction must be preceded by an SEL P instruction. Since only one word of the SLW instruction is shown by the programmer in his coding, the General Assembly Program automatically generates the second word during the assembly process.

This instruction cannot be modified.

Example: Slew the paper in the HSP 4 lines.

GAP Coding:

Symbol	Opr	Operand	X
	SEL	6	
	SLW	4	

GAP Assembly:

Octal Memory	Contents	Coding
02144	2500620	SEL 6
02145	0600000	SLW 4
02146	0400000	

SLT K	0X00000	Word Times: 2
	XX00000	

Functional Description: SLEW PAPER TO TAPE PUNCH. The printer paper is spaced until a hole is detected in the vertical format tape. K is the specified tape channel and X, the instruction octal code, varies with the channel specified.

This instruction must be preceded by an SEL P instruction. Since only one word of the SLT instruction is written by the programmer in his coding, the General Assembly Program automatically generates the second word during the assembly process

This instruction cannot be modified.

Example: Slew the paper in the high-speed printer to the punch in vertical format tape channel 6.

GAP Coding:

Symbol	Opr	Operand	X
	SEL	6	
	SLT	6	

GAP Assembly:

Octal Contents	Coding
2500620	SEL 6
0100000	SLT 6
0000000	

As a matter of convention, vertical format unit 8 (VFU 8) in instruction word 2 is used for slewing to the top of the page.

Printer Test and Branch Instructions

The following branch instructions test whether a particular printer controller condition is true or false. If the condition tested is true, the computer executes the next sequential instruction. If false, the computer skips the next instruction and executes the second sequential instruction.

BCS BPR P 2514P20 Word Times: 2

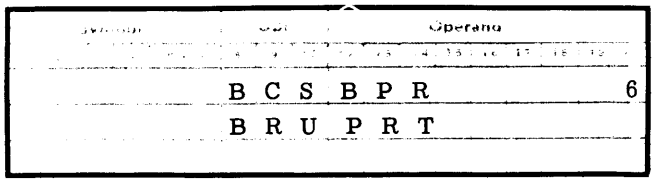
Functional Description: BRANCH ON PRINTER READY. The printer controller P is tested for the ready status.

BCS BPN P 2516P20 Word Times: 2

Functional Description: BRANCH ON PRINTER NOT READY. The printer controller P is tested for not ready status.

Example 1: Test the printer controller for ready status and, if ready, branch to symbolic memory location PRT.

GAP Coding:

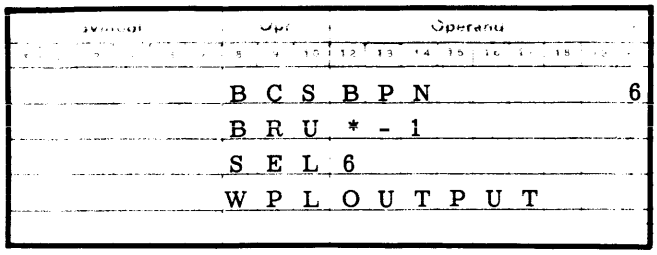


GAP Assembly:

Octal Contents	Coding
2514620	BCS BPR 6
2601750	BRU PRT

Example 2: Test the printer controller for ready status and, if ready, print a line of data from symbolic memory location PRT. If not ready, delay.

GAP Coding:



GAP Assembly:

Octal Memory	Contents	Coding
01113	2516620	BCS BPN 6
01114	2601113	BRU *-1
01115	2500620	SEL 6
01116	2600000	WPL OUTPUT
01117	0101373	

Comments: It is good programming practice to test the printer controller for ready status before issuing a print command.

BCS BOP P 2514P22 Word Times: 2

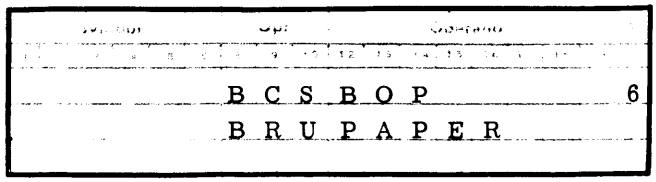
Functional Description: BRANCH ON OUT OF PAPER. The printer controller P is tested to determine if the printer is out of paper.

BCS BNP P 2516P22 Word Times: 2

Functional Description: BRANCH IF NOT OUT OF PAPER. The printer controller P is tested to determine if it is not out of paper.

Example: Test the printer controller for an out of paper condition. Branch to PAPER if condition exists.

GAP Coding:



GAP Assembly:

Octal Contents	Coding
2514622	BCS BOP 6
2603613	BRU PAPER

Comments: It is the responsibility of the programmer to test for the printer being out of paper before issuing a print command. The out-of-paper condition is indicated when only a few inches of paper remain.

BCS BER P 2514P27 Word Times: 2

Functional Description: BRANCH ON ERROR. The printer controller P is tested for any error, such as a parity or input-output buffer error.

BCS BNE P 2516P27 Word Times: 2

Functional Description: BRANCH ON NO ERROR. The printer controller P is tested for a no-error condition.

Example: Delay until the printer completes the printing of a line, then test for any error condition. If any exists, branch to symbolic memory location PRT ERR.

GAP Coding:

Symbol	Opr	Operand	X
PRTLIN	BCS	BPN	6
	BRU	* - 1	
	SEL	6	
	WPL	PRT	
	BCS	BPN	6
	BRU	* - 1	
	BCS	BER	6
	BRU	PRTERR	

Print Cycle

The print cycle consists of three basic phases: filling the 120-character core buffer, printing the contents of the buffer, and slewing the paper.

The fill phase begins when format and data words begin entering the printer controller. This filling begins when the printer (through the controller selector) is granted access to memory. To illustrate, assume that the following print instructions were programmed:

GAP Coding:

Symbol	Opr	Operand	X
	BCS	BPN	6
	BRU	* - 1	
	SEL	6	
	WFL	FORM	
	WPL	PRT	

When SEL 6 enters the central processor I register, instruction words two and three (WFL and WPL) are automatically sent to the printer controller on connector plug 6. Here, they enter the data word buffer where the starting addresses (symbolic location FORM for format words and symbolic location PRT for data words) are transferred to the printer controller format and data address counters. These instructions then initiate printer operations.

Note that instruction words, format words, and data words all enter the printer controller through the data buffer.

The printer controller then requests the first word of format data from memory. This request to memory is made on a priority interrupt basis during each access time. First, a format word is requested, then a data word, then a format word, etc. After each access, words enter the 21-bit data buffer where parity is checked. Format words are sent to the format buffer; data words stay in the data buffer.

CORE BUFFER CAPACITY. The loading process continues until the core buffer is filled with the desired number of characters to be printed in one print line. Note that this buffer need not always hold 120 printable characters before printing can occur. If only 30 characters (10 BCD words, each word containing 3 characters) need to be printed, then only 30 characters need to be loaded into the core buffer. It should also be noted that, for the high-speed printer, the core buffer can hold a maximum of 120 characters. This includes both data and format words. Thus, if 35 data words are held and the programmer has programmed to insert 6 words of format control characters, the 40 word limit is exceeded and the core buffer overflows and an overflow alarm is shown on the printer controller display panel.

In addition, whenever data is loaded into the core buffer, the sign of the last word (3 BCD characters) to be printed should always be set to minus. This rule must be followed, regardless of the number of

words loaded into the buffer, whether it is 4 words or 40 words.

Example: Print eight words of data starting with symbolic memory location PRT.

GAP Coding:

Symbol	Opr	Operand	X
	LDA	PRT + 7	
	BPL		
	CHS		
	STA	PRT + 7	
	BCS	BPN	6
	BRU	* - 1	
	SEL	6	
	WPL	PRT	

Comments: It is a serious programming error to fail to set to minus the sign bit of the last data word to be printed.

After the core buffer is filled, the printing phase of the print cycle occurs, resulting in the printing of the data line.

SLEWING PHASE. Once printing is completed, the printer controller core buffer is ready to receive new data (fill phase of the print cycle), and slewing, or spacing, of the paper begins.

Slewing is the third phase of the print cycle. At this point, note that slewing can be part of a print instruction or it can be a separate instruction. The normal procedure involves slewing as part of the print instruction which automatically slews paper one line after each line is printed. When it is desired to slew paper more than one line, and a separate slew instruction (SLW) is programmed, it is possible to slew only, or print and then slew a specified number of lines. To combine the operations of printing and then slewing more than one line, the slewing instruction must be written in octal as shown in the example:

Example: Write a print line from location 1000 and slew 10 lines.

GAP Coding:

Slewing by Instruction:

Symbol	Opr	Operand	X
	BCS	BPN	6
	BRU	* - 1	
	SEL	6	
	WPL	1000	
	BCS	BPN	6
	BRU	* - 1	
	SEL	6	
	SLW	10	

Slewing as Part of Print Instruction:

Symbol	Opr	Operand	X
	BCS	BPN	6
	BRU	* - 1	
	SEL	6	
	OCT	260000000	
	OCT	120175 0	

Slewing can be programmed by either of two methods: (1) slew by the countdown or (2) slew to a paper tape channel.

Slewing by Countdown. If the programmer desires to slew by countdown, he can write printer instructions in octal and conform to the printer instruction format illustrated in Figure 9-5. Note that bit positions of instruction words two and three, designated by VFU 1 through VFU 6, contain the binary count for the number of lines to be slewed. Bit positions designated by VFU 7 and VFU 8 must always contain 1-bits to accomplish the countdown slew method. If a 1-bit does not appear in both of these bit positions, then slewing is by the paper tape channel method.

Slewing to a Channel. This method of slewing is accomplished by an eight-channel tape loop located within the printer unit. The tape loop gives the programmer the ability to control vertical spacing on printed forms by slewing to a punch in a specified channel in the paper tape loop. If only one of the eight VFU bits is set to 1, paper is slewed until a hole is detected in the VFU tape channel designated by that bit. For example, if VFU 3 contains a 1-bit, paper slews until a hole is detected in VFU tape channel three. VFU tape channel eight should be used for slewing paper to the top of the page.

Automatic Format Control

In the printer controller, when a data line is to be printed under format control, format words are stored in the central processor memory in blocks of words the same as the print line data. Format control data consists of (1) any printable character, such as a dollar sign, and (2) any control characters, such as ignore, delete, skip, etc. In assemblying a formatted print line, the printer controller reads from memory one format word and one data word. The first format character of a format word is examined by the printer controller. If it is a printable character, it is stored in the controller core buffer for printing. If it is a control character, the printer controller takes the action required. The format character is always examined before the data character.

FORMAT CONTROL CHARACTERS. Five special control characters can be programmed in a format word to control the horizontal format of a printed line. In addition, the $ character (octal 53) exercises a control function, as well as being a printable character. These characters, their octal representations, and their functions are shown below. How to program them into format words is described in the subsequent automatic format control examples.

1. Ignore (Octal 35)

If the format character is an Ignore, the next data character is immediately considered for storage in the printer controller core buffer. This next data character will occupy the position on the printed line that the format character would have occupied had it been a printable character.

Example:

Format	35	35	35	35	35	35
Data	G	E	-	2	2	5
Print Line	G	E	-	2	2	5

2. Ignore/Skip (Octal 36)

If the format character is an Ignore/Skip, the next data character is immediately considered for storage in the printer controller core buffer. This data character will not occupy the same position that the format character would have occupied on the printed page, but is placed in the next position. In other words, a blank is inserted on the printed page in the position of the format character.

Example:

Format	36	35	35	35	35	35
Data	G	E	-	2	2	5
Print Line		G	E	-	2	2	5

3. Delete (Octal 37)

If the format character is a Delete, neither this character nor the following data character are stored in the printer controller core buffer. The next character that can be stored in the core buffer is the next format character. In examining the format/data word pair, if the second format character is a delete character, this character and the second data character are not stored in the core buffer; the third format character would be the next character that could possibly be stored.

Example:

Format	37	35	35	35	35	35
Data	G	E	-	2	2	5
Print Line	E	-	2	2	5	

4. Delete/Skip (Octal 56)

If the format character is a Delete/Skip, an action similar to format delete occurs. The difference is that a blank is inserted in that position on the print line. Format character 3 of the format word would occupy the next print line position, if it is a printable character.

Example:

Format	56	35	35	35	35	35
Data	G	E	-	2	2	5
Print Line		E	-	2	2	5

5. Zero Suppress (Octal 57)

If the format character is a Zero Suppress, neither this character nor the following data character is stored in the printer controller core buffer. The next format character can, however, be stored in the core buffer, if it is a printable character. After this last format character is considered, blanks are inserted in the print line for each of the succeeding data characters and printable format characters, until either a non-zero data character is detected or a period character is found in a format word.

For example, if format character 1 is a Zero Suppress, data character 1 cannot be stored in the core buffer. Format character 2 can be stored, if it is a printable character. After format character 2 has been considered, blanks are inserted for each succeeding data character until a non-zero data character or a period format character is found. When either of these characters is encountered, it is stored in the core buffer and the Zero Suppress condition is ended.

Note that, once a Zero Suppress has been put into effect, print line data is examined by the printer controller only for a non-zero data character. Format control data is examined only for a period character.

Examples:

Format	57	35	73	35	35	73	35	35	33	35	
Data	0	8	9	9	9	9	9	9	9	9	
Print Line	8	,	9	9	9	,	9	9	9	.	9	9

Format	57	35	73	35	35	73	35	35	33	35	
Data	0	0	0	8	9	9	9	9	9	9	
Print Line	Δ	Δ	8	9	,	9	9	9	.	9	9

Format	57	35	73	35	35	73	35	35	33	35	
Data	8	9	9	9	9	9	9	9	9	9	
Print Line*	9	,	9	9	9	,	9	9	9	.	9	9

*Note the loss of the first data character in this example.

Format	57	35	35	35	35
Data	9	,	9	9	9
Print Line**	,	9	9	9	

**Note the loss of the first data character and printing of a comma within the data.

6. $ Zero Suppression (Octal 53)

A $ symbol in the format data word initiates spaces in the print line in the same manner as Zero Suppress, except that the next data character is not ignored after the $ symbol is printed. In this case, if the next data character is a non-zero character, it will be printed.

Examples:

Format	53	73	35	35	73	35	35	33	35		
Data	8	9	9	9	9	9	9	9	9		
Print Line	$	8	,	9	9	9	,	9	9	9	.	9	9

Format	53	73	35	35	73	35	35	33	35		
Data	0	8	9	9	9	9	9	9	9		
Print Line	$		8	9	9	,	9	9	9	.	9	9

Format	53	35	35	35	35
Data	0	,	9	9	9
Print Line*	$		,	9	9	9

*Note the zero suppression and printing of the comma which is a printable character.

Note that when zero suppression on the print line is in effect as a result of a dollar sign or an octal 57 (zero suppress) in the format words, a comma in the format word is not printed (but a blank is printed instead), unless the data character preceding the comma in the format word is a non-zero character.

Automatic Format Control Coding

In coding for automatic format control, format words are loaded into a user's program in octal form:

Symbol	Opr	Operand	X
FORMAT	OCT	0 3 5 3 5 3 5	
	OCT	0 3 5 3 5 3 5	
	OCT	0 3 5 3 5 3 5	

Then, when the write format line (WFL) instruction is given, these constants are called from memory and used to control the format of the print line. A format character is examined, then a data character, then a format character, etc., until printing is accomplished. Also in the process of examining a format word character, then a data word character, it is possible for an Ignore or an Ignore/Skip character (octal 35 or 36) to be encountered in the print line data (as well as in the format control data). If a data character is an Ignore, the next format character is immediately considered and nothing is printed for the Ignore character. If a data character is an Ignore/Skip character, a blank is printed and the next format character is considered.

In sequence, the second format character, then the second data character, are considered, followed by the third format and the third data characters. Following the consideration of the third data character, another data word and another format word are requested from the central processor in memory. Upon receiving these new words, the procedure described above is repeated. This routine is continued until a 1 (minus) in the sign-bit position of the last data word to be printed is encountered; after consideration of this data word and its respective format word, the print line control sequence is ended.

The procedure described makes it possible for a print line format arrangement to be stored in memory once and to be used as often as needed to print lines of data in that same format, such as for payrolls, etc. The data may, within the limitations imposed by the editing control as described, be stored in sequence in computer memory; the printer controller automatically constructs the print line according to the prescribed format. By using such stored format routines, any number of different printed forms can be used.

HIGH-SPEED PRINTER (OFF-LINE)

The model 4WPA690 printer is designed for both on-line and off-line operation. When used on-line, the printer is similar to model 4WP225A except for additional field-spreading ability. This additional capability is discussed later. However, in the off-line mode, model 4WPA690 has very distinctive features, such as a separate 1024-character core buffer within a special magnetic tape controller. This buffer is used only during off-line operations.

On-Line Operations

Operation of the high-speed printer model 4WPA690 in the on-line mode differs from that of the on-line printer, model 4WP225A, because of the field-spreading feature. This feature allows the insertion of up to 120 characters in the print line from format information, without requiring dummy characters to be included as part of the print data. To the programmer, this feature provides the ability to spread fields by inserting as many blank spaces (or other printable format data) as necessary in the print line, without deleting existing print line data.

When field-spreading occurs, the normal one-for-one comparison of a format word character and a data word character is interrupted until a field is spread as desired. After blanks or additional printable format characters are inserted to spread the field, the regular one-for-one character examination is resumed. This feature is described more fully below.

NEW FORMAT CHARACTERS. In automatic format control, various control characters, such as ignore (octal 35) or delete (octal 37), can be programmed as format constants in the user's program. These format characters are stored in the central processor memory and, when called by the printer controller, are used to edit a print line.

For the on/off-line printer operating in the on-line mode, format characters are programmed and used in the same manner. However, to allow error-free formats to be used for printing operations in either mode, two new format control characters must be used: an octal 55 to indicate field-spreading and an octal 77 to indicate end-of-line data. Octal 55 is described here because of its relationship to on-line printing operations with the on/off-line printer. Octal 77 is discussed with off-line programming considerations.

FIELD SPREADING. An octal 55 in a format line causes the printer controller to store succeeding octal characters of the memory format information in the 120-character core buffer until an octal 35 (ignore) is encountered in the format information. During this time, data word characters are ignored. Once an octal 35 is encountered, the normal process of examining a format word character against a data word character is resumed.

This operation is shown in Figure 9-6, which shows a line of format words and a line of data words. In normal print operations, word 2000 would enter the printer controller format word buffer, word 3020 would enter the data word buffer, and the first format word character would be examined against the first data word character. If the first format word character were an octal 35 (ignore), then the first data word character (B) would be stored in the printer controller 120-character core buffer. However, in the example shown, the first format word character is an octal 55, which indicates to the printer controller to begin field spreading.

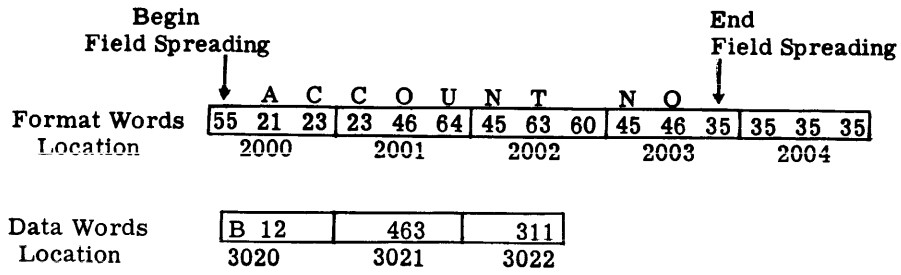


Figure 9-6. Field Spreading in Off-Line Operation

In this case, it is desired to spread the field by inserting the words ACCOUNT NO before resuming the normal examination of format word characters against data word characters. Thus, data word characters B12 remain in the printer controller data word buffer while format characters in words 2000 through 2003 corresponding to the various octal codes are brought in from memory through the format word buffer and inserted in the printer controller buffer. Note: the octal codes for ACCOUNT NO are shown beneath each alphabetic character. Field spreading continues until the printer controller encounters an octal 35, signifying an end of field spreading. At this point, characters in format word 2004 enter the format word buffer. It should be recalled that the characters B12 of data word 3020 were already in the data word buffer when field spreading began. The normal process of examining format word characters then begins.

OCTAL 55/35 RELATIONSHIP. A word of caution about programming for field spreading: When programming for on-line printing operation, the octal 35 must be in the same relative position of a word in which the field spreading code (octal 55) appears. Figure 9-6 shows a line of format words and a line of data words. In this case, the octal 35 is not in the same relative position of format word 2003 as the field spreading octal 55 in word 2000. This format line would work satisfactorily only in the off-line mode. To accomplish the same field spreading results in both the on- and off-line modes, then the octal 35 which terminates field spreading must be in the same relative position of a word as the first octal 55 appears. This is shown in Figure 9-7.

Note, in this case, that the first octal 55 appears in position one of the word at location 0800. To keep the proper octal 55/35 relationship, the octal 35 must appear as the first position of the word at location 0804. If the information to be printed in the on-line mode is such that this 55/35 relationship does not exist, then either one or two (in this case, one) octal 55 codes must be placed in the format line to create that relationship. In the example shown, an octal 55 has been placed in the third position of the format word at location 0803 so that octal 35 in the format word at 0804 occupies the same relative position in the word as the first octal 55 does at location 0800.

The important conclusion is that, due to this relationship, the example in Figure 9-7 will now work properly in both the on- and off-line printing modes. By contrast, the example in Figure 9-6 will work properly in the off-line mode only.

COMMAND WORDS. Printer command words for on-line printing are the same as specified for the regular on-line printer model 4WP225A, except for the following additional instructions.

Programs written for the on-line printer, Model 4WP225A, can be run with the on/off-line printer in the on-line mode.

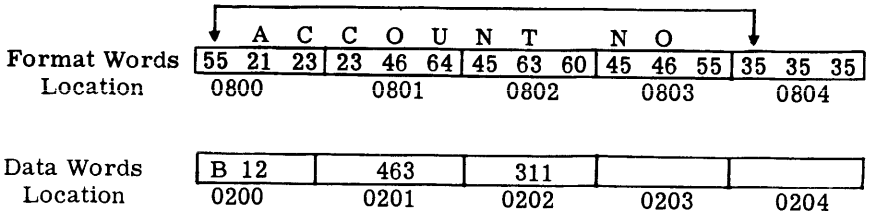


Figure 9-7. Field Spreading for Both On- and Off-Line Printing Modes

Additional Test-and-Branch Instructions

BCS BAA P 2514P21 Word Times: 2

Functional Description: BRANCH ON ANY ALERT. The printer controller P is tested for any alert condition. Any alert includes the following conditions:

1. Slew Alert
2. Buffer Overflow
3. Parity Error
4. Out of Paper

Example: Branch to print error routine, if any alert condition exists.

GAP Coding:

Opr	Operand	X	REMARKS
BCS	BAA	6	
BRU	PRTERR		BRANCH TO ROUTINE TO DETERMINE TYPE OF ALERT CONDITION

BCS BNA P 2516P21 Word Times: 2

Functional Description: BRANCH ON NO ALERT. The printer controller P is tested for no alert condition.

Example: Branch to print routine, if no alert condition exists.

GAP Coding:

Opr	Operand	X	REMARKS
BCS	BNA	6	
BRU	PRT		BRANCH TO PRINT ROUTINE

BCS BSA P 2514P24 Word Times: 2

Functional Description: BRANCH ON SLEW ALERT. The printer controller P is tested for a slew alert condition.

Example: Branch to slew error routine.

GAP Coding:

Opr	Operand	X	REMARKS
BCS	BSA	6	
BRU	SLWERR		BRANCH TO SLEW ERROR MESSAGE

BCS BNS P 2516P24 Word Times: 2

Functional Description: BRANCH ON NO SLEW ALERT. The printer controller P is tested for no slew alert condition.

Example: Branch to load print routine, if no slew alert.

GAP Coding:

Opr	Operand	X	REMARKS
BCS	BNS	6	
BRU	LOAD		BRANCH TO LOAD PRINT ROUTINE

BCS BOV P 2514P23 Word Times: 2

Functional Description: BRANCH ON PRINTER BUFFER OVERFLOW. The printer controller P is tested for buffer overflow condition.

Example: Branch to buffer error message, if printer buffer overflow.

GAP Coding:

Opr	Operand	X	REMARKS
BCS	BOV	6	
BRU	BUFERR		BRANCH TO BUFFER ERROR MESSAGE

BCS BNO P 2516P23 Word Times: 2

Functional Description: BRANCH ON NO PRINTER BUFFER OVERFLOW. The printer controller P is tested for no buffer overflow condition.

Example: Branch to processing routine, if no buffer overflow.

GAP Coding:

Symbol	Opr	Operand	X
	BCS	BNO	6
	BRU	PROCES	

Off-Line Operation Characteristics

When the on/off-line printer is operating off-line, it functions as an independent system consisting of a magnetic core buffered tape reader, a magnetic tape

controller, a 900 line per minute printer mechanism, a printer controller, and a self-contained power supply. In the off-line mode, automatic format control is provided, along with selective file printing (file stacking). The buffered (1024 characters) magnetic tape reader in this system requires the use of 200 character per inch density tapes operating at a rate of 15,000 characters per second. Tape reading speed is 75 ips with a rewind speed of 150 ips.

