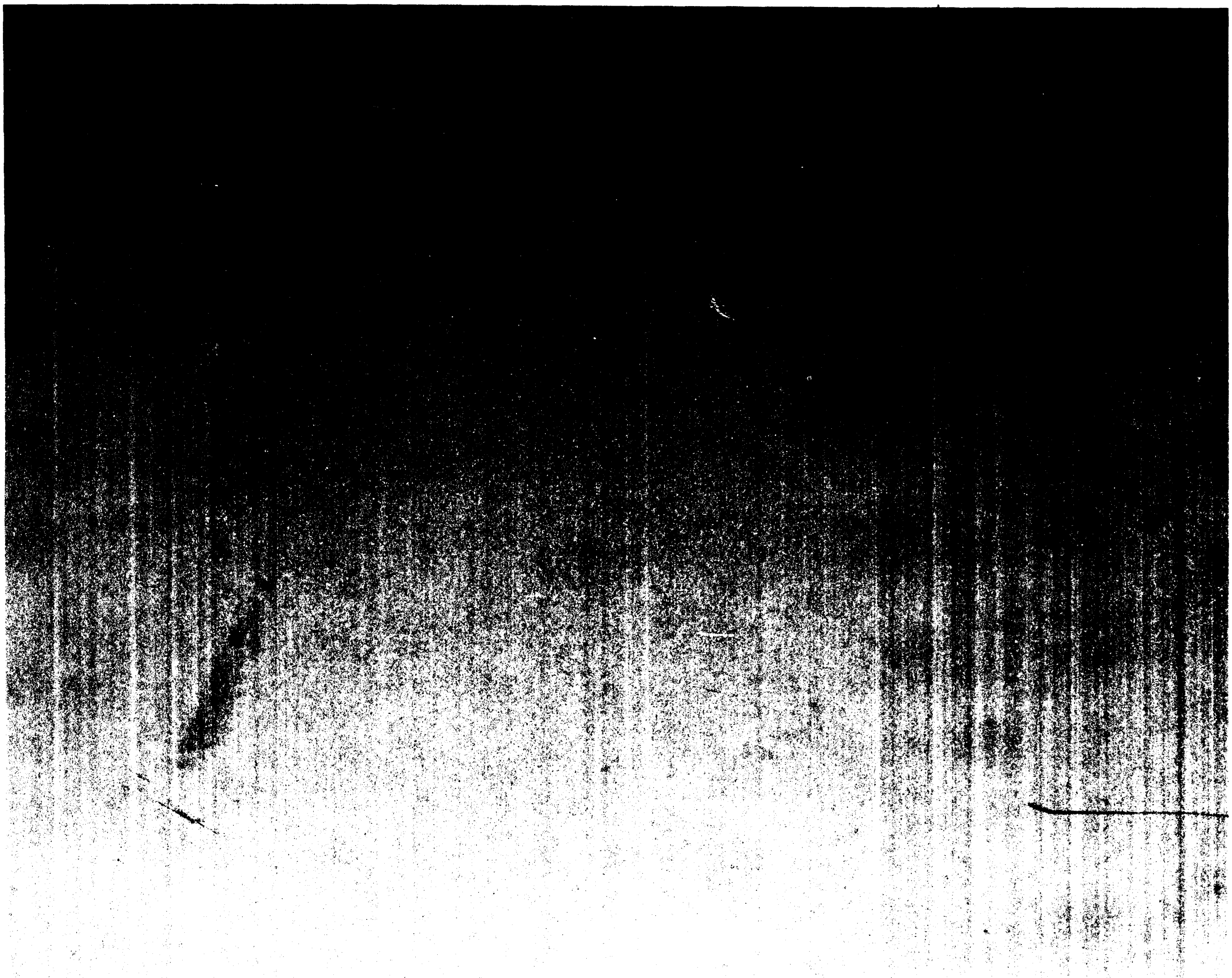
GENERAL ELECTRIC
COMPUTERS

GE-225 Programming Reference Manual



GENERAL ELECTRIC

CPB-252B

GE-225
PROGRAMMING
REFERENCE MANUAL

October 1963

Rev. June 1966

GENERAL ELECTRIC

INFORMATION SYSTEMS DIVISION

PREFACE

The GE-225 Programming Reference Manual has been prepared both as a reference manual for programming the GE-225 information processing system and as a training aid. It includes a brief description of the major components of the system, machine language and number systems, central processor and console typewriter operations, controller selector operations, programming conventions, and an octal and alphabetical listing of General Assembly Program instructions.

The information in this manual also applies essentially to the GE-205 and GE-215 systems. Equivalent coverage for the GE-235 is included in the GE-235 Central Processor Reference Manual, CPB-374.

This manual is a condensed version of an earlier edition which also contained information on the General Assembly Program and the various peripheral subsystems used with the GE-225. Separate manuals have been published to cover these subjects, as listed below:

Subject	Manual Title and Publication No.
General Assembly Program	GE-200 Series General Assembly Program II (Publication No. CPB-1180)
400-cpm Card Reader 1000-cpm Card Reader 100-cpm Card Punch 300-cpm Card Punch	GE-200 Series Punched Card Subsystem Reference Manual (Publication No. CPB-302)
15- & 15/41Kc. Magnetic Tape Subsystems	GE-200 Series Magnetic Tape Subsystems Reference Manual (Publication No. CPB-339)
Paper Tape Reader/Punch	GE-200 Series Paper Tape Subsystem Reference Manual (Publication No. CPB-308)
900-Lpm On-Line High Speed Printer	GE-200 Series High Speed On-Line Printer Reference Manual (Publication No. CPB-321)
900-Lpm Off/On-Line High Speed Printer	GE200 Series High Speed Off-Line/On-Line Printer Reference Manual (Publication No. CPB-1075)
12-Pocket Document Handler (1200-dpm)	GE-225/235 Document Handler Reference Manual (Publication No. CPB-307)

Suggestions and criticisms relative to form, content, purpose, or use of this manual are invited. Comments may be sent on the Document Review Sheet in the back of this manual or may be addressed directly to Engineering Publications Standards, B-90, Computer Equipment Department, General Electric Company, 13430 North Black Canyon Highway, Phoenix, Arizona 85029.

CONTENTS

APPENDIXES

ILLUSTRATIONS

1. THE GE - 225 INFORMATION PROCESSING SYSTEM

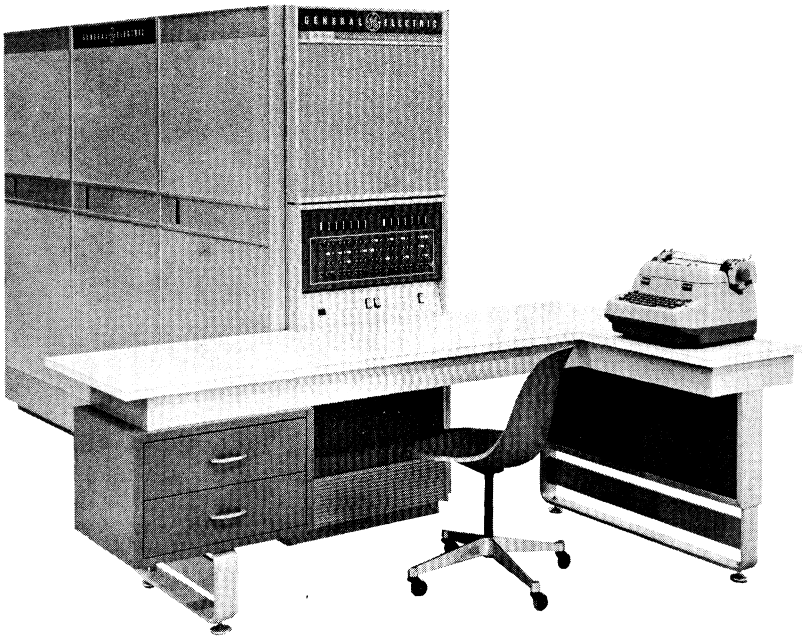
The GE-225 Information Processing System is a medium-scale, general-purpose digital computer that permits an integrated approach to the total information processing needs of business, government, and science, while providing an economical means of processing large volumes of data at high speed.

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.

* DATANET is a registered trademark of the General Electric Company.



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 the manuals listed in the Preface.

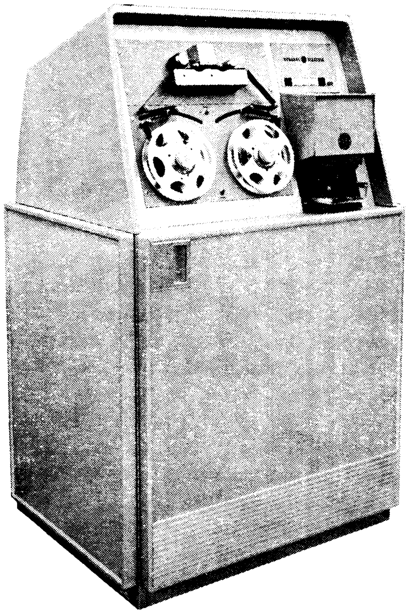
Central Processor

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.

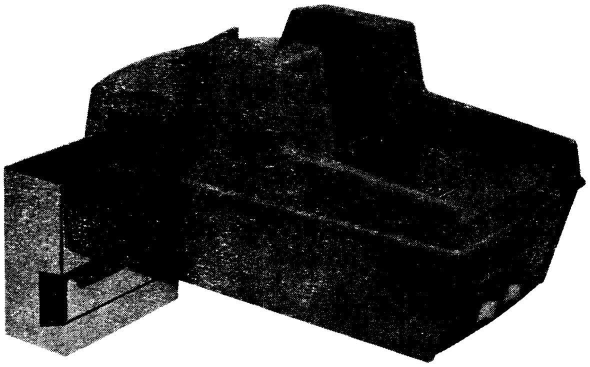
Control Console

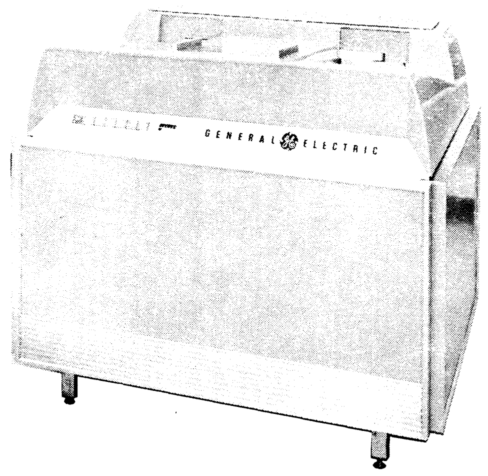
The GE-225 Control Console, attached to the central processor, provides manual control of operations, visual display of the contents of appropriate registers, program monitoring facilities for the operator, and typed output via the console typewriter, under program control. From the console, the operator 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.



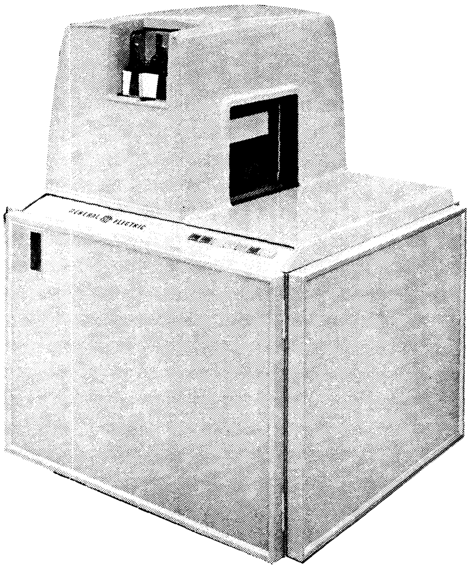


Card Reader

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 300 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.

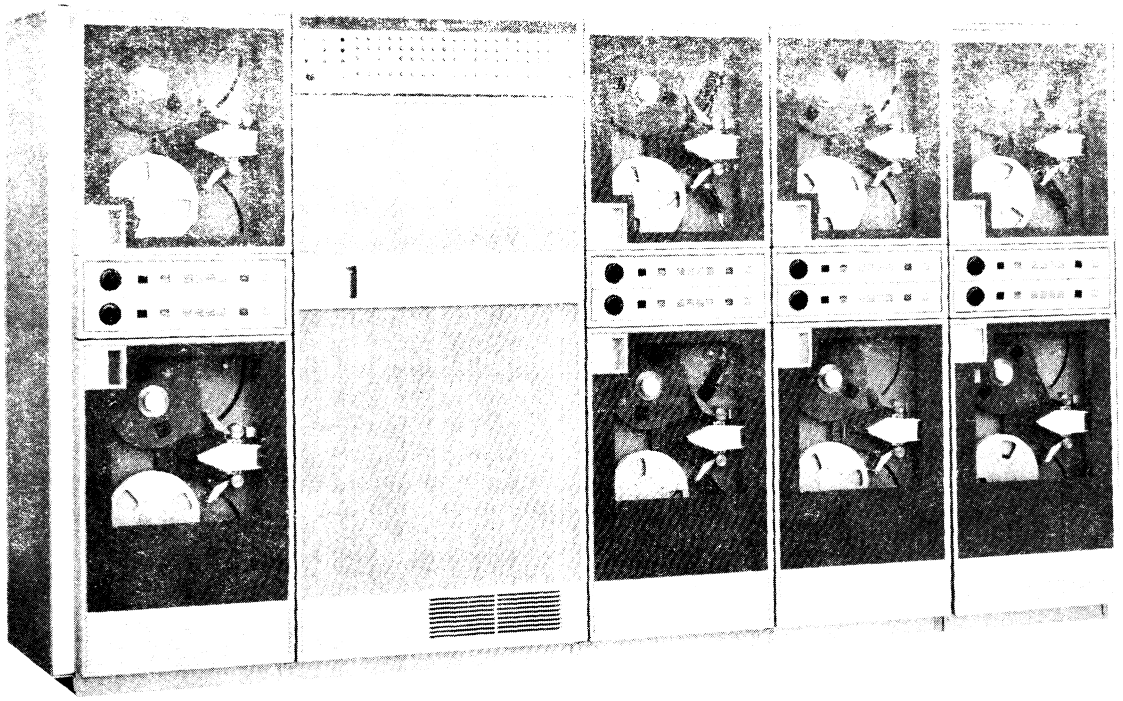
The card punch is primarily an on-line peripheral and receives basic control signals from the central processor. However, gang punching, or duplication of many cards from a master card, can be performed off-line.

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.

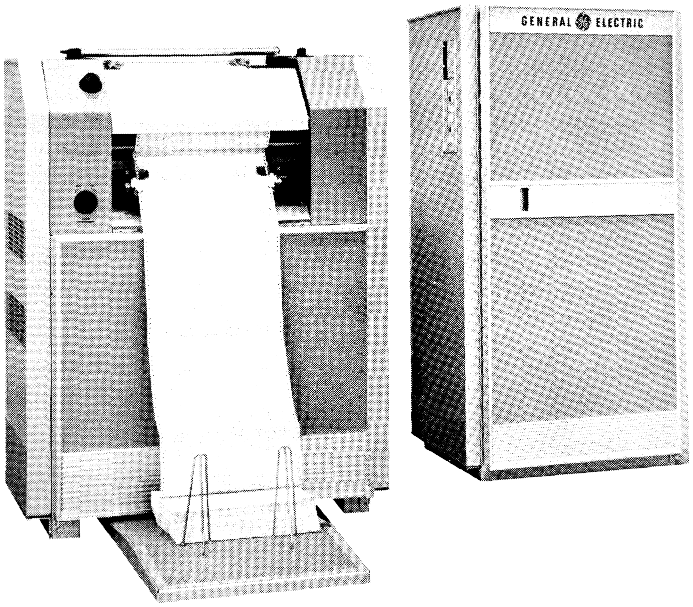


Magnetic Tape

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 <u>magnetic tape subsystem.</u> A subsystem of one tape controller and multiple tape handlers permits reading <u>or</u> writing concurrently with other operations. A subsystem containing two or more tape controllers permits reading <u>and</u> writing simultaneously with other operations.

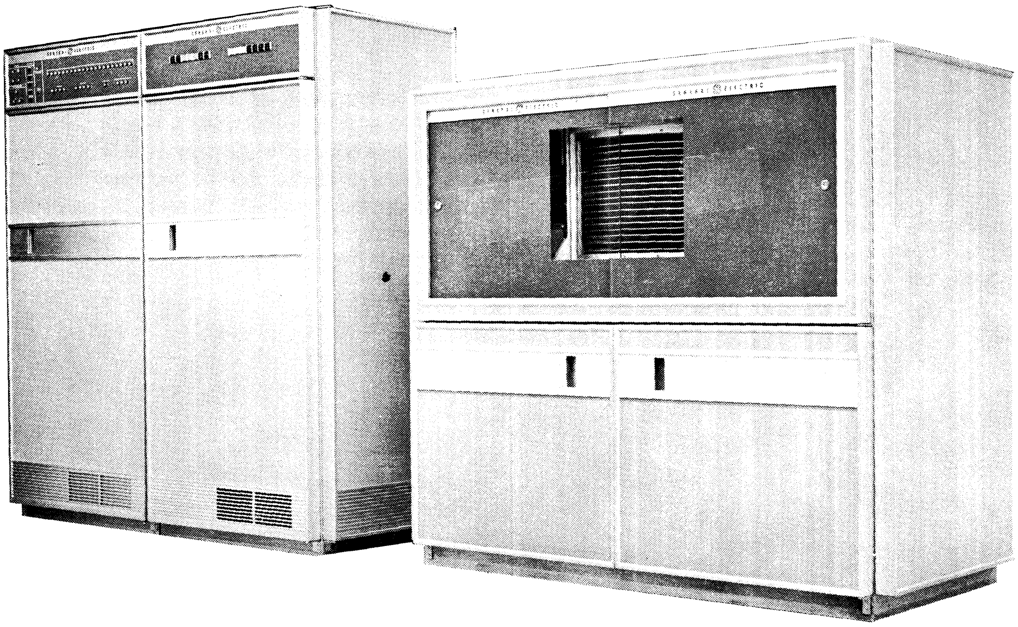


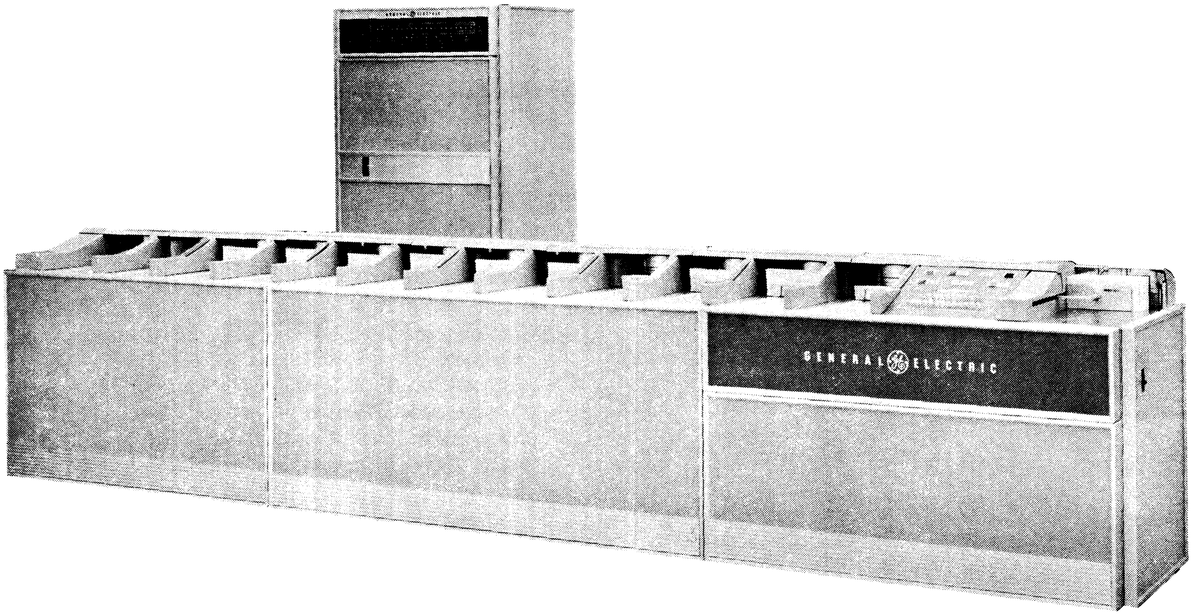
High Speed Printer

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.