CONTROLLERS. The printer controller has all the capabilities provided in the regular on-line printer sub-system previously described, including a 120-character core buffer. The major difference between the on/off-line and the on-line printer systems is the additional format control capability, field-spreading.

The magnetic tape controller used in off-line printing operations is distinctly different from other magnetic tape controllers in that it contains a 1024-character addressable core memory or buffer. The entire buffer can be used for print data only or, as described below, for some combination of data and format. This tape controller can control reading of data reels which have been prepared by either the GE-225 or similar tape handler systems.

PROGRAMMING. Before printing off-line, all programming instructions, format, and data are stored on reels of magnetic tape. These reels of tape serve essentially the same function as does core memory in normal on-line print operations. From the programming viewpoint, three key differences exist:

1. Because tape handlers do not have a buffer register as does the central processor or core memory, the magnetic tape controller contains a 1024-character addressable memory, or buffer, into which magnetic tapes are read before entering the printer controller.

2. Write format line (WFL) and write print line (WPL) instructions are never programmed when coding for off-line operations. Print instructions are always given as write tape instructions.

3. All data to be written on tape for off-line printing must always be written in the 18-bit special binary mode. This includes both format and data instruction words. Thus, the WTS instruction always is used.

These and other programming considerations for off-line printing are discussed more fully in off-line programming considerations.

CONTROL SWITCHES. The following additional controls are located on the high-speed printer mechanism.

1. START:

When this switch is depressed, off-line printing operation starts. The system must be in the automatic mode.

2. AUTO-MANUAL:

Normal off-line printing mode is automatic. Certain operations such as skip, print one line, rewind, and space, operate only in the manual mode. Switching from automatic to manual mode causes all off-line operations to halt as soon as the print line in process is completed.

3. PRINT ONE LINE:

When the AUTO-MANUAL switch is in the MANUAL position, this switch causes one line to be printed each time the switch is depressed. Format and slew are still determined by the command words for each line of print.

4. SPACE PAPER:

In the manual mode, depressing this switch causes the printer to slew one line.

5. BACKSPACE:

Causes the tape reader to backspace one tape record. When the AUTO-MANUAL switch is in the AUTOMATIC position, the tape record is re-read automatically. If backspacing is accomplished while in the manual mode, the tape halts after backspacing.

6. SKIP:

Causes tape to advance one record. The selected file is read into the core buffer but is not printed. This switch operates in the manual mode only.

7. REWIND:

Causes tape to be rewound and positioned at load point. Operates only in the manual mode.

8. MANUAL CLEAR:

Clears all alert indicators.

9. MEMORY DUMP:

Causes octal printout of the off-line printer core buffer during off-line mode or the on-line GE-225 core memory during on-line mode.

10. ON-LINE/OFF-LINE SWITCH:

This switch sets the mode of operation.

Two additional switches are also provided on the magnetic tape controller. These switches are:

1. FILE SELECT:

Selects one or more of eight codes. Code 0 causes the entire tape record to be written sequentially (one character at a time) into the core buffer, starting at location 0000. Codes 1 through 7 permit only those print lines with the pre-selected file codes to be stored in the appropriate locations. Code 0 applies, regardless of the condition set by the other file select switches.

2. HIGH-LOW DENSITY:

A switch is provided to select tape density for use in systems capable of reading high-density tape.

INDICATOR LIGHTS. In addition to the indicator lights on the on-line printer mechanism, other lights are provided for off-line operation as described below:

1. TAPE ALERT:

A tape alert includes longitudinal parity, lateral parity, modulo check, core buffer overflow, and interlock. Each of the above alerts is indicated separately on the buffered tape reader control panel.

2. END-OF-FILE:

Indicates that an end-of-file code was detected on tape (this is a tape file code and should not be confused with the logical print file code).

3. END-OF-TAPE:

Indicates that the physical end-of-tape or beginning-of-tape marker has been detected.

4. BUFFER PARITY:

Indicates that incorrect parity has been detected on a character being read out of the core buffer.

5. INPUT OVERFLOW:

Indicates an overflow condition in the printer controller core buffer.

6. SLEW ALERT:

Indicates that an attempt was made to slew paper beyond the limitations allowed under program control.

7. PAPER ALERT:

Indicates loss of paper tension due to tearing or out-of-paper condition.

8. INPUT PARITY:

Indicates that a word received from the central processor during on-line operation has incorrect parity.

9. ON-LINE/OFF-LINE:

Indicates operating mode.

ERROR CHECKING. Checking features of the off-line printer sub-system assure accurate performance. Three of these checks apply to the printer itself and four apply to the operations of the special magnetic tape controller. One check, computer parity, is used for on-line operation only.

The printer checking features include:

1. Print Buffer Overflow:

This check detects whether 120 characters have been written into the core buffer without an end-of-line code being detected. The printer halts without printing the line in question.

2. Slew Alert:

This check detects paper slew runaway. If the top of page channel is detected twice during a single slew operation, printing is inhibited and an alert indicator light is turned on.

3. Paper Alert:

An out-of-paper condition or loss of tension due to paper tearing is revealed by this check.

Checking features involving the operations of the magnetic tape controller include:

1. Tape Parity Alert:

When a lateral or longitudinal parity error on tape has been detected, the system halts as soon as the record on tape has been read. No portion of that record is printed unless the backspace button is depressed and the record is reread without recurrence of an alert.

2. Mod Alert:

Checks to see that the number of characters in the tape record is a multiple of 3.

3. Core Buffer Overflow:

If more than 1023 characters are transferred from tape to the core buffer, an alert is indicated. No characters are printed until recovery action is taken.

4. Core Buffer Parity:

Each character read from the off-line core buffer is checked for correct parity.

Off-Line Programming Considerations

As discussed previously, off-line printing operations differ from on-line operations in that magnetic tape is used for holding both format and data words, a separate magnetic tape controller with its own 1024-character core buffer is required, and all print instructions are given as write tape instructions in the special binary mode (WTS). Format and print lines can still be programmed to perform automatic editing of data to be printed. However, the particular method of specifying editing information emphasizes three points that the programmer should keep in mind when preparing format and data to be written on tape for off-line printing.

1. An 18-bit command word must precede the actual format line (Figure 9-8).

2. Two command words must precede the data line.

3. Special end-of-line codes are required at the end of both format and data lines.

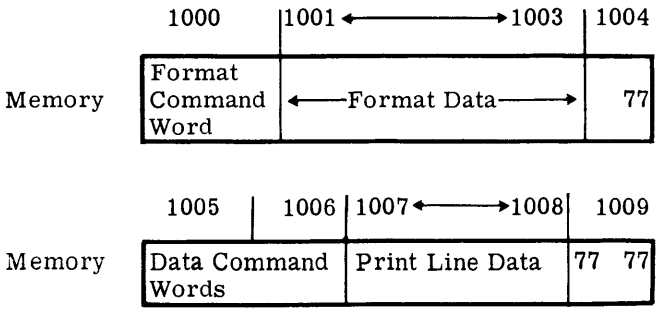


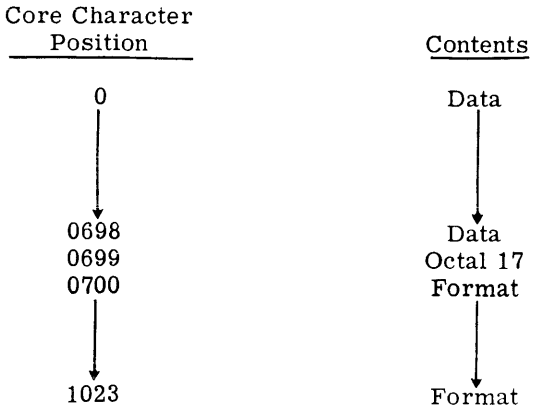
Figure 9-8. Command Words for Off-Line Printing

Format and data lines are discussed below. First, it is important to explain the use of the octal 77 character mentioned earlier.

OCTAL 77. An octal 77 is the code used in an end-of-line word to indicate the end of either a format or data line including tape identification labels. Three possibilities exist for the last word in a data or format line. Two of these possibilities are shown in Figure 9-8. In the case of the format line, only one end-of-line octal 77 is needed, because two printable or special characters occupy the remainder of the word. In the last word of the data line, two octal 77's are required. The third possibility, obviously, is the use of three octal 77's.

In all cases, the programmer should make the last character on magnetic tape pertaining to any format or data line an octal 77. When it is necessary to include more than one octal 77 to fill out a partial word, such as shown in Figure 9-8, all but one octal 77 is stripped off automatically so that only one end-of-line code for each format or data line need be stored in the 1024-character magnetic tape controller core buffer.

The programmer also should note that the tape controller generates an octal 17 and automatically stores it in the character position following the last character of the last print line stored in memory from a block of tape. The octal 17 tells the controller to read another block from tape after printing the last line of print in the tape controller buffer. For example, assume that format data is stored in the tape controller core buffer from character 0700 through character 1023. The data to be printed from any one tape block may then not exceed 699 characters (0-698) because the octal 17 control character is held in character position 0699 as shown below:



Format Line Information

In the off-line printing mode, it is possible to edit print line data by storing appropriate format codes (octal 35, 36, etc.) as constants. Thus, when magnetic tape is read into the controller 1024-character core buffer, format data is stored or held until such time as the user's program calls upon it. The coding for calling format data from the buffer is described below.

To understand format editing by the high-speed printer in the off-line mode, the programmer should recognize that a format line consists of some combination of format codes, preceded by one format command word, and followed by an end-of-line word (a word with format or data always contains three characters as was illustrated in Figure 9-8).

A format command word provides such information as identification, file number, and the starting address in the core buffer where the format is to be sorted. The specific bit configuration for this command word is shown in Figure 9-9.

Bit Position 0-19	Format Line Command Word
0	Not used
1	Not used
2	Zero
3	One
4	Zero
5	File Select 2^2
6	File Select 2^1
7	File Select 2^0
8	Zero
9	Zero
10	Format Address 2^9
11	" " 2^8
12	" " 2^7
13	" " 2^6
14	" " 2^5
15	" " 2^4
16	" " 2^3
17	" " 2^2
18	" " 2^1
19	" " 2^0

Figure 9-9. Format Line Command Word Bit Configuration

Bit positions 0 and 1 are not used. Bit positions 2, 3, and 4 specify the operation code for a format command word. Bit positions 5, 6, and 7 specify or select the format file number, or rather, the kind of report to be printed and edited by the format data following the command word. Because there are three bits, each with a binary weight, the maximum number of different kinds of reports that can be printed are 8. If no 1-bits are shown, then zero is implied. Code zero (in both the format line command word and data line command words) cause the entire magnetic tape record to be written sequentially, regardless of the kind of data, one character at a time into the tape controller 1024-character core buffer, starting at core location 0000. The meanings of codes 1 through 7 are assigned by the programmer; for example, he may desire to have code 1 (001) be a master file report, a code 2 (010) represent an inventory report, etc.

Returning to the command word, bit positions 8 and 9 are always zeros. Bit positions 10 through 19 specify the starting address in the tape controller core buffer where format data for the report is to be stored. Because bit position 19 has a binary weight of 0 or 1, bit 18 a weight of 2, bit 17 a weight of 4, and bit 16 a weight of 8, a 1-bit in any of these bit positions specifies the starting location where format data is to be stored.

The programmer should note that it is advisable to store format information in the upper portion of the core buffer. This is recommended so that the area where print data is read in will not overlay it. Thus, a 1-bit should generally be specified in bit position 10 so that format data is stored in the core buffer above character location 0512 (2^9 = 0512).

FORMAT EXAMPLE. Assume that an inventory report is to be printed (code 2, for example) and that the format to edit this report is to be stored in the core buffer, beginning at character location 0512. The programmer would prepare the command word as shown in Figure 9-10.

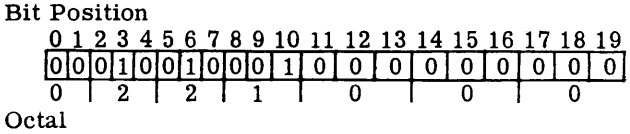


Figure 9-10. Sample Completed Format Command Word

FORMAT LINE CODING. Format line data is loaded into the user's program by programming the octal representations for the control actions desired (including the format command word) as constants in the beginning of a program (Figure 9-11). In this respect, off-line programming of format data is similar to on-line format control coding (see Automatic Format Control Coding).

As illustrated, the format command word and all format data associated with format one (shown as FMT1) are programmed as constants. In the example shown, note that the octal equivalent in coding line 1 is for the format command word shown in Figure 9-10. Other format data is shown in lines 2 through 4.

Symbol	Opr	Operand	X	REMARKS
FMT1	OCT	022100 0		FORMAT COMMAND WORD
	OCT	035355 7		FORMAT DATA
	OCT	036355 7		
	OCT	035777 7		
DATA1	OCT	042570 0		DATA COMMAND WORD
	OCT	060200 0		DATA COMMAND WORD
	ALF	ACC		DATA
	ALF	OUN		
	OCT	063677 7		

Figure 9-11. Sample Coding of Format and Data Information For Off-Line Printing

Note that as many formats can be prepared in constant form as desired; however, in the format command word only seven different kinds of reports can be formatted. Thus, if a programmer wants to print three different reports, with each report requiring five or six different formats, then he must code these formats (including the appropriate format command word) as described, store them as constants in the beginning of the program with the data line information and then, during program execution, write them out on a specified magnetic tape to be used subsequently for off-line printing. Observe, in the sample coding illustrated below, that the constant format data was written on magnetic tape by executing a WTS instruction. The operating logic of off-line printing is described in the material on printing logic.

Example:

Symbol	Opr	Operand	X
	BCS	BTN	2
	BRU	* - 1	
	SEL	2	
	WTS	FMT1	1
		9	

Data Line Information

Regardless of whether data is to be printed unedited or with format control lines, certain considerations must be followed in preparing data for off-line printing.

As indicated previously, a line of data consists of some combination of codes and printable characters preceded by two command words and followed by a termination or end-of-line word (see Figure 9-8). In contrast to format line data, print line data requires two command words which provide such information as shown in Figure 9-12.

Bit Position 0 - 19	Data Line	
	Command Word 1	Command Word 2
0	Not used	Not used
1	Not used	Not used
2	One	VFU-8
3	Zero	VFU-7
4	Zero	VFU-6
5	File Select 2^2	Zero
6	File Select 2^1	Zero
7	File Select 2^0	Zero
8	Print (1) or Slew (0)	VFU-5
9	Format (1) or Not (0)	VFU-4
10	Format Address 2^9	Zero
11	" " 2^8	Zero
12	" " 2^7	Zero
13	" " 2^6	Zero
14	" " 2^5	VFU-3
15	" " 2^4	VFU-2
16	" " 2^3	VFU-1
17	" " 2^2	Zero
18	" " 2^1	Zero
19	" " 2^0	Zero

Figure 9-12. Data Line Command Words and Bit Configurations

In command word 1, bit positions 0 and 1 are not used. Bit positions 2, 3, and 4 indicate a line of data. Bit positions 5, 6, and 7 indicate the file number selected or rather, the kind of report to be printed. If a report is to be printed with format control, the particular report code specified by these bits must always agree exactly with the report code indicated in the appropriate format line command word.

Bit position 8 indicates print, or slew without printing, and bit position 9 indicates print under format control or print without using format. Then, if format is specified, the exact format address indicated in the corresponding format command word is marked in bit positions 10 through 19.

In the second command word, bit positions 2, 3, 4, 8, 9, 14, 15, and 16 are used to specify slewing by the countdown method or slewing to position X, where X is a channel in the printer vertical format unit (VFU)

tape. Note that the same bit configuration rules apply for the off-line as for the on-line printing mode. If slewing by the countdown method is selected, bit positions 2 and 3 in the second command word must contain 1-bits with the subsequent weighted value VFU bit positions also containing 1-bits to indicate the number of lines to slew. Thus, 1-bits in positions 2 and 3 and a 1-bit in bit position 9 would indicate 'slew by countdown method 8 lines.' Remember that bit positions representing VFU1 through VFU6 designate, in binary form, any decimal number from 0 to 63. Thus, the countdown method permits slewing any number of lines from 0 to 63.

Examples of two filled-in data line command words, and their octal equivalent values are shown in Figure 9-13. Note that the kind of report and format address specified agrees, for this illustration, with the report and address specified in the format line command word shown previously in Figure 9-10.

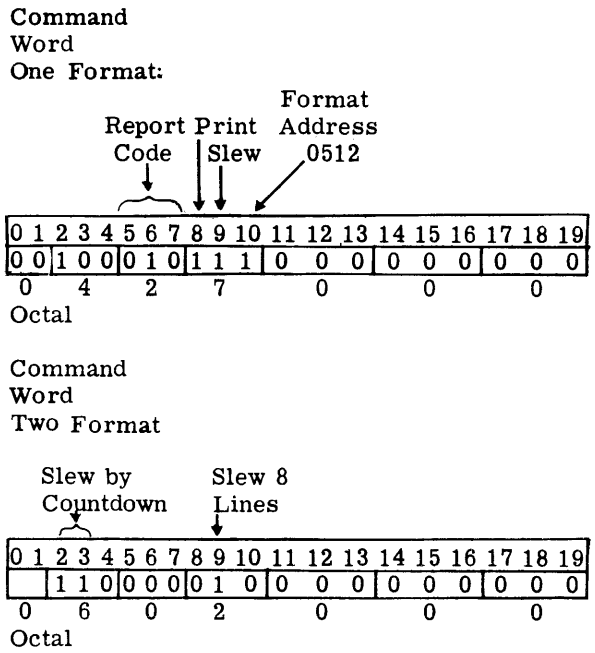


Figure 9-13. Sample Data Line Command Words

DATA LINE CODING. Data to be printed, whether format edited or not, is programmed at the beginning of a user's program as constants. This can be seen in Figure 9-11. Note that the data to be printed under format control immediately follows the necessary format control characters (FMT1 and DATA 1).

There is, however, no need to separate format and data lines, successive lines of data or format, or files of information to be written into the core buffer by an Inter-Record Gap (IRG). Care must be exercised to ensure that not more than 1023 characters are transferred to the 1024-character core buffer during the reading of one tape record. To assure off-line operation at 900 lines per minute (alpha-numeric-single space only), physical tape records containing 750 characters or less are recommended. This recommendation is for tape written at 200 characters per inch density.

Further, the programmer should look at his format and data line constants to determine how many characters (3 characters per word) of the tape controller 1024-character core buffer that a particular tape record will occupy. In the example shown in Figure 9-11, there are 6 data words (not counting command words) at three characters each, or 18 characters. To this is added the six characters comprising the data command words (characters 2 and 3 in word two, 3 characters in word three, and 1 character for a total of 24 to be read into the controller counter). The format command word is not transferred from tape to the 1024-character buffer. Thus, when the WTS FMT 1 instruction is given to write 9 words onto magnetic tape as one tape block, 24 characters actually fill the core buffer when off-line printing occurs. Of course, if several formats and several lines of print data were written on tape, the number of total characters would be much greater. In all cases, however, it is recommended that tape records be 750 characters or less.

Printing Logic

When a WTS instruction is executed, the appropriate format and/or data line information is written from core memory onto a reel of magnetic tape (Figure 9-14). After the desired number of records are so written, this reel of tape is mounted on the special tape handler used for off-line printing.

As shown, the tape handler then begins reading format and data words into the tape controller 1024-character core buffer. As data enters the controller, it checks the first word of each tape block. The first word of each block of information on tape and the word immediately following an 'end-of-line' code are either data or format command words. The file-select code in the command word determines whether or not the succeeding characters are to be decoded and the information read into the 1024-character core buffer. If the information to be stored is a line of format, the command word is decoded to determine the starting address. If it is a line of data, the command words must be stored, with the data, for future use by the printer controller. Lines of data, their command words, and the end-of-line code are stored sequentially by character, starting at memory location zero.

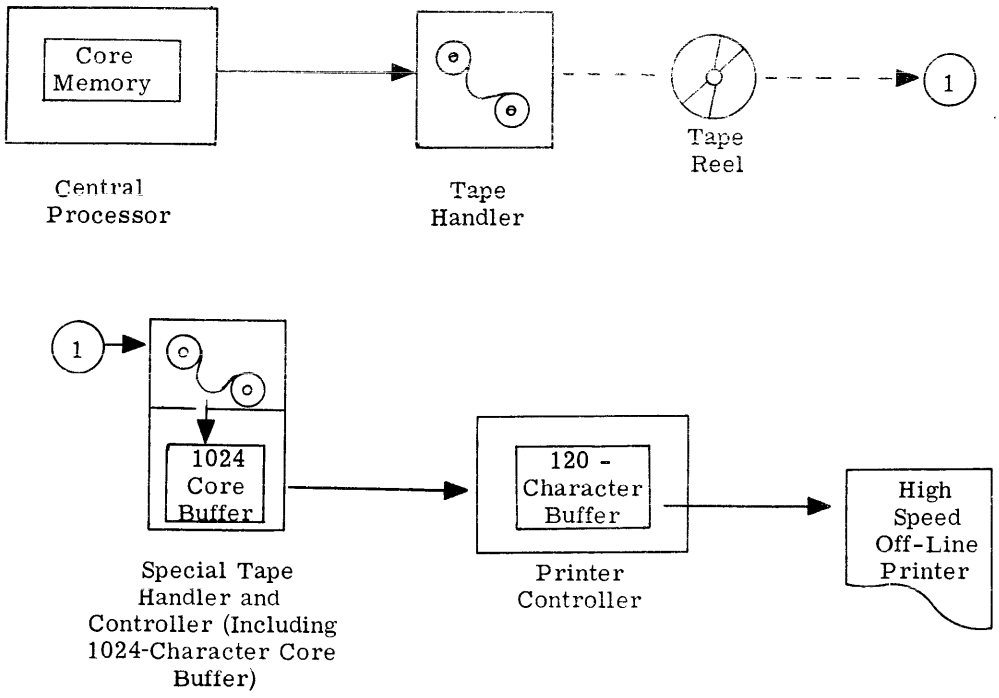


Figure 9-14. Off-Line Printing Block Diagram

Core buffer storage allocation can be calculated by allowing seven additional characters over and above the actual number of data characters for each line of print data. These additional characters include the following:

	Total
Command Word 1	
Characters 2 and 3	2
Command Word 2	
Characters 1, 2 and 3	3
Data Characters	
(any number)	
Octal 77 Character	1
(indicating end-of-line)	
Octal 17	1
(control character inserted automatically by controller once for each tape record)	7

In addition, include one additional octal 77 end-of-line character for each line of format. It should be reemphasized that no octal 17 control character is inserted, not for format data but for print line data only to signal the tape controller that it can read in another block from tape.

When the core buffer is filled, or when a complete tape record has been read into the buffer, transfer operations to the printer controller begin automatically, one print line at a time.

Format and print line data enter the printer controller and are processed as previously described. To repeat, each word is examined alternately, character-by-character starting with a format word. If a character is a printable character, it is stored in the 120-character core buffer. If it is a format character, such as an ignore code, the printer controller takes the format action required. The 120-character core buffer continues to be loaded either until it is full or all data for one print line is filled. In any case, the buffer cannot hold more than 40 words total.

SECTION X

DOCUMENT HANDLER OPERATIONS

The information on the GE-225 12-pocket, 1200-document-per-minute document handlers that was originally in this manual has been superseded by a new and separate manual entitled "GE-225/235 DOCUMENT HANDLER REFERENCE MANUAL" (Publication No. CPB-307).

Use this new manual for the latest and most complete coverage of GE-225 Document Handler operations. Copies are available from General Electric computer representatives.

SECTION XI

MASS RANDOM ACCESS DATA STORAGE OPERATIONS

Mass Random Access Data Storage (MRADS) provides the GE-225 with the ability to store vast quantities of data which can be randomly accessed and quickly processed at low cost.

Each MRADS unit consists of 16 circular magnetic data storage discs which provide 32 recording surfaces. Up to four 16-disc MRADS units can be connected to each MRADS controller. The controller, in turn, is connected to the GE-225 system through the controller selector as shown in Figure 11-1.