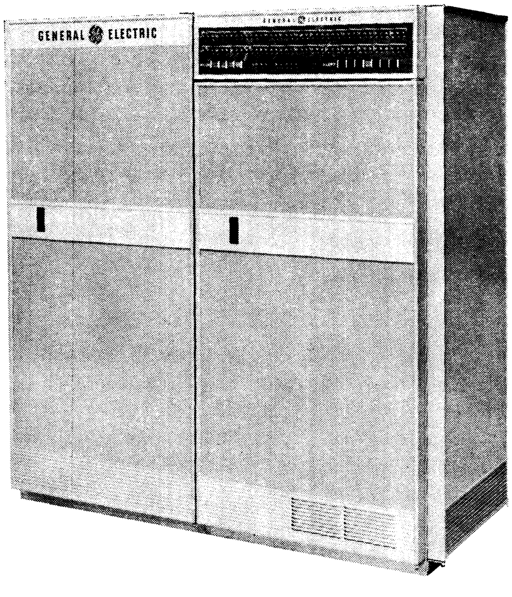
Disc Storage Unit

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.

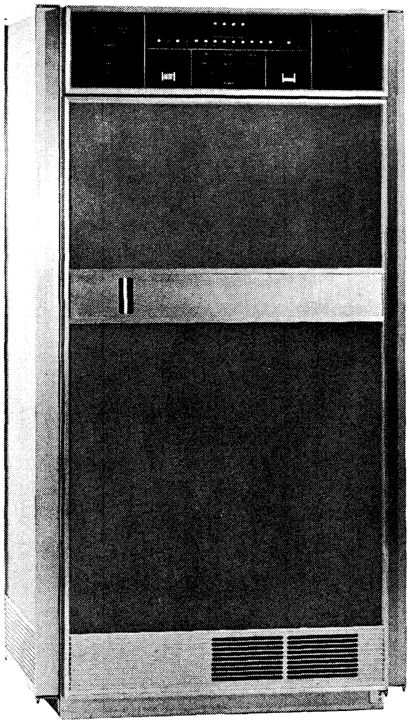


12-Pocket Document Handler

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 facility 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.



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).

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. GE-225 System Components

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 unloaded 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

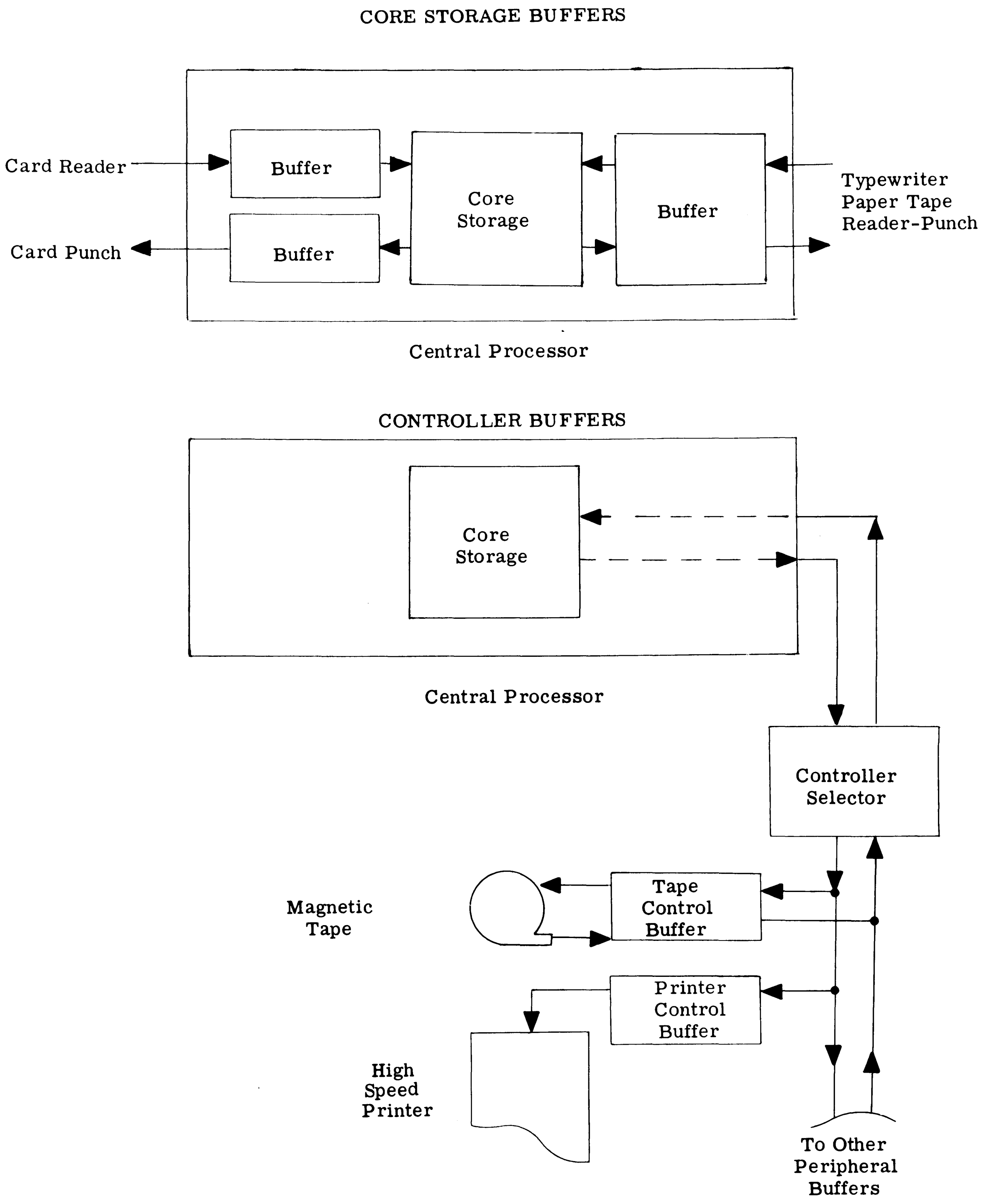


Figure 2. Central Processor and Controller Buffers

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 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, disc storage 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 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.

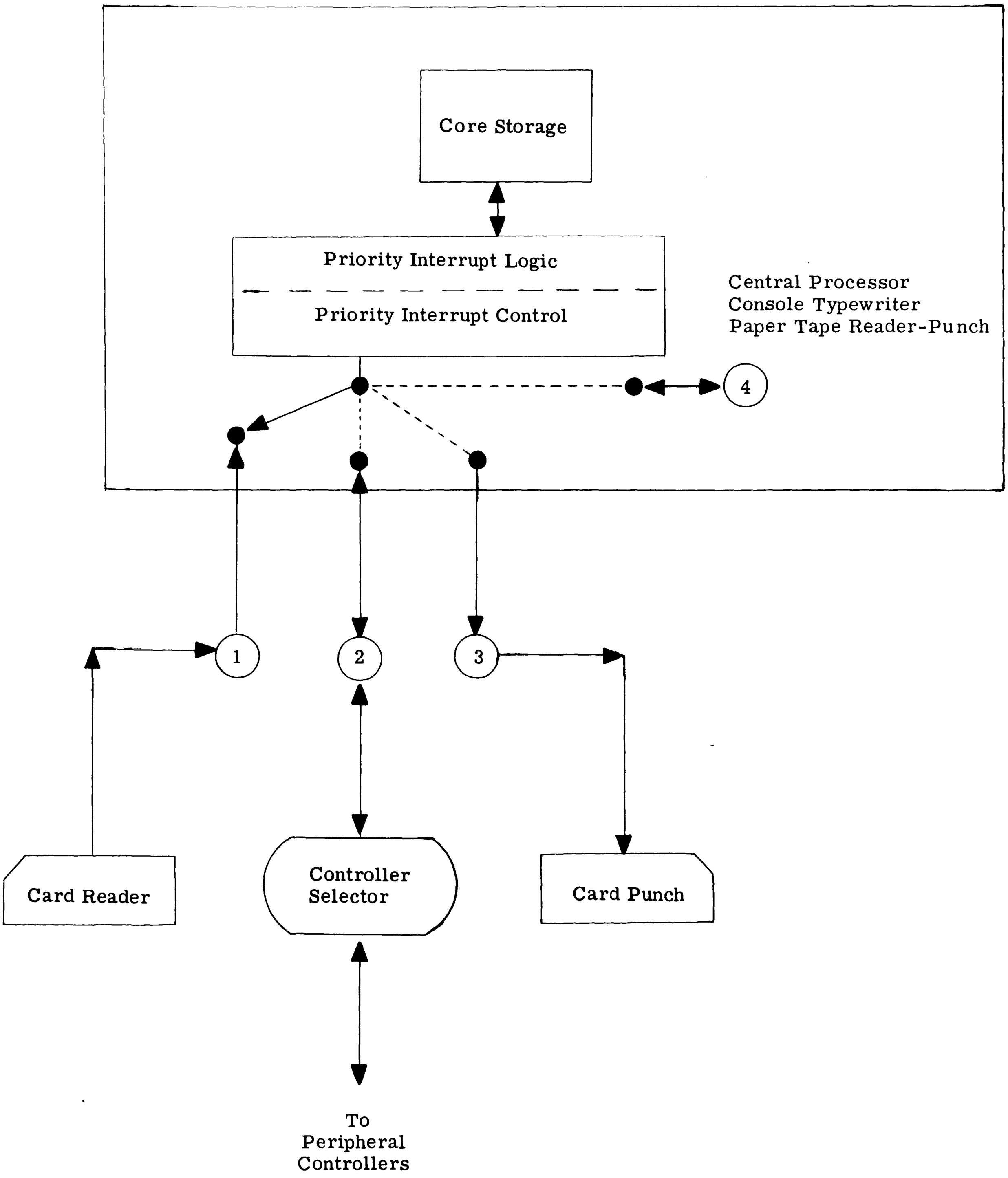


Figure 3. GE-225 Priority Access System

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 4 illustrates one possible system configuration. Smaller or different configurations are also possible.

In Figure 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 5 is an expansion of the priority interrupt control system shown previously in Figure 3. This diagram futher 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 a DSU.

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

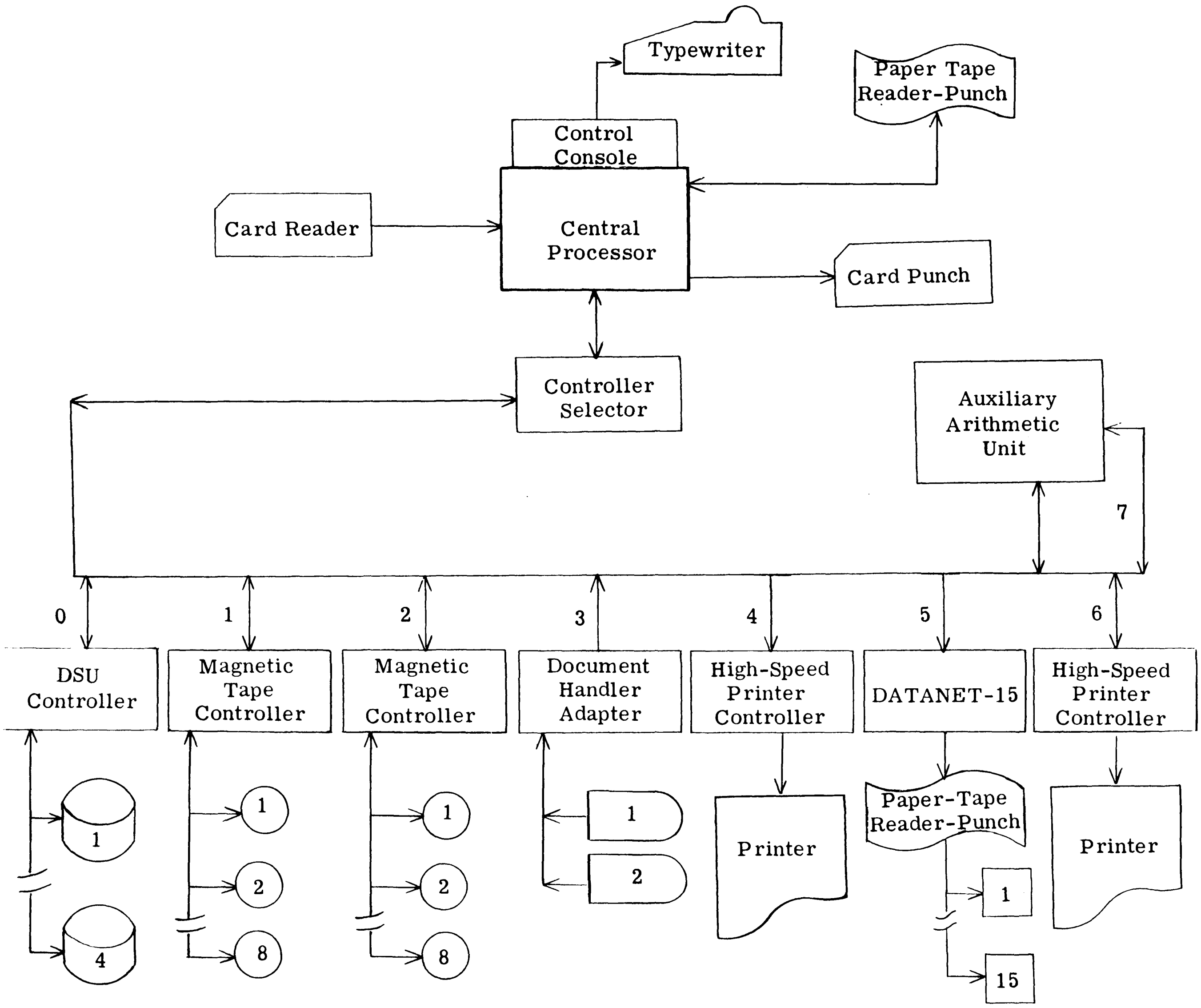


Figure 4. Large GE-225 System Configuration

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.

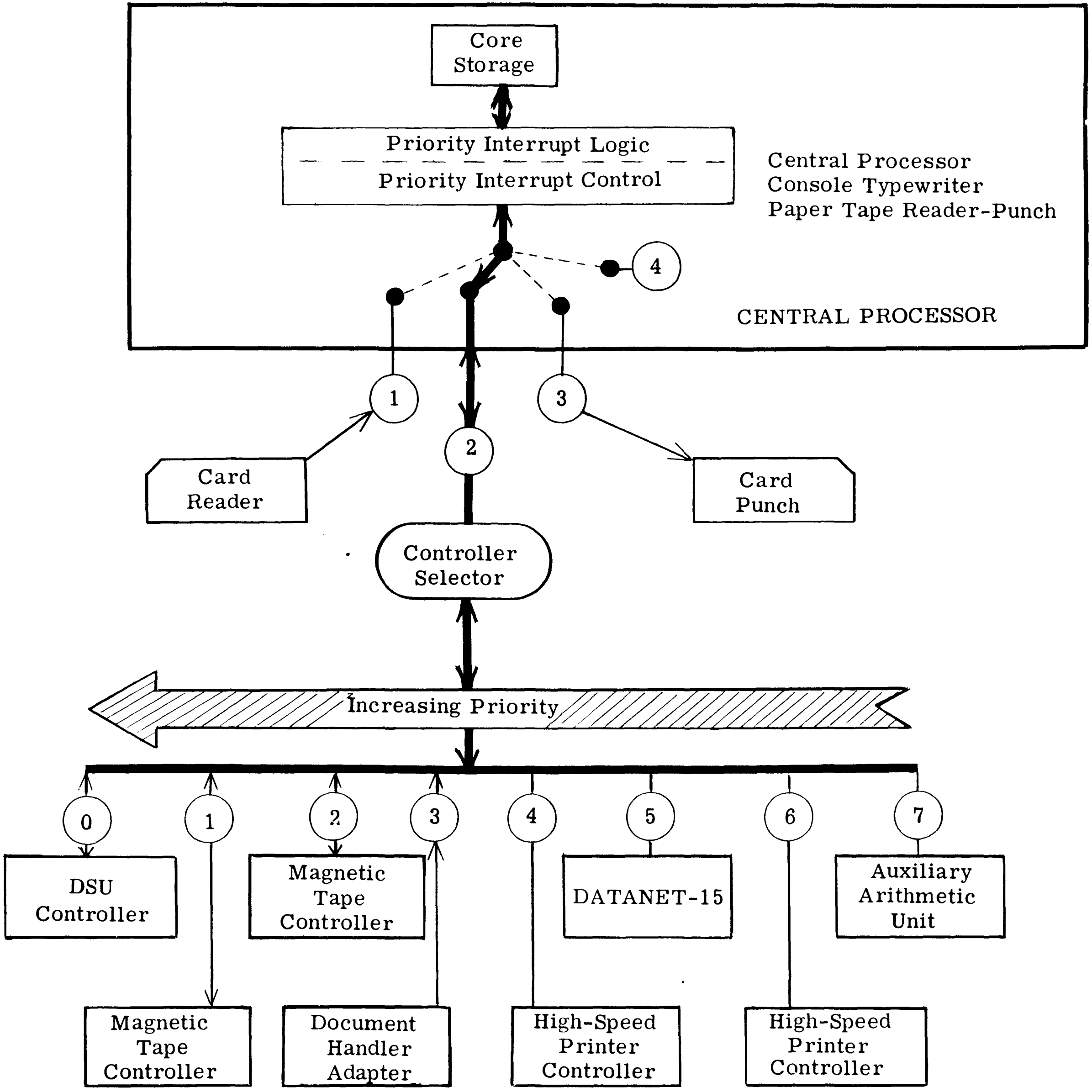


Figure 5. Controller Selector Priority

2. 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 6. Other arithmetic operations are similarly easy.

+	0	1
0	0	1
1	1	10

Figure 6. **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 x 8^3 + 3 x 8^2 + 7 x 8^1 + 6 x 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 7.

Octal Digits

	+	0	1	2	3	4	5	6	7
Octal Digits	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

Figure 7. 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.

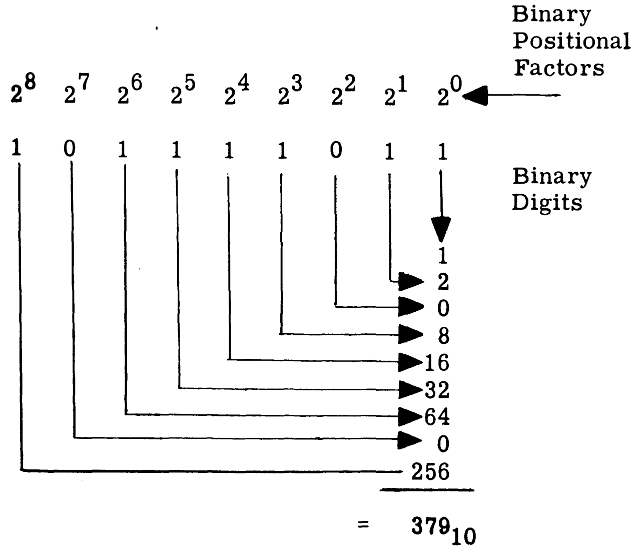
$$
\begin{array}{rrl}
 & \underline{100101} & \\
1010 \,\big) & 101111011 & \\
 & \underline{1010} & \\
 & 1110 & \\
 & \underline{1010} & \\
 & 10011 & \\
 & \underline{1010} & \\
 & 1001 & = 9_{10} \longrightarrow \text{9 units digit}
\end{array}
$$

$$
\begin{array}{rrl}
 & \underline{11} & \\
1010 \,\big) & 100101 & \\
 & \underline{1010} & \\
 & 10001 & \\
 & \underline{1010} & \\
 & 111 & = 7_{10} \longrightarrow \text{79 tens and units digits}
\end{array}
$$

$$
\begin{array}{rrl}
 & \underline{0} & \\
1010 \,\big) & 11 & \\
 & \underline{0} & \\
 & 11 & = 3_{10} \longrightarrow \text{379 hundreds, tens, and unit digits.}
\end{array}
$$

Another method would be simply to look up the decimal equivalents of the corresponding powers of two in the table shown in Figure 8 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 8. 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.