Since a MRADS unit can transfer data at a maximum rate of 500,000 bits per second, (23,700 words per second), it requires memory access every other word time. Thus, the recommended address for a MRADS

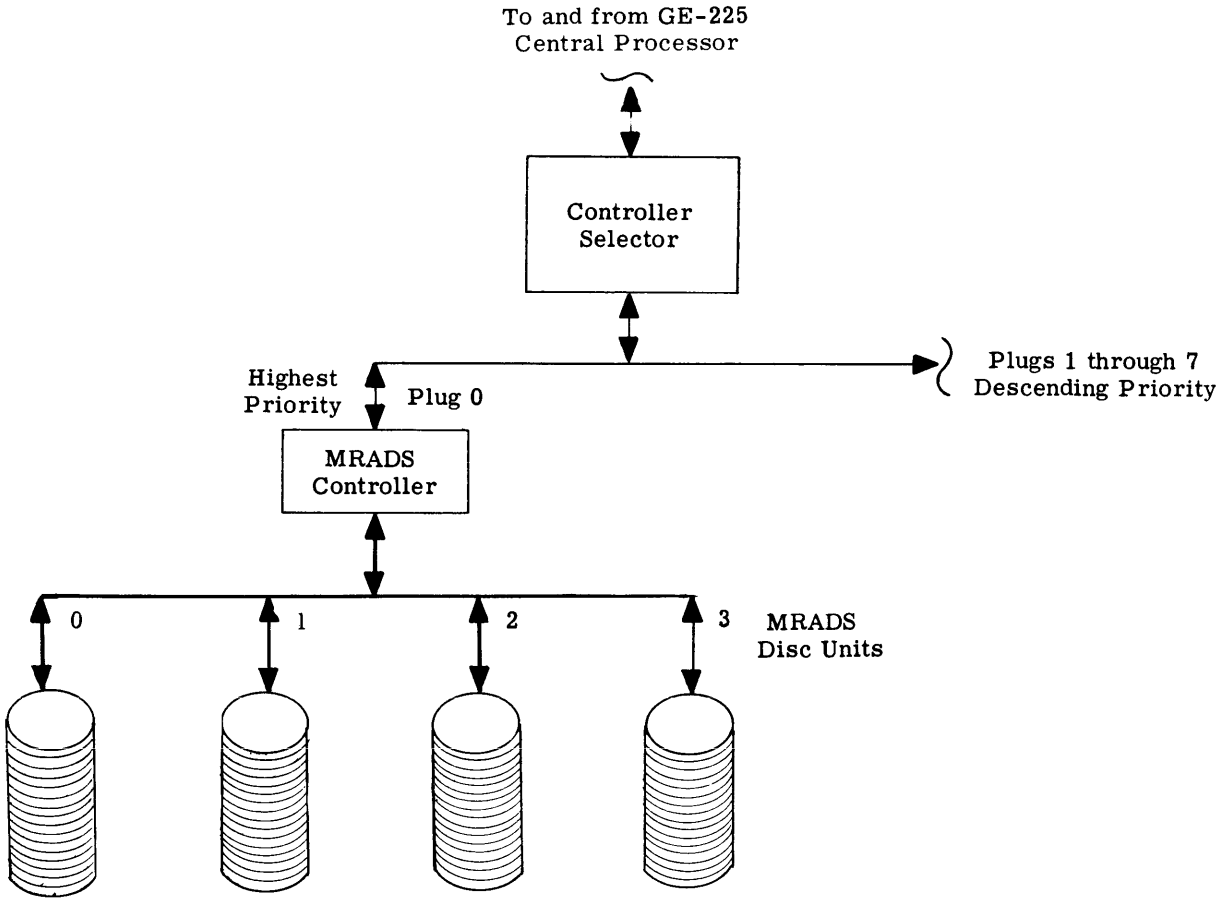


Figure 11-1. MRADS Sub-System

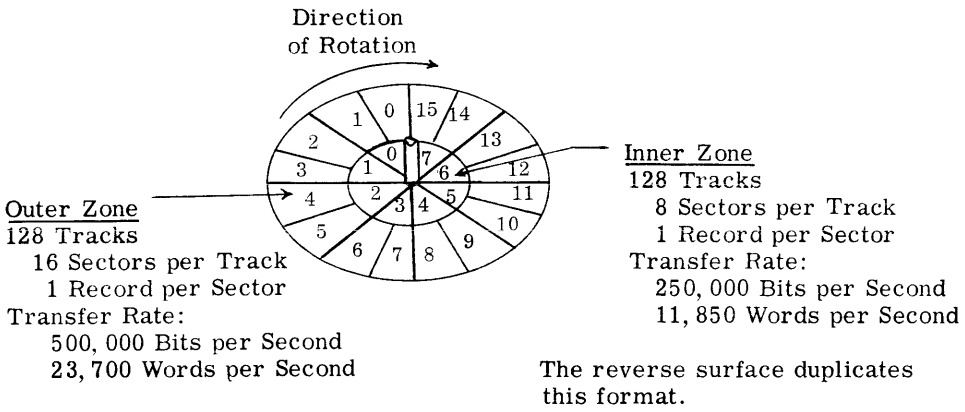


Figure 11-2. MRADS Disc Format

unit is plug zero, but it can have any plug address from zero to seven, provided that it has the highest priority plug (lowest plug number) of the particular configuration connected into the controller selector.

Each 16-disc unit has a capacity of 98,304 records with each record containing 64 words.

FILE DISCS

Each surface of a disc is divided into an inner and an outer zone as shown in Figure 11-2.

There are 128 circular tracks in each zone, providing a total of 256 tracks on each side of a disc. The 128 outer tracks are divided into 16 sectors, while the 128 inner tracks are divided into 8 sectors. Each track sector can store one 64-word record. The transfer rate to or from the 128 outer tracks is 500,000 bits, or 23,700 words per second. The transfer rate to or from the 128 inner tracks is 250,000 bits, or 11,850 words per second.

Each word recorded on a disc is an image of the corresponding memory word; that is, it consists of 20 information bits, plus an odd parity bit. The minimum amount of information which can be transferred in either direction by one instruction is 64 words, or one disc record; data is recorded serially. The maximum amount of information which can be transferred in either direction by one instruction is sixteen 64-word records. A summary of MRADS characteristics is given in Figure 11-3.

Sixteen consecutive records can be read or written by one instruction. Input or output commands which sequentially address the tracks served by the 8 read/write heads, make it possible to read or write 96 consecutive records without moving the positioning arm. The inner heads can address 32 records and the outer heads 64 records.

Operating Times	
Speed of rotation of discs	1200 rpm
Effective transfer rate:	
Outer zone	23,700 words/sec
Inner zone	11,850 words/sec
Average latency time	26 ms
Maximum latency time	52 ms
Average positioning time	199 ms

Physical Characteristics	
Number of data storage discs per file	16
Number of recording surfaces	32
Number of positioning arms	16
Number of read/write heads per positioning arm	8
Number of read/write heads per surface	4
Number of tracks per surface	256
Inner zone	128
Outer zone	128
Number of words per record	64
Number of records per track	
Inner zone	8
Outer zone	16
Number of records per surface	3,072
Number of records per 16 disc file	98,304
Number of data words per file	6,291,456

Figure 11-3. Summary of MRADS Characteristics

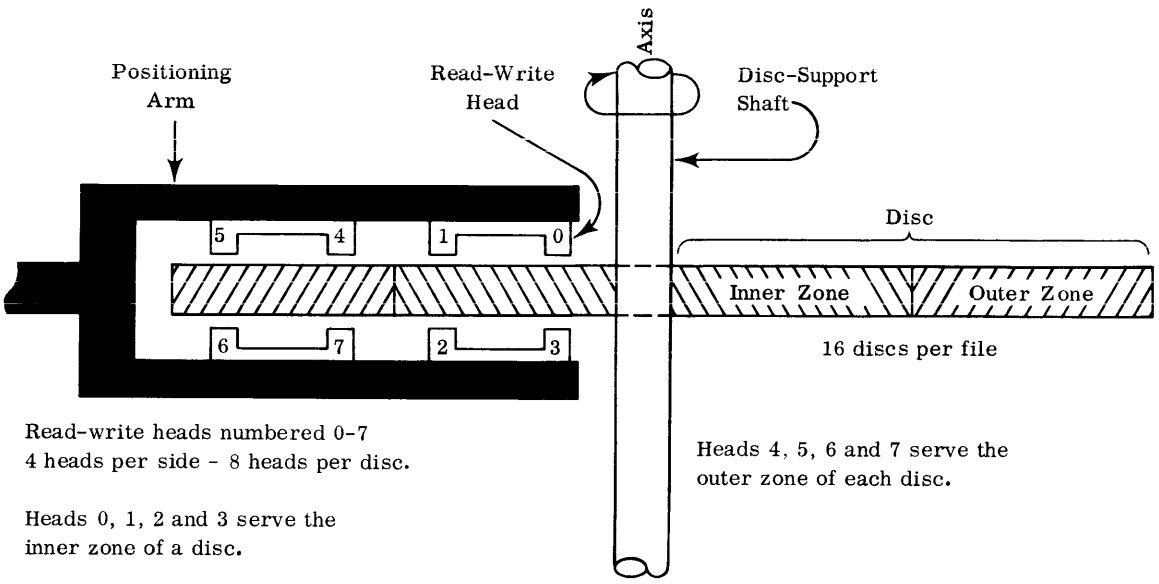


Figure 11-4. MRADS Positioning Arm and Head Configuration

CONTROL WORDS

In addition to the 64 words of information in a record, three additional words are associated with each sector. These are the header word, the longitudinal parity check word, and the synchronization word.

Header Word

A header word precedes each sector and contains the record address. This word is permanently recorded and is not available to the programmer except for addressing the record. When a search is made to locate the correct record for a read or write operation, the header address and the address contained in the instruction word are compared. When the two addresses are confirmed electronically, the addressed record is read or written, depending upon the commands issued.

Longitudinal Parity Check Word

The second additional word associated with each record is generated in the controller when the record is written. This word (word 65) consists of twenty longitudinal parity check bits, one parity bit for each of the twenty bit positions of all 64 data words.

Synchronization Word

The third additional word in each sector is the synchronization word which identifies the end of a record. The purpose of the synchronization word is for timing and need not directly concern the programmer.

ERROR CHECKING FEATURES

There are many built-in precautions to make certain that there is a reliable transfer of data between computer memory and the MRADS. These checks are:

1. Parity Check
   The controller odd parity error check is used during both read and write transfers. If a parity error is sensed at the controller during transfer between the MRADS controller and memory, the controller parity light is turned on. The GE-225 can be programmed so that, when this condition is sensed, the data can be reread or rewritten until a correct transfer is completed.

2. Memory Access Check
   A memory access error occurs when memory access is requested by the MRADS controller but access is not granted. However, the MRADS controller is equipped with a buffer which allows a delay of one access time before this type of input/output error condition can occur. Controller

selector plug 0 is normally assigned to the MRADS controller to reduce the occurrence of this type of error. If a memory access error occurs during an operation, the programmer must repeat the operation.

3. Clocking Check
The clocking check enables the MRADS controller error light to be turned on if there is an inconsistency in the frequency of data transfer during a read or write operation.

4. Address Confirmation
Each record on the MRADS disc file has a header word preceding the first word of the 64-word record. Before a read or write instruction is processed, a comparison is made between the header word and the address contained in the instruction. The record is not considered as addressed until both addresses have been confirmed electronically. After confirmation, reading or writing occurs when the head senses the record to be processed.

5. Longitudinal Parity Word
A longitudinal parity or check word is written as part of each record, following record word 64. If a parity error is detected by the controller during the read process, word 65, the check word, is automatically read into memory along with the 64-word record and the parity error indicator is set. If no parity error is detected on the file read process, word 65 does not enter memory. Word 65 becomes significant when several unsuccessful attempts are made to correctly read a record. If this occurs, a subroutine can check word 65; if only one bit is lost, the record can be reconstructed by inserting the correct bit, and processing will continue without interruption. If it should ever happen that more than one bit is lost, the record cannot be restored internally. However, the program can provide for typing out the record with the incorrect words and continue processing without interruption. Manual correction from reference sources is then necessary.

6. Read After Write Check
This error check is under program control. (See Special Bit Configuration description below.)

7. Additional Error Checks
Other error conditions, such as an invalid address, parity error in address, selection of a file not online, and electro-mechanical failures can also be detected. Except for the electro-mechanical failure error, all the error conditions can be corrected through programming. In addition to controls mentioned, there are other electronic checks used in the equipment to insure accuracy.

MRADS ADDRESSING

MRADS addressing is accomplished in a manner similar to other devices connected to the controller selector in that a SELECT command must be used to specify the plug desired.

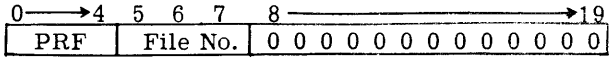
When the MRADS controller is selected, it receives the next two positioning words from memory before the operation is executed. The word format for positioning an MRADS file follows:

PRF	F	2500000	Word Times: 2
OCT	(MRADS Address)	MMMMMMM	

Functional Description: POSITION MRADS FILE. One of the MRADS units F (0 through 3) is positioned to receive or transmit a specific record. The line OCT contains the actual MRADS address (octal) of the selected MRADS unit. This instruction holds power upon completion of the operation. These commands must precede any read or write operations.

Comments: The bit configurations for words two and three are:

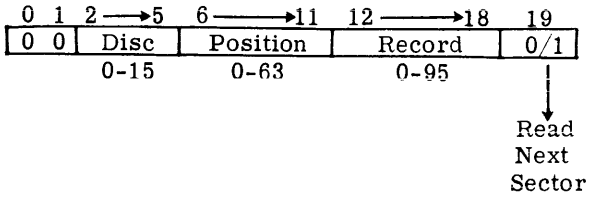
Word Two, PRF



Bit Positions	Contents
0 through 4	Position MRADS instruction
5 through 7	File number, as follows: File 0 = 001 File 1 = 010 File 2 = 111 File 3 = 100
8 through 19	Must be zero

Bit combinations in positions 5, 6, and 7 are automatically developed during assembly. These combinations provide additional file protection, in that bit pick-up or drop in these positions is immediately detected by error-checking logic.

Word Three, OCT



Bit Positions	Contents
0 and 1	Must be zero
2 through 5	Selects the disc number, 0 through 15.
6 through 11	Position, 0 through 63 involved in information transfer.
12 through 18	Record, 0 through 95, to be read or written.
19	Zero. See last paragraph of this Comments section.

The seven bits, 12 through 18, designate the head which will read or write, as well as the number of the record. All 96 records that can be read when the arm is in a given position can be addressed by the seven record bits whose binary value varies from 0000000 for record zero to 1011111 for record 95.

Note: To read 96 records, the instruction must be modified to obtain groups of 16 records. However, positioning time is eliminated.

The bit configurations shown in Figure 11-5 indicate the manner in which specific records are addressed. (Bits 14 and 15 determine inner zone heads 0 through 3; bits 12, 13, and 14 determine outer zone heads 4 through 7. Bits 16, 17, and 18 determine inner zone sectors 0 through 7; and bits 15, 16, 17, and 18 determine outer zone sectors 0 through 15.)

The following disc positions must be reserved for diagnostic routines:

Disc	Position	Head	Sector
0	0	0	0
15	63	7	0-15

Selection of the record to be transferred automatically selects the head to perform the read or write operation. Each of the eight heads on the arm can read a specified number of records as follows:

Head Number	Record Number
0	0 - 7
1	8 - 15
2	16 - 23
3	24 - 31
4	32 - 47
5	48 - 63
6	64 - 79
7	80 - 95

As has been pointed out, the maximum number of records which can be transferred by one instruction is 16. It is not necessary that these 16 records (or any part of 16 records) all be in the outer tracks or all be in the inner tracks. The transfer of information during the execution of an instruction can start in the inner tracks and continue in the outer tracks. As records are being transferred, a count is maintained in the MRADS controller so that the read or write operation continues for the specified number of records. The sequential incrementing of the RECORD address in the controller automatically results in the proper head switching. Because record 95 is the highest valid address, incrementing the address in the controller beyond 95 causes the counter to be set to zero and record zero of the specified position is the next record transferred. Thus it is possible to address record 95, request 16 records to be transferred, and still have 16 records transferred.

NOTE: If 96 records are written, starting from record 0 as indicated in Figure 11-5, five instruction modifications are required. If (by programming) an attempt is made to increment the bit configuration for record 95 (1011111) by one, bits 12 and 13 will become ones; this produces an invalid record address which results in an error signal when an attempt is made to use the address.

It should also be noted that bit 19 of word three, identified by READ NEXT SECTOR, is a variable. When bit 19 of word three is ON (contains a one) the RECORD address is ignored and the subsequent reading or writing operation takes place in the next sector. Bit 19 of word three is used when it is desired to sample a record from a given position of the positioning arm. Rather than search for a specific record out of the 96 possible records, the next record can be read. This form of addressing can also be used when it is known that every sector in a track is to be transferred and it does not make any difference which sector is read first.

Bit Positions									
12	13	14	15	16	17	18		Record	Position
0	0	0	0	0	0	0	=	0	Inner tracks 0 - 63 (top of disc)
			to					to	
0	0	0	0	1	1	1	=	7	
0	0	0	1	0	0	0	=	8	Inner tracks 0 - 63 (top of disc)
			to					to	
0	0	0	1	1	1	1	=	15	
0	0	1	0	0	0	0	=	16	Inner tracks 0 - 63 (bottom of disc)
			to					to	
0	0	1	0	1	1	1	=	23	
0	0	1	1	0	0	0	=	24	Inner tracks 0 - 63 (bottom of disc)
			to					to	
0	0	1	1	1	1	1	=	31	
0	1	0	0	0	0	0	=	32	Outer tracks 0 - 63 (top of disc)
			to					to	
0	1	0	1	1	1	1	=	47	
0	1	1	0	0	0	0	=	48	Outer tracks 0 - 63 (top of disc)
			to					to	
0	1	1	1	1	1	1	=	63	
1	0	0	0	0	0	0	=	64	Outer tracks 0 - 63 (bottom of disc)
			to					to	
1	0	0	1	1	1	1	=	79	
1	0	1	0	0	0	0	=	80	Outer tracks 0 - 63 (bottom of disc)
			to					to	
1	0	1	1	1	1	1	=	95	
1	1	0	0	0	0	0	=	96 through 127 are invalid record addresses. Thus, when an address is generated, only one test (bits 12 and 13) is needed to detect an illegal address.	
			to						
1	1	1	1	1	1	1			

Figure 11-5. MRADS Record Address Bit Configurations

MRADS DATA TRANSFER INSTRUCTIONS

A MRADS unit which has been positioned can be addressed for a read or write operation. It is first necessary to give a SEL P instruction which selects the controller selector plug (P) into which the MRADS unit is connected. Once the MRADS unit has been selected by the controller selector, the MRADS controller goes busy and waits for the next two words from memory. These words are sent to the MRADS controller and indicate the operation to be performed (read or write), the file which is to perform the operation, the number of records to be transferred and the starting address in memory where information is to be sent or retrieved. The GAP instructions for read/write operations follow.

RRF	N	F	1200000	Word Times: 2
(blank)	M		00MMMMM	

Functional Description: READ FROM MRADS UNIT F. N is the number (1 through 16) of 64-word records to be transmitted from MRADS to core storage. F is the number (0 through 3) of the selected MRADS unit. M is the core memory address into which the first word of the record is stored. Power is held on the arm after execution of this instruction.

WRF	N	F	3700000	Word Times: 2
(blank)	M		00MMMMM	

Functional Description: WRITE ON MRADS UNIT F. N is the number (1 through 16) of 64-word records to be transmitted from core storage to MRADS. F is the number (0 through 3) of the selected MRADS unit. M is the core memory address from which the first word is to be transmitted. Power is held on the arm after execution of this instruction.

RRD	N	F	1201000	Word Times: 2
(blank)	M		00MMMMM	

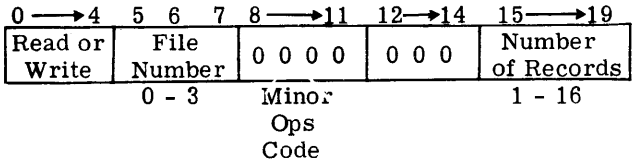
Functional Description: Performs the same function as RRF, except that power is dropped from the arm upon completion of the read operation.

WRD	N	F	3701000	Word Times: 2
(blank)	M		00MMMMM	

Functional Description: Performs the same function as WRF, except that power is dropped from the arm upon completion of the write operation.

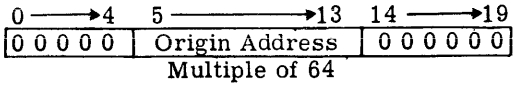
Comments: Note that a position instruction sequence (SEL, PRF, OCT) must precede any read or write sequence. The format for MRADS data transfer instructions is:

Word Two



Bit Positions	Contents
0 through 4	Operation code; read = octal 12 write = octal 37
5 through 7	File number, 0 through 3
8 through 11	Reserved for minor operation codes, described in Special Bit Configurations
12 through 14	Must be zero
15 through 19	Number of records to be transferred, 1 through 16

Word Three



Bit Positions	Contents
0 through 4	Must be zero
5 through 13	Starting location for read or write operation. Must be multiple of 64.
14 through 19	Must be zero, because origin address is multiple of 64.

Special Bit Configurations

Normally, bit positions 8 through 11 of word two of the read and write instructions contain zeros. However, there are special bit characteristics for these positions that are valuable for checking and timing. These are described below.

READ-AFTER-WRITE PARITY CHECK - BIT POSITION 9

A read-after-write parity check can be performed by the controller if bit 9 in the read instruction is set to 1. If an error should occur, it may be detected under program control. During this check, no data (including the 65th check word) is transferred to memory. Further, the controller is busy until the parity check is completed. When the controller becomes ready, the program must check the parity error branch, and if an error occurs, the record should be rewritten. This check requires a minimum of 52 milliseconds.

RAW	N	F	1202000	Word Times: 2
(blank)	zero		0000000	

Functional Description: READ AFTER WRITE CHECK. N is the number (1-16) of 64-word records to be checked. F is the number (0-3) of the MRADS unit selected. The second command word of this read instruction contains zeros. A seek command (SEL, PRF, OCT) must precede any read or write commands.

Comments: GAP II will assemble the mnemonics for dropping power and for the read-after-write check. Thus, if GAP II is used it is not necessary to set these bits.

This command enables a parity check to be made on all words of a record(s) transferred to the file. The check

is made after the data has been transferred to the controller, but no transfer of data is made to the central processor.

POSITION RELEASE - BIT POSITION 10

Bit positions 8 through 11 of the read and write instructions are normally zero. When bit position 10 is zero, power remains on for the actuator being used and the arm position is held. Thus, it is possible to read a record, update it, and write it back into random access memory in the same location without consuming time for positioning, although latency time must still be considered. The RRF and WRF, read and write, instructions hold power after the operation is complete. RRD and WRD, read and write, instructions will drop power after the operation and the position will not be retained.

There are other advantages to holding power. For example, positioning time can be saved when it is known that the next operation can be performed without changing the arm position.

In many cases of purely random processing it is advantageous to drop power on the actuator after reading or writing. This saves 9 ms each time a different arm or position is addressed. When a 1-bit is inserted in position 10 of the read or write instructions, power is turned off and the position is not held after the operation has been completed.

READ NEXT SECTOR - BIT POSITION 19

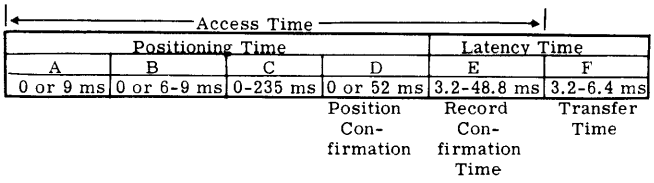
Bit 19, READ NEXT SECTOR, of the OCT instruction, word 3 of the positioning instructions, is a valuable tool. When set to zero, only the specific record indicated by the instruction is read. However, when the bit is set to 1, the sector part of the instruction is ignored, the specified head is selected, and the next record to come under the read-write head is read into memory. The entire track can be read into memory by so indicating in the read instruction. For example, if it is desired to read eight records from head 0 and the first record read is three, the transfer to memory is in the sequence: 3, 4, 5, 6, 7, 0, 1, 2. Under similar circumstances, with bit 19 set to zero (normal) the sequence would be 3, 4, 5, 6, 7, 8, 9, 10. In this case, head switching is accomplished automatically. This cuts latency time almost to zero.

Bit 19 is useful for dumping disc storage onto magnetic tape or cards for file protection or sorting. Since the data in disc storage is in random sequence, the sequence of dumping the data onto magnetic tape before sorting is not important. (This assumes that one record contains a unit or several units of information.) The savings with this type of dump routine would be the average latency time of 25 milliseconds per track. At the lower tape speeds this would not be of particular advantage as the dump is tape bound.

MRADS OPERATING TIMES

Access Time

Access time to a record stored in a disc file is defined as the time from the initiation of a seek command to the time when the first data bit in the record is read. This assumes that no delays are incurred because of the computer program. Figure 11-6 illustrates the major components of total access time. The numbers below each letter represents milliseconds (ms) of time.



A. A Power Off Delay is incurred when the seek command calls for a new position or arm. This delay will be incurred if power was not dropped after the last read or write command, or power was dropped after the last read or write command and a new seek command was received before power had been dropped.

B. A Power On Delay is incurred any time an arm (with power off) must be moved to a specified position.

C. Travel Time. (Basically, this depends upon the starting position of the arm, the direction of movement, and the number of positions to be moved.)

D. Position Confirmation Time is the amount of time required to verify that the head is ready to read or write on the correct track.

E. Latency Time is the rotational delay incurred from completion of address confirmation until the proper record is available.

Figure 11-6. MRADS Timing

Positioning Time

Positioning time is A + B + C + D (Figure 11-6). However, it may be zero. See Case 2 below.