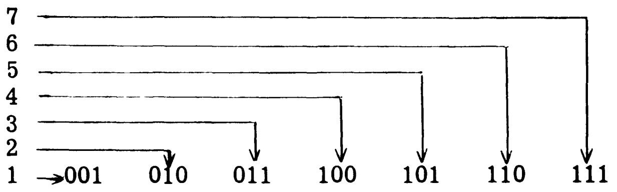
Example: Convert 101001110111001_2 into octal notation.

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.

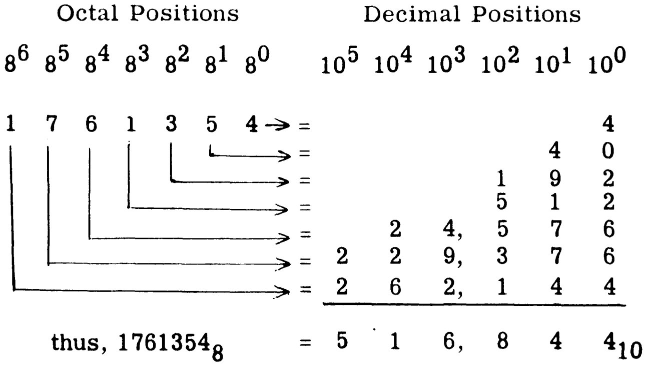


Octal-to-Decimal Conversion

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 9 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 9. 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 10. 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 10. 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
 |    |    |    |    L------>=                    1    0   6
 |    |    |    L----------->=               1    6    0   4
 |    |    L---------------->=          1    1    6    1   0
 |    L--------------------->=     1    1    6    1    0   0
 L-------------------------->= 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 disc storage 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 11. 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.

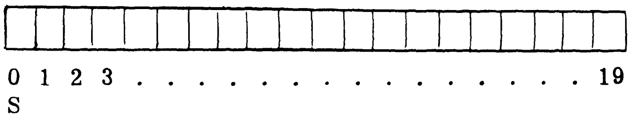


Figure 11. Basic GE-225 Word

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:

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

S 1 2 3 19

Inverting all bit positions gives:

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

S 1 2 3 19

Adding a 1-bit to bit position 19:

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

S 1 2 3 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:

																	2^{21}		2^{19}
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

S 1 2 3 19

Memory Location 1

			2^{16}		2^{14}			2^{11}			2^{8}			2^{5}					2^{0}
0	0	1	0	1	1	1	0	1	1	1	1	1	1	0	1	0	0	1	1

S 1 2 3 19

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

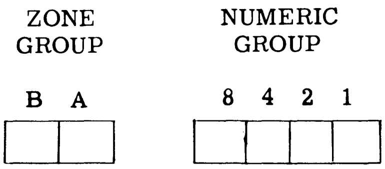
The auxiliary arithmetic unit (AAU) expands the arithmetic capability of the GE-225 to include normalized and unnormalized floating-point operations. Representation of floating-point numbers is discussed in the section, Auxiliary Arithmetic Unit Operations.

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:

		B	A	8	4	2	1	B	A	8	4	2	1	B	A	8	4	2	1
0	0	0	1	0	0	1	0	0	0	0	1	1	0	0	0	0	0	1	0
S	1	B						6						2					

- end of [illegible]

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:

DO THIS	X X	WITH DATA LOCATED HERE
0 4	5 6	7 19

OR

OPERATION CODE	X X	OPERAND ADDRESS
0 4	5 6	7 19

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 Chapter 4.

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 Sthrough 4 (10 101, or 25_8). Format II has three variations, corresponding to the three general categories mentioned.

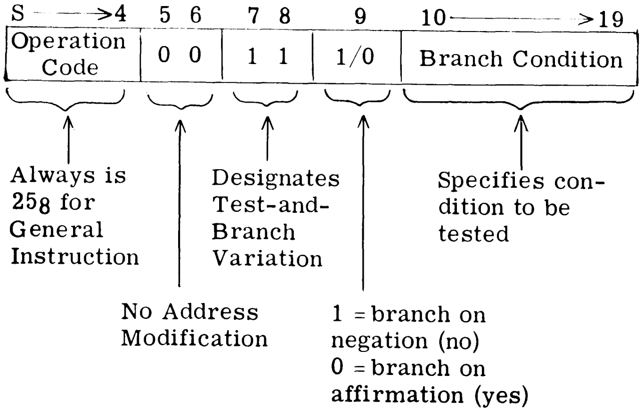
The word movement variation is for instructions involving full word transfers between arithmetic registers and the arithmetic unit. They assume this format:

S → 4	5 6	7 8	9 → 19
Operation Code	0 0	0 1	Specifies Exact Operation
Always is 25_8 for General Instruction	No Address Modification	01 indicates Word Movement Variation	Interpretation of these bits is explained under 'Micro-programming'

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:

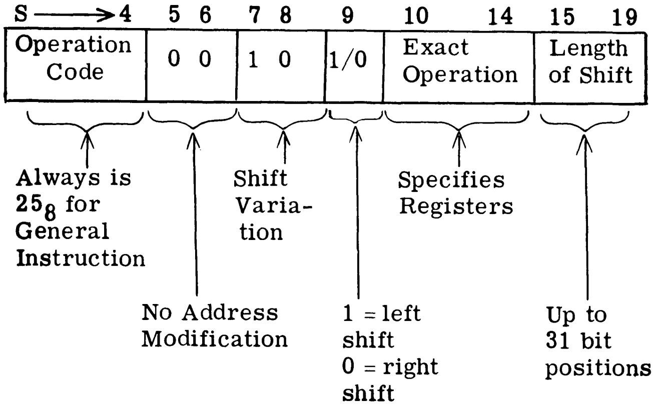
S → 4	5 6	7 8	9 → 13	14 → 19
Operation Code	0 0	0 0	Starting Address	Specific Operation
Always is 25_8 for General Instruction	No Address Modification	Designates Input-Output Variation	Either a memory location or peripheral controller address	Designates the specific input-output operation

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:


S ——>4
5 6
7 8
9
10——————>19
Operation Code
0 0
1 1
1/0
Branch Condition
Always is 25₈ for General Instruction
Designates Test-and-Branch Variation
Specifies condition to be tested
No Address Modification
1 = branch on negation (no)
0 = branch on affirmation (yes)

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 assembly by the General Assembly Program) 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:

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

S 1 19

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:

01	101	011	100	101	100	111	←Bits
①	②	③	④	⑤	⑥	⑦	←Octal Group No.

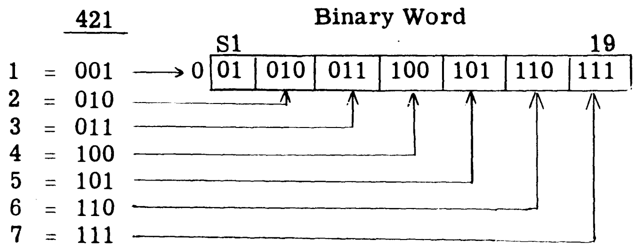
Evaluate each group and write the equivalent octal digit:

$$\left.\begin{array}{lrcl} \textcircled{1} & 01 & = & 1 \\ \textcircled{2} & 101 & = & 5 \\ \textcircled{3} & 011 & = & 3 \\ \textcircled{4} & 100 & = & 4 \\ \textcircled{5} & 101 & = & 5 \\ \textcircled{6} & 100 & = & 4 \\ \textcircled{7} & 111 & = & 7 \end{array}\right\} = 1534547_8$$

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

Programs for the GE-225 information processing system are generally written in symbolic coding. The programmer is thus able to write instructions in meaningful symbolic codes, rather than the absolute numeric code language of the computer. This relieves him of much time-consuming clerical detail, especially important in writing lengthy programs.

The General Assembly Program

The General Assembly Program transforms symbolic mnemonic codes into numeric machine language for each instruction in the repertoire of the GE-225 system. These mnemonic codes have been chosen to provide significance and easy recognition of the operation performed. For example, the mnemonic code "ADD" instructs the General Assembly Program to build a numeric instruction by which the GE-225 system performs algebraic addition.

The General Assembly Program is comprised of two parts:

1. The symbolic language used by the programmer in coding the source program.
2. The actual assembly program (on punched cards, perforated tape, or magnetic tape) supplied by General Electric that processes the source (or symbolic) program into a ready to execute machine language (or object) program.

The symbolic language consists of these standardized mnemonic codes divided into two general categories:

1. The pseudo-instructions used by the General Assembly Program for memory location assignments, program control constants, program constant storage, and program control during the assembly operation. These do not correspond to "real" GE-225 machine instructions.
2. The mnemonic operation codes corresponding to the more than 300 machine instructions of the GE-225 system.

There generally is a one-to-one relationship between the mnemonic operation code prepared by the programmer and the machine instruction appearing in the object program as assembled by the General Assembly Program. A single pseudo-instruction, however, can result in the generation of from one to several machine instructions during the assembly operation. The pseudo-instructions are described in a separate manual, GE-200 Series General Assembly Program II, CPB-1180, which also discusses all phases of the assembly operation and operating procedures.

The machine instructions for the GE-225 central processor are described in Chapters 4, 5, and 6 of this manual. Instructions for the various peripheral subsystems are described in the separate manuals covering these subsystems, as listed in the Preface.

A complete, brief listing of General Assembly Program instructions in both alphabetical and octal order is given in Appendixes A and B.

MICROPROGRAMMING

The flexibility of the instruction repertoire is further enhanced by the addition of a feature known as microprogramming. Microprogramming is the building of a computer instruction under programmer control by the specification of a series of elementary operations. In the table, Figure 12, a 1-bit in any of the labeled bit positions results in the elemental action described therein when the instruction is executed.

For example, a 1 in bit positions 10 and 11 of the "general" Shift Right instruction instructs the computer to take the actions $A_{\overline{19}} \rightarrow Q_1$ and $A_{\overline{19}} \rightarrow N_1$. The octal operation code for this specific command is 2511400. Reference to the octal listing in the Appendix shows this to be an ANQ (Shift A into N and Q) command. The instruction repertoire describes ANQ on the following page.

> The contents of Register A (1-19) are shifted K places to the right into both Register N and Register Q. Bits shifted out of Register A (19) enter both Register Q (1) and Register N (1). Bits shifted out of Register N (6) and Register Q (19) are lost. If the sign of Register A is plus, the vacated positions of Register A are filled with zeros; if the sign of Register A is minus, ones fill the vacated positions of Register A.

In addition, the programmer can create instructions which are not listed in the instruction repertoire. For example, a 1-bit in bit positions 12 and 13 of the general instruction Shift Right shifts the contents of Q into A ($Q_{19} \longrightarrow A_1$) and N into A ($N_6 \longrightarrow A_1$). The bits from N and Q are logically added as they shift into A. The octal code for this created instruction would be 2510300.

This information on microprogramming is included only for the use of the advanced programmer who desires to create his own special instructions. Normal programming will employ only the mnemonic or octal codes that have been assigned to the most common combinations of "micro" operations.

S 1 2 3 4	5 6	7	8	9	10	11	12	13	14	15	16	17	18	19
INSTRUCTION	X	SUB INSTRUCTIONS & ADDRESSES												
1 0 1 0 1														
SHIFT RIGHT		1	0	0	$A_{19} \rightarrow Q_1$	$A_{19} \rightarrow N_1$	$Q_{19} \rightarrow A_1$	$N_6 \rightarrow A_1$	$A_{19} \rightarrow A_1$	LENGTH OF SHIFT				
SHIFT LEFT		1	0	1	NORMALIZE		$Q_1 \rightarrow A_{19}$			LENGTH OF SHIFT				
WORD MOVEMENT (REGISTER TRANSFER)		0	1			FIRST PART A→B		SECOND PART B→B	CHS	ADD ONE	A→A. U.	A→Q	A. U.→A	Q→A
INPUT/OUTPUT		0	0	DATA ORIGIN MULTIPLE OF 128							DECODE			
PRIORITY CONTROL		0	0		CONTROLLER ADDRESS					1				
BRANCH TRUE		1	1	0						1	DECODE			
BRANCH FALSE		1	1	1						1	DECODE			
PRIORITY CONTROL BRANCH		1	1	0 or 1	CONTROLLER ADDRESS					1	DECODE			

Figure 12. Table of Elemental Instructions

3. 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 13 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

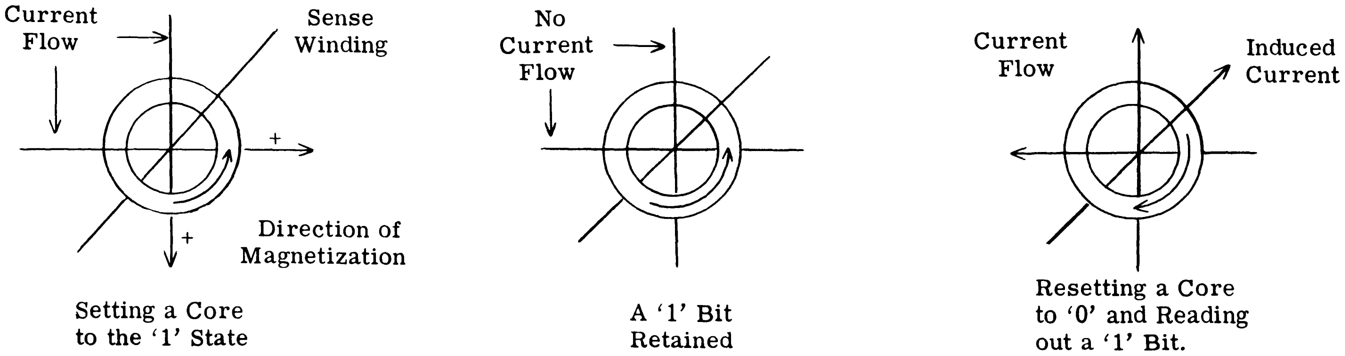

Current Flow
Sense Winding
+
+
Direction of Magnetization
Setting a Core to the '1' State
No Current Flow
A '1' Bit Retained
Current Flow
Induced Current
Resetting a Core to '0' and Reading out a '1' Bit.

Figure 13. 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 microseconds (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.

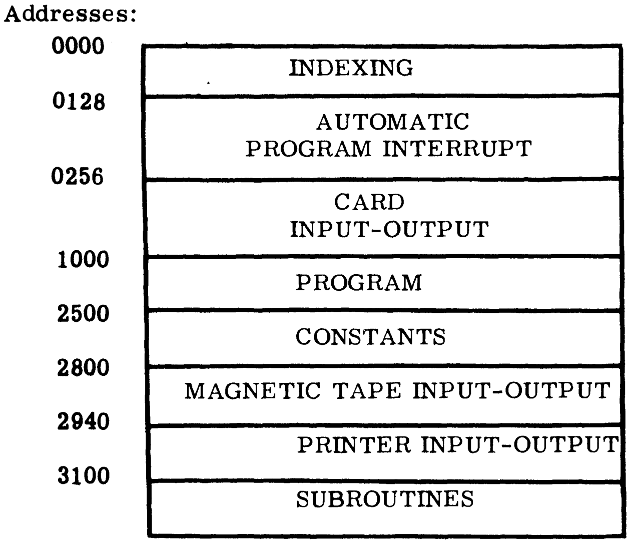


Figure 14. Representative Allocation of Memory

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 14 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 15). 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 16)

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)

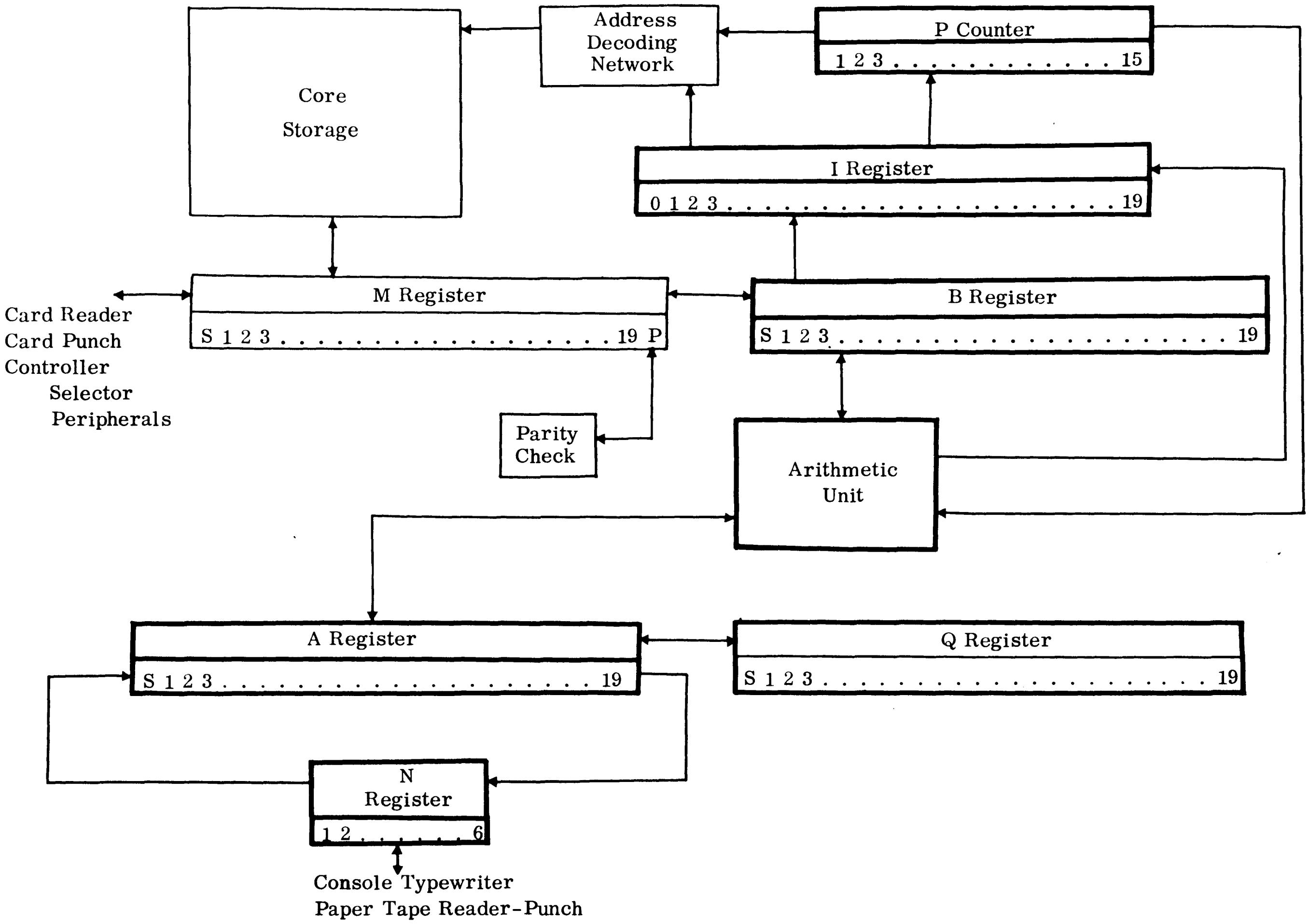


Figure 15. GE-225 Arithmetic and Control Register

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 Chapter 4.