Relationship Between Commands and Hardware

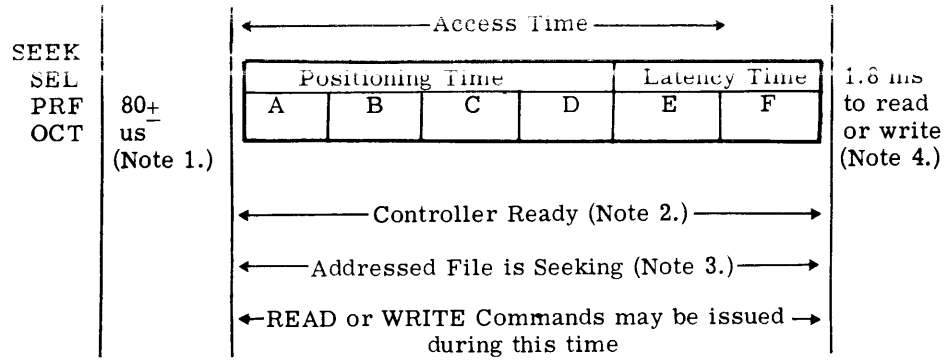
To minimize access time, read or write instructions should be issued as soon as the controller becomes ready, following the issuance of the seek instruction. The controller holds the instructions and automatically instructs the file to read or write upon receiving the file ready signal. The following cases illustrate possible conditions that may occur in connection with access time. In all cases, assume that the controller is ready.

Case 1 - There are no commands held by the controller and power is not applied to any actuator. In this case, as soon as a seek command is received by the controller and an address is sent to the file, power is brought up on the actuator and positioning starts. Thus, arm positioning time is B + C + D. If there are no file errors during this time, after the record address has been confirmed, the file will signal the controller 'file ready.' A read or write command may be issued to the controller any time after the seek command is transferred and when controller ready (80 microseconds) occurs.

Case 2 - A read or write operation has just been completed and the instruction executed did not cause actuator power to be turned off. In this case, if a new seek command specifies a record on the same disc which does not require movement of the arm, there is no positioning time. The only delay is due to latency time. However, if a new disc or new position is addressed, the positioning time is A + B + C + D, since power must be dropped on the actuator and then brought up for the new position.

Case 3 - Seek and read instructions have been sent to the controller, observing proper programming procedures, but a file error occurs prior to completion of time period D. In this case, the file will not send a ready pulse to the controller. However, the file error and controller ready branch condition will be set. It is up to the program to interrogate and initiate error recovery procedures.

The diagram in Figure 11-7 summarizes the interrelationship of commands and the physical units.



The controller becomes busy when a read or write instruction is given, and remains busy until the operation is completed.

Notes:

1. During this time, the seek instruction goes to the controller, and the controller becomes ready after 80 microseconds.

2. Seeks may be overlapped if more than one unit is in the configuration.

3. A read or write command (or, if desired, a new seek) may be given during this period. The unit will signal the controller when it is ready. If read or write commands are given during access time, the controller will hold either until the unit signals that it is ready.

4. When the unit is ready to read or write and instructions have not yet been issued, 1.8 milliseconds are available to issue the commands. If the issuance of the command is delayed beyond the 1.8 millisecond period, another 52 milliseconds will elapse before reading or writing occurs. Editing, etc, may be accomplished during positioning time. If, under program control, it is found to be undesirable to read or write a record for which a seek command has been issued, a new seek instruction for a specific unit may be issued.

Figure 11-7. MRADS Instruction Timing

MRADS CONTROLLER TEST-AND-BRANCH INSTRUCTIONS

The following branch instructions test the controller to determine whether a particular MRADS condition is true or false. If the condition tested is true, the computer executes the next sequential instruction. If the condition tested is false, the computer skips the next instruction and executes the second sequential instruction. The controller must be 'ready' before any other branch condition is tested.

BCS BRN P 2516P20 Word Times: 2

Functional Description: BRANCH ON MRADS CONTROLLER NOT READY. The MRADS controller on Plug P is tested for a not-ready condition.

GAP Coding:

Symbol	Opr	Operand	X
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	B C S	B R N	0
	B R U	* - 1	

The above example is typical of programming all MRADS test and branch instructions.

BCS BRR P 2514P20 Word Times: 2

Functional Description: BRANCH ON MRADS CONTROLLER READY. The MRADS controller on Plug P is tested for the ready condition.

BCS BIO P 2514P25 Word Times: 2

Functional Description: BRANCH ON INPUT/OUTPUT ERROR. The MRADS controller on Plug P is tested for input/output error indicator on.

BCS BIC P 2516P25 Word Times: 2

Functional Description: BRANCH ON INPUT/OUTPUT CORRECT. The MRADS controller on Plug P is tested for the input/output error indicator off.

BCS RPE P 2514P26 Word Times: 2

Functional Description: BRANCH ON PARITY ERROR. The MRADS controller on Plug P is tested for parity error indicator on.

BCS RPC P 2516P24 Word Times: 2

Functional Description: BRANCH ON PARITY CORRECT. The MRADS controller on Plug P is tested for parity error indicator off.

BCS BER P 2514P27 Word Times: 2

Functional Description: BRANCH ON ANY ERROR. The MRADS controller on Plug P is tested for error indicator on. These conditions are:

1. Illegal command for specific controller
2. Input/output error
3. Parity error
4. Any type of file error

BCS BNE P 2516P27 Word Times: 2

Functional Description: BRANCH ON NO ERROR. The MRADS controller on Plug P is tested for error indicator off.

MRADS UNIT TEST-AND-BRANCH INSTRUCTIONS

BCS FKR P 2514P21 (File 0) or 2514P22 (File 1) or 2514P23 (File 2) or 2514P24 (File 3) Word Times: 2

Functional Description: BRANCH ON FILE K READY. The specified MRADS unit K on controller Plug P is tested for ready status. K in the mnemonic is replaced by the file number 0, 1, 2, or 3, and may be program modified. Ready indicates that a seek command has been executed and that the file is ready to read or write.

BCS FKN P 2516P21 (File 0) or 2516P22 (File 1) or 2516P23 (File 2) or 2516P24 (File 3) Word Times: 2

Functional Description: BRANCH ON FILE K NOT READY. The specified MRADS unit K on controller Plug P is tested for a not-ready status. K in the

mnemonic is replaced by the file number 0, 1, 2, or 3, and may be program modified. Ready indicates that a seek command has been executed and that the file is ready to read or write.

BCS	FAE	P	2514P31	Word Times: 2

Functional Description: BRANCH ON ERROR - ON ANY FILE. The MRADS units on controller Plug P are tested for file error indicator on. If any file has an error, this condition will be set and can be tested with this instruction.

BCS	FAC	P	2516P31	Word Times: 2

Functional Description: BRANCH ON NO ERROR - ANY FILE. The MRADS units on controller Plug P are tested for file error indicator off. If any file has an error, the condition will be set and can be tested with this instruction.

BCS	FKE	P	2514P32 (File 0) or 2514P33 (File 1) or 2514P34 (File 2) or 2514P35 (File 3)	Word Times: 2

Functional Description: BRANCH ON FILE K, FILE ERROR. The specified MRADS unit on controller Plug P is tested for a file error. K in the mnemonic is replaced by file number 0, 1, 2, or 3, and may be program modified.

BCS	FKC	P	2516P32 (File 0) or 2516P33 (File 1) or 2516P34 (File 2) or 2516P35 (File 3)	Word Times: 2

Functional Description: BRANCH ON UNIT K, NO UNIT ERROR. The specific MRADS unit K on selector Plug P is tested for no file error. K in the mnemonic is replaced by file number 0, 1, 2, or 3. K may be program modified.

NOTE: When a file error is set, it will remain set until a seek instruction for that specific file is issued. (For instance, to clear an error on File 0, a seek instruction for File 0 must be given.) As long as any file holds an error, the FAE and FAC branches will indicate that an error exists.

MRADS PROGRAMMING SEQUENCE

Before attempting to execute a MRADS operation, the programmer must test the MRADS controller for a ready status and for any error condition resulting from a previous operation. Figure 11-8 is a flow chart illustrating the proper programming sequence. This sequence of instructions is necessary for every read or write operation regardless of whether or not a new arm position is required. There is no rule requiring that the ready branch following a seek command be executed within any given length of time. It is also good programming practice to check for 'any error' after read or write operations. If 'any error' occurs, the read or write operation should be repeated.

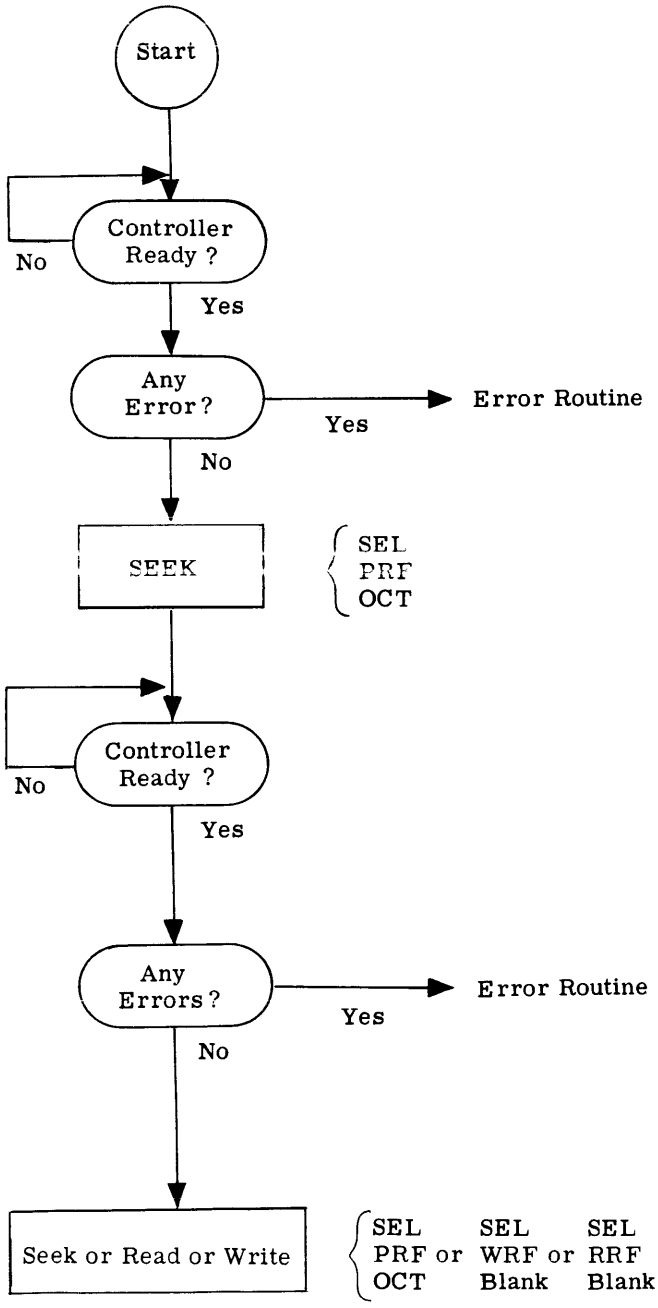


Figure 11-8. MRADS Programming Sequence

GAP Coding:

```
Opr     Operand     X   REMARKS
        8  9  10  12 13 14 15 16 17 18 19  20  31                    75
BCS     BRN         0   CHECK FOR READY STATUS
BRU     *-1
BCS     BER         0
BRU     ERROR           BRANCH TO ERROR ROUTINE
```

If the controller is ready and there are no errors, the first set of commands to be executed will initiate a seek operation on the specified file.

GAP Coding:

```
Opr     Operand     X   REMARKS
        8  9  10  12 13 14 15 16 17 18 19  20  31                    75
SEL     0               SELECT PLUG 0
PRF     1               POSITION MRADS UNIT
OCT     0521436
```

Following the transfer of the seek command to the controller, another 'Branch on Controller Ready' command should be programmed. This delay is approximately 80 microseconds.

GAP Coding:

```
Opr     Operand     X   REMARKS
        8  9  10  12 13 14 15 16 17 18 19  20  31                    75
BCS     BRN         0
BRU     *-1             DELAY FOR COMPLETION OF OPERATION
```

Any time after the controller is ready, the program may either initiate a seek on another disc file, transfer a read or write command to the file already selected, or alter the previous seek command.

GAP Coding:

```
Opr     Operand     X   REMARKS
        8  9  10  12 13 14 15 16 17 18 19  20  31                    75
SEL     0
RRF     1           1   READ RECORD FROM MRADS
        64
```

MRADS PROGRAMMING EXAMPLES

Example 1

Read a record from MRADS unit 1; MRADS 1 is plugged into controller selector Plug 0. The record is to be read into symbolic memory location RAMIN. The record is located at disc 10, track position 35, and record 15.

GAP Coding:

```
    Opr     Operand     X   REMARKS
            8  9  10  12 13 14 15 16 17 18 19  20  31                    75
1   BCS     BRN         0   DELAY UNTIL CONTROLLER IS READY
2   BRU     *-1
3   SEL     0               SELECT PLUG 0
4   PRF     1               POSITION MRADS UNIT 1
5   OCT     0521436
6   BCS     BRN         0   DELAY UNTIL CONTROLLER IS READY
7   BRU     *-1
8   SEL     0
9   RRF     1           1   READ RECORD FROM MRADS
10          RAMIN
11  BCS     BRN         0   DELAY UNTIL READ IS COMPLETED
12  BRU     *-1
13
```

The OCT command of line 5 is constructed as shown below:

Bit Position	0←→5	6←→11	12←→18	19
Address	Disc	Track	Record	RNS
Bits	001010	100011	0001111	0
Octal Code	0 5	2 1	4 3	6

Example 2

Using the data of Example 1, assume that all address data is known except the record number. Read the record within range of access on the same file, disc, and track. Assume that the record to be read will furnish further information for the 'search' and further processing will interrogate the data within the record itself.

GAP Coding:

```
Opr     Operand     X   REMARKS
        8  9  10  12 13 14 15 16 17 18 19  20  31                    75
BCS     BRN         0   CHECK FOR CONTROLLER READY
BRU     *-1
SEL     0
PRF     1               POSITION MRADS UNIT 1
OCT     0521401
BCS     BRN         0
BRU     *-1
SEL     0
RRF     1           1   READ RECORD FROM MRADS
        RAMIN
BCS     BRN         0
BRU     *-1
```

The preceding coding is identical with that of Example 1, except for the address of the OCT instruction. This address is constructed as follows.

Bit Position	0←→5	6←→11	12←→18	19
Address	Disc	Track	Record	RNS
Bits	001010	100011	0000000	1
Octal Code	0 5	2 1	4 0	1

Example 3

With reference to Example 1, write a record into the MRADS unit from symbolic location RAMIN. Assume the MRADS has been positioned at the desired address by a seek command.

GAP Coding:

Opr	Operand	X	REMARKS
B C S	B R N	0	
B R U	* - 1		
S E L	0		
W R F	1	1	WRITE RECORD ON MRADS UNIT 1
	R A M I N		
B C S	B R N	0	
B R U	* - 1		

MRADS WITH AUTOMATIC PROGRAM INTERRUPT (API)

The MRADS controller contains a manual switch which may be used during normal programmed operation, if the computer is equipped with the Automatic Program Interrupt feature. In the off position, interrupt pulses will not be transmitted. If the switch is on, interrupt can occur as described below.

The system design will determine whether an MRADS program is to be run in conjunction with another 'main' program. If this should be the case, with the switch on (and depending upon the manner in which the interrupt is programmed) automatic program interrupt will occur under the following circumstances:

1. A read or write operation has been completed.
2. A seek instruction has been completed and the file is ready to read or write. If the controller is holding a read or write instruction, no interrupt will be provided.
3. A file error is received during the seek operation and the controller is holding a read or write instruction for the file that has the error.
4. The controller receives a read or write instruction for a file which is holding an error condition.

SECTION XII

AUXILIARY ARITHMETIC UNIT OPERATIONS

The addition of the Auxiliary Arithmetic Unit (AAU) extends the arithmetic capability of the GE-225 system; it is particularly useful in applications where numerous floating-point or double word calculations are required. The AAU processes floating-point arithmetic at much higher speeds than is possible when using a programmed floating-point package. The latter method is used only when a system does not include an AAU. Floating-point word format is the same in either case, however.

AUXILIARY ARITHMETIC UNIT

Although the AAU is connected through the controller selector to the central processor, it is not an input/output device; it is an extension of the basic arithmetic unit. By routing access to the AAU through connector plug 7 of the controller selector and the automatic program interrupt controls, simultaneous AAU and central processor arithmetic unit operations are possible. Due to its control logic, the AAU can be connected to controller selector plug 7 only.

Function

The central processor is readily adaptable to computing ranges of numbers with fixed decimal points, such as whole numbers or whole numbers with decimal fractions (example: dollars and cents, where the fractional portion of the number is always two decimal digits). When the calculations involve numbers that have varying format, such are often found in scientific calculations, the programmer must keep track of the location of the radix point. This can be laborious and can easily lead to errors. The AAU enables the programmer to use floating-point arithmetic with both ease and speed, thereby avoiding most scaling considerations. In some respects, the AAU can be thought of as an extension of the central processor with larger registers, thereby permitting calculations of greater complexity than does the arithmetic unit alone.

Modes

Three modes of calculations are performed by the AAU: unnormalized floating-point, normalized floating-point, and fixed-point operations. It is usual to do floating-point operations in the normalized mode. A normalized number is one in which the most significant non-zero bit of the mantissa is not preceded by leading zeros. For example, the decimal number 6786 might be represented in unnormalized form as:

$$.006786 \times 10^6$$

In normalized form, this number would be:

$$.6786 \times 10^4$$

Addition, subtraction, multiplication, and division can be done in any of the three modes of operation. Special commands, discussed later in this section, enable the execution of the desired type of calculation.

Controller

All AAU operations are conducted through the AAU controller. However, unlike controller selector peripherals, the controller and auxiliary arithmetic unit are not two separate physical units. They are one unit with all normal controller-type functions being performed from the AAU to the central processor core memory through the controller selector. All instructions for the AAU are routed through the AAU controller logic which decodes and executes the desired operations. A GE-225 system can have only one AAU connected to the controller selector.

Priority

The AAU is granted access to memory according to its priority level in the controller selector priority interrupt system. This level is pre-established by means of convenient plug-in connectors to the controller selector. The AAU occupies plug 7 which is the station with the lowest priority. Thus, if any peripheral with

a higher level priority desires access to memory, the AAU must wait. Through this priority interrupt system of time sharing, the AAU decodes and executes its instructions and permits calculations to occur simultaneously with central processor computations.

All communication with the central processor core memory is performed on a priority basis with two exceptions: (1), when the AAU controller receives a new instruction during the execution of a current instruction, and (2) when the AAU receives a floating-point divide instruction and the dividend in the AX and QX registers is negative. These two conditions are discussed later in this section.

Registers Affected

All AAU operations process data in the AAU after data passes through the central processor M register, where each 20-bit word is checked for parity. The AAU has an instruction register to hold AAU instructions, a 40-bit buffer register which accepts two 20-bit words from the central processor to form one 40-bit AAU word, a 40-bit adder, and two other 40-bit registers, known as the AX and the QX registers whose function is similar to the A and Q registers in the central processor (these registers are described under AAU Operating Logic). Note that the size of these two registers permits both floating-point and fixed-point calculations on larger numbers that could be conveniently processed otherwise.

INSTRUCTION WORDS

Setting the Mode

Unlike the requirements for controller selector peripherals, three instruction words are not needed to initiate operation of the AAU. One reason is because there can be only one AAU in a GE-225 system. Also, because there is no separate controller with a number of operating units like the controller selector peripherals, there is no need to select a plug. As stated, the AAU is always on controller plug 7 and thus no SEL instruction is needed. All that is required is a set mode instruction to select one of the three modes before giving an arithmetic instruction. Following a set mode are the instructions required to perform the desired arithmetic operations.

Once the set mode instruction is given for the desired calculation mode, it need not be given again until the programmer wants to change modes.

To illustrate, Figure 12-1 shows a program which begins calculations in the normalized floating-point mode and then switches to the unnormalized floating-point mode as directed by the new mode instruction shown on line 11.

GAP Coding:

	Symbol	Opr	Operand
1	START	SET	NFLPOINT
2		LDA	X
3		FSU	Y
4		FST	TFM1
5			
6			
7			
8			
9			
10		FST	ADIDO
11		SET	UFLPOINT
12		LDA	X
13			
14			

Figure 12-1. Setting the Calculating Mode

AAU instructions are similar in performance to double word length central processor instructions. For example, the AAU load instruction, FLD 3200, double loads the contents of locations 3200 and 3201 into the 40-bit AX register. As is shown later in this section, AAU instructions are available to use the AX and QX registers together, such as might be required in normalized floating-point mode multiply operations.

Format

Instruction words for the AAU are contained in core memory in one 20-bit word of the same format as central processor instructions. Figure 12-2 shows the word organization of a store AAU (FST 3200) instruction.

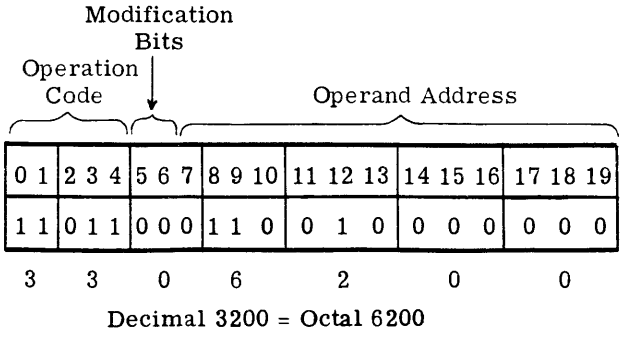


Figure 12-2. AAU Instruction Word Format

Bits 0 through 4 contain the operation code; bits 5 and 6 contain the automatic modification bits, if required; bits 7 through 19 contain the address of the operand.

As in the case of the central processor, the floating-point instructions are available to the user through the use of mnemonics which are listed in the instruction repertoire. The use of any of the AAU instructions for floating-point operations assumes that the operands to be acted upon are already in floating-point format. Note: information can be placed in floating-point format using existing subroutines.

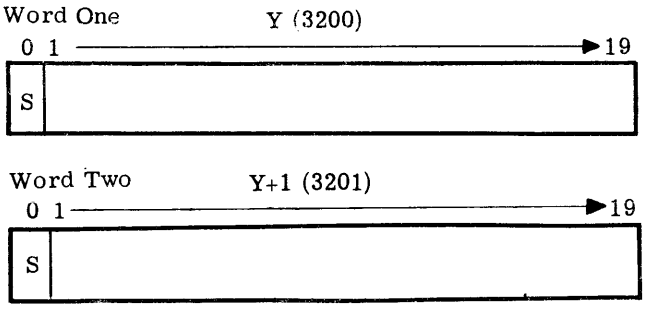
DATA WORD FORMAT

Data for AAU operations can exist in memory in any of three different modes; fixed-point, normalized floating-point and unnormalized floating-point. All AAU data words exist in the central processor core memory as two 20-bit words, with bits of each word having meaning according to the mode selected previously. Thus, when an instruction such as load the AAU with the contents of location 3200 (FLD 3200) is received and executed by the AAU, the contents of 3200 and 3201 are brought into the AAU. The format in which the contents of 3200 and 3201 are interpreted depends upon the mode in which the AAU is operating.

Fixed-Point Words

Fixed-point is a mode of operation whereby the binary information in a register (AX or QX) is decoded in standard binary; that is, each bit has a binary value. The least significant bit, bit 19 of the first data word, has a binary value of 0 or 1, bit 18 has a binary value of 2, bit 17 has a value of 4, bit 16 has a value of 8, etc.

In Memory:



In the AAU AX Register:

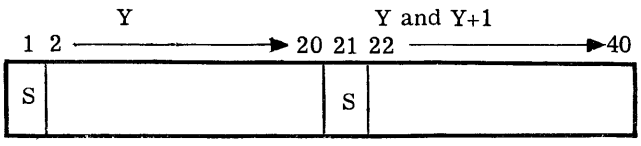


Figure 12-3. Fixed-Point Data Word Format.

Fixed-point information can be of the form as shown below:

62487 (integer)
.62487 (fraction)
62.487 (mixed number)

The format for fixed-point words in memory is shown in Figure 12-3. To illustrate, the FLD 3200 instruction is used. Note that the signs of word one and word two are identical for fixed-point double words. Thus, when two data words from memory enter the AAU, they appear in the AX register as one 40-bit word. As shown, the fixed-point word in the AX register consists of 38 information bits, plus two identical sign bits. Similarly, a fixed-point word in the QX register also consists of 38 information bits and 2 sign bits. Note that bit one is the sign bit of the entire word. Bit 21 (the sign of the mantissa in floating-point format) has no significance in fixed-point operation.

Floating-Point Words

A floating-point number consists of two parts, an exponent and a mantissa. For clarification, it is helpful to review some of the basic terms used in floating-point arithmetic operations.

1. Exponent. As used with the AAU, the exponent (or characteristic) is the nine most-significant bits (eight numeric bits and a sign bit) of a double word (see Figure 12-4). These bits designate to what power of two the mantissa portion of the word must be raised.

2. Mantissa. In the AAU, the mantissa is the 30 least-significant bits of a double word. The radix point for these 30 bits is assumed to be to the left of the most significant of the 30 bits. Thus, the mantissa is fractional in value (see Figure 12-4).

The mantissa is multiplied by two raised to the exponential power expressed in bits 0 through 8 to give the entire word the desired numeric value.

3. Normalization. In the AAU, positive and negative numbers are normalized, or adjusted, so that the mantissa lies in the prescribed range; the absolute value of the mantissa must be greater than (or equal to) 1/2 and less than (or equal to) 1. Algebraically, this is expressed as: $1/2 \leq |\text{mantissa}| \leq 1$.

Positive numbers are normalized by shifting the mantissa left until its most significant bit (in the AX register) is not a 0-bit. For each position shifted left, one is subtracted from the exponent.

Negative numbers (in 2's complement form) are normalized by shifting the mantissa left until its most significant bit is a 0-bit, or until the most significant bit is a 1-bit and all other mantissa bits are 0-bits. The exponent is then adjusted by the number of shifts.

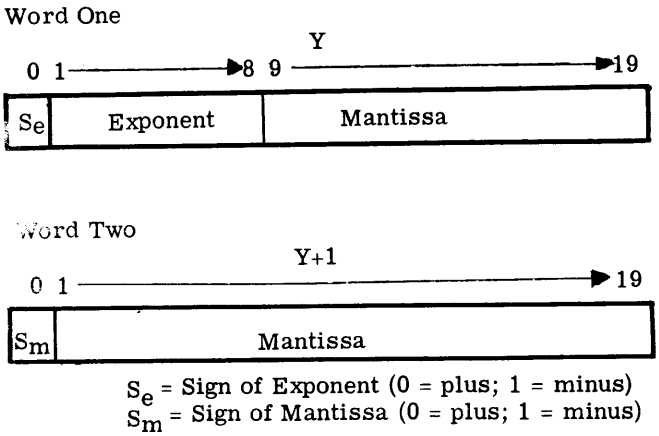
The mentioned significant 1-bit in the mantissa is a special condition which exists only when the mantissa of a negative number is all zeros. In such cases, the most significant bit is a 1-bit and the exponent is adjusted so that the value in the AX register is unchanged.

4. Radix Point. The point in any numbering system which separates the whole integers from the fraction. Thus, the decimal point is a radix point for the decimal system; a binary point is the radix point for the binary system.

Floating-point numbers occupy two 20-bit words in memory, as shown in Figure 12-4. The binary point is assumed to be before the first bit (bit 9) of the mantissa. This format produces a binary number with a 30-bit mantissa and a binary characteristic range of -256 to +255. This is approximately equal to a decimal number with a 9-digit mantissa and a decimal range of 10^{-77} to 10^{+77}. The fact that two words are used allows one of the sign positions to be applied to the exponent.

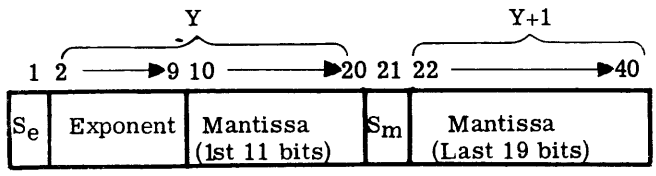
When data in the two-word floating-point format enters the AAU, it is converted into one word, 40 bits long, as shown.

In Memory:



S_e = Sign of Exponent (0 = plus; 1 = minus)
S_m = Sign of Mantissa (0 = plus; 1 = minus)

In the AAU:



S_e = Sign of Exponent
S_m = Sign of Mantissa

Figure 12-4. Floating-Point Data Word Format

The QX register is an extension of the AX register. It also consists of an 8-bit exponent with sign and a 30-bit mantissa with sign. The value of the exponent of the QX register is the value of the AX register exponent minus 30.

EXPONENTIAL ARITHMETIC

To perform arithmetic operations in the floating-point format, several requirements must be met. For addition and subtraction problems, the exponents of the numbers involved must be equal. It is not probable that a common exponent will be used in all problems; however, the AAU automatically adjusts exponents. The AAU adjusts the exponent with the smaller numeric value. Adjustment is accomplished by automatically shifting the mantissa of the word with the smaller exponent right and incrementing its exponent by the number of positions shifted.

Addition and Subtraction

Once exponents are equalized, addition or subtraction of the mantissas can occur. The exponent of the answer will be the adjusted exponent while the mantissa of the answer will be the sum or difference of the shifted mantissas.

Multiplication

For multiplication, exponents are added and mantissas are multiplied. The resultant exponent in multiplication is the algebraic sum of the original exponents, while the resultant mantissa is the product of multiplying the original mantissas.

Division

In division, the exponent of the divisor is subtracted from the exponent of the dividend, and the mantissa of the dividend is divided by the mantissa of the divisor. The resultant exponent is the algebraic difference between the dividend and the divisor exponents. The resultant mantissa is the result of the algebraic division of the dividend mantissa by the divisor mantissa. In summary, floating-point division causes the subtraction of exponents and division of mantissas.

Overflow and Underflow

Floating-point arithmetic operations can result in overflow or underflow during both normalize and unnormalize modes.

Overflow occurs if the exponent of a partial or final result in the AX register exceeds $+377_8$ ($+255_{10}$).

Underflow occurs when the exponent of a partial or final result in the AX register becomes less than -400_8 (-256_{10}).

Overflow or underflow can result at any of these times:

1. During the formation of the initial estimate of the result exponent.

2. During a right shift one and add one as a result of mantissa overflow.

3. During the normalization of a result (normalize mode). Recall that normalization involves shifting the mantissa left N places and adding N to the exponent, possibly causing overflow, N being the number of leading zeros in a positive mantissa or the 2's complement of the negative mantissa.

FAD Instruction. Overflow or underflow can occur during events 2 and 3, above, never during event 1.

FSU Instruction. Overflow or underflow can occur during events 2 and 3, above, never during event 1.

FMP Instruction. Overflow or underflow can occur during events 1 and 3, above. Overflow during event 2 could only occur if both operands were the maximum negative values (this is illegal and results in undetected errors).

FDV Instruction. Overflow or underflow can occur during events 1 and 3. Also, overflow can occur during event 2 if Am $\geq$ BM and AM/2 $<$ Bm, where Am = the dividend mantissa and Bm = the divisor mantissa.

AAU OPERATING LOGIC

Before AAU operations can be performed, the appropriate instruction such as load the AAU (FLD) is sent to the AAU control circuitry, once the AAU is granted access to memory. This instruction is stored in the AAU 20-bit instruction register (Figure 12-5).

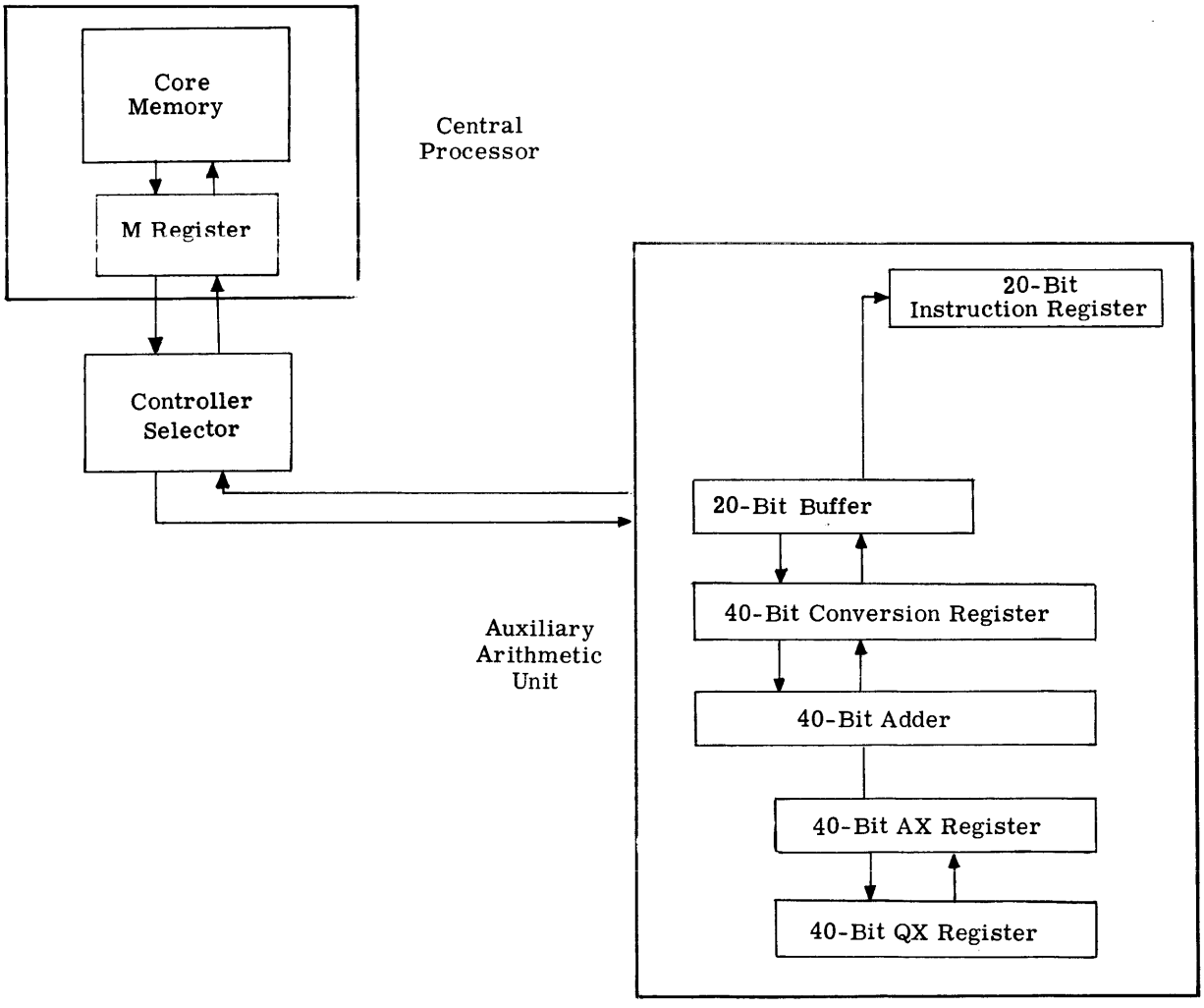


Figure 12-5. AAU Register Operations

Priority

As discussed earlier, all communication with central processor core memory is performed through the AAU when, under the priority interrupt system, it is granted access to memory. There are two basic exceptions:

1. Once the AAU is given an arithmetic function to perform, the AAU operates independently of the central processor. If a second instruction is given to the AAU before the first instruction is completed, the AAU prevents the computer program from advancing by removing computer priority through the controller selector.

2. The normal priority arrangement also is interrupted if the AAU receives a floating-point divide instruction and the dividend in the AX and QX registers is negative. During the request for priority, the 2's complement of the dividend is formed in the AX and QX registers.

At the elimination of either of these two conditions, priority is returned to the central processor so that it may complete its function with respect to the current operation. Note that, when priority is taken away from the computer, it is also taken away from the card punch. However, the AAU can not monopolize priority to the extent that card punch operations are affected. Also, a unit of higher priority can override a priority request from the AAU.

Registers

Once the AAU is granted access to memory, two data words are brought from memory through the central processor M register and checked for parity. As shown in Figure 12-5, each data word first enters a 20-bit buffer and then the 40-bit conversion register where the 40-bit double word format of the AAU is formed. The format of data in this register is pre-determined by a previous set mode instruction.

The function of the conversion register is similar to that of the B register in the central processor; it acts as a data link for words being transferred to the 40-bit AX register.

The 40 bits from the conversion register enter the 40-bit adder which performs essentially the same arithmetic operations as the arithmetic unit of the central processor. Another input to the adder is from either the 40-bit AX or the QX register. Within the adder, arithmetic operations are performed which, in the case of floating-point mode calculations, might include carries from the mantissa portion of a data word into the exponent portion. In calculations, the AX and QX registers play the same functional role as the A and Q registers in the central processor. Thus, in the floating-point mode, for example, mantissa bits shifted out of AX are shifted into QX.

After the designated calculations are completed, and the AAU again gains access to memory, the results are transferred out of the AAU, through the controller selector, and stored in memory as programmed.

Subroutines

Subroutines are required initially to create floating-point words from BCD data or create BCD words from floating-point format. This conversion from one form to another is possible using GE-225 utility routines.

AAU INSTRUCTIONS

Mode Control Instructions

These instructions establish the mode in which subsequent AAU arithmetic instructions will be executed. Once a set mode instruction is executed, the AAU will execute all AAU arithmetic instructions in that mode until the mode is changed by another set mode instruction. The execution times given are in addition to access time of 43 microseconds (usec).

SET UFLPOINT	3200010	Timing: 49.5 usec

Functional Description: SET UNNORMALIZED FLOATING-POINT MODE. The AAU is set to execute AAU arithmetic instructions in unnormalized floating-point mode.

SET NFLPOINT	3100010	Timing: 49.5 usec

Functional Description: SET NORMALIZED FLOATING-POINT MODE. The AAU is set to execute AAU arithmetic instructions in normalized floating-point mode.

SET FIXPOINT	3500010	Timing: 49.5 usec

Functional Description: SET FIXED-POINT MODE. The AAU is set to execute AAU arithmetic instructions in double-word fixed-point mode.

Data Transfers Within the AAU

These instructions are similar to certain central processor data transfer instructions in that they involve transfers of data between arithmetic registers and are specified by similar mnemonics. However, the AAU data transfer instructions are identified by placing the letter A in the X column (card column 20) of the GAP coding sheet.

AAU error conditions are reset when any AAU data transfer instruction is executed. Execution times are given below in microseconds; to these times add 36 usec for access time. The AAU mode of operation does not affect the execution of these instructions.

LAQ	A	3600002	Timing: 49.5 usec

Functional Description: LOAD AX FROM QX. The contents of the QX register replace the contents of the AX register. The contents of the QX register are unchanged.

MAQ	A	3100002	Timing: 49.5 usec

Functional Description: MOVE AX TO QX. The contents of the AX register replace the contents of the QX register. The AX register is reset to zero.

LQA	A	3200002	Timing: 49.5 usec

Functional Description: LOAD QX FROM AX. The contents of the AX register replace the contents of the QX register. The contents of the AX register are unchanged.

XAQ	A	3500002	Timing: 117 usec

Functional Description: EXCHANGE AX AND QX. The contents of the AX and QX registers are interchanged.

Data Transfers Between AAU and Memory

For proper execution, instructions causing data transfers between the AAU and memory must specify even effective memory addresses (Y) greater than 0015. Execution times are given below in microseconds; to these times add 72 usec for access time. The AAU mode of operation does not affect the execution of these instructions.

FLD	Y	3000000	Timing: 72 usec

Functional Description: LOAD AUXILIARY ARITHMETIC UNIT. The contents of memory locations Y and Y+1 replace the contents of the AX register. The contents of Y and Y+1 are not changed.

FST	Y	3300000	Timing: 72 usec

Functional Description: STORE AUXILIARY ARITHMETIC UNIT. The contents of the AX register replace the contents of memory locations Y and Y+1. The contents of the AX register are not changed.

AAU Arithmetic Instructions

The modified address in all arithmetic operations must be even and greater than 15 for proper execution. In addition to the execution times listed, 99 usec must be added for access and resynchronization, except for the FDV with a negative dividend (135 usec). Each arithmetic instruction is described once for each of the three operating modes: normalized floating-point, unnormalized floating-point, and double-word fixed-point.

FAD	Y	31YYYYY	Timing: Min. 162 usec Max. 709 usec

Functional Description: NORMALIZED FLOATING-POINT MODE - ADD. The floating-point number in memory location Y and Y+1 is added algebraically to the floating-point number in the AX register. The result is placed in the AX register in normalized form. The contents of Y and Y+1 are unchanged.

UNNORMALIZED FLOATING-POINT MODE - ADD. Execution is the same as above, except that the result is placed in the AX and QX registers and may or may not be in normalized form.

DOUBLE-WORD FIXED-POINT MODE - ADD. The contents of Y and Y+1 are algebraically added to the contents of the AX register. The result is placed in the AX register as a 38-bit fixed-point number. The contents of Y and Y+1 are not changed.

FMP	Y	35YYYYY	Timing: Min. 297 usec Max. 1062 usec

Functional Description: NORMALIZED FLOATING-POINT MODE - MULTIPLY. The floating-point number in memory locations Y and Y+1 is algebraically multiplied by the floating-point number in the QX register. The 60-bit product of the two mantissas is normalized. The most significant half of the normalized product is placed with its exponent in the AX register; the least significant half is placed in the QX register (with the exponent less, by 30, than the floating-point exponent in the AX register). The previous contents of the AX register are destroyed.

UNNORMALIZED FLOATING-POINT MODE - MULTIPLY. Execution is the same as above except that the result may or may not be normalized.

DOUBLE-WORD FIXED-POINT MODE - MULTIPLY. The fixed-point number in memory locations Y and Y+1 is algebraically multiplied by the fixed-point number in the QX register, giving a 76-bit product and 4 identical sign bits. The most significant half of

the product is placed with 2 sign bits in the AX register; the least significant half is placed with 2 sign bits in the QX register. The previous contents of the AX register are destroyed.

FDV	Y	36YYYYY	Timing: Min. 837 usec Max. 1062 usec +168.25 if dividend is neg.

Functional Description: NORMALIZED FLOATING-POINT MODE - DIVIDE. The floating-point number in the AX and QX registers is automatically divided by the floating-point number in memory locations Y and Y+1. The normalized quotient is stored in the AX register and the remainder (which may or may not be normalized) is stored in the QX register. A Divide Check Error condition will occur if the absolute value of the mantissa in the AX register is equal to or greater than twice the absolute value of the mantissa in the divisor. If the contents of Y and Y+1 are zero, the division is invalid and a Divide Check Error condition occurs.

Timing: Min. 814.5 usec
Max. 837.25 usec
+168.25 if dividend is neg.

UNNORMALIZED FLOATING-POINT MODE - DIVIDE. Execution is the same as above, except that the quotient is stored in the AX register and may or may not be in normalized form.

Timing: Min. 1017 usec
Max. 1095 usec
+168.25 if dividend is neg.

DOUBLE-WORD FIXED-POINT MODE - DIVIDE. The contents of the AX and QX registers is divided by the contents of memory locations Y and Y+1. The quotient is stored in the AX register and the remainder is stored in the QX register. The magnitude of the divisor in memory locations Y and Y+1 must be greater than the absolute value of the dividend in the AX register. If not, a Divide Check Error condition occurs. If the divisor is zero, the division is invalid and a Divide Check Error condition occurs.

FSU	Y	32YYYYY	Timing: Min. 162 usec Max. 709 usec

Functional Description: NORMALIZED FLOATING-POINT MODE - SUBTRACT. The floating-point number in memory locations Y and Y+1 is subtracted algebraically from the floating-point number in the AX register. The result is placed in the AX and QX registers in normalized form. The contents of Y and Y+1 are unchanged.

UNNORMALIZED FLOATING-POINT MODE - SUBTRACT. Execution is the same as above, except that the result is placed in the AX register in unnormalized form.

DOUBLE-WORD FIXED-POINT MODE - SUBTRACT. The contents of Y and Y+1 are algebraically subtracted from the contents of the AX register. The result is placed in the AX register as a 38-bit fixed-point number. The contents of Y and Y+1 are unchanged.

AAU Test-and-Branch Instructions

All AAU test-and-branch instructions conform to the following format:

Operation	Operand	P
BAR	XXX	7

The mnemonic XXX identifies the specific test to be performed. If the condition tested is true, the computer executes the next sequential instruction. If the condition tested is false, the next instruction is skipped and the second sequential instruction is executed.

Before any test (except tests for ready status) can be correctly made, any previous AAU instruction must have been completed. This can be assured by testing the AAU for ready status before testing for specific conditions. Either the Branch on AAU Not Ready (BAR BAN) or the Branch on AAU Ready (BAR BAR), described below, can be used for this preliminary test.

All other AAU instructions are correctly executed without being preceded by a ready test because the AAU will not accept arithmetic or data transfer instructions until a previous instruction is completed.

BAR	BAN	7	2516720	Word Times: 2

Functional Description: BRANCH ON AAU NOT READY. The AAU is tested to determine if it is not ready to accept another instruction.

BAR	BAR	7	2514720	Word Times: 2

Functional Description: BRANCH ON AAU READY. The AAU is tested to determine if it is ready to receive another instruction.

BAR BPL 7 2516721 Word Times: 2

Functional Description: BRANCH ON AAU PLUS. The AX register is tested for a plus sign.

BAR BMI 7 2514721 Word Times: 2

Functional Description: BRANCH ON AAU MINUS. The AX register is tested for a minus sign.

BAR BZE 7 2514722 Word Times: 2

Functional Description: BRANCH ON AAU ZERO. The AX register is tested for all zero content.

BAR BNZ 7 2516722 Word Times: 2

Functional Description: BRANCH ON AAU NOT ZERO. The AX register is tested for non-zero content.

BAR BOV 7 2514723 Word Times: 2

Functional Description: BRANCH ON OVERFLOW. The AAU is tested for overflow indicator ON.

BAR BNO 7 2516723 Word Times: 2

Functional Description: BRANCH ON NO OVERFLOW. The AAU is tested for overflow indicator OFF.

BAR BUF 7 2514724 Word Times: 2

Functional Description: BRANCH ON UNDERFLOW. The AAU is tested for underflow indicator ON.

BAR BNU 7 2516724 Word Times: 2

Functional Description: BRANCH ON NO UNDERFLOW. The AAU is tested for underflow indicator OFF.

BAR BER 7 2514727 Word Times: 2

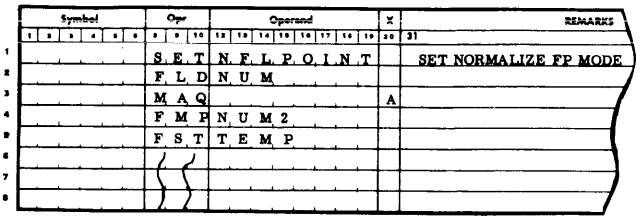
Functional Description: BRANCH ON ERROR. The error indicator is tested for ON.

BAR BNE 7 2516727 Word Times: 2

Functional Description: BRANCH ON NO ERROR. The error indicator is tested for OFF.

Example 1: Normalized floating-point operation with the AAU.

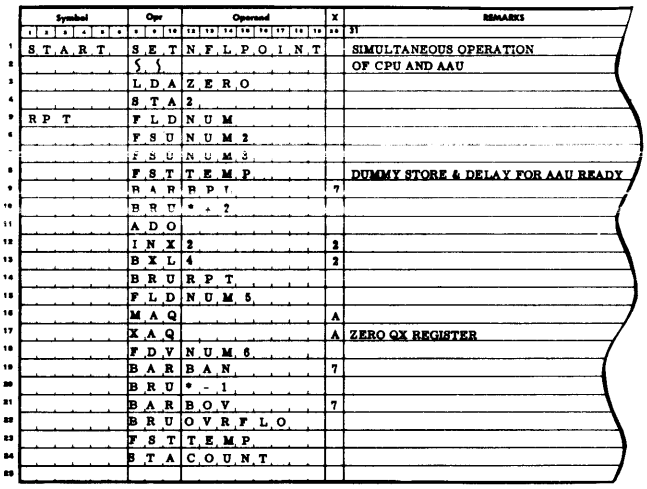
GAP Coding:



	Symbol	Opr	Operand	X	REMARKS
1		SET	NFLPOINT		SET NORMALIZE FP MODE
2		FLD	NUM		
3		MAQ		A	
4		FMP	NUM2		
5		FST	TEMP		
6					
7					
8					

Example 2: Normalized floating-point with simultaneous central processor and AAU operations.

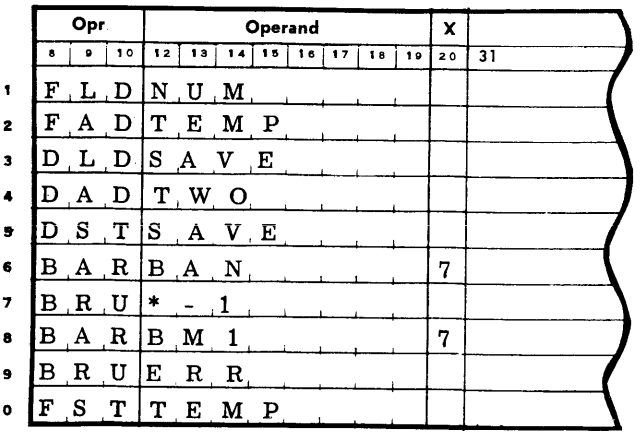
GAP Coding:



	Symbol	Opr	Operand	X	REMARKS
1	START	SET	NFLPOINT		SIMULTANEOUS OPERATION
2		,,			OF CPU AND AAU
3		LDA	ZERO		
4		STA	2		
5	RPT	FLD	NUM		
6		FSU	NUM2		
7		FSU	NUM3		
8		FST	TEMP		DUMMY STORE & DELAY FOR AAU READY
9		BAR	RPT	7	
10		BRU	*+2		
11		ADO			
12		INX	2	2	
13		BXL	4	2	
14		BRU	RPT		
15		FLD	NUM5		
16		MAQ		A	
17		XAQ		A	ZERO QX REGISTER
18		FDV	NUM6		
19		BAR	BAN	7	
20		BRU	*-1		
21		BAR	BOV	7	
22		BRU	OVRFLO		
23		FST	TEMP		
24		STA	COUNT		
25					

Example 3: Test AAU results for minus and show delay necessary.