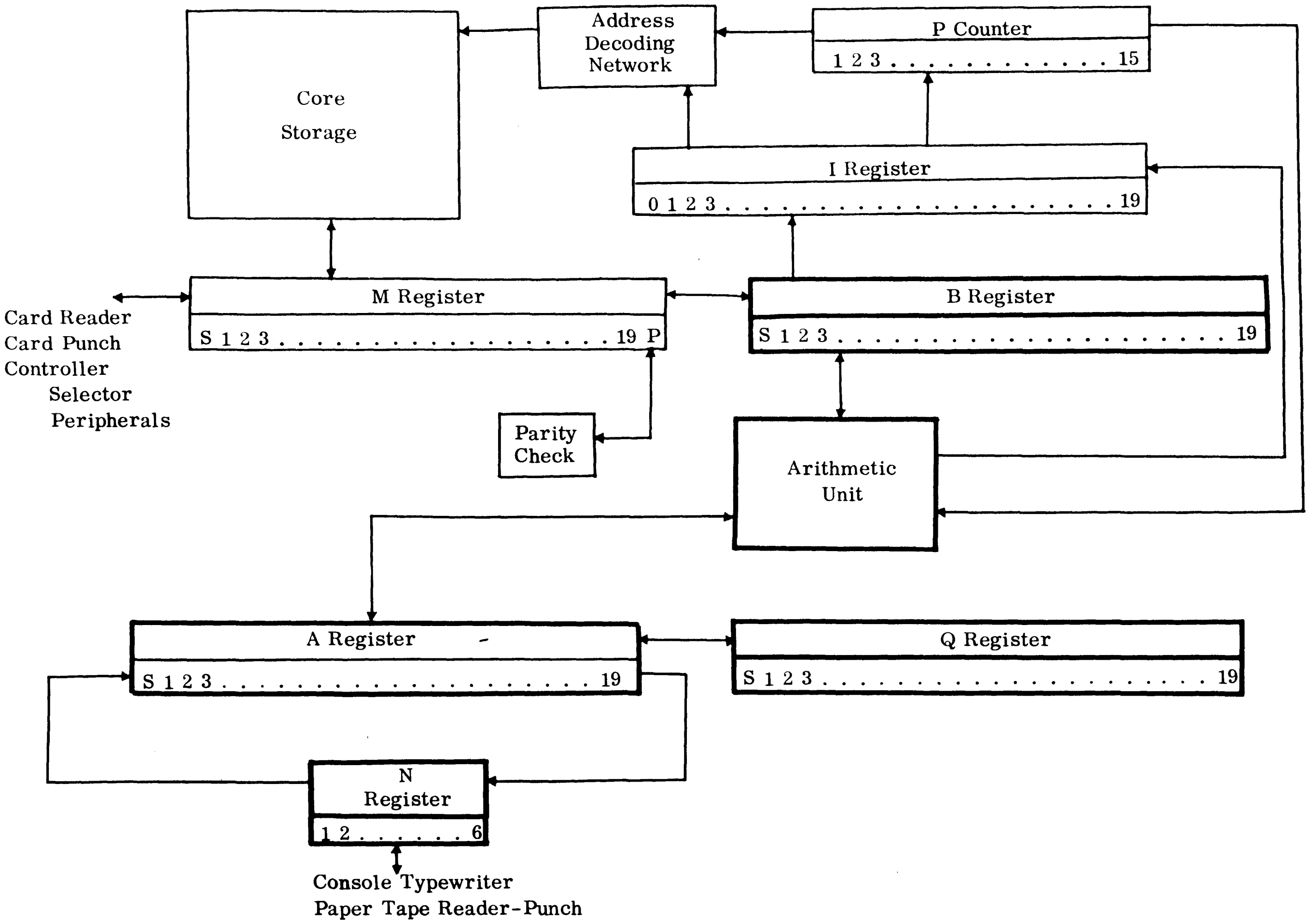


Figure 16. GE-225 Arithmetic Registers

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 17)

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.

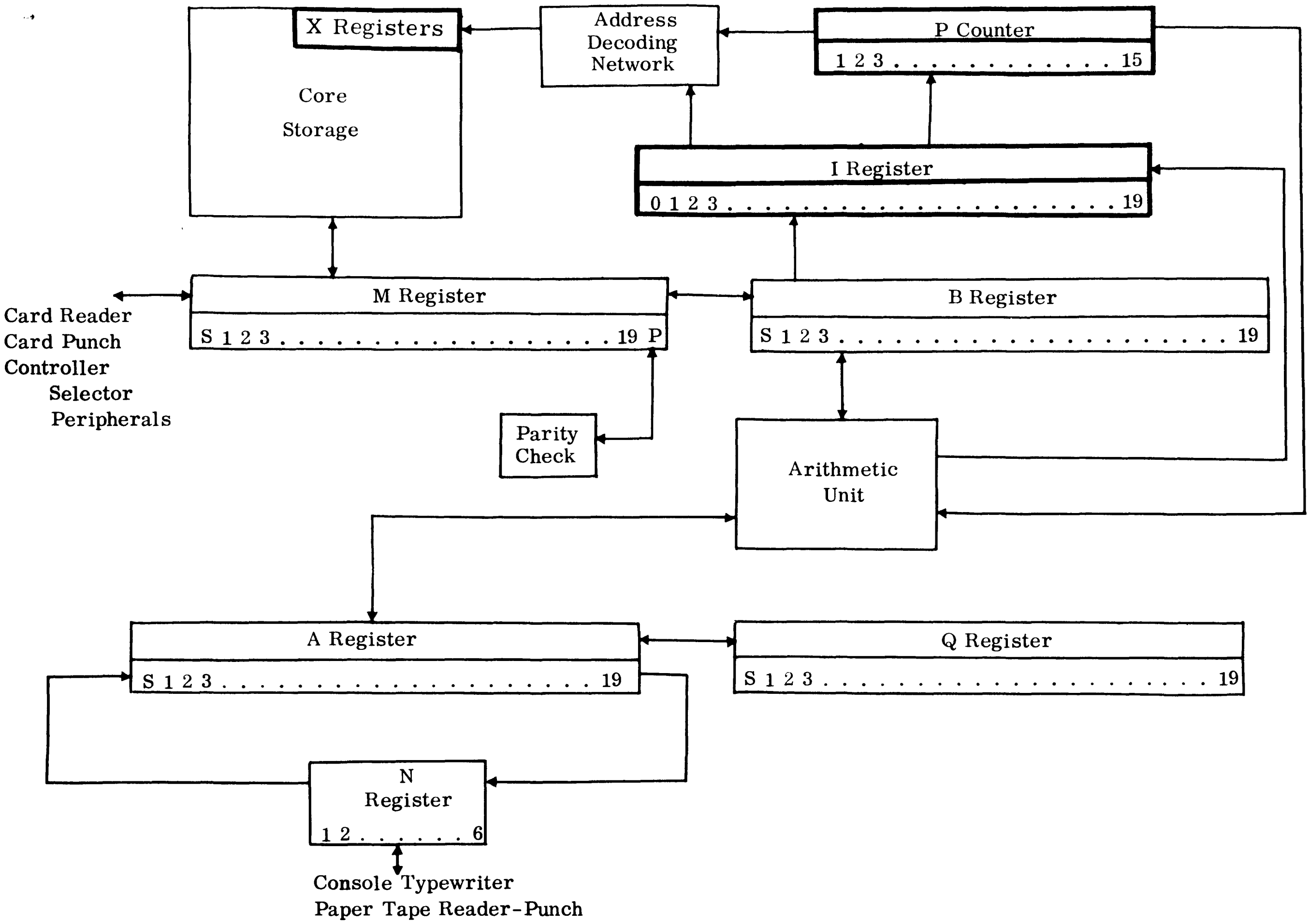


Figure 17. GE-225 Control Registers

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

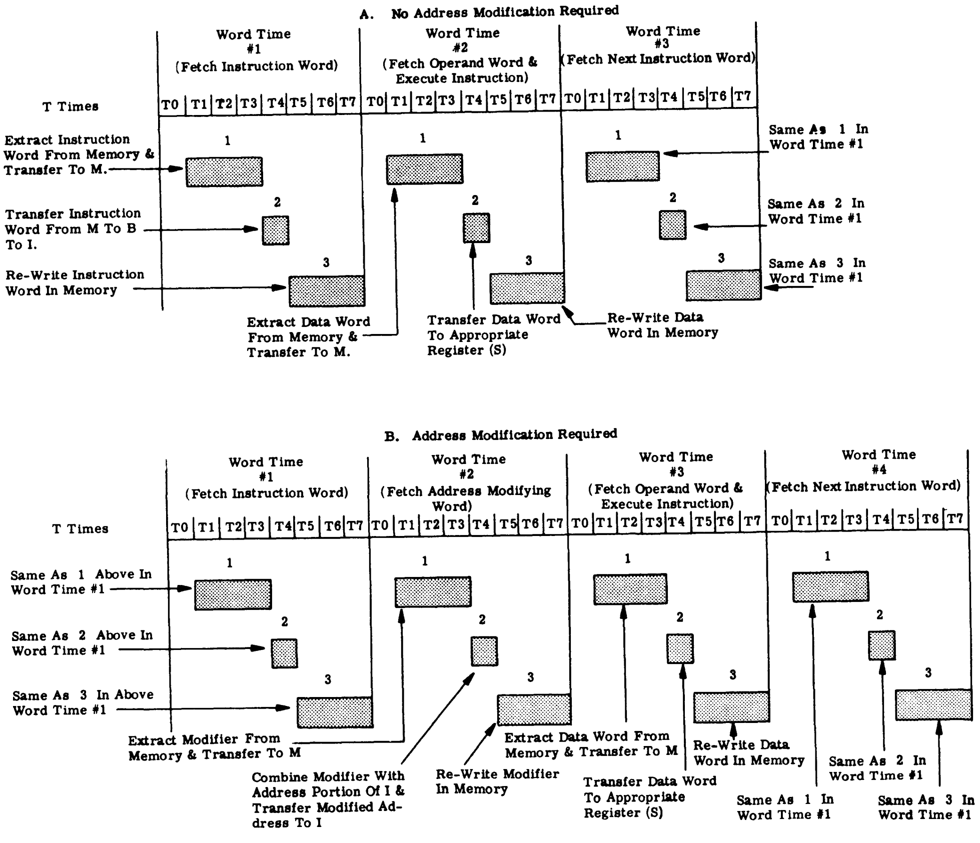
P COUNTER. The P Counter (or register) is a 15-bit location counter that contains the memory address of the next instruction to be executed. The contents of the P counter are incremented by one before the execution of an instruction so that the P counter indicates the next instruction in sequence. The 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. Part A of Figure 18, 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, Part A of Figure 18) 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 18, Part B.

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


A. No Address Modification Required

Word Time #1 (Fetch Instruction Word)
Word Time #2 (Fetch Operand Word & Execute Instruction)
Word Time #3 (Fetch Next Instruction Word)
T Times
T0 T1 T2 T3 T4 T5 T6 T7 T0 T1 T2 T3 T4 T5 T6 T7 T0 T1 T2 T3 T4 T5 T6 T7
1
2
3
Extract Instruction Word From Memory & Transfer To M.
Transfer Instruction Word From M To B To I.
Re-Write Instruction Word In Memory
Extract Data Word From Memory & Transfer To M.
Transfer Data Word To Appropriate Register (S)
Re-Write Data Word In Memory
Same As 1 In Word Time #1
Same As 2 In Word Time #1
Same As 3 In Word Time #1

B. Address Modification Required

Word Time #1 (Fetch Instruction Word)
Word Time #2 (Fetch Address Modifying Word)
Word Time #3 (Fetch Operand Word & Execute Instruction)
Word Time #4 (Fetch Next Instruction Word)
T Times
T0 T1 T2 T3 T4 T5 T6 T7 T0 T1 T2 T3 T4 T5 T6 T7 T0 T1 T2 T3 T4 T5 T6 T7 T0 T1 T2 T3 T4 T5 T6 T7
1
2
3
Same As 1 Above In Word Time #1
Same As 2 Above In Word Time #1
Same As 3 In Above Word Time #1
Extract Modifier From Memory & Transfer To M
Combine Modifier With Address Portion Of I & Transfer Modified Address To I
Re-Write Modifier In Memory
Extract Data Word From Memory & Transfer To M
Transfer Data Word To Appropriate Register (S)
Re-Write Data Word In Memory
Same As 1 In Word Time #1
Same As 2 In Word Time #1
Same As 3 In Word Time #1

Figure 18. 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-exeution 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 19. 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.

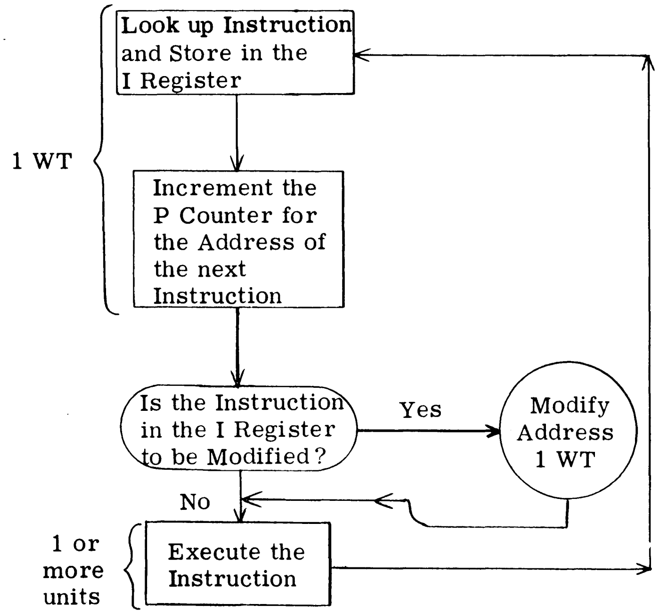

Look up Instruction and Store in the I Register
1 WT
Increment the P Counter for the Address of the next Instruction
Is the Instruction in the I Register to be Modified?
Yes
Modify Address 1 WT
No
1 or more units
Execute the Instruction

Figure 19. 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 20 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 20 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.

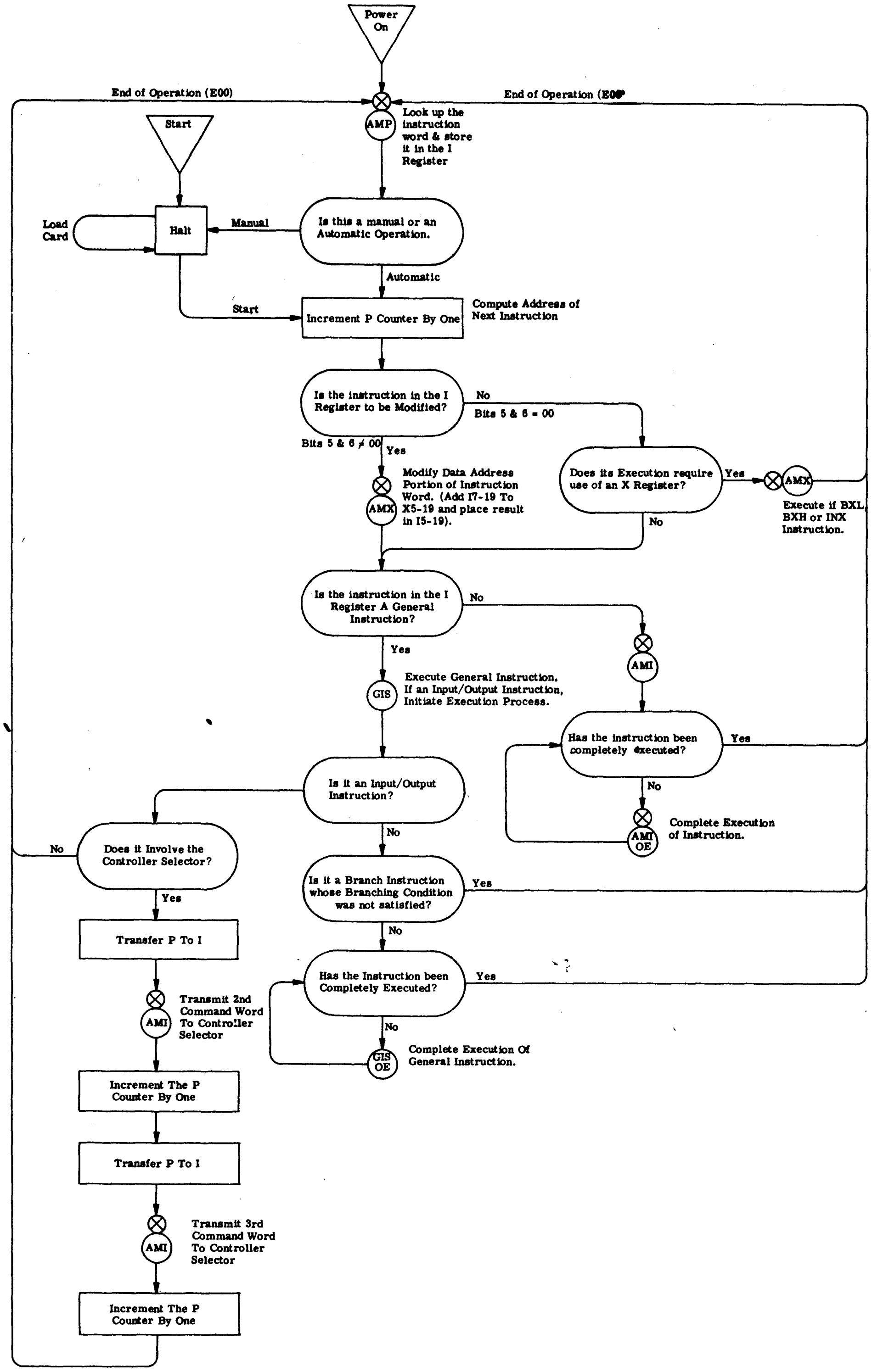

Power On
End of Operation (E00)
End of Operation (E0Ø
Start
AMP
Look up the instruction word & store it in the I Register
Load Card
Halt
Manual
Is this a manual or an Automatic Operation.
Automatic
Start
Increment P Counter By One
Compute Address of Next Instruction
Is the instruction in the I Register to be Modified?
No
Bits 5 & 6 = 00
Bits 5 & 6 ≠ 00
Yes
Modify Data Address Portion of Instruction Word. (Add I7-19 To X5-19 and place result in I5-19).
AMX
Does its Execution require use of an X Register?
Yes
AMX
Execute if BXL, BXH or INX Instruction.
No
Is the instruction in the I Register A General Instruction?
No
AMI
Yes
GIS
Execute General Instruction. If an Input/Output Instruction, Initiate Execution Process.
Has the instruction been completely executed?
Yes
No
AMI OE
Complete Execution of Instruction.
Is it an Input/Output Instruction?
No
Does it Involve the Controller Selector?
No
Yes
Is it a Branch Instruction whose Branching Condition was not satisfied?
Yes
No
Has the Instruction been Completely Executed?
Yes
No
GIS OE
Complete Execution Of General Instruction.
Transfer P To I
AMI
Transmit 2nd Command Word To Controller Selector
Increment The P Counter By One
Transfer P To I
AMI
Transmit 3rd Command Word To Controller Selector
Increment The P Counter By One

Figure 20. Flow Chart Showing Central Processor Operating Cycle

4. 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 General Assembly Program (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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	SUB	AMT#3	

Register Contents in Octal

	A	Q
Before execution:	0146626	?
After execution:	3747011	?

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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	DSU	AMT#6	

Register Contents in Octal

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

Example 2: Subtract the positive 155,926,921,828$_{10}$ (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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	ADO		

Register Contents in Octal

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

Example 2: Add one to the negative number -42189$_{10}$ (3655463$_8$), which has been previously loaded into the A register.

GAP Coding:

Symbol	Opr	Operand	X
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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.

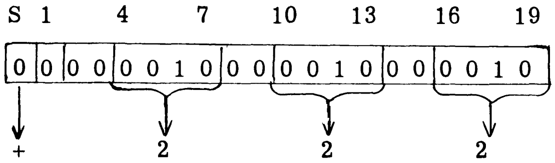
the Power On switch on the control console. The initial power on sequence automatically sets the GE-225 in the binary mode.

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.