GAP Coding:



	Opr	Operand	X
1	FLD	NUM	
2	FAD	TEMP	
3	DLD	SAVE	
4	DAD	TWO	
5	DST	SAVE	
6	BAR	BAN	7
7	BRU	*-1	
8	BAR	BM1	7
9	BRU	ERR	
10	FST	TEMP	

Comments: Note the use of the FST TEMP (line 8, Example 2), which is a 'dummy' store to provide a delay until the AAU completes the preceding floating-point instruction before testing for a plus condition. The FST at this point serves no other function than to delay before testing.

In line 17, Example 2, an XAQ A instruction is used to zero the QX register before executing an FDV. If the QX register is not zeroed before division, results may be inaccurate.

SECTION XIII

PROGRAMMING CONVENTIONS

The efficiency of any computer installation depends to a great extent upon proper organization of programming procedures and techniques. This section contains suggestions and lists items that should be considered in establishing installation procedures.

MEMORY LAYOUTS

Many installations have (as standard procedure) allocation of memory areas which all programmers must observe. A few advantages of such a system are:

1. Standardization of input and output, subroutine, constant, and main program areas.
2. Programmer familiarization with the operating program is increased.
3. Changes and modifications are more easily and correctly made.
4. Debugging is accomplished more readily.

Because operating conditions and requirements vary from installation to installation, the memory layout used may be unique and suitable only for that particular installation. A typical layout is shown in Figure 13-1.

Decimal Location	Description
0000 to 0003	Index Registers
0004 to 0127	Optional Index Registers
0128 to 0169	Reserved for Automatic Program Interrupt
0170 to 0255	Miscellaneous Constants or Working Storage
0256 to 0283	Card Read-In Area
0284 to 0383	Miscellaneous Constants or Working Storage
0384 to 0401	Card Read-In Area
0402 to 0511	Miscellaneous Constants or Working Storage
0512 to 0539	Card Punch Area
0540 to 0639	Reserved for Automatic Program Interrupt
0640 to 0719	Printout and Format Areas
0720 to 0839	Magnetic Tape Input and Output Areas
0840 to 1999	Subroutines
2000 to 8191	Main Program

Figure 13-1. Typical Memory Allocation

INPUT/OUTPUT DOCUMENTATION

Proper documentation and layout of input and output data is the responsibility of the programmer; in addition, good documentation is a valuable tool for the programmer, because it enables the programmer to modify or change data with a minimum of effort, debugging is made easier and program operation is possible in less time. Typical forms available are shown in Figures 13-2 through 13-7.

GE 225

MAGNETIC TAPE RECORD LAYOUT

CK - 62

RUN ______________________________ PROGRAMMER ______________

MODE: BCD ______________________________ DATE ______________

BIN ______________________________ PAGE ________ OF ________

SPEC BIN ______________________________

WORD	BIT POSITIONS																			
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0																				
1																				
2																				
3																				
4																				
5																				
6																				
7																				
8																				
9																				
0																				
1																				
2																				
3																				
4																				
5																				
6																				
7																				
8																				
9																				
0																				
1																				
2																				
3																				
4																				
5																				
6																				
7																				
8																				
9																				

Figure 13-2. Magnetic Tape Record Layout

CK 52 (2M 6-61)

RUN: _____ GENERAL ELECTRIC DATE: _____

Computer Department
Phoenix, Arizona

FILE: _____ PROGRAMMER: _____

RECORD TYPE: _____ **GE 225 MAGNETIC TAPE RECORD LAYOUT SHEET** PAGE: _____ OF: _____

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87

Figure 13-3. Magnetic Tape Record Layout Sheet

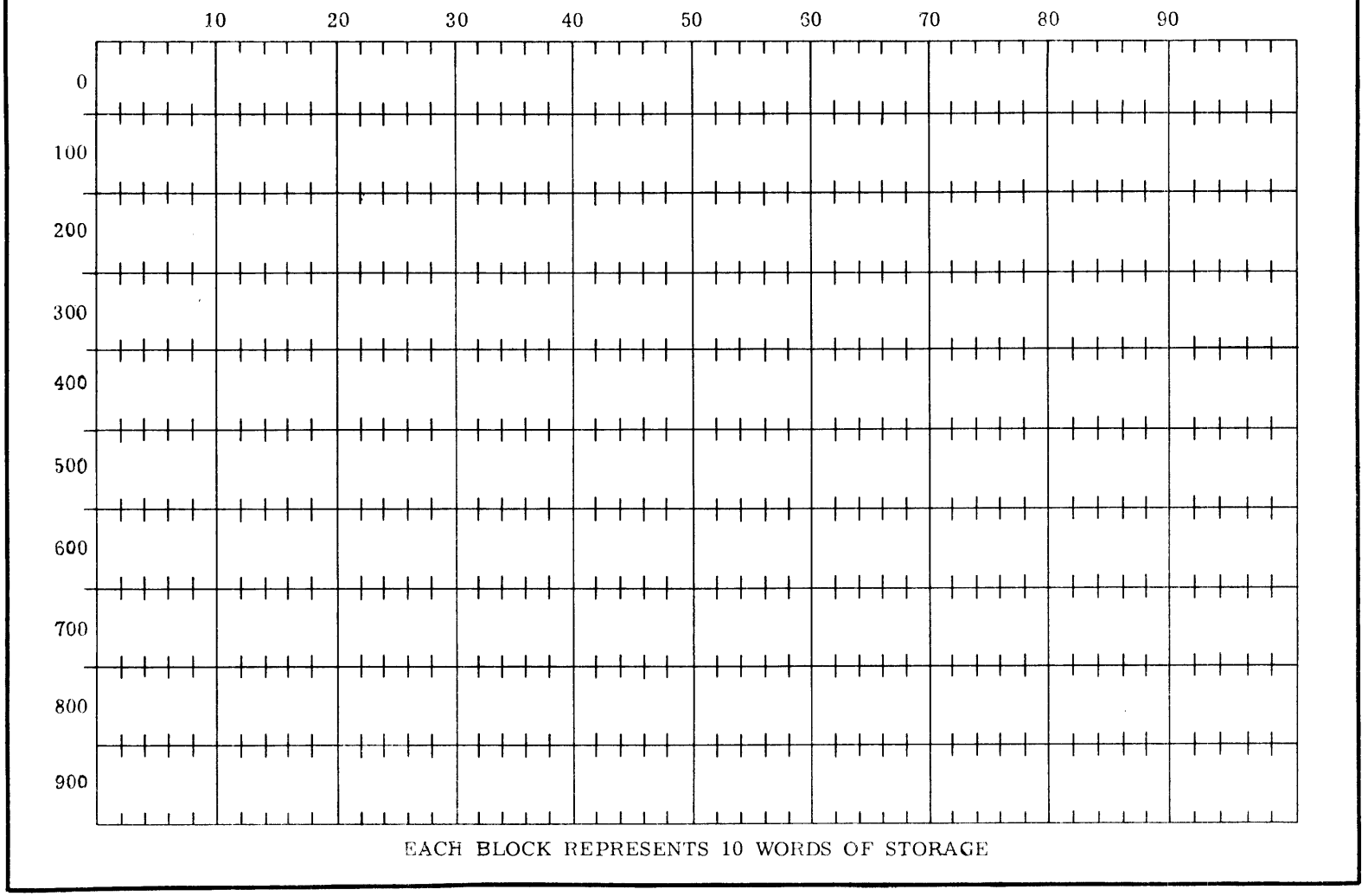
GE 225

MEMORY LAYOUT

CK 67

RUN _____ DATE _____

SYSTEM _____ PROGRAMMER _____ PAGE _____ OF _____



EACH BLOCK REPRESENTS 10 WORDS OF STORAGE

Figure 13-4. Memory Layout Sheet

GE 225 MULTIPLE CARD LAYOUT
BINARY CODED DECIMAL DATA

CK 65

RUN _____ PROGRAMMER _____

SYSTEM _____ DATE _____

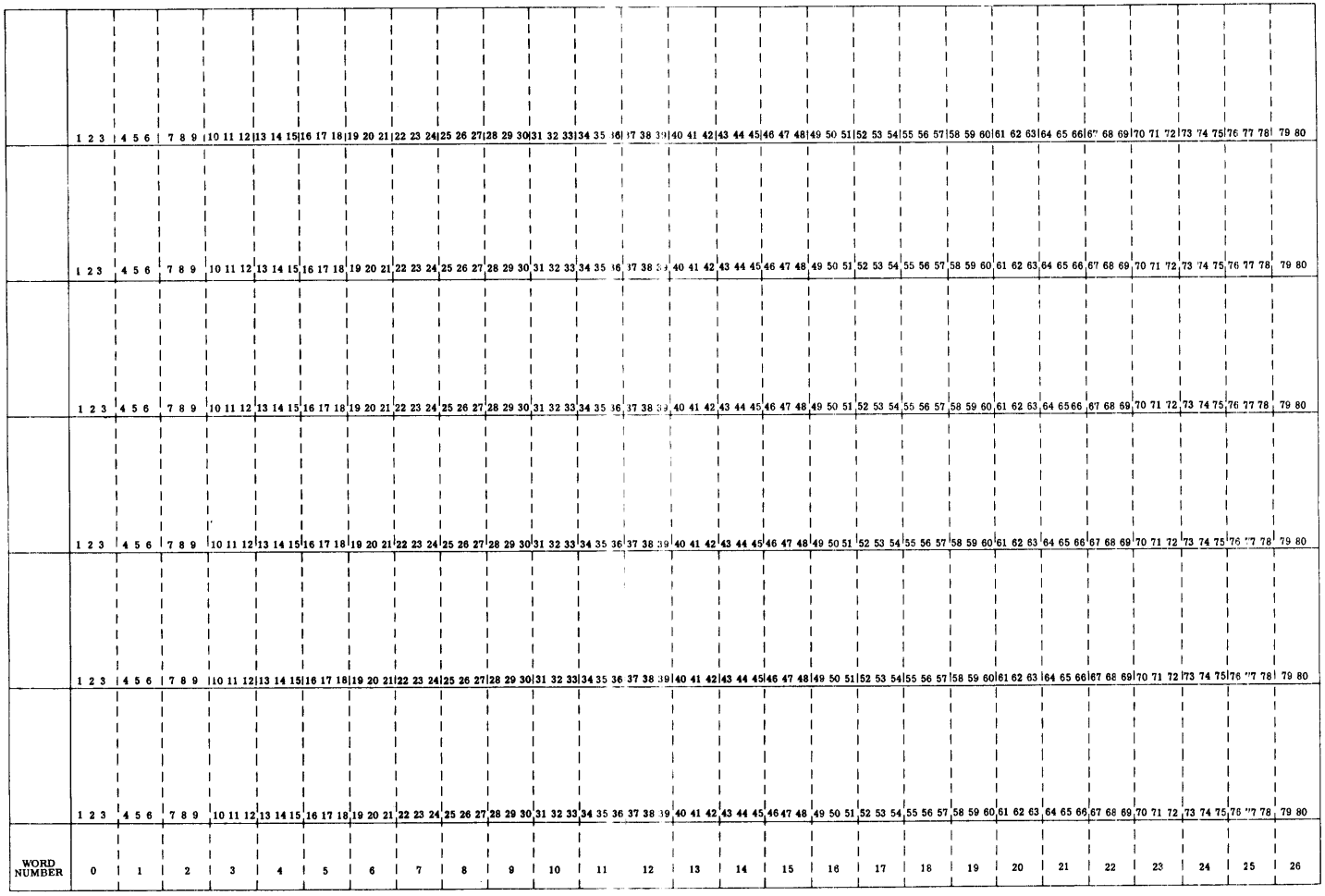


Figure 13-5. BCD Multiple Card Layout Sheet

GE 225

MEMORY ALLOCATION LAYOUT SHEET

INPUT - - OUTPUT

CK 68

RUN ____________ PROGRAMMER ____________ DATE ____________ PAGE ________ OF ________

WORD NBR	MEMORY LOCATION	CARD COL	BIT POSITIONS 0 - 1	2 - 7	8 - 13	14 - 19	DESCRIPTION OF DATA	WORD NBR	MEMORY LOCATION	CARD COL	BIT POSITIONS 0 - 1	2 - 7	8 - 13	14 - 19	DESCRIPTION OF DATA
0								0							
1								1							
2								2							
3								3							
4								4							
5								5							
6								6							
7								7							
8								8							
9								9							
0								0							
1								1							
2								2							
3								3							
4								4							
5								5							
6								6							
7								7							
8								8							
9								9							
0								0							
1								1							
2								2							
3								3							
4								4							
5								5							
6								6							
7								7							
8								8							
9								9							
0								0							
1								1							
2								2							
3								3							
4								4							
5								5							
6								6							
7								7							
8								8							
9								9							

Figure 13-6. Memory Allocation Layout Sheet

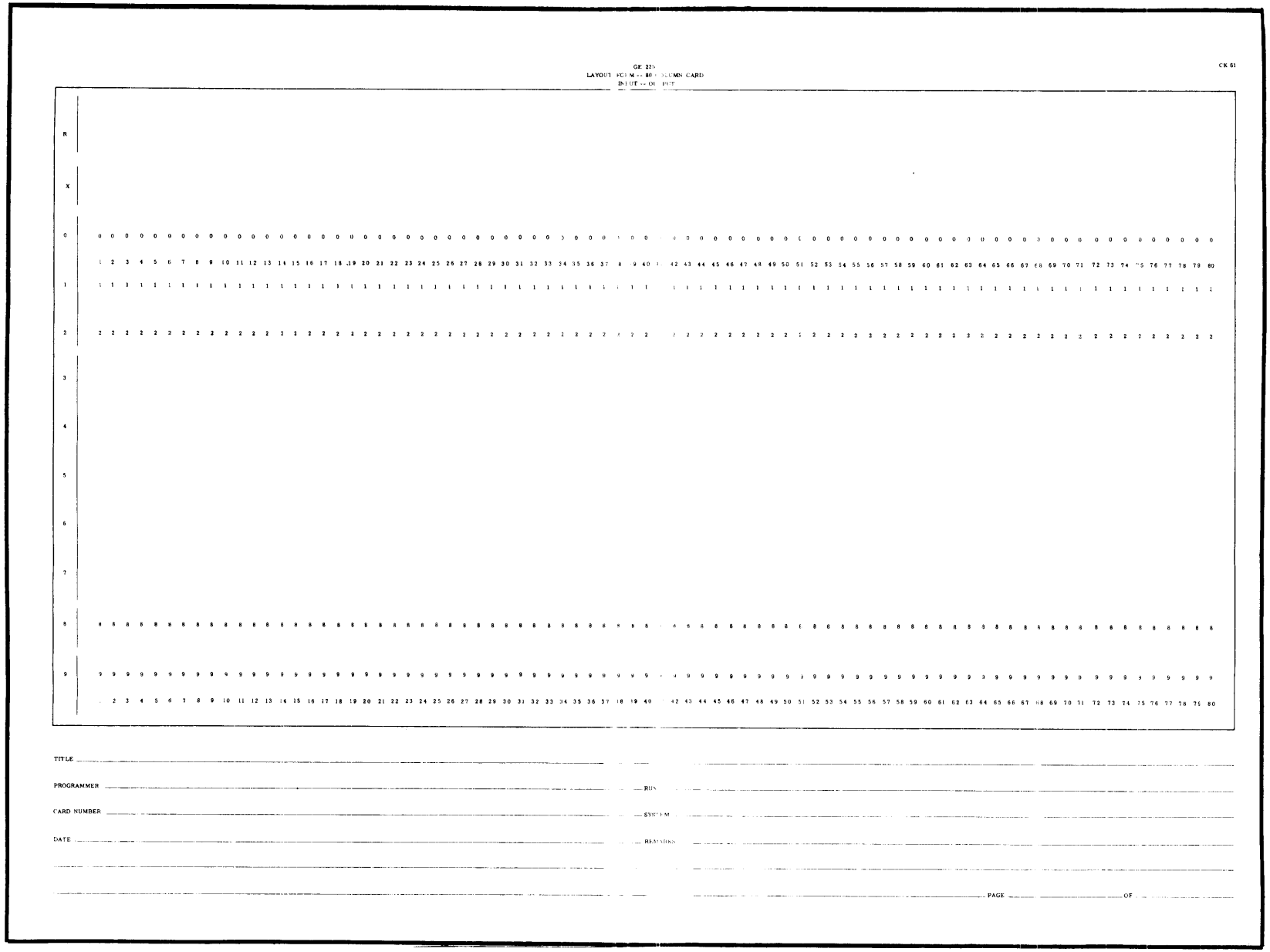


Figure 13-7. 80-Column Card Layout Form

USE OF SYMBOLS

The use of symbolic memory addresses rather than absolute addresses is of utmost importance to the programmer because it relieves him of having to keep track of the location of each constant or instruction in memory. By shifting the burden of memory location to the assembly program, the programmer can code with less errors and thus produce an operating program more quickly. In addition, the symbol used can convey information as to the action taking place within the program. Figure 13-8 illustrates typical symbols.

Symbol	Opr	Operand	X
TWO	DEC	2	
TEN	DEC	10	
CARDIN	BSS	27	
STORE	DDC	0	
CDEOF	ALF	ZZZ	

Figure 13-8. Typical Symbolic Addresses

SUBROUTINE USAGE

The use of subroutines can result in saving of both programming and machine running time. Subroutines can control all input and output operations and many internal operations of a program and use less memory. Normally, a subroutine is a series of instructions which perform a repetitive function for the main program.

The use of subroutines enables the programmer to employ the 'building block principle' in the construction of the program. All frequently-used data processing functions at an installation can be prepared in subroutine form. It is then only necessary for the programmer to use these routines to construct a major portion of the main program with less effort and time than would otherwise be necessary.

The ability to jump to a subroutine and return to the main program requires the retention of information for the return. This concept of informing the subroutine how to get back is termed 'linkage'. In the GE-225, the SPB command provides the 'link' for returning control to the main program after the subroutine function is performed.

In addition to linkage, it is also necessary to specify the parameters which define the problem to the subroutine. Subroutines are usually written in a form for general applicability and must be self-specializing to the particular problem at hand.

The calling sequence which supplies the information (parameters and linkage) needed by the subroutine can vary in size and form. An example of a simple subroutine is illustrated in Figure 13-9.

Symbol	Opr	Operand	X
	DLD	NUM	
	DAD	NUM 2	
	SPB	MPYTEN	1
	DST	RESULT	
	(		
	}		
	(		
MPYTEN	SLD	1	
	STA	TEMP	
	SLD	2	
	DAD	TEMP	
	BRU	1	1
TEMP	DDC	0	

Figure 13-9. Representative Subroutine

This type of subroutine requires no parameters or elaborate calling sequence. The data needed is contained in the A and Q registers before entry and the results from the routine are in the A and Q registers upon exit.

A subroutine requiring a set of parameters in the calling sequence is shown in Figure 13-10.

	Opr	Operand	X	
	8 9 10	12 13 14 15 16 17 18 19	20	31
1	SPB	STRIP	1	
2	DEC	128		
3	DEC	1		
4	DEC	3		
5	BRU	ERROR		
6	STA	AMT#1		
7				

Figure 13-10. Subroutine Requiring a Calling Sequence

```
          00620              ORG 0400
                             REM                    BCD CARD READ SUBROUTINE
                             REM                    CALLING SEQUENCE
                             REM                    A SPB * CRDIN 1
                             REM                    A+1 DEC CARD INPUT AREA
                             REM                    A+2 DEC WORKING STORAGE
                             REM                    A+3 ALF PROGRAM EOF
                             REM                    A+4 EOF RETURN
                             REM                    A+5 NORMAL RETURN
00620     0020001  *CRDIN    LDA 1          1       CARD INPUT AREA
00621     2700636            STO *RD#1              RCD #1
00622     2700663            STO *RD#2              RCD #2
00623     2700650            STO *EOF               EOF LOCATION
00624     2700656            STO *MOVE              MOVE LOCATION
00625     0100644            ADD *SYCON             SYNC CONSTANT LOCATION
00626     0300644            STA *SYCON
00627     0020002            LDA 2          1
00630     0300657            STA *STORE             WORKING STORAGE AREA
00631     0000707            LDA *ENCON             ENTRY CONSTANT
```

Figure 13-11. Subroutine Calling Sequence

Since the parameters necessary for a subroutine can vary over a wide range, the exits from a routine can vary, depending upon the condition encountered within the routine. In the example above, an error in the routine results in the return to line 5 on the coding sheet. In programming, this can be accomplished within the routine by an instruction consisting of

```
BRU  4  1
```

A subroutine calling sequence and the use of the parameters within the sequence is illustrated in Figure 13-11.

The exits from the routine are handled in this manner.

GAP Coding:

Opr			Operand								X	
8	9	10	12	13	14	15	16	17	18	19	20	31
B	R	U	4								1	EOF Return
B	R	U	5								1	Normal Return

In summary, the use of subroutines makes possible considerable saving of memory space and programming time at the very slight expense of the space and complexity of linkages and calling sequences.

TYPEWRITER UTILIZATION

The GE-225 console typewriter can be used by the programmer to type messages concerning conditions within a program and also to instruct the computer operator as to program needs. Using the typewriter for operation control can help reduce human errors.

Typical messages on program conditions are:

1.
```
        0  ERRORS  TAPE  3
        0  ERRORS  TAPE  4
END OF PASS 0
```

2.
```
END OF JOB
```

3.
```
EOF P  1 T  2002 - 004 PREMATURE START
002 - 004 PREMATURE START
003 - 010 PREMATURE START
```

Typewriter messages concerning the operator will be similar to these:

1.
```
JOB DONE. TAPE 7 IS NEW MONITOR TAPE. SAVE 6 AND 7

TAPE 7 XONT READ
```

2.
```
REMOV TAPE 2
MOUNT TAPE 5
TOGGLE SWITCH 18
```

Since the typewriter is a relatively slow output device, messages and operator instructions should be as brief as possible.

DEBUGGING TECHNIQUES

Debugging can be extremely expensive and wasteful of time unless done properly. A few simple and basic rules can do much to reduce the expense involved in getting an operational program. Because debugging methods vary with the individual and the situation, the following is offered merely as a guide.

Desk Checking

When the symbolic program is returned from key punching, a listing is usually sent with the card deck. Check this listing for discrepancies due to misinterpretation by the key punch operator and any possible key punch machine errors. In scanning the symbolic program listing, watch for mistakes in the operation codes and for punches in card columns 7 and 11. During GAP assembly any card containing punches in columns 7 and 11 will be rejected. Correct any errors found before proceeding to the GAP assembly.

Correcting Errors Detected By Gap

After the symbolic program has been assembled by GAP and returned, correct the errors detected and listed by the General Assembly Program. If there were numerous errors listed, make the corrections to the symbolic program deck and reassemble. If relatively few errors were detected, make these corrections in the symbolic program deck without reassembling but by punching octal correction cards to place with the GAP binary program deck.

Flow Chart Utilization

A flow chart is a valuable debugging aid in that it provides for easier detection of logic errors and can be used by the programmer to check off debugged paths within the program. Because this provides the programmer with an indication of what portions are completed, debugging time and check-out time can be reduced.

During debugging, if the programmer uses valid input data and predetermined answers at various program check-points, he can use flow charts as an aid in error location or bracketing, thereby reducing debugging time and machine time requirements.

Memory Dumps

During debugging, memory dumps are essential. Several types of dumps are available but the most frequently used are octal dumps.

The quickest dump of memory is obtained by pressing the memory dump button on the Printer Controller. This automatically produces an octal dump (Figure 13-12), starting at memory location 0000 and continues until the manual clear button on the printer controller is depressed. The printer does not stop automatically when the entire memory has been dumped, but continues looping through memory until the clear button is pressed.

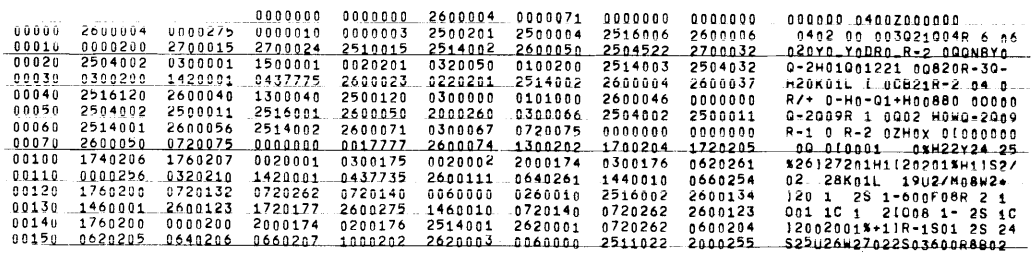
The octal dump that is most frequently used provides the octal memory location for each eight (8) word line of print (column 1 of Figure 13-13). If the words for a line of print are identical to those of the last line printed, the line is skipped. This saves the number of lines printed and machine time required for dumping. This routine is a program feature and thus must be either in memory or read in from cards or tape.

Memory can also be dumped on magnetic tape. Normally, this type of dump is intended for later use by rerun or recovery routines and, in the case of long running programs, should be done periodically. Routines are then available to list these tapes via the high-speed printer.