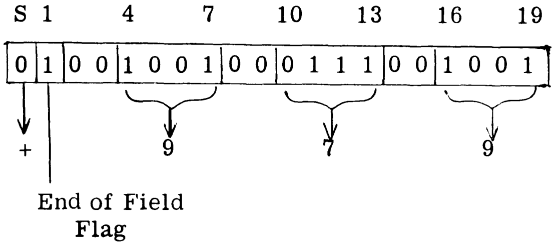
The sign of the decimal number is in the S position of the word containing the most significant decimal digit; a 0-bit designates a positive decimal number, while a 1-bit indicates a negative quantity.

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.

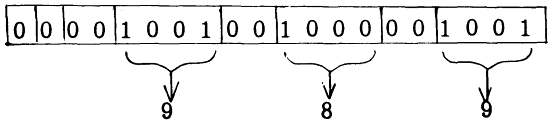
The decimal word containing the most significant (high-order) digit must be marked or flagged to define the end of the decimal field by placing a 1-bit in bit position 1.

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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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

	A	Q
After execution:	+ 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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	D S U	N E G R	

Memory and A and Q Register Contents in BCD

	A	Q
Before execution:	+ 5 4 3	2 1 0
	NEGR	**NEGR+1**
	+ 1 2 3	4 5 6
After execution:	+ 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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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.

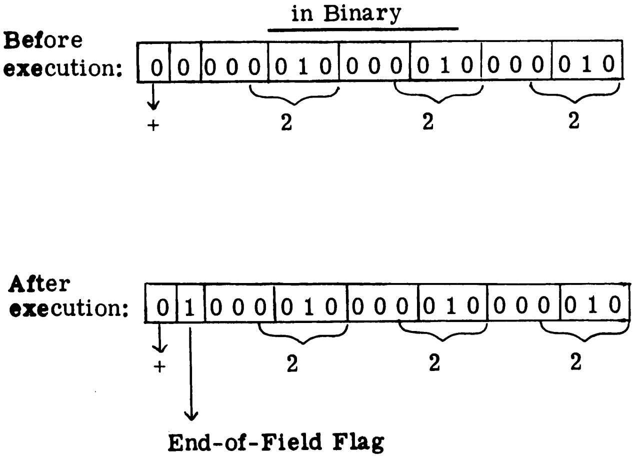
PROGRAM INSERTION OF END-OF-FIELD FLAGS

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



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.

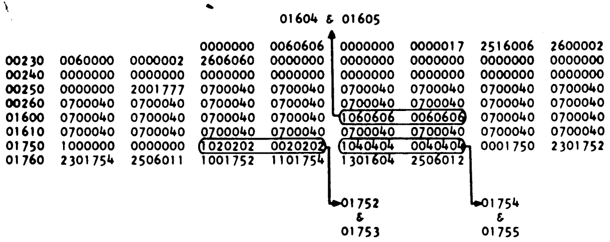
Command 01761 converts the internal operation of the computer to BCD prior to the addition and command 1765 restores the computer to the binary mode.

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. Locations 01604 contains the sum which also is automatically flagged.

Memory Printout



```
                                  0000000  0060606  0000000  0000017  2516006  2600002
00230  0060000  0000002  2606060  0000000  0000000  0000000  0000000  0000000
00240  0000000  0000000  0000000  0000000  0000000  0000000  0000000  0000000
00250  0000000  2001777  0700040  0700040  0700040  0700040  0700040  0700040
00260  0700040  0700040  0700040  0700040  0700040  0700040  0700040  0700040
01600  0700040  0700040  0700040  0700040  1060606  0060606  0700040  0700040
01610  0700040  0700040  0700040  0700040  0700040  0700040  0700040  0700040
01750  1000000  0000000  1020202  0020202  1040404  0040404  0001750  2301752
01760  2301754  2506011  1001752  1101754  1301604  2506012
```

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:

S

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

Maximum Positive Number = $+524,287_{10}$

S

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	0	0	0	0	0	0	0	0	0	0

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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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} x -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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
	DVD	WRDS	

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:

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

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 21.

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	1	1	1	1	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

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

Figure 21. **Two Numbers in Memory before Scaling**

When these two numbers are added, the result would appear in the A register as 1569_{10} (Figure 22). 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	1	1	0	0	0	1	0	0	0	0	1

= 1569_{10}

Figure 22. **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 23.

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	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 23. **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 the GE-200 Series General Assembly Program II reference manual, CPB-1180 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 24.

	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 24. **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 526301_{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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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: 2

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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	ORY	$OUT	

Memory and A Register before Execution (binary):

$OUT

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

A Reg

Memory and A Register after Execution (binary):

$OUT

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

A Reg

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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	ORY	PRICE	

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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	EXT	MOD	

Memory and A Register before 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 1 0	1 0 1	0 1 1	0 0 1	1 1 1

A Reg

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

Registers

A

0002260 = 1200_{10}

Q

3777766 = -10_{10}

Memory and Register Contents in Octal:

After execution:

Memory

Octal Address	Contents
0261	0123456
0262	0246531
0263	1234567
0264	0765432
0265	0135764
0266	2345670
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

Registers

A	Q
0000000	3777766

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 7654321_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
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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

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
8 9 10	12 13 14 15 16 17 18 19	20	31 75
L A C			TRANSFER TIME TO A REG
M A Q			TRANSFER TIME TO Q REG FOR DVD
D V D	C O N 1		COMPUTE HOURS
S T A	H O U R S		
L D Z			CLEAR A REG
D V D	C O N 2		COMPUTE MINUTES
S T A	M I N S		
L D Z			
D V D	C O N 3		COMPUTE SECONDS
S T A	S E C S		
X A Q			TRANSFER SIXTH-SECONDS TO A REG
S T A	S X T H S		

Initial Register Contents:

C	A	Q
1205701	?	?

Registers Affected by Each Instruction:

GAP Coding	Registers A	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 conver**sion** factors. Perhaps the simplest method would be to **con**vert the binary contents of the C register to octal, **then decimal, and divide by decimal conversion factors. The** conversion chart in Figure 9 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 9 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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	L D Z		

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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	L D O		

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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	N O P		

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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	S R A	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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 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 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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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 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	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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 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 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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	SLA	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	SRD	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	SRD	2	

Register Contents in Binary

Before execution:

A Reg

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

Q Reg

After execution:

A Reg

0 0	0 0 1	1 0 1	0 1 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
0 0	1 0 0	1 1 0	1 0 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
1	2	3	4	5	6	8	9	10	12	13	14	15	16	17	18	19	20
						S	L	D	4								

Register Contents
in Binary

Before execution:

A Reg

0 0	0 0 0	0 1 0	1 1 0	0 1 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	1 0 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
1	2	3	4	5	6	8	9	10	12	13	14	15	16	17	18	19	20
						S	C	A	8								

Register Contents
in Binary

Before execution:
A Reg

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

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

After execution:
A Reg

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

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

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
1	2	3	4	5	6	8	9	10	12	13	14	15	16	17	18	19	20
						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 (1–6)	Opr (8–10)	Operand (12–19)	X (20)
	SAN	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 (1–6)	Opr (8–10)	Operand (12–19)	X (20)
	SNA	6	

Register Contents (BCD)

	A	N
Before execution	123	8
After execution	+12	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 (1–6)	Opr (8–10)	Operand (12–19)	X (20)
	ANQ	6	

Register Contents in Octal

	A	Q	N
Before execution:	1460430	0000000	?
After execution:	0014604	0600000	30

P25th [illegible]

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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	N A Q	6	

Register Contents in Octal

	A	Q
Before execution:	888	???
	N 7	
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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	N O R	5	

Memory and Register Contents in Octal

	A	0000
Before execution:	0012345	00000??
After execution:	0516240	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	N O R	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 the GE-205/215/225 Auxiliary Arithmetic Unit reference manual, CPB-325.

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 (9 leading 0-bits) to 6 bit positions (K = 6, R = 9).

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 N O	6	

Memory and Register Contents in Octal

	A	Q	0000
Before execution:	0001234	0076543	00000 ??
After execution:	0123403	1654300	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	D N O	1 5	

Memory and Register Contents in Octal

	A	Q	0000
Before execution:	0001777	0000177	00000 ??
After execution:	1777000	0177000	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
00460	BRU	STORE	

	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
02676	SPB	RERUN	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
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 X 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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	
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 X 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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	INX	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:

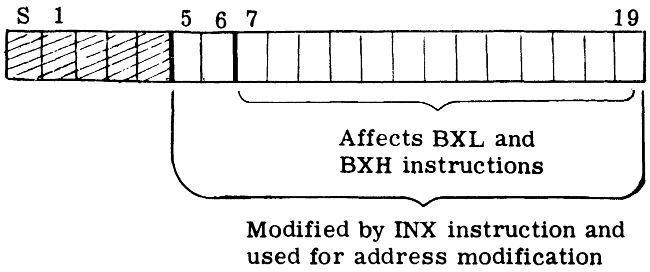
Symbol	Opr	Operand	X
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	INX	8186	3

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.

X Register Contents


S 1
5 6 7
19
Affects BXL and BXH instructions
Modified by INX instruction and used for address modification

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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	BXH	4	2
FICA	BRU	0777	
	STA	TEMP	
	⌇		

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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	LDX	SET1	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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	STX	RESET	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 2506YY3 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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
	SXG	27	

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.

<u>Comments:</u> 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.

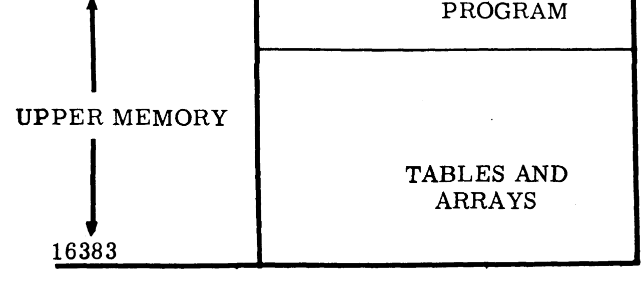


Figure 25. 16k Memory Layout

In addition, the proper allocation and use of memory becomes essential. Figure 25 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	REMAR
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
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 26 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 26. 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: 1756

Opr	Operand	X
8 9 10	12 13 14 15 16 17 18 19	20
B R U	3 8 0 8	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: 12250

Opr	Operand	X
8 9 10	12 13 14 15 16 17 18 19	20
B R U	3 8 0 8	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: **12250**

Opr	Operand	X
8 9 10	12 13 14 15 16 17 18 19	20
B R U	3 8 0 8	2

Next Instruction Location is 03808

Controls are changed to the lower bank starting at memory location 03808.

In summary, it is essential that the programmer 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
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20
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
	8 9 10	12 13 14 15 16 17 18 19	20
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
	8 9 10	12 13 14 15 16 17 18 19	20
12228	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
	8 9 10	12 13 14 15 16 17 18 19	20
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	Operand	X
	8 9 10	12 13 14 15 16 17 18 19	20
12160	LDA	3000	2
12161	SPB	* + 1	1
12162	STO	2	1
12163	ADD	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 27 illustrates a portion of the flowcharting for a rejected parts cost program. GAP coding sheets corresponding to that portion of the flowchart are shown in Figures 28 through 31. 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 28, 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 29, 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 30), DAREA calculations are skipped, and EAREA calculations are made.

Line 20 of Figure 29 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 30 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 OVERFLO, line 2 of Figure 30. After overflow recovery the BRU 0002, modified by index word 1 returns to the main routine, line 13, Figure 30.

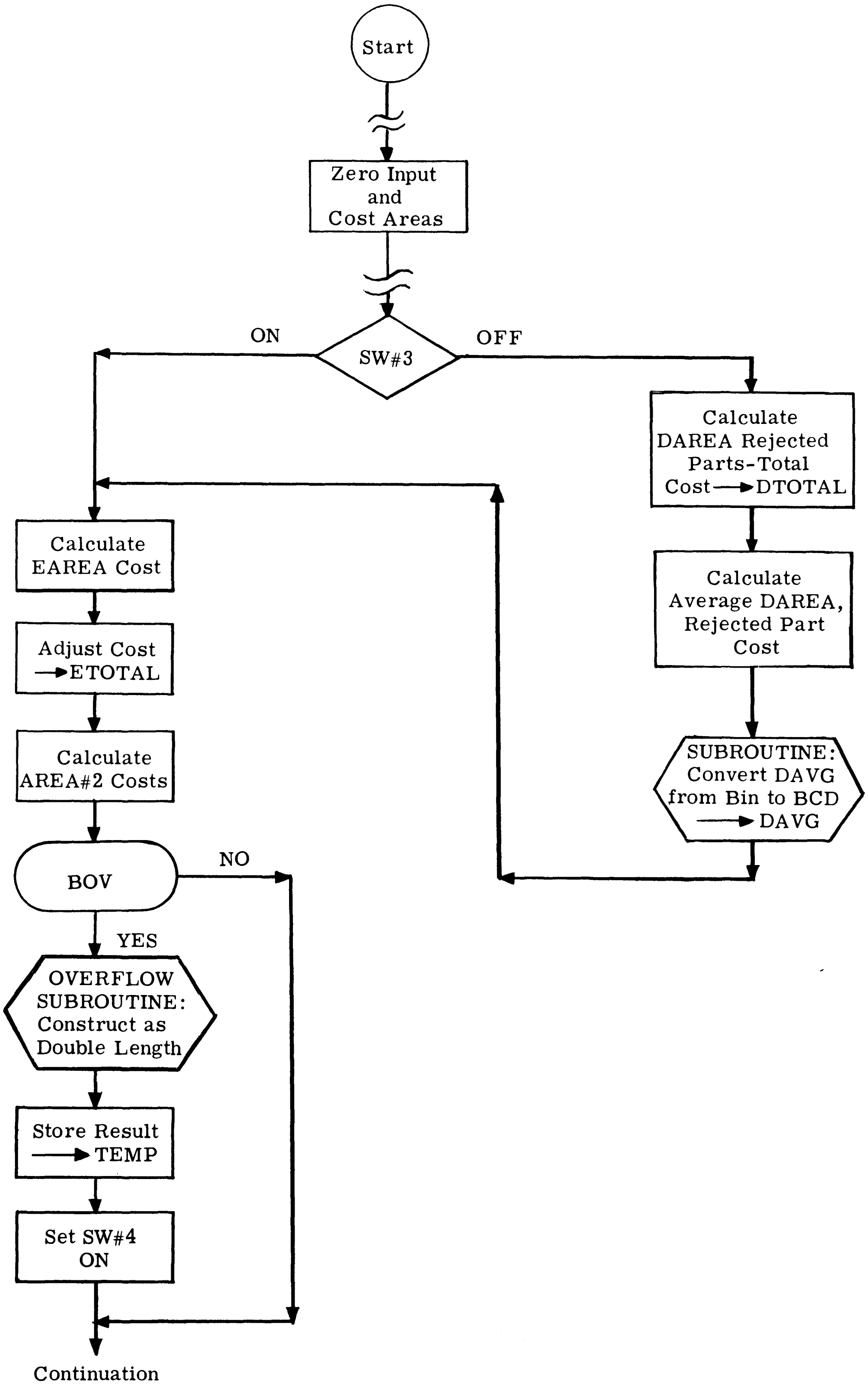
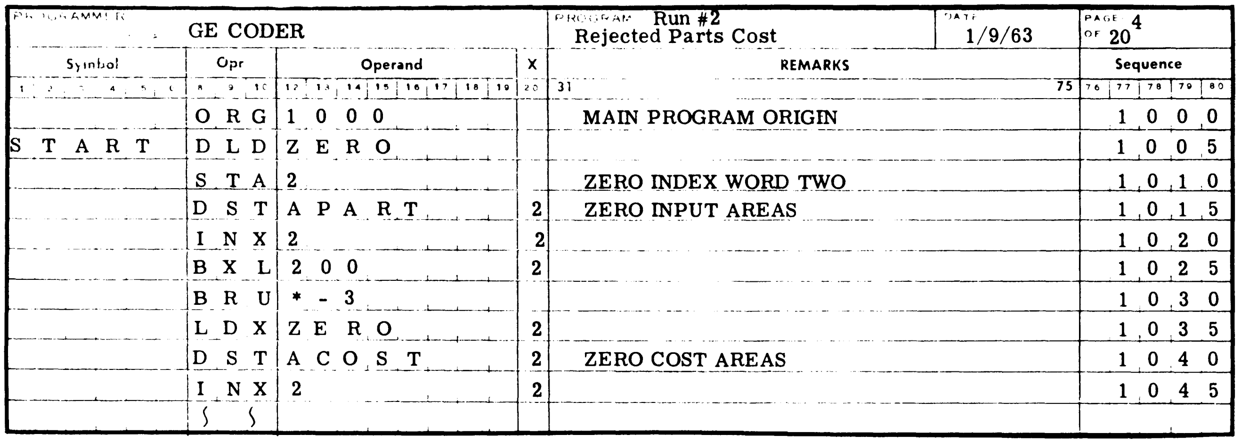

Start
Zero Input and Cost Areas
ON
SW#3
OFF
Calculate DAREA Rejected Parts-Total Cost ——► DTOTAL
Calculate EAREA Cost
Calculate Average DAREA, Rejected Part Cost
Adjust Cost ——► ETOTAL
Calculate AREA#2 Costs
SUBROUTINE: Convert DAVG from Bin to BCD ——► DAVG
BOV
NO
YES
OVERFLOW SUBROUTINE: Construct as Double Length
Store Result ——► TEMP
Set SW#4 ON
Continuation

Figure 27. Rejected Parts Cost Flow Chart

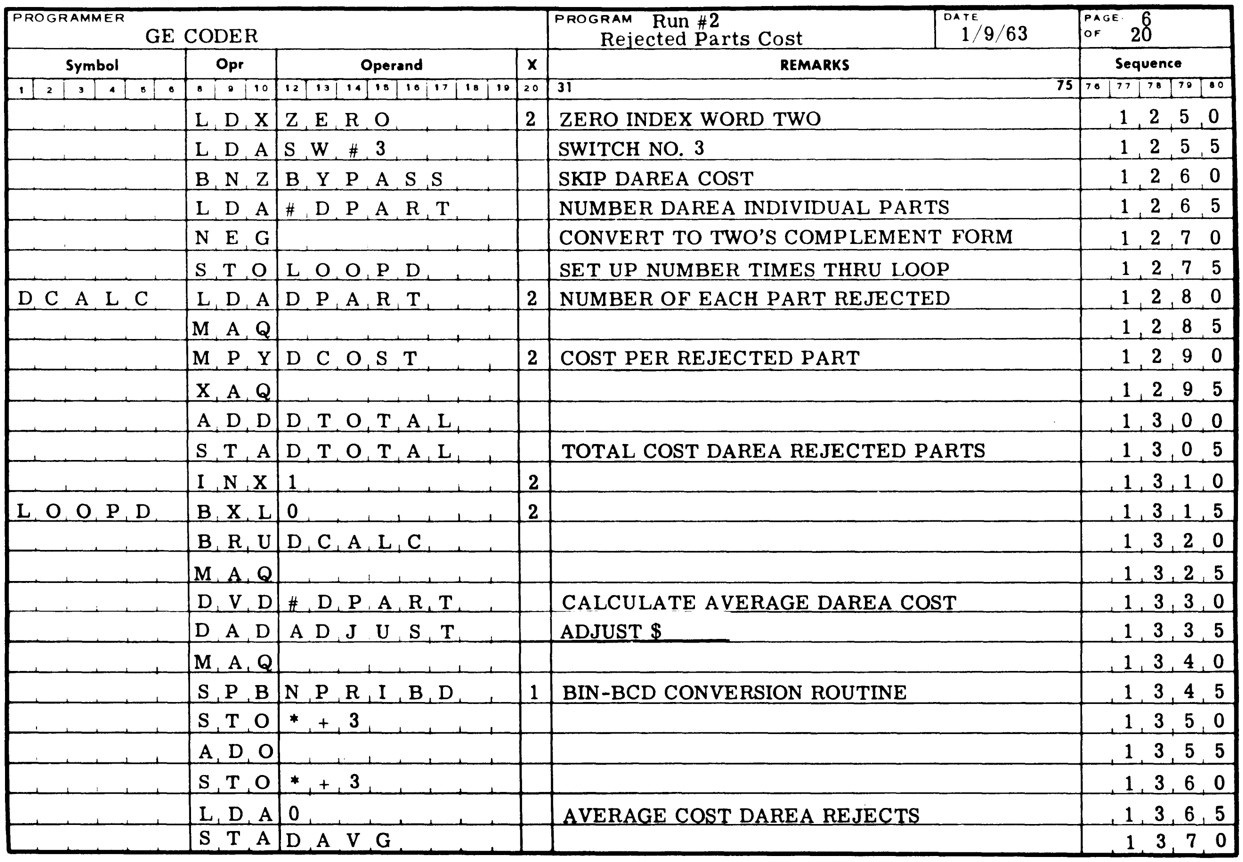
GAP Coding:



	PROGRAMMER GE CODER				PROGRAM Run #2 Rejected Parts Cost	DATE 1/9/63	PAGE 4 OF 20
	Symbol	Opr	Operand	X	REMARKS		Sequence
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31 75		76 77 78 79 80
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		ʃ ʃ					

Figure 28. RPC Program - Initialization

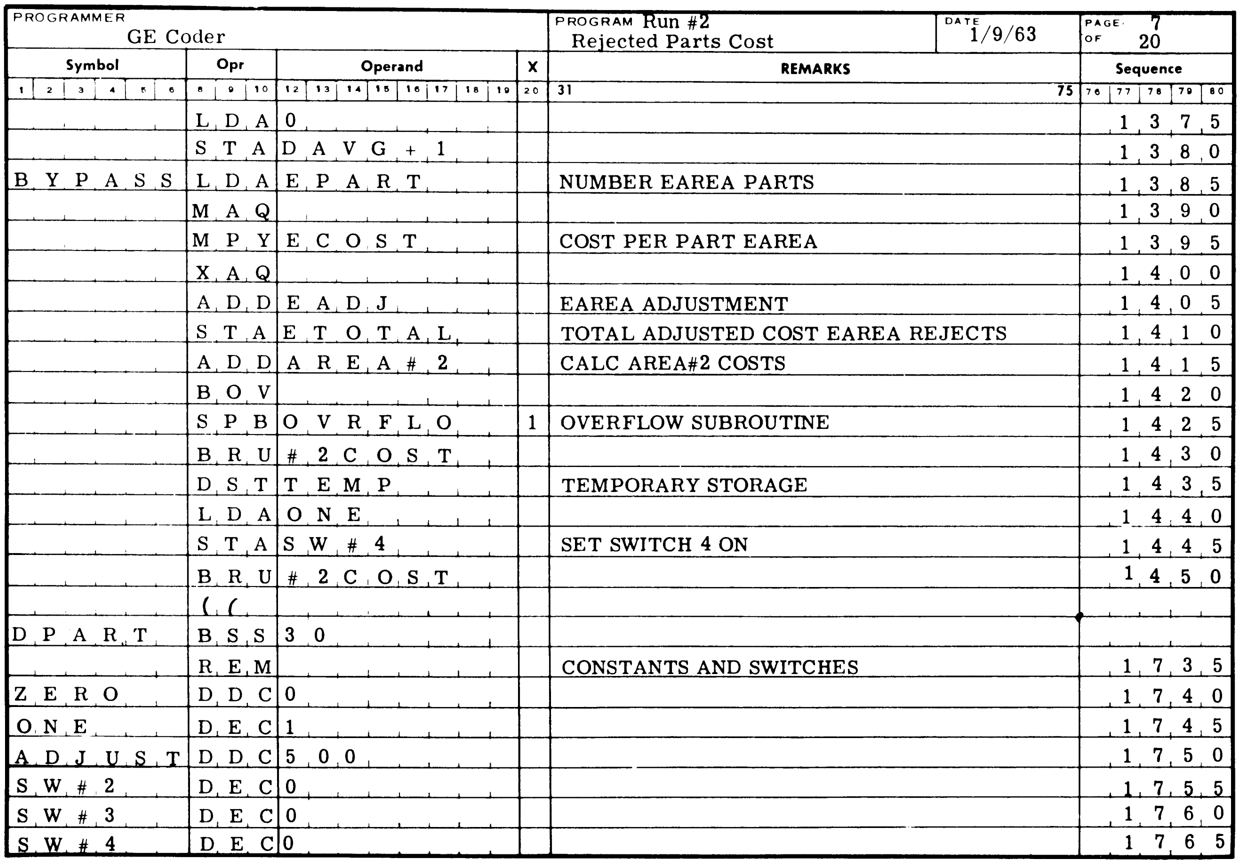
GAP Coding:



	PROGRAMMER GE CODER				PROGRAM Run #2 Rejected Parts Cost	DATE 1/9/63	PAGE 6 OF 20
	Symbol	Opr	Operand	X	REMARKS		Sequence
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31 75		76 77 78 79 80
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 29. RPC Program - DPARTS Calculations

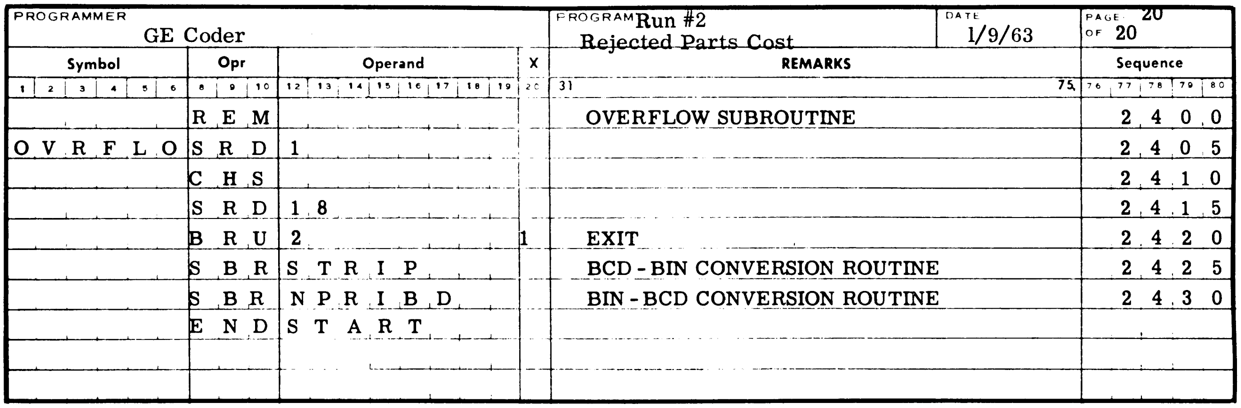
GAP Coding:



PROGRAMMER GE Coder				PROGRAM Run #2 Rejected Parts Cost	DATE 1/9/63 | PAGE 7 OF 20
Symbol	Opr	Operand	X	REMARKS	Sequence
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31 75	76 77 78 79 80
	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 30. **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
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31 75	76 77 78 79 80
	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 31. RPC Program - OVRFLO Routine

5. 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 32. 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 DSU, 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 86. Visual

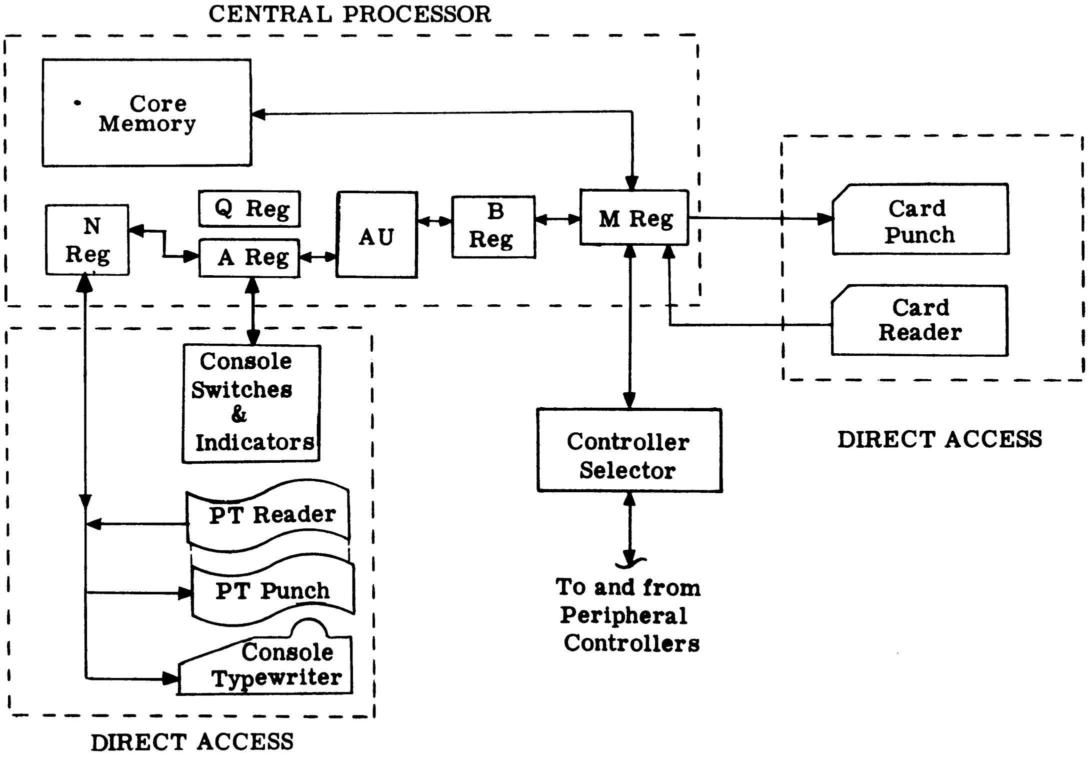


Figure 32. **Units Directly Accessing Memory**

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 33. 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 33, 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.

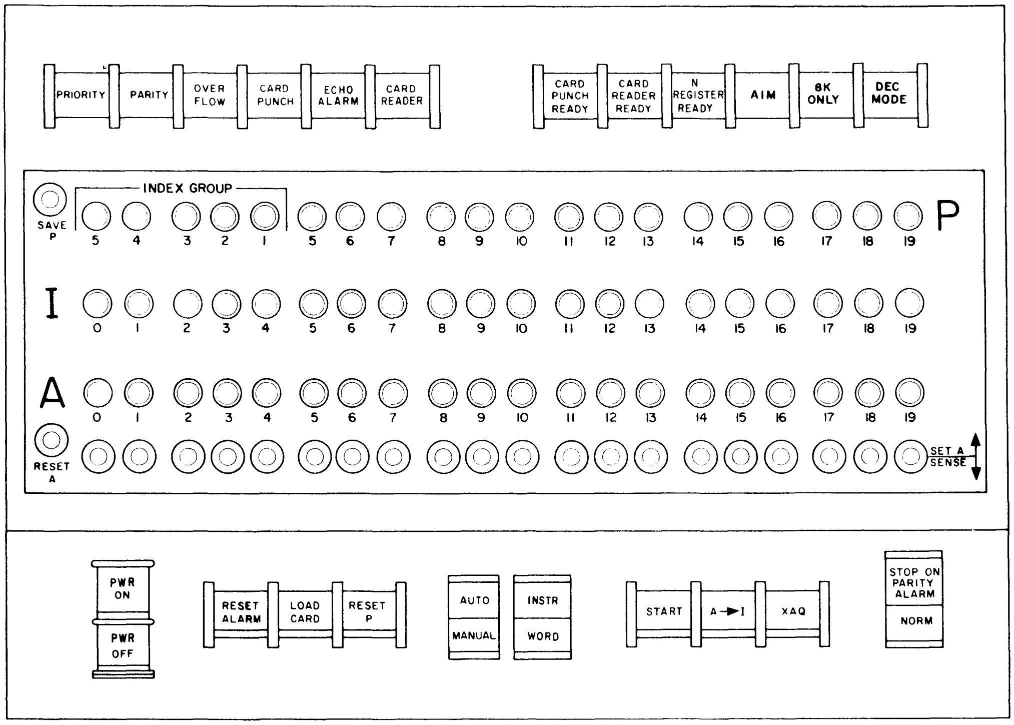


Figure 33. The Control Console Panel

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
	RCS		

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 34, 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

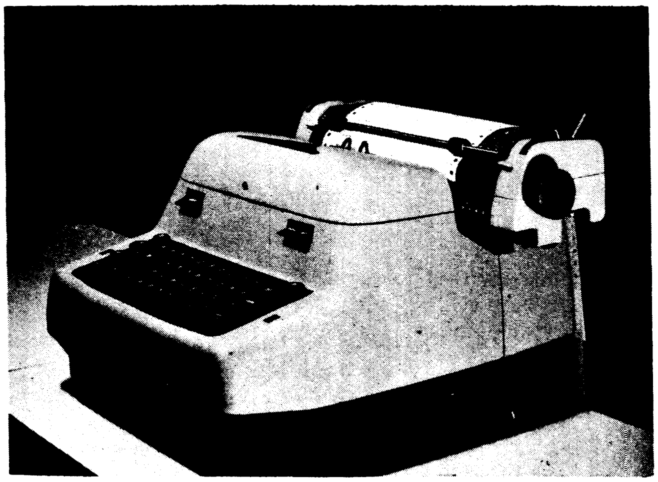


Figure 34. Console Typewriter

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
,	73
Print Black	75
Tab	76

Figure 35. 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 35); 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.

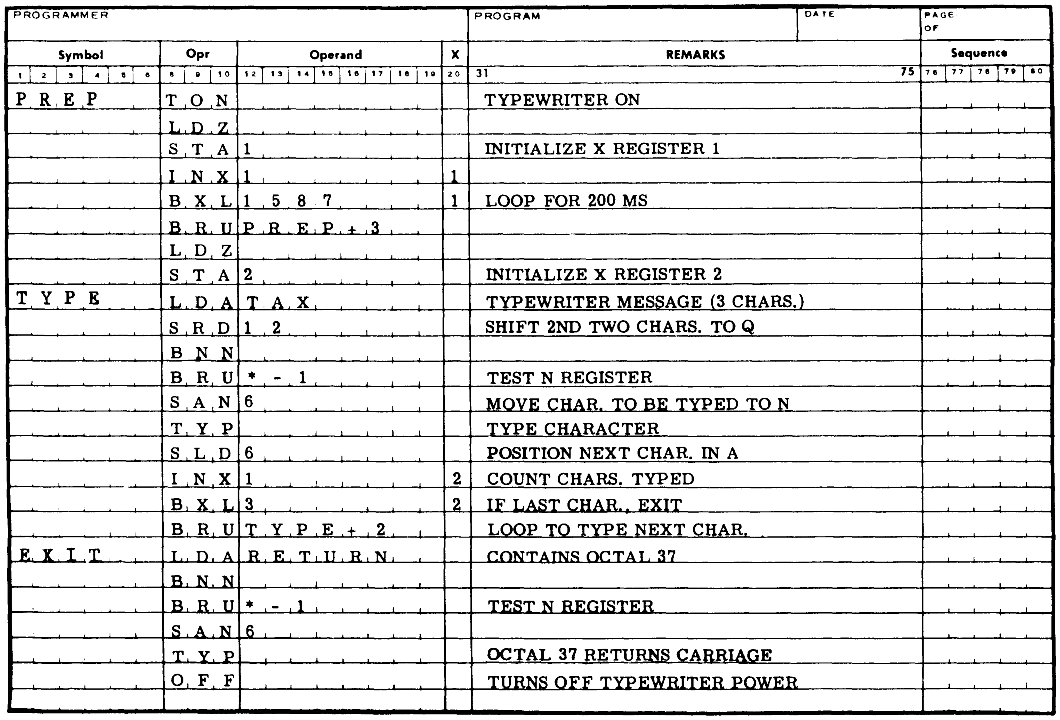
Example: Examples of all typewriter instructions are provided in the coding sample following this instruction description.

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 36.



```
PROGRAMMER                          PROGRAM                     DATE     PAGE OF
Symbol  Opr  Operand        X   REMARKS                                   Sequence
PREP    TON                     TYPEWRITER ON
        LDZ
        STA  1                  INITIALIZE X REGISTER 1
        INX  1              1
        BXL  1587           1   LOOP FOR 200 MS
        BRU  PREP+3
        LDZ
        STA  2                  INITIALIZE X REGISTER 2
TYPE    LDA  TAX                TYPEWRITER MESSAGE (3 CHARS.)
        SRD  12                 SHIFT 2ND TWO CHARS. TO Q
        BNN
        BRU  *-1                TEST N REGISTER
        SAN  6                  MOVE CHAR. TO BE TYPED TO N
        TYP                     TYPE CHARACTER
        SLD  6                  POSITION NEXT CHAR. IN A
        INX  1              2   COUNT CHARS. TYPED
        BXL  3              2   IF LAST CHAR., EXIT
        BRU  TYPE+2             LOOP TO TYPE NEXT CHAR.
EXIT    LDA  RETURN             CONTAINS OCTAL 37
        BNN
        BRU  *-1                TEST N REGISTER
        SAN  6
        TYP                     OCTAL 37 RETURNS CARRIAGE
        OFF                     TURNS OFF TYPEWRITER POWER
```

Figure 36. 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).

6. 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 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 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 disc storage unit (DSU), 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	Disc Storage Unit (DSU)
	2nd DSU 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 DSU controllers, programs must not refer to peripherals used by another program in the same load. When magnetic tape and DSU controllers are both used, the same handler on the DSU 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 DSU 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
A1	DEC	512		First card read-in area
	DEC	640		Second card read-in area
A2	DEC	512		First card read-in area
	DEC	640		Second card read-in area
CONST1	DEC	142		Transfer location
CONST2	DDC	0		Storage area for contents of the A and Q registers
	BCN			Test for card reader not ready
	BRU	A6		Exit if card reader is not ready
A3	RCD	0512		Read card into memory beginning at 0512
	HCR			Halt card reader
	DST	CONST2		Store contents of A and Q registers for main
				program
	DLD	A1		Load read-in area constants
	XAQ			Switch read-in areas
	DST	A1		Store read-in areas as constants
	STO	A2		Set up alternate card read-in area
A4	BRU	A7		Bypass writing a tape record the first time
				through
	BCS	BTN	2	Test for tape controller not ready
	BRU	*-1		Delay until tape controller is ready
	SEL	2		Select controller selector address two
A5	WTD	0512	1	Write tape in decimal mode from memory
				locations beginning at 0512 onto tape 1
		27		Write a maximum of 27 words
	DLD	A2		Load read-in area constants
	XAQ			Switch read-in area constants
	DST	A2		Store read-in area constants
	STO	A5		Set up memory address from which tape record is
				to be written
A6	DLD	CONST2		Load contents of the A and Q registers from main
				program
	SET	PST		Set priority interrupt mode on
	BRU	0	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
A7	LDA	CONST1		Load binary equivalent of 142
	STO	A4		Will cause the writing of tape records all succeed-
				ing times through the program
	BRU	A6		Transfer to exit

Figure 37. Assembly Program 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.

7. 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.