2514003	0001340	0000002	2600002	2500201	2500004	2516006	2600006
0000200	2700023	2700015	2510015	2514002	2601277	2504522	2700031
2504002	0300001	0020201	0321277	0100200	2514003	2504032	0300200
1420001	0437732	2600022	0220201	2514002	2600002	2514006	2600036
2600002	0000000	0000000	0000000	0000000	0000000	0000000	0000000
0000000	2001777	0000000	0000000	0000000	0000000	0000000	0000000

Figure 13-12. Printer Controller Octal Memory Dump

Octal
Location



Octal Location										
			0000000	0000000	2600004	0000071	0000000	0000000	000000 0400Z000000	
00000	2600004	0000275	0000010	0000003	2500201	2500004	2516006	2600006	0402 00 003021004R 6 06	
00010	0000200	2700015	2700024	2510015	2514002	2600050	2504522	2700032	020Y0 Y0DR0 R-2 0Q0NBY0	
00020	2504002	0300001	1500001	0020201	0320050	0100200	2514003	2504032	Q-2H01Q01221 00820R-30-	
00030	0300200	1420001	0437775	2600023	0220201	2514002	2600004	2600037	H20K01L I 0CB21R-2 04 0	
00040	2516120	2600040	1300040	2500120	0300000	0101000	2600046	0000000	R/+ 0-H0-01+H00880 00000	
00050	2504002	2500011	2516001	2600050	2000260	0300066	2504002	2500011	Q-2009R 1 0Q02 H0WQ-2Q09	
00060	2514001	2600056	2514002	2600071	0300067	0720075	0000000	0000000	R-1 0 R-2 0ZH0X 0[000000	
00070	2600050	0720075	0000000	0017777	2600074	1300202	1700204	1720205	00 0[0001 0%H22Y24 25	
00100	1740206	1760207	0020001	0300175	0020002	2000174	0300176	0620261	%261272[1H1[202[1%H11S2/	
00110	0000256	0320210	1420001	0437735	2600111	0640261	1440010	0660254	02 28K01L 19U2/M08W2•	
00120	1760200	0720132	0720262	0720140	0060000	0260010	2516002	2600134	]20 1 2S 1-600F08R 2 1	
00130	1460001	2600123	1720177	2600275	1460010	0720140	0720262	2600123	001 1C 1 2[008 1- 2S 1C	
00140	1760200	0000200	2000174	0200176	2514001	2620001	0720262	0600204	]2002001%+1]R-1S01 2S 24	
00150	0620205	0640206	0660207	1000202	2620003	0060000	2511022	2000255	S25U26W27022S03600R8802	

Figure 13-13. Memory Dump Printout
Showing Octal and BCD Representation

Memory dumps when properly utilized are very informative and very efficient since only a small amount of computing time is used when dumping through the high-speed printer. The advantages of a memory dump are:

a. It gives the results of any program modification that may have been done.

b. Programmer can check memory to see that information is correct and in the proper locations.

c. It gives temporary or final results in key memory locations up to the time memory was dumped.

d. It shows input or test data being used as a program is run.

Memory dumps via the typewriter or card punch consume computer time and should be used only when a high-speed printer is not available.

Memory dumps should be used frequently during debugging. However, if they prove insufficient, then tracing may provide the solution.

TRACING

The TRACE routine can be used when other techniques have failed. However, at first, trace only the portions of the program that are known or suspected to contain bugs. If it then becomes necessary, trace as much of the program as required. Tracing can be an extremely powerful debugging tool but often its use is abused. Tracing is time-consuming and thus is expensive when used to excess.

Options are available with most trace routines that may supply the desired information without tracing each program instruction.

Typical options are:

1. Snapshot Option

   This type lists index registers 0, 1, 2, and 3, and registers P, I, A, and Q before a BRU, SPB, or SEL instruction is executed.

2. Single Address Option

   This type lists the same registers as the SNAPSHOT option, only when a specific address is referenced.

3. Normal Option

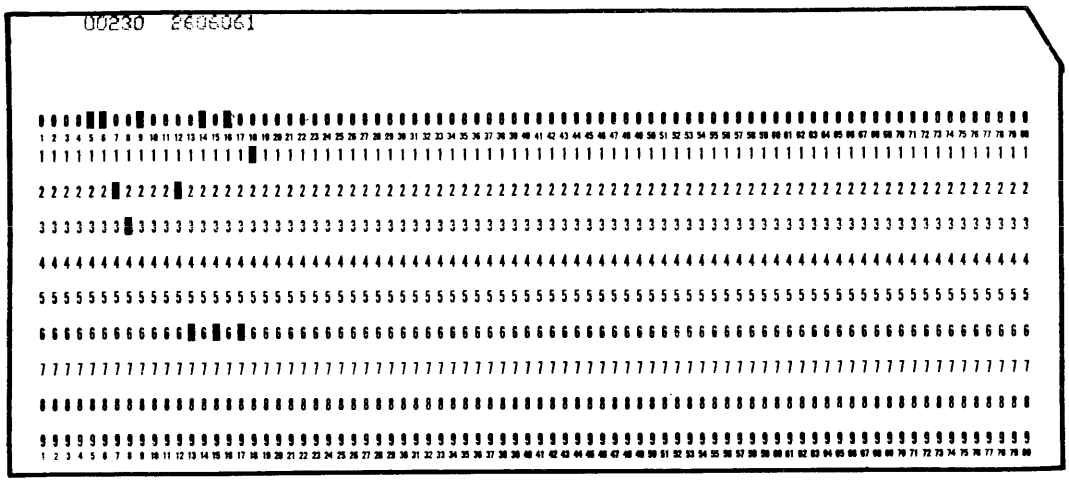
   The same registers listed in the other types are printed before each instruction is executed.

Tracing output is normally through the high-speed printer.

Loaders

When the program deck from GAP is in binary form, a binary loader deck is used to read the GAP program deck into the proper memory locations.

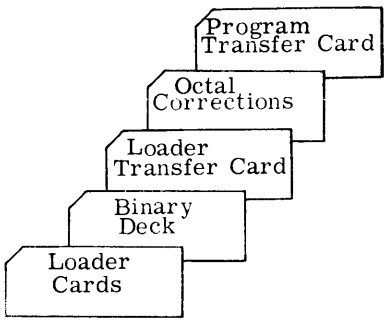
During debugging, it is best to use a binary loader with octal correction cards. This type of loader will read into memory the binary deck and then read the octal corrections into the specified memory words. Thus, errors can be corrected without repeated reassembly of the symbolic deck and, when the program is completely debugged, a corrected symbolic deck can be produced in a single new GAP assembly. Normally a loader, such as the Lower Memory Binary Loader for Binary Deck with Octal Correction Cards,



For use with a binary loader containing an octal correction subroutine to change a specific memory location.

Figure 13-14. Octal Correction Card

CD225B1.3, is used. To illustrate loader and correction card usage, the deck set-up using octal corrections for Routine CD225B1.3 is shown:



The octal corrections are punched as follows: One card is punched for each change to be made. Columns 5 through 9 contain the octal address and columns 12 through 18 contain the octal contents required. Figure 13-14 shows a sample octal correction card.

When the program is considered debugged, the corrections should be made to the source deck and the program reassembled. A final checkout should now be made with the new object deck.

PROGRAM DOCUMENTATION

Accurate, up-to-date program documentation can produce considerable savings in programming and operator effort, as well as computer time. Efficient operation of a computer system requires that changes or correction to operating programs be made quickly and correctly. Without adequate documentation, changes and corrections may become difficult to accomplish. Since each computer installation has different characteristics, program documentation can vary from site to site. However, a basic pattern can be used by each system.

Run Book

A RUN BOOK should exist for every program run within a system and should contain documentation so complete that modifications can be made with minimum effort. Also, if trouble develops, the source can be readily found. A typical run book would contain the following:

A. Run Number and Title.

B. Name of Programmer, Date Completed, and Date of Last Modification.

C. A Concise Description of what the run is to accomplish.

D. A Write-up containing all internal and external controls pertinent to the program, including:

   1. A completed Operator Instruction or Run Form that contains,

      a. Average run time and procedure to follow if established time limit is exceeded.

      b. Console switch settings and brief description of each.

c. Error and special procedure loops with brief explanation of each.

d. Tape controller and input and output tape handler numbers.

e. Identification and disposition of tapes.

f. Rerun and restart procedure.

g. All peripheral device set-ups and plug designation.

2. Completed description and Layout Forms for all input and output.

3. Memory Allocation Layout

a. Mark input and output areas

b. Program and subroutine areas

c. Working storage areas identifying each location used

d. If overlays are used, identify areas in which it occurs.

E. Run Diagram and Flow Chart

An up-to-date run diagram and an accurate flow chart should be in the Run Book. GAP coding reference points should be marked or identified on the flow chart. This provides references between the operating program and the flow chart providing for easier program corrections.

F. Sample Printer Output

If the high-speed printer is used during the run, a sample of the output can be extremely useful. The samples should be marked with the run number.

G. GAP Listing

The GAP program listing can be included in the run book. If the GAP listing is in a separate binder, indicate the binder number for quick location of the program. Any corrections or modifications to the listing should be entered in red and initialled and dated if the program is not to be reassembled at this time.

APPENDICES

I. REPRESENTATION OF GE-225 CHARACTERS

II. OCTAL LIST OF GE-225 INSTRUCTIONS

III. ALPHABETIC LIST OF GE-225 INSTRUCTIONS

APPENDIX I.

REPRESENTATION OF GE-225 CHARACTERS

CHARACTER	HIGH SPEED PRINTER SYMBOLS	CONSOLE TYPEWRITER CHARACTER OR ACTION	PAPER TAPE CHARACTER (8 CHANNEL)	HOLLERITH CODE (PUNCH IN ROWS)	BCD MEMORY (OCTAL)**	BCD MAGNETIC TAPE (OCTAL)
0	0	0	Space	0	00	12
1	1	1	1	1	01	01
2	2	2	2	2	02	02
3	3	3	3	3	03	03
4	4	4	4	4	04	04
5	5	5	5	5	05	05
6	6	6	6	6	06	06
7	7	7	7	7	07	07
8	8	8	8	8	10	10
9	9	9	9	9	11	11
A	A	A	/	12-1	21	61
B	B	B	S	12-2	22	62
C	C	C	T	12-3	23	63
D	D	D	U	12-4	24	64
E	E	E	V	12-5	25	65
F	F	F	W	12-6	26	66
G	G	G	X	12-7	27	67
H	H	H	Y	12-8	30	70
I	I	I	Z	12-9	31	71
J	J	J	J	11-1	41	41
K	K	K	K	11-2	42	42
L	L	L	L	11-3	43	43
M	M	M	M	11-4	44	44
N	N	N	N	11-5	45	45
O	O	O	O	11-6	46	46
P	P	P	P	11-7	47	47
Q	Q	Q	Q	11-8	50	50
R	R	R	R	11-9	51	51
S	S	S	B	0-2	62	22
T	T	T	C	0-3	63	23
U	U	U	D	0-4	64	24
V	V	V	E	0-5	65	25
W	W	W	F	0-6	66	26
X	X	X	G	0-7	67	27
Y	Y	Y	H	0-8	70	30
Z	Z	Z	I	0-9	71	31
+	+		0	12	20	60
-	-	-	-	11	40	40
Space	Blank	Blank	&	Blank	60	20
/	/		A	0-1	61	21
				2-8	12	12
#	#	/	Stop	3-8	13	13
@	@			4-8	14	14
(Underline)	-			5-8	15	15
=	=			6-8	16	16
				7-8	17	
				12-2-8	32*	72
+0				12-0	32*	
.	.	.		12-3-8	33	73
*				12-4-8	34	74
				12-5-8	35	75
			Tab	12-6-8	36	76
		Carriage Return		12-7-8	37	77
-0				11-0	52*	52
				11-2-8	52*	52
$	$	$	$	11-3-8	53	53
*	*			11-4-8	54	54
				11-5-8	55	55
				11-6-8	56	56
				11-7-8	57	57
		Print Red		0-2-8	72	32
,	,			0-3-8	73	33
%	%			0-4-8	74	34
(	⊏	Print Black		0-5-8	75	35
)	⊐	Tab		0-6-8	76	36
			Delete	0-7-8	77	37

*The 400 card per minute card reader reads 11-0 and 11-2-8 as 52 and 12-0 and 12-2-8 as 32. The 1000 cards per minute card reader treats 11-2-8 and 12-2-8 as invalid characters. The card punch punches only 11-0 for 52 and 12-0 for 32.

**The OCTAL notation is a shorthand for binary representation. Conversion between the two representations can be done mentally. In the OCTAL system, there are eight admissible symbols: 0, 1, 2, 3, 4, 5, 6, 7. Each may represent (when used) a maximum of three binary bits.

APPENDIX II. OCTAL LIST OF GE-225 INSTRUCTIONS

Octal	Mnemonic	Word Times	Page
0000000	LDA Y X Load A Register	2	V - 14
0100000	ADD Y X	2	V - 2
	*Decimal Add	2	V - 7
0200000	SUB Y	3	V - 7
	*Decimal Subtract		
0200000	SUB Y X Subtract	3	V - 2
0200000 TTNNNNN	WEF T Write End of File	2	VIII - 11
02MMMMM TTNNNNN	WTD M T (blank) N Write Tape Decimal	2	VIII - 10
0300000	STA Y X Store A	2	V - 15
03MMMMM TTNNNNN	WTB M T (blank) N Write Tape Binary	2	VIII - 10
0400000	BXL K X Branch If X Is Less Than	3	V - 35
0420000(N=1)	RSD M N Read Document Single	2	X - 3
0440000(N=1)	RDC M N Read Document Continuously	2	X - 4
0460000(N=1)	PKT X N Pocket Select	2	X - 4
04MMMMM TTNNNNN	RTD M T (blank) N Read Tape Decimal	2	VIII - 11
0500000	BXH K X Branch If X Is Higher Than or Equal To	3	V - 34
0500000(N=1)	HLT M N Halt Continuous Feeding	2	X - 5
0520000(N=1) 0000000	ERB N End Read Busy	2	X - 5
05MMMMM TTNNNNN	RTB M T (blank) N Read Tape Binary	2	VIII - 11
0600000	LDX Y X Load X	3	V - 36
0600000 NN00000	SLW N Slew Paper N Lines	2	IX - 6
0700000	SPB Y X Store P and Branch	2	V - 31
0X00000 XX00000	SLT K Slew Paper to Tape Punch	2	IX - 6
1000000	DLD Y X Double Length Load	3	V - 15
1020000(N=2)	RSD M N Read Document Single	2	X - 3
1040000(N=2)	RDC M N Read Document Continuously	2	X - 4
1060000(N=2)	PKT X N Pocket Select	2	X - 4
1100000	DAD Y X	3	V - 2
	*Double Decimal Add	3	V - 8
1100000	DAD Y X Double Length Add	3	V - 2
1100000(N=2)	HLT M N Halt Continuous Feeding	2	X - 5
1120000(N=2) 0000000	ERB N End Read Busy	2	X - 5
1200000	DSU Y X	5	V - 3
	*Double Decimal Subtract	5	V - 8
1200000	DSU Y X Double Length Subtract	5	V - 3
1200000 00MMMMM	RRF N F (blank) M Read from MRADS Unit F	2	XI - 6

*Optional Instruction

Octal	Mnemonic	Word Times	Page
1201000 00MMMMM	RRD N F (blank) M Read from MRADS Unit F	2	XI - 7
1202000 0000000	RAW N F (blank) zero Read After Write Check	2	XI - 7
1300000	DST Y X Double Length Store	3	V - 16
1400000	INX K X Increment X	3	V - 34
14MMMMM TTNNNNN	RBD M T (blank) N Read Backward Decimal	2	VIII - 12
1500000	MPY Y X Multiply	9 to 23	V - 5
15MMMMM TTNNNNN	RBB M T (blank) Read Backward Binary	2	VIII - 12
1600000	DVD Y X Divide	26 to 29	V - 5
1600000 TT00000	BKW T Backspace and Position Write Head	2	VIII - 12
1700000	STX Y X Store X	3	V - 36
2000000	EXT Y X Extract	3	V - 17
2000000 01YYYYY	WPL Y N Write Print Line	2	IX - 5
2000000 TT00000	RWD T Rewind	2	VIII - 12
2100000	*CAB Y Compare and Branch	2 to 4	V - 33
2200000	*DCB Y Double Compare and Branch	2 to 6	V - 33
2300000	ORY Y X Or A into Y	3	V - 17

Octal	Mnemonic	Word Times	Page
23MMMMM TTNNNNN	WTS M T (blank) N Write Tape Special Binary Mode	2	VIII - 11
2400000	*MOV Y Move	4 + 2N	V - 18
2500000 MMMMMMM	PRF F OCT (MRADS Address) Position MRADS File	2	XI - 4
2500004	HCR Halt Card Reader	2	VI - 26
2500005	OFF Power Off (Direct I/O Devices)	2	VI - 8
2500006	RPT Read Paper Tape	2	VI - 16
2500006	TYP Type	2	VI - 8
2500006	WPT Write Paper Tape	2	VI - 18
2500007	TON Typewriter On	2	VI - 8
2500011	RCS Read Control Switches	2	VI - 6
2500015	PON Punch On	2	VI - 18
2500016	HPT Halt Paper Tape Reader	2	VI - 17
2500014	RON Paper Tape Reader On	2	VI - 16
2500P20	SEL P X Select	2	VII - 2
2504001	LAQ Load A from Q	3	V - 19
2504002	LDZ Load Zero into A Register	3	V - 22
2504004	LQA Load Q from A	3	V - 19
2504005	XAQ Exchange A and Q	3	V - 20
2504006	MAQ Move A to Q	3	V - 19

* This instruction is an optional feature.

Octal	Mnemonic	Word Times	Page
2504012	NOP No Operation	3	V - 24
2504022	LDO Load One into A Register	3	V - 22
2504032	ADO Add One	3	V - 4
2504032	ADO Add One *Add One Decimal	3 3	V - 4 V - 9
2504040	CHS Change Sign of A Register	2	V - 23
2504102	LMO Load Minus One into A Register	3	V - 23
2504112	SBO Subtract One *Subtract One Decimal	3 3	V - 4 V - 9
2504202	*LAC Load A Register from C Register	3	V - 20
2504210	*LCA Load C Register from A Register	3	V - 20
2504302	CPL Complement A	3	V - 23
2504522	NEG Negate A	3	V - 23
2506003	*SXG Y Select X Register Group	2	V - 36
2506011	SET DECMODE Set Decimal Mode	2	V - 9
2506012	SET BINMODE Set Binary Mode	2	V - 10
2506015	SET PST Set Automatic Priority Interrupt On	2	VII - 3
2506016	SET PBK Set Automatic Priority Interrupt Off	2	VII - 3
250YY00	RCD Y Read Cards Decimal	2	VI - 26
250YY01	RCB Read Cards Binary	2	VI - 27
250YY02	WCD Y Write Card Decimal	2	VI - 37
250YY03	WCB Y Write Card Binary	2	VI - 37
250YY10	RCF Y Read Cards Full	2	VI - 28
250YY12	RCM Y Read Cards Mixed	2	VI - 29
250YY17	WCF Y Write Cards Full	2	VI - 37
2510000	SRA K Shift Right A Register	2 to 12	V - 24
2510040	SCA K Shift Circular A Register	2 to 12	V - 27
2510100	SNA K Shift N and A Right	2 to 12	V - 28
2510400	SAN K Shift A and N Right	2 to 12	V - 28
2511000	SRD K Shift Right Double	2 to 12	V - 26
2511100	NAQ K Shift N, A, and Q Right	2 to 12	V - 29
2511200	SCD K Shift Circular Double	2 to 12	V - 27
2511400	ANQ K Shift A into N and Q	2 to 12	V - 28
2512000	SLA K Shift Left A Register	2 to 12	V - 25
2512200	SLD K Shift Left Double	2 to 12	V - 26
2513000	NOR K Normalize the A Register	3 to 12	V - 29
2513200	DNO K Double Length Normalize	2 to 12	V - 30
2514000	BOD Branch on Odd	2	V - 32
2514001	BMI Branch on Minus	2	V - 32

* This instruction is an optional feature.

Octal	Mnemonic	Word Times	Page
2514002	BZE Branch on Zero	2	V - 32
2514003	BOV Branch on Overflow	2	V - 32
2514004	BPE Branch on Parity Error	2	V - 32
2514005	BNR Branch on N Register Ready	2	VI - 9
2514006	BCR Branch on Card Reader Ready	2	VI - 33
2514007	BPR Branch on Card Punch Ready	2	VI - 37
2514720	BAR BAR 7 Branch on AAU Ready	2	XII - 8
2514721	BAR BMI 7 Branch on AAU Minus	2	XII - 9
2514722	BAR BZE 7 Branch on AAU Zero	2	XII - 9
2514723	BAR BOV 7 Branch on AAU Overflow	2	XII - 9
2514724	BAR BUF 7 Branch on AAU Underflow	2	XII - 9
2514727	BAR BER 7 Branch on AAU Error	2	XII - 9
2514P20	BCS BPR P Branch on Printer Ready	2	IX - 7
2514P20	BCS BRR P Branch on MRADS Controller Ready	2	XI - 10
2514P20	BCS BTR P Branch on Tape Controller Ready	2	VIII - 14
2514P20(K=1)	BCS SKR P Branch on Document Handler K Ready	2	X - 6
2514P21	BCS BAA P Branch on Any Alert	2	IX - 14
2514P21	BCS BEF P Branch on End of File	2	VIII - 14
2514P21(File 0)	BCS FKR P Branch on File K Ready	2	XI - 10
2514P21(K=2)	BCS SKR P Branch on Document Handler K Ready	2	X - 6
2514P22	BCS BET P Branch on End of Tape	2	VIII - 14
2514P22	BCS BOP P Branch on Printer Out of Paper	2	IX - 7
2514P22(File 1)	BCS FKR P Branch on File K Ready	2	XI - 10
2514P22(K=1)	BCS NPK P Branch on No Pocket Decision, Document Handler K	2	X - 6
2514P23	BCS BOV P Branch on Printer Buffer Overflow	2	IX - 14
2514P23	BCS BRW P Branch on Tape Rewinding	2	VIII - 15
2514P23(File 2)	BCS FKR P Branch on File K Ready	2	XI - 10
2514P23(K=2)	BCS NPK P Branch on No Pocket Decision, Document Handler K	2	X - 6
2514P24	BCS BPE P Branch on Mag Tape Parity Error	2	VIII - 14
2514P24	BCS BSA P Branch on Printer Slew Alert	2	IX - 14
2514P24(File 3)	BCS FKR P Branch on File K Ready	2	XI - 10
2514P24(K=1)	BCS FSK P Branch on Feeding, Document Handler K	2	X - 6
2514P25	BCS BIO P Branch on Mag Tape I/O Buffer Error DSU	2	VIII - 14 XI - 10
2514P25(K=2)	BCS FSK P Branch on Feeding, Document Handler K	2	X - 6
2514P26	BCS BME P Branch on Mod 3 or 4 Error	2	VIII - 14
2514P26(K=1)	BCS ICK P Branch on Invalid Character, Document Handler K	2	X - 6
2514P26	BCS ICK P Branch on DSU Parity Error	2	X1-10

Octal	Mnemonic	Word Times	Page
2514P27	BCS BER P Branch on Error	2	VIII - 15
2514P27(K=2)	BCS ICK P Branch on Invalid Character, Document Handler K	2	X - 6
2514P30(K=1)	BCS SKE Branch on Any Error, Document Handler K	2	X - 7
2514P31	BCS FAE P Branch on Error - On Any File	2	XI - 11
2514P31(K=2)	BCS SKE Branch on Any Error, Document Handler K	2	X - 7
2514P32(K=1)	*BCS DQK Branch on Document TCD Correct, Document Handler K.	2	X - 7
2514P32(File 0)	BCS FKE P Branch on File K, File Error	2	XI - 11
2514P33(K=2)	*BCS DQK Branch on Document TCD Correct, Document Handler K.	2	X - 7
2514P33(File 1)	BCS FKE P Branch on File K, File Error	2	XI - 11
2514P34(File 2)	BCS FKE P Branch on File K, File Error	2	XI - 11
2514P35(File 3)	BCS FKE P Branch on File K, File Error	2	XI - 11
2514PCC	BCS XXX P Branch on Controller Selector	2	VII - 2
2516000	BEV Branch on Even	2	V - 32
2516001	BPL Branch on Plus	2	V - 32
2516002	BNZ Branch on Non-Zero	2	V - 32
2516003	BNO Branch on No Overflow	2	V - 32
2516004	BPC Branch on Parity Correct	2	V - 32
2516005	BNN Branch on N Register Not Ready	2	VI - 9
2516006	BCN Branch on Card Reader Not Ready	2	VI - 33
2516007	BPN Branch on Card Punch Not Ready	2	VI - 37
2516720	BAR BAN 7 Branch on AAU Not Ready	2	XII - 8
2516721	BAR BPL 7 Branch on AAU Plus	2	XII - 9
2516722	BAR BNZ 7 Branch on AAU Not Zero	2	XII - 9
2516723	BAR BNO 7 Branch on AAU No Overflow	2	XII - 9
2516724	BAR BNU 7 Branch on AAU No Underflow	2	XII - 9
2516727	BAR BNE 7 Branch on AAU No Error	2	XII - 9
2516P20	BCS BPN P Branch on Printer Not Ready	2	IX - 7
2516P20	BCS BRN P Branch on MRADS Controller Not Ready	2	XI - 10
2516P20	BCS BTN P Branch on Tape Controller Not Ready	2	VIII - 14
2516P20(K=1)	BCS SKN P Branch on Document Handler K Not Ready	2	X - 6
2516P21	BCS BNA P Branch on Printer No Alert	2	IX - 14
2516P21	BCS BNF P Branch on No End of File	2	VIII - 14
2516P21(File 0)	BCS FKN P Branch on File K Not Ready	2	XI - 10
2516P21(K=2)	BCS SKN P Branch on Document Handler K Not Ready	2	X - 6
2516P22	BCS BNP P Branch if Printer Not Out of Paper	2	IX - 7

* This instruction is an optional feature.