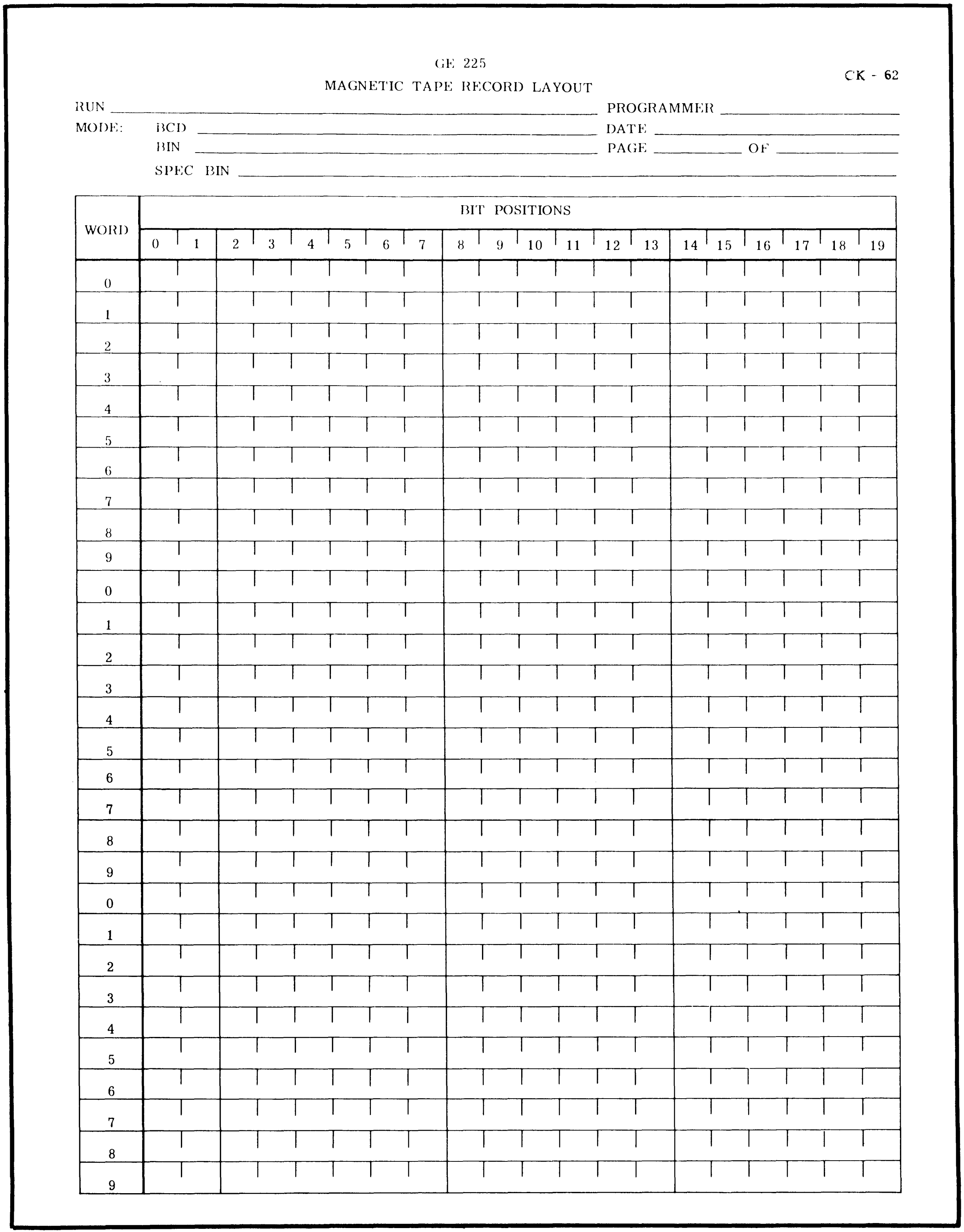
Because operating conditions and requirements vary from installation to installation, the memory layout used may be unique and suitable only for that particular installation. A typical layout is shown in Figure 38.

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 39 through 43.

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 38. Typical Memory Allocation



GE 225

MAGNETIC TAPE RECORD LAYOUT

CK - 62

RUN ____________________ PROGRAMMER ____________________

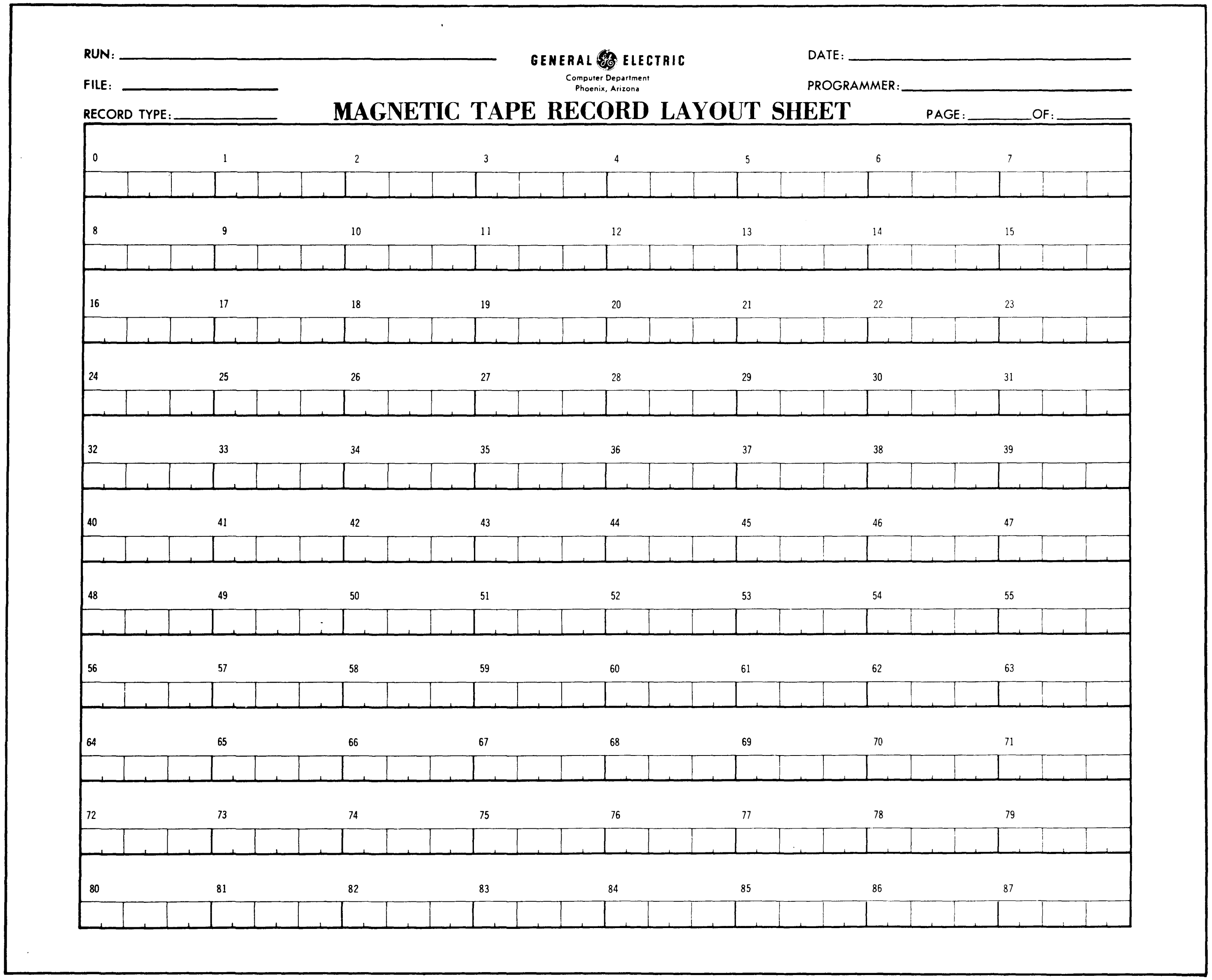
MODE: BCD ____________________ DATE ____________________

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 39. **Magnetic Tape Record Layout**



RUN: ______________________

GENERAL ELECTRIC
Computer Department
Phoenix, Arizona

DATE: ______________________

FILE: ______________

PROGRAMMER: ______________________

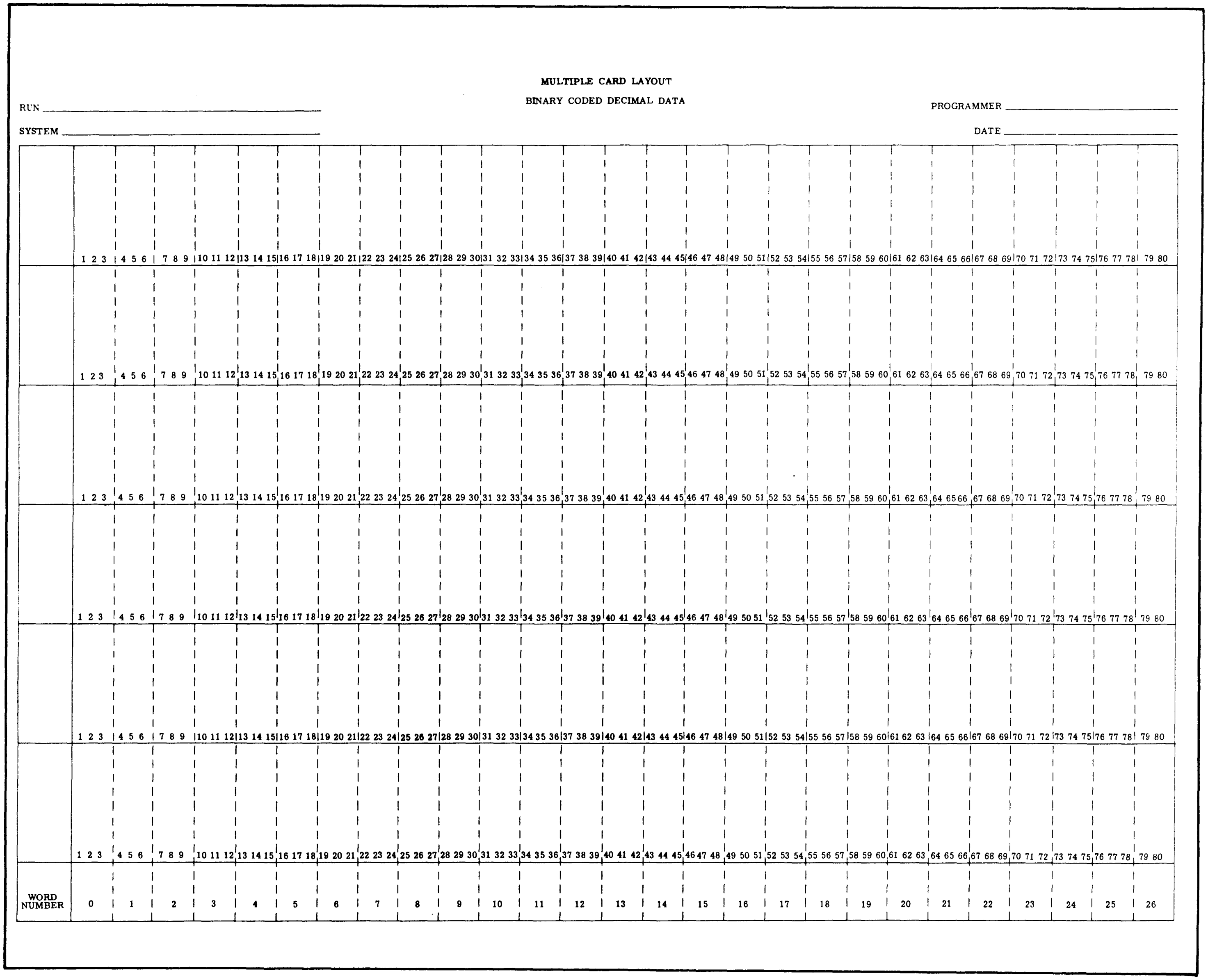
RECORD TYPE: ______________

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 40. Magnetic Tape Record Layout Sheet



MULTIPLE CARD LAYOUT

BINARY CODED DECIMAL DATA

RUN ____________

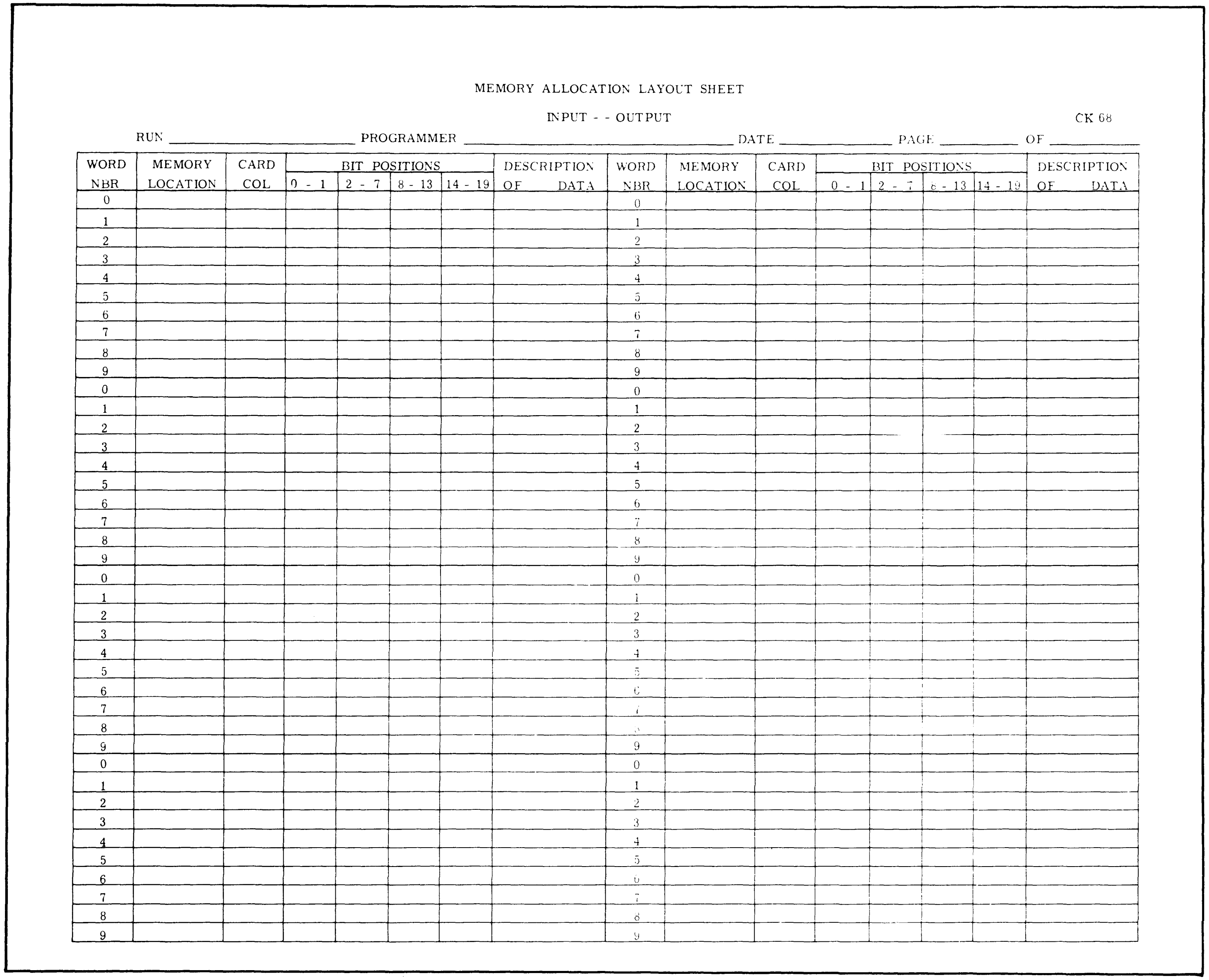
SYSTEM ____________

PROGRAMMER ____________

DATE ____________

	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
	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
	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 6566	67 68 69	70 71 72	73 74 75	76 77 78	79 80
	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
	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
	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	4647 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
WORD NUMBER	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

Figure 41. BCD Multiple Card Layout Sheet



MEMORY ALLOCATION LAYOUT SHEET

INPUT - - OUTPUT

CK 68

RUN ______________ PROGRAMMER ______________ DATE __________ PAGE ________ OF __________

WORD	MEMORY	CARD	BIT POSITIONS				DESCRIPTION	WORD	MEMORY	CARD	BIT POSITIONS				DESCRIPTION
NBR	LOCATION	COL	0 - 1	2 - 7	8 - 13	14 - 19	OF DATA	NBR	LOCATION	COL	0 - 1	2 - 7	8 - 13	14 - 19	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 42. **Memory Allocation Layout Sheet**

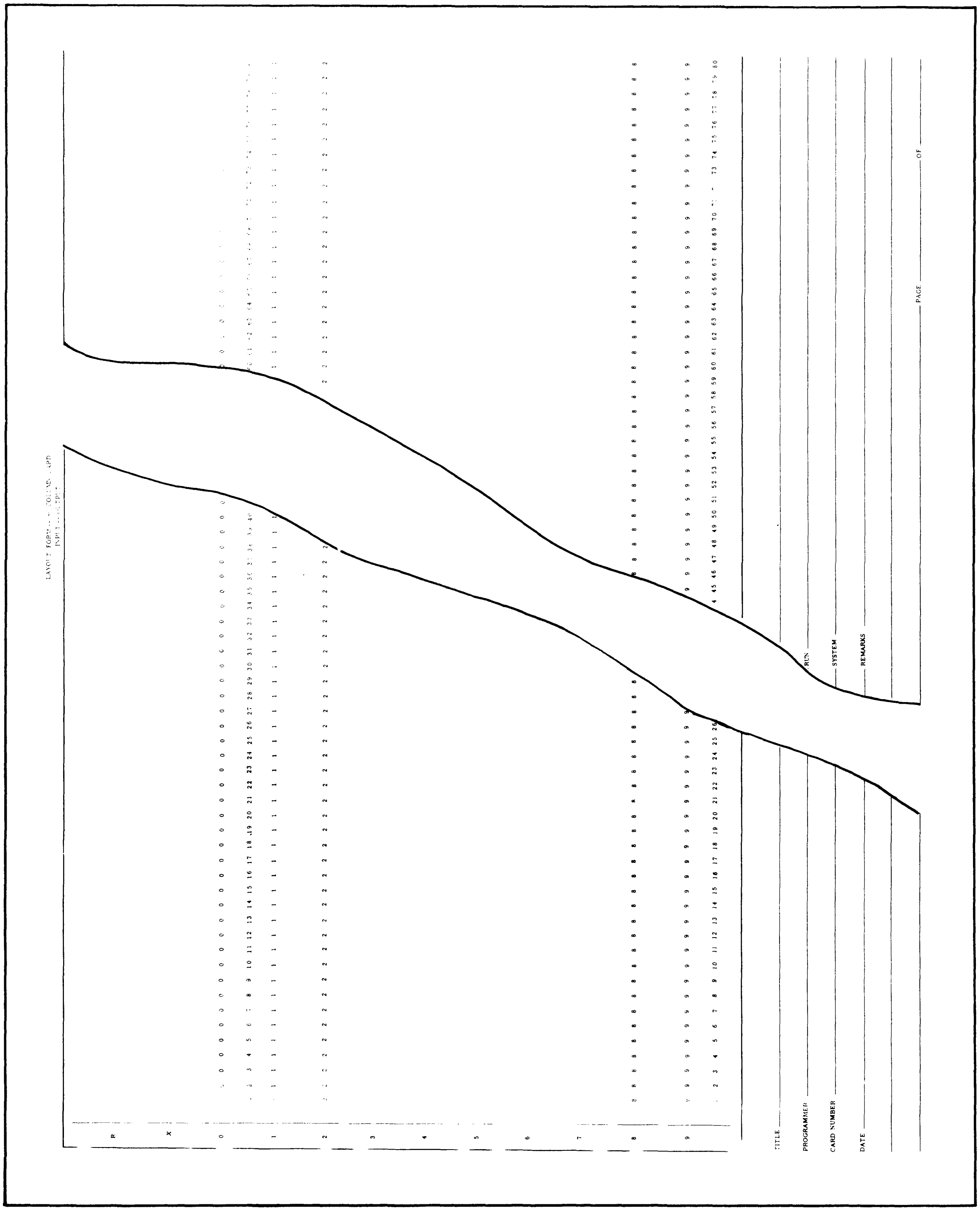
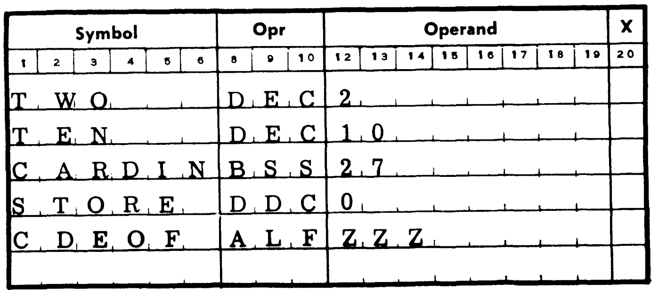


Figure 43. 80-Column Card Layout Form

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 44 illustrates typical symbols.