Octal	Mnemonic	Word Times	Page
2516P22	BCS BNT P Branch on No End of Tape	2	VIII - 14
2516P22(File 1)	BCS FKN P Branch on File K Not Ready	2	XI - 10
2516P22(K=1)	BCS PDK P Branch on Pocket Decision, Document Handler K	2	X - 6
2516P23	BCS BNO P Branch on No Printer Buffer Overflow	2	IX - 14
2516P23	BCS BNR P Branch on No Tape Rewinding	2	VIII - 15
2516P23(File 2)	BCS FKN P Branch on File K Not Ready	2	XI - 10
2516P23(K=2)	BCS PDK P Branch on Pocket Decision, Document Handler K	2	X - 6
2516P24	BCS BNS P Branch on No Printer Slew Alert	2	IX - 14
2516P24	BCS BPC P Branch on Mag Tape Parity Correct	2	VIII - 14
2516P24(File 3)	BCS FKN P Branch on File K Not Ready	2	XI - 10
2516P24(K=1)	BCS NFK P Branch on Not Feeding, Document Handler K	2	X - 6
2516P25	BCS BIC P Branch on Mag Tape I/O Buffer Correct	2	VIII - 14
2516P25(K=2)	BCS NFK P Branch on Not Feeding, Document Handler K	2	X - 6
2516P26	BCS BNM P Branch on No Mod 3 or 4 Error	2	VIII - 15
2516P26	BCS RPC P Branch on DSU Parity Correct	2	XI - 10
2516P26(K=1)	BCS VCK Branch on Valid Character, Document Handler K	2	X - 7
2516P27	BCS BNE P Branch on No Error	2	VIII - 15
2516P27(K=2)	BCS VCK Branch on Valid Character, Document Handler K	2	X - 7
2516P30(K=1)	BCS SKC Branch on Document Handler K Correct	2	X - 7
2516P31	BCS FAC P Branch on No Error - Any File	2	XI - 11
2516P31(K=2)	BCS SKC Branch on Document Handler K Correct	2	X - 7
2516P32(File 0)	BCS FKC P Branch on File K, No Unit Error	2	XI - 11
2516P32(K=1)	*BCS NQK Branch on Document TCD Not Correct, Document Handler K	2	X - 7
2516P33(File 1)	BCS FKC P Branch on File K, No Unit Error	2	XI - 11
2516P33(K=2)	*BCS NQK Branch on Document TCD Not Correct, Document Handler K	2	X - 7
2516P34(File 2)	BCS FKC P Branch on File K, No Unit Error	2	XI - 11
2516P35(File 3)	BCS FKC P Branch on File K, No Unit Error	2	XI - 11
2516PCC	BCS XXX P Branch on Controller Selector	2	VII - 2
25MMMMM TTNNNNN	RTS M T (blank) N Read Tape Special Binary Mode	2	VIII - 11
2600000	BRU Y X Branch Unconditionally	1	V - 31
2700000	STO Y X Store Operand Address	3	V - 16
3000000	FLD Y Load Auxiliary Arithmetic Unit	72 usec	XII - 7
30YYYYY 01XXXXX	WFL Y X N (WPL) Write Format Line	2	IX - 6

* This instruction is an optional feature

Mnemonic	Octal			Word Times	Page
3100002	MAQ	A		49.5 usec	XII - 7
	Move AX to QX				
3100010	SET NFLPOINT			49.5 usec	XII - 6
	Set Normalized Floating-Point Mode				
31YYYYY	FAD	Y		Min. 162 usec Max. 709 usec	XII - 7
	AAU Add				
3200002	LQA	A		49.5 usec	XII - 7
	Load QX From AX				
3200010	SET UFLPOINT			49.5 usec	XII - 6
	Set Unnormalized Floating-Point Mode				
32YYYYY	FSU	Y		Min. 162 usec Max. 709 usec	XII - 8
	AAU Subtract				
3300000	FST	Y		72 usec	XII - 7
	Store Auxiliary Arithmetic Unit				
3500002	XAQ	A		117 usec	XII - 7
	Exchange AX and QX				
3500010	SET FIXPOINT			49.5 usec	XII - 6
	Set Fixed-Point Mode				
35MMMMM TTNNNNN	RBS	M	T	2	VIII - 12
	(blank)	N			
	Read Backward Special Binary				
35YYYYY	FMP	Y		Min. 297 usec Max. 1062 usec	XII - 7
	AAU Multiply				
3600002	LAQ	A		49.5 usec	XII - 7
	Load AX From QX				
36YYYYY	FDV	Y		Min. 814.5 usec Max. 1095 usec	XII - 8
	AAU Divide				
3700000 00MMMMM	WRF	N	F	2	XI - 7
	(blank)	M			
	Write on MRADS Unit F				
3701000 00MMMMM	WRD	N	F	2	XI - 7
	(blank)	M			
	Write on MRADS Unit F				

APPENDIX III. ALPHABETIC LIST OF GE-225 INSTRUCTIONS

Mnemonic		Octal	Word Times	Page
ADD	Y X	0100000	2	V - 2
	*Decimal Add		2	V - 7
ADD	Y X	0100000	2	V - 1
	Add			
ADO		2504032	3	V - 4
	Add One			
ADO		2504032	3	V - 9
	*Add One Decimal			
ALF	(Pseudo)			IV -10
	Alphanumeric			
ANQ	K	2511400	2 to 12	V -28
	Shift A into N and Q			
BAR	BAN 7	2516720	2	XII - 8
	Branch on AAU Not Ready			
BAR	BAR 7	2514720	2	XII - 8
	Branch on AAU Ready			
BAR	BER 7	2514727	2	XII - 9
	Branch on AAU Error			
BAR	BMI 7	2514721	2	XII - 9
	Branch on AAU Minus			
BAR	BNE 7	2516727	2	XII - 9
	Branch on AAU No Error			
BAR	BNO 7	2516723	2	XII - 9
	Branch on AAU No Overflow			
BAR	BNU 7	2516724	2	XII - 9
	Branch on AAU No Underflow			
BAR	BNZ 7	2516722	2	XII - 9
	Branch on AAU Not Zero			
BAR	BOV 7	2514723	2	XII - 9
	Branch on AAU Overflow			
BAR	BPL 7	2516721	2	XII - 9
	Branch on AAU Plus			
BAR	BUF 7	2514724	2	XII - 9
	Branch on AAU Underflow			
BAR	BZE 7	2514722	2	XII - 9
	Branch on AAU Zero			
	*Optional Instruction			

Mnemonic		Octal	Word Times	Page
BCN		2516006	2	VI - 33
	Branch on Card Reader Not Ready			
BCR		2514006	2	VI - 33
	Branch on Card Reader Ready			
BCS	BAA P	2514P21	2	IX - 14
	Branch on Any Alert			
BCS	BEF P	2514P21	2	VIII - 14
	Branch on End of File			
BCS	BER P	2514P27	2	VIII - 15
	Branch on Error			IX - 8
				XI - 10
BCS	BET P	2514P22	2	VIII - 14
	Branch on End of Tape			
BCS	BIC P	2516P25	2	VIII - 14
	Branch on Input/Output Buffer Correct			XI - 10
BCS	BIO P	2514P25	2	VIII - 14
	Branch on Input/Output Buffer Error			XI - 10
BCS	BME P	2514P26	2	VIII - 14
	Branch on Mod 3 or 4 Error			
BCS	BNA P	2516P21	2	IX - 14
	Branch on Printer No Alert			
BCS	BNE P	2516P27	2	VIII - 15
	Branch on No Error			IX - 8
				XI - 10
BCS	BNF P	2516P21	2	VIII - 14
	Branch on No End of File			
BCS	BNM P	2516P26	2	VIII - 15
	Branch on No Mod 3 or 4 Error			
BCS	BNO P	2516P23	2	IX - 14
	Branch on No Printer Buffer Overflow			
BCS	BNP P	2516P22	2	IX - 7
	Branch if Printer Not Out of Paper			
BCS	BNR P	2516P23	2	VIII - 15
	Branch on No Tape Rewinding			

Mnemonic		Octal	Word Times	Page
BCS	BNS P Branch on No Printer Slew Alert	2516P24	2	IX - 14
BCS	BNT P Branch on No End of Tape	2516P22	2	VIII- 14
BCS	BOP P Branch on Printer Out of Paper	2514P22	2	IX - 7
BCS	BOV P Branch on Printer Buffer Overflow	2514P23	2	IX - 14
BCS	BPC P Branch on Tape Parity Correct	2516P24	2	VIII - 14
BCS	BPE P Branch on Tape Parity Error	2514P24	2	VIII - 14
BCS	BPN P Branch on Printer Not Ready	2516P20	2	IX - 7
BCS	BPR P Branch on Printer Ready	2514P20	2	IX - 7
BCS	BRN P Branch on MRADS Controller Not Ready	2516P20	2	XI - 10
BCS	BRR P Branch on MRADS Controller Ready	2514P20	2	XI - 10
BCS	BRW P Branch on Tape Rewinding	2514P23	2	VIII- 15
BCS	BSA P Branch on Printer Slew Alert	2514P24	2	IX - 14
BCS	BTN P Branch on Tape Controller Not Ready	2516P20	2	VIII- 14
BCS	BTR P Branch On Tape Controller Ready	2514P20	2	VIII- 14
*BCS	DQK Branch on Document TCD Correct, Document Handler K.	2514P32(K=1) or 2514P33(K=2)	2	X - 7
BCS	FAC P Branch on No Error - Any File	2516P31	2	XI - 11
BCS	FAE P Branch on Error - On Any File	2514P31	2	XI - 11
BCS	FKC P Branch on File K, No Unit Error	2516P32(File 0) or 2516P33(File 1) or 2516P34(File 2) or 2516P35(File 3)	2	XI - 11
BCS	FKE P Branch on File K, File Error	2514P32(File 0) or 2514P33(File 1) or 2514P34(File 2) or 2514P35(File 3)	2	XI - 11
BCS	FKN P Branch on File K Not Ready	2516P21(File 0) or 2516P22(File 1) or 2516P23(File 2) or 2516P24(File 3)	2	XI - 10
BCS	FKR P Branch on File K Ready	2514P21(File 0) or 2514P22(File 1) or 2514P23(File 2) or 2514P24(File 3)	2	XI - 10
BCS	FSK P Branch on Feeding, Document Handler K	2514P24(K=1) or 2514P25(K=2)	2	X - 6
BCS	ICK P Branch on Invalid Character, Document Handler K	2514P26(K=1) or 2514P27(K=2)	2	X - 6
BCS	NFK P Branch on Not Feeding, Document Handler K	2516P24(K=1) or 2516P25(K=2)	2	X - 6
BCS	NPK P Branch on No Pocket Decision, Document Handler K	2514P22(K=1) or 2514P23(K=2)	2	X - 6
*BCS	NQK Branch on Document TCD Not Correct, Document Handler K	2516P32(K=1) or 2516P33(K=2)	2	X - 7

* This instruction is an optional feature

Mnemonic		Octal	Word Times	Page
BCS	PDK P Branch on Pocket Decision, Document Handler K	2516P22(K=1) or 2516P23(K=2)	2	X - 6
BCS	RPC P Branch on DSU Parity Correct	2516P26	2	XI - 10
BCS	RPE P Branch on DSU Parity Error	2514P26	2	XI - 10
BCS	SKC Branch on Document Handler K Correct	2516P30(K=1) or 2516P31(K=2)	2	X - 7
BCS	SKE Branch on Any Error, Document Handler K	2514P30(K=1) or 2514P31(K=2)	2	X - 7
BCS	SKN P Branch on Document Handler K Not Ready	2516P20(K=1) or 2516P21(K=2)	2	X - 6
BCS	SKR P Branch on Document Handler K Ready	2514P20(K=1) or 2514P21(K=2)	2	X - 6
BCS	VCK Branch on Valid Character, Document Handler K	2516P26(K=1) or 2516P27(K=2)	2	X - 7
BCS	XXX P Branch on Controller Selector	2514PCC or 2516PCC	2	VII - 2
BEV	Branch on Even	2516000	2	V - 32
BKW	T Backspace and Position Write Head	1600000 TT00000	2	VIII - 12
BMI	Branch on Minus	2514001	2	V - 32
BNN	Branch on N Register Not Ready	2516005	2	VI - 9
BNO	Branch on No Overflow	2516003	2	V - 32
BNR	Branch on N Register Ready	2514005	2	VI - 9
BNZ	Branch on Non-Zero	2516002	2	V - 32
BOD	Branch on Odd	2514000	2	V - 32
BOV	Branch on Overflow	2514003	2	V - 32
BPC	Branch on Parity Correct	2516004	2	V - 32
BPE	Branch on Parity Error	2514004	2	V - 32
BPL	Branch on Plus	2516001	2	V - 32
BPN	Branch on Card Punch Not Ready	2516007	2	VI - 37
BPR	Branch on Card Punch Ready	2514007	2	VI - 37
BRU	Y X Branch Unconditionally	2600000	1	V - 31
BSS(Pseudo)	Block Started by Symbol			IV - 15
BXH	K X Branch if X is Higher Than or Equal To	0500000	3	V - 34
BXL	K X Branch If X is Less Than	0400000	3	V - 35
BZE	Branch on Zero	2514002	2	V - 32
*CAB	Y Compare and Branch	2100000	2 to 4	V - 33
CHS	Change Sign of A Register	2504040	2	V - 23
CPL	Complement A	2504502	3	V - 23
DAD	Y X Double Decimal Add	1100000	3 3	V - 2 V - 8

* This instruction is an optional feature.

Mnemonic			Octal	Word Times	Page
DAD	Y	X	1100000	3	V - 2
	Double Length Add				
* DCB	Y		2200000	2 to 6	V -33
	Double Compare and Branch				
DDC (Pseudo)					IV -12
	Double Length Decimal				
DEC (Pseudo)					IV -11
	Decimal				
DLD	Y	X	1000000	3	V - 15
	Double Length Load				
DNO	K		2513200	2 to 12	V - 30
	Double Length Normalize				
DST	Y	X	1300000	3	V - 16
	Double Length Store				
DSU	Y		1200000	5	V - 8
	* Double Decimal Subtract				
DSU	Y	X	1200000	5	V - 3
	Double Length Subtract				
DVD	Y	X	1600000	26 to 29	V - 5
	Divide				
EJT (Pseudo)					IV - 17
	Eject Printer Paper				
END(Pseudo)					IV - 16
	End of Program				
EQO (Pseudo)					IV - 15
	Equals Octal				
EQU (Pseudo)					IV - 15
	Equals				
ERB	N		0520000(N=1) 0000000	2	X - 5
		or	1120000(N=2) 0000000		
	End Read Busy				
EXT	Y	X	2000000	3	V - 17
	Extract				

* This instruction is an optional feature.

Mnemonic		Octal	Word Times	Page
FAD	Y	31YYYYY	Min.162 usec Max.709 usec	XII- 7
	AAU Add			
FDC (Pseudo)				IV - 12
	Floating Point Decimal			
FDV	Y	36YYYYY	Min. 814.5 usec Max.1095 usec	XII - 8
	AAU Divide			
FLD	Y	3000000	72 usec	XII - 7
	Load Auxiliary Arithmetic Unit			
FMP	Y	35YYYYY	Min. 297 usec Max. 1062 usec	XII - 7
	AAU Multiply			
FST	Y	3300000	72 usec	XII - 7
	Store Auxiliary Arithmetic Unit			
FSU	Y	32YYYYY	Min. 162 usec Max. 709 usec	XII - 8
	AAU Subtract			
HCR		2500004	2	VI-26
	Halt Card Reader			
HLT	M N	0500000(N=1) or 1100000(N=2)	2	X - 5
	Halt Continuous Feeding			
HPT		2500016	2	VI - 17
	Halt Paper Tape Reader			
INX	K X	1400000	3	V -34
	Increment X			
*LAC		2504202	3	V -20
	Load A Register from C Register			
LAQ		2504001	3	V -19
	Load A from Q			
LAQ	A	3600002	49.5 usec	XII - 7
	Load AX from QX			
*LCA		2504210	3	V -20
	Load C Register from A Register			
LDA	Y X	0000000	2	V -14
	Load A Register			

Mnemonic		Octal	Word Times	Page
LDO		2504022	3	V - 22
	Load One into A Register			
LDX	Y X	0600000	3	V - 36
	Load X			
LDZ		2504002	3	V - 22
	Load Zero into A Register			
LMO		2504102	3	V - 23
	Load Minus One into A Register			
LOC (Pseudo)				IV - 15
	Location in Octal			
LQA		2504004	3	V - 19
	Load Q from A			
LQA	A	3200002	49.5 usec	XII- 7
	Load QX From AX			
LST (Pseudo)				IV - 17
	List			
MAL (Pseudo)				IV - 11
	Multiple Alphanumeric			
MAQ		2504006	3	V - 19
	Move A to Q			
MAQ	A	3100002	49.5 usec	XII- 7
	Move AX to QX			
*MOV	Y	2400000	4 + 2N	V - 18
	Move			
MPY	Y X	1500000	9 to 23	V - 5
	Multiply			
NAL (Pseudo)				IV - 10
	Negative Alphanumeric			
NAM (Pseudo)				IV - 17
	Print Name or Title on Each Page			
NAQ	K	2511100	2 to 12	V - 29
	Shift N, A, and Q Right			
NEG		2504522	3	V - 23
	Negate A			
NLS (Pseudo)				IV - 17
	No List.			

* This instruction is an optional feature.

Mnemonic		Octal	Word Times	Page
NOP		2504012	3	V - 24
	No Operation			
NOR K		2513000	3 to 12	V - 29
	Normalize the A Register			
OCT (Pseudo)				IV - 13
	Octal			
OFF		2500005	2	VI - 8
	Power Off (Direct I/O Devices)			
ORG (Pseudo)				IV - 14
	Origin			
ORY	Y X	2300000	3	V - 17
	Or A into Y			
PAL (Pseudo)				IV - 11
	Multiple Alphanumeric for Printer with Print Line Indicator			
PKT	X N	0460000(N=1) or 1060000(N=2)	2	X - 4
	Pocket Select			
PLD (Pseudo)				IV - 16
	Punch Loader Cards			
PON		2500015	2	VI - 18
	Punch On			
PRF OCT (MRADS Address) MMMMMMM	F	2500000	2	XI - 4
	Position MRADS File			
RAW (blank)	N F zero	1202000 0000000	2	XI - 7
	Read After Write Check			
RBB (blank)	M T N	15MMMMM TTNNNNN	2	VIII- 12
	Read Backward Binary			
RBD (blank)	M T N	14MMMMM TTNNNNN	2	VIII- 12
	Read Backward Decimal			
RBS (blank)	M T N	35MMMMM TTNNNNN	2	VIII- 12
	Read Backward Special Binary			

Mnemonic		Octal	Word Times	Page
RCB		250YY01	2	VI - 27
	Read Cards Binary			
RCD	Y	250YY00	2	VI - 26
	Read Cards Decimal			
RCF	Y	250YY10	2	VI - 28
	Read Cards Full			
RCM	Y	250YY12	2	VI - 29
	Read Cards Mixed			
RCS		2500011	2	VI - 6
	Read Control Switches			
RDC	M N	0440000(N=1) or 1040000(N=2)	2	X - 4
	Read Document Continuously			
REM (Pseudo)				IV - 16
	Remarks			
RON		2500014	2	VI - 16
	Paper Tape Reader On			
RPT		2500006	2	VI - 16
	Read Paper Tape			
RRD	N F	1201000	2	XI - 7
(blank)	M	00MMMMM		
	Read from MRADS Unit F			
RRF	N F	1200000	2	XI - 6
(blank)	M	00MMMMM		
	Read from MRADS Unit F			
RSD	M N	0420000(N=1) or 1020000(N=2)	2	X - 3
	Read Document Single			
RTB	M T	05MMMMM	2	VIII- 11
(blank)	N	TTNNNNN		
	Read Tape Binary			
RTD	M T	04MMMMM	2	VIII- 11
(blank)	N	TTNNNNN		
	Read Tape Decimal			
RTS	M T	25MMMMM	2	VIII- 11
(blank)	N	TTNNNNN		
	Read Tape Special Binary Mode			
RWD	T	2000000 TT00000	2	VIII- 12
	Rewind			
SAN	K X	2510400	2 to 12	V - 28
	Shift A and N Right			
SBO		2504112	3	V - 4
	Subtract One			
SBO		2504112	3	V - 9
	*Subtract One Decimal			
SBR (Pseudo)				IV - 14
	Subroutine Call			
SCA	K X	2510040	2 to 12	V - 27
	Shift Circular A Register			
SCD	K X	2511200	2 to 12	V - 27
	Shift Circular Double			
SEL	P X	2500P20	2	VII - 2
	Select			
SEQ (Pseudo)				IV - 17
	Check Source Program Card Sequence Numbers			
SET BINMODE		2506012	2	V - 10
	Set Binary Mode			
SET DECMODE		2506011	2	V - 9
	Set Decimal Mode			
SET FIXPOINT		3500010	49.5 usec	XII - 6
	Set Fixed-Point Mode			
SET NFLPOINT		3100010	49.5 usec	XII - 6
	Set Normalized Floating-Point Mode			
SET	PBK	2506016	2	VII - 3
	Set Automatic Priority Interrupt Off			
SET	PST	2506015	2	VII - 3
	Set Automatic Priority Interrupt On			
SET UFLPOINT		3200010	49.5 usec	XII - 6
	Set Unnormalized Floating-Point Mode			
SLA	K X	2512000	2 to 12	V - 25
	Shift Left A Register			
SLD	K X	2512200	2 to 12	V - 26
	Shift Left Double			
SLT	K	0X00000 XX00000	2	IX - 6
	Slew Paper to Tape Punch			

Mnemonic			Octal	Word Times	Page
SLW	N		0600000 NN00000	2	IX - 6
	Slew Paper N Lines				
SNA	K	X	2510100	2 to 12	V - 28
	Shift N and A Right				
SPB	Y	X	0700000	2	V - 31
	Store P and Branch				
SRA	K	X	2510000	2 to 12	V - 24
	Shift Right A Register				
SRD	K		2511000	2 to 12	V - 26
	Shift Right Double				
STA	Y	X	0300000	2	V - 15
	Store A				
STO	Y	X	2700000	3	V - 16
	Store Operand Address				
STX	Y	X	1700000	3	V - 36
	Store X				
SUB	Y	X	0200000	3	V - 2
	Subtract				
SUB	Y	X	0200000	3	V - 7
	*Decimal Subtract				
*SXG	Y		2506003	2	V - 36
	Select X Register Group				
TCD(Pseudo)					IV - 15
	Punch Transfer Card				
TON			2500007	2	VI - 8
	Typewriter On				
TYP			2500006	2	VI - 8
	Type				
WCB	Y		250YY03	2	VI - 37
	Write Card Binary				
WCD	Y		250YY02	2	VI - 37
	Write Card Decimal				
WCF	Y		250YY17	2	VI - 37
	Write Cards Full				
WEF		T	0200000 TT00000	2	VIII - 11
	Write End of File				
WFL		N	36YYYYY	2	IX - 6
(WPL)	Y	X	01XXXXX		
	Write Format Line				
WPL	Y	N	2000000 01YYYYY	2	IX - 5
	Write Print Line				
WPT			2500006	2	VI -18
	Write Paper Tape				
WRD	N	F	3701000	2	XI - 7
(blank)	M		00MMMMM		
	Write on MRADS Unit F				
WRF	N	F	3700000	2	XI - 7
(blank)	M		00MMMMM		
	Write on MRADS Unit F				
WTB	M	T	03MMMMM	2	VIII - 10
(blank)	N		TTNNNNN		
	Write Tape Binary				
WTD	M	T	02MMMMM	2	VIII - 10
(blank)	N		TTNNNNN		
	Write Tape Decimal				
WTS	M	T	23MMMMM	2	VIII - 11
(blank)	N		TTNNNNN		
	Write Tape Special Binary Mode				
XAQ			2504005	3	V - 20
	Exchange A and Q				
XAQ	A		3500002	117 usec	XII- 7
	Exchange AX and QX				
Z (Pseudo)					IV - 13
	Octal Operation Code				

* This instruction is an optional feature.

Progress Is Our Most Important Product

GENERAL ELECTRIC

COMPUTER DEPARTMENT • PHOENIX, ARIZONA

CPB-252

Litho in U.S.A.