Symbol (1–6)	Opr (8–10)	Operand (12–19)	X (20)
TWO	DEC	2	
TEN	DEC	10	
CARDIN	BSS	27	
STORE	DDC	0	
CDEOF	ALF	ZZZ	

Figure 44. Typical Symbolic Addresses

SUBROUTINE USAGE

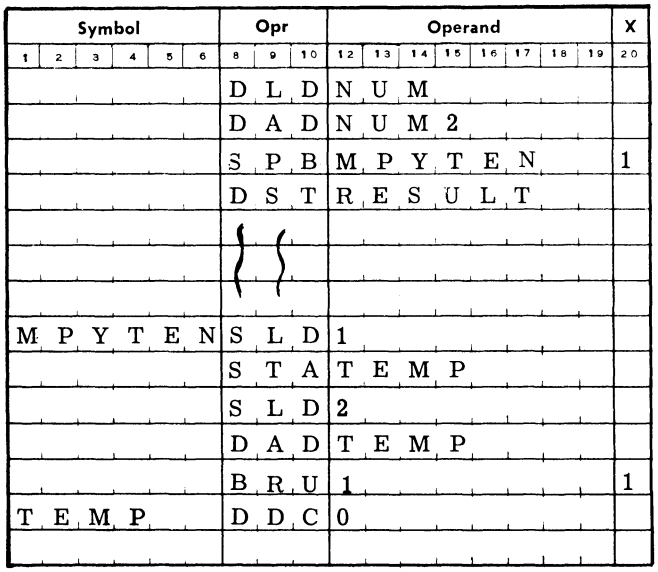
The use of subroutines can result in saving of both programming and machine running time. Subroutines can control all input and output operations and many internal operations of a program and use less memory. Normally, a subroutine is a series of instructions which perform a repetitive function for the 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 45.

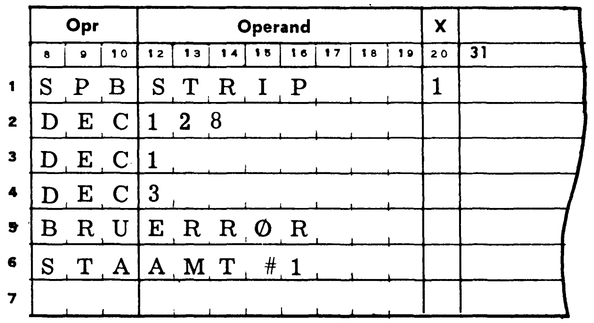


Symbol (1–6)	Opr (8–10)	Operand (12–19)	X (20)
	DLD	NUM	
	DAD	NUM2	
	SPB	MPYTEN	1
	DST	RESULT	
MPYTEN	SLD	1	
	STA	TEMP	
	SLD	2	
	DAD	TEMP	
	BRU	1	1
TEMP	DDC	0	

Figure 45. 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 46.



	Opr (8–10)	Operand (12–19)	X (20)	31
1	SPB	STRIP	1	
2	DEC	128		
3	DEC	1		
4	DEC	3		
5	BRU	ERRØR		
6	STA	AMT#1		
7				

Figure 46. 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 47. **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 47.

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
BRU	4	1	EØF Return
BRU	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.
```
      O  ERRORS  TAPE  3
      O  ERRORS  TAPE  4
END OF PASS O
```

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.

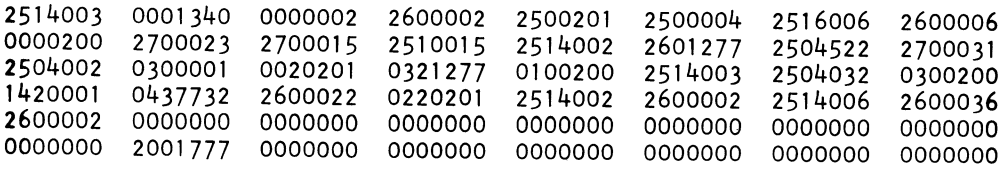
Memory Dumps

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 48), starting at memory location 0000 and continues until the manual clear button on 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 49). 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 48. **Printer Controller Octal Memory Dump**

```
Octal
Location

                                0000000  0001340  0000000  0000017  2516006  2600002
00230   0060000  0000002  2606060  0000000  0000000  0000000  0000000  0000000
00240   0000000  0000000  0000000  0000000  0000000  0000000  0000000  0000000
00250   0000000  2001777  0000000  0000000  0000000  0000000  0000000  0000000
00260   0000000  0000000  0000000  0000000  0000000  0000000  0000000  0000000
00400   0060126  0040171  0030606  0012604  0112100  0052322  0510101  0446060
00410   0606060  0606001  0004002  0606060  0060126  0040271  0000606  0012604
00420   0112200  0032322  0510200  0446060  0606060  0606001  0004001  0606060
00430   0606060  0606060  0606060  2606077  0000000  0000000  0000000  0000000
00440   0000000  0000000  0000000  0000000  0000000  0000000  0000000  0000000
```

Figure 49. **Programmed Octal Memory Dump**

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

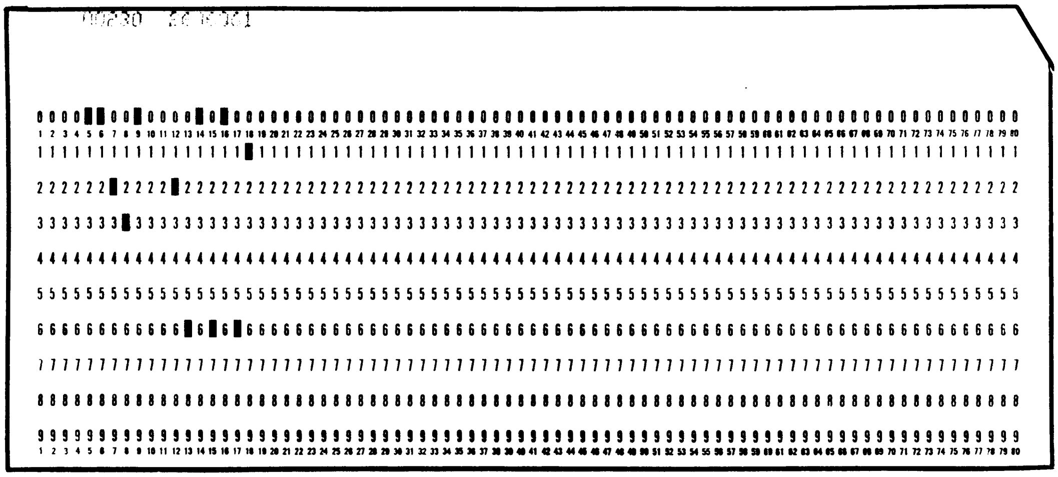
   The same registers listed in the other types are printed before each instruction is executed.

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.

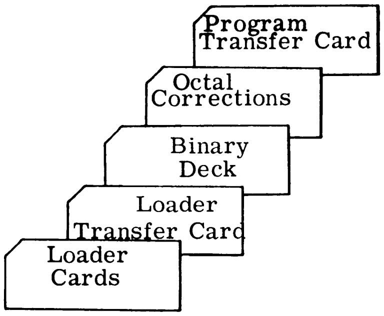
During debugging, it is best to use a binary loader with octal correction cards. This type of loader will read into memory the binary deck and then read the octal corrections into the specified memory words. Thus, errors can be corrected without repeated reassembly of the symbolic deck and, when the 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 50. **Octal Correction Card**

CD225B1.003 is used. To illustrate loader and correction card usage, the deck set-up using octal corrections for Routine CD225B1.003 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** 50 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.

APPENDIXES

APPENDIX A.

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 ll-0 and ll-2-8 as 52 and 12-0 and 12-2-8 as 32. The 1000 cards per minute card reader treats ll-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 B. OCTAL LIST OF GE-225 INSTRUCTIONS

Octal	Mnemonic	Word Times
0000000	LDA Y X Load A Register	2
0100000	ADD Y X *Decimal Add	2 2
0200000	SUB Y X *Decimal Subtract	3
0200000	SUB Y X Subtract	3
0200000 TT00000	WEF T Write End of File	2
02MMMMM TTNNNNN	WTD M T (blank) N Write Tape Decimal	2
0300000	STA Y X Store A	2
03MMMMM TTNNNNN	WTB M T (blank) N Write Tape Binary	2
0400000	BXL K X Branch If X Is Less Than	3
0420000(N=1)	RSD M N Read Document Single	2
0440000(N=1)	RDC M N Read Document Continuously	2
0460000(N=1)	PKT X N Pocket Select	2
04MMMMM TTNNNNN	RTD M T (blank) N Read Tape Decimal	2
0500000	BXH K X Branch If X Is Higher Than or Equal To	3
0500000(N=1)	HLT M N Halt Continuous Feeding	2
0520000(N=1) 0000000	ERB N End Read Busy	2

*Optional Instruction

Octal	Mnemonic	Word Times
05MMMMM TTNNNNN	RTB M T (blank) N Read Tape Binary	2
0600000	LDX Y X Load X	3
0600000 NN00000	SLW N Slew Paper N Lines	2
0700000	SPB Y X Store P and Branch	2
0X00000 XX00000	SLT K Slew Paper to Tape Punch	2
1000000	DLD Y X Double Length Load	3
1020000(N=2)	RSD M N Read Document Single	2
1040000(N=2)	RDC M N Read Document Continuously	2
1060000(N=2)	PKT X N Pocket Select	2
1100000	DAD Y X *Double Decimal Add	3 3
1100000	DAD Y X Double Length Add	3
1100000(N=2)	HLT M N Halt Continuous Feeding	2
1120000(N=2) 0000000	ERB N End Read Busy	2
1200000	DSU Y X *Double Decimal Subtract	5 5
1200000	DSU Y X Double Length Subtract	5
1200000 00MMMMM	RRF N F (blank) M Read from DSU F	2

Octal	Mnemonic	Word Times
1201000 00MMMMM	RRD N F (blank) M Read from DSU F	2
1202000 0000000	RAW N F (blank) zero Read After Write Check	2
1300000	DST Y X Double Length Store	3
1400000	INX K X Increment X	3
14MMMMM TTNNNNN	RBD M T (blank) N Read Backward Decimal	2
1500000	MPY Y X Multiply	9 to 23
15MMMMM TTNNNNN	RBB M T (blank) Read Backward Binary	2
1600000	DVD Y X Divide	26 to 29
1600000 TT00000	BKW T Backspace and Position Write Head	2
1700000	STX Y X Store X	3
2000000	EXT Y X Extract	3
2000000 01YYYYY	WPL Y N Write Print Line	2
2000000 TT00000	RWD T Rewind	2
2100000	*CAB Y Compare and Branch	2 to 4
2200000	*DCB Y Double Compare and Branch	2 to 6
2300000	ORY Y X Or A into Y	3

Octal	Mnemonic	Word Times
23MMMMM TTNNNNN	WTS M T (blank) N Write Tape Special Binary Mode	2
2400000	*MOV Y Move	4 + 2N
2500000 MMMMMMM	PRF F OCT (DSU Address) Position DSU File	2
2500004	HCR Halt Card Reader	2
2500005	OFF Power Off (Direct I/O Devices)	2
2500006	RPT Read Paper Tape	2
2500006	TYP Type	2
2500006	WPT Write Paper Tape	2
2500007	TON Typewriter On	2
2500011	RCS Read Control Switches	2
2500015	PON Punch On	2
2500016	HPT Halt Paper Tape Reader	2
2500014	RON Paper Tape Reader On	2
2500P20	SEL P X Select	2
2504001	LAQ Load A from Q	3
2504002	LDZ Load Zero into A Register	3
2504004	LQA Load Q from A	3
2504005	XAQ Exchange A and Q	3
2504006	MAQ Move A to Q	3

* This instruction is an optional feature.

Octal	Mnemonic	Word Times	Octal	Mnemonic	Word Times
2504012	NOP No Operation	3	250YY02	WCD Y Write Card Decimal	2
2504022	LDO Load One into A Register	3	250YY03	WCB Y Write Card Binary	2
2504032	ADO Add One	3	250YY10	RCF Y Read Cards Full	2
2504032	ADO Add One *Add One Decimal	3 3	250YY12	RCM Y Read Cards Mixed	2
2504040	CHS Change Sign of A Register	2	250YY17	WCF Y Write Cards Full	2
2504102	LMO Load Minus One into A Register	3	2510000	SRA K Shift Right A Register	2 to 12
2504112	SBO Subtract One *Subtract One Decimal	3 3	2510040	SCA K Shift Circular A Register	2 to 12
2504202	*LAC Load A Register from C Register	3	2510100	SNA K Shift N and A Right	2 to 12
2504210	*LCA Load C Register from A Register	3	2510400	SAN K Shift A and N Right	2 to 12
2504502	CPL Complement A	3	2511000	SRD K Shift Right Double	2 to 12
2504522	NEG Negate A	3	2511100	NAQ K Shift N, A, and Q Right	2 to 12
2506YY3	*SXG Y Select X Register Group	2	2511200	SCD K Shift Circular Double	2 to 12
2506011	SET DECMODE Set Decimal Mode	2	2511400	ANQ K Shift A into N and Q	2 to 12
2506012	SET BINMODE Set Binary Mode	2	2512000	SLA K Shift Left A Register	2 to 12
2506015	SET PST Set Automatic Priority Interrupt On	2	2512200	SLD K Shift Left Double	2 to 12
2506016	SET PBK Set Automatic Priority Interrupt Off	2	2513000	NOR K Normalize the A Register	3 to 12
250YY00	RCD Y Read Cards Decimal	2	2513200	DNO K Double Length Normalize	2 to 12
250YY01	RCB Read Cards Binary	2	2514000	BOD Branch on Odd	2
			2514001	BMI Branch on Minus	2

* This instruction is an optional feature.

Octal	Mnemonic	Word Times
2514002	BZE Branch on Zero	2
2514003	BOV Branch on Overflow	2
2514004	BPE Branch on Parity Error	2
2514005	BNR Branch on N Register Ready	2
2514006	BCR Branch on Card Reader Rea	2
2514007	BPR Branch on Card Punch Ready	2
2514720	BAR BAR 7 Branch on AAU Ready	2
2514721	BAR BMI 7 Branch on AAU Minus	2
2514722	BAR BZE 7 Branch on AAU Zero	2
2514723	BAR BOV 7 Branch on AAU Overflow	2
2514724	BAR BUF 7 Branch on AAU Underflow	2
2514727	BAR BER 7 Branch on AAU Error	2
2514P20	BCS BPR P Branch on Printer Ready	2
2514P20	BCS BRR P Branch on DSU Controller Ready	2
2514P20	BCS BTR P Branch on Tape Controller	2
2514P20(K=1)	BCS SKR P Branch on Document Handler K Ready	2
2514P21	BCS BAA P Branch on Any Alert	2
2514P21	BCS BEF P Branch on End of File	2
2514P21(File 0)	BCS FKR P Branch on File K Ready	2

Octal	Mnemonic	Word Times
2514P21(K=2)	BCS SKR P Branch on Document Handler Ready	2
2514P22	BCS BET P Branch on End of Tape	2
2514P22	BCS BOP P Branch on Printer Out of Paper	2
2514P22(File 1)	BCS FKR P Branch on File K Ready	2
2514P22(K=1)	BCS NPK P Branch on No Pocket Decision, Document Handler K	2
2514P23	BCS BOV P Branch on Printer Buffer Overflow	2
2514P23	BCS BRW P Branch on Tape Rewinding	2
2514P23(File 2)	BCS FKR P Branch on File K Ready	2
2514P23(K=2)	BCS NPK P Branch on No Pocket Decision, Document Handler K	2
2514P24	BCS BPE P Branch on Mag Tape Parity Error	2
2514P24	BCS BSA P Branch on Printer Slew Alert	2
2514P24(File 3)	BCS FKR P Branch on File K Ready	2
2514P24(K=1)	BCS FSK P Branch on Feeding, Document Handler K	2
2514P25	BCS BIO P Branch on Mag Tape I/O Buffer Error DSU	2
2514P25(K=2)	BCS FSK P Branch on Feeding, Document Handler K	2
2514P26	BCS BME P Branch on Mod 3 or 4 Error	2
2514P26(K=1)	BCS ICK P Branch on Invalid Character, Document Handler K	2
2514P26	BCS ICK P Branch on DSU Parity Error	2

Octal	Mnemonic	Word Times
2514P27	BCS BER P Branch on Error	2
2514P27(K=2)	BCS ICK P Branch on Invalid Character, Document Handler K	2
2514P30(K=1)	BCS SKE Branch on Any Error, Document Handler K	2
2514P31	BCS FAE P Branch on Error - On Any File	2
2514P31(K=2)	BCS SKE Branch on Any Error, Document Handler K	2
2514P32(K=1)	*BCS DQK Branch on Document TCD Correct, Document Handler K.	2
2514P32(File 0)	BCS FKE P Branch on File K, File Error	2
2514P33(K=2)	*BCS DQK Branch on Document TCD Correct, Document Handler K.	2
2514P33(File 1)	BCS FKE P Branch on File K, File Error	2
2514P34(File 2)	BCS FKE P Branch on File K, File Error	2
2514P35(File 3)	BCS FKE P Branch on File K, File Error	2
2514PCC	BCS XXX P Branch on Controller Selector	2
2516000	BEV Branch on Even	2
2516001	BPL Branch on Plus	2
2516002	BNZ Branch on Non-Zero	2
2516003	BNO Branch on No Overflow	2
2516004	BPC Branch on Parity Correct	2

* This instruction is an optional feature.

Octal	Mnemonic	Word Times
2516005	BNN Branch on N Register Not Ready	2
2516006	BCN Branch on Card Reader Not Ready	2
2516007	BPN Branch on Card Punch Not Ready	2
2516720	BAR BAN 7 Branch on AAU Not Ready	2
2516721	BAR BPL 7 Branch on AAU Plus	2
2516722	BAR BNZ 7 Branch on AAU Not Zero	2
2516723	BAR BNO 7 Branch on AAU No Overflow	2
2516724	BAR BNU 7 Branch on AAU No Underflow	2
2516727	BAR BNE 7 Branch on AAU No Error	2
2516P20	BCS BPN P Branch on Printer Not Ready	2
2516P20	BCS BRN P Branch on DSU Controller Not Ready	2
2516P20	BCS BTN P Branch on Tape Controller Not Ready	2
2516P20(K=1)	BCS SKN P Branch on Document Handler K Not Ready	2
2516P21	BCS BNA P Branch on Printer No Alert	2
2516P21	BCS BNF P Branch on No End of File	2
2516P21(File 0)	BCS FKN P Branch on File K Not Ready	2
2516P21(K=2)	BCS SKN P Branch on Document Handler K Not Ready	2
2516P22	BCS BNP P Branch if Printer Not Out of Paper	2

Octal	Mnemonic	Word Times
2516P22	BCS BNT P Branch on No End of Tape	2
2516P22(File 1)	BCS FKN P Branch on File K Not Ready	2
2516P22(K=1)	BCS PDK P Branch on Pocket Decision, Document Handler K	2
2516P23	BCS BNO P Branch on No Printer Buffer Overflow	2
2516P23	BCS BNR P Branch on No Tape Rewinding	2
2516P23(File 2)	BCS FKN P Branch on File K Not Ready	2
2516P23(K=2)	BCS PDK P Branch on Pocket Decision, Document Handler K	2
2516P24	BCS BNS P Branch on No Printer Slew Alert	2
2516P24	BCS BPC P Branch on Mag Tape Parity Correct	2
2516P24(File 3)	BCS FKN P Branch on File K Not Ready	2
2516P24(K=1)	BCS NFK P Branch on Not Feeding, Document Handler K	2
2516P25	BCS BIC P Branch on Mag Tape I/O Buffer Correct	2
2516P25(K=2)	BCS NFK P Branch on Not Feeding, Document Handler K	2
2516P26	BCS BNM P Branch on No Mod 3 or 4 Error	2
2516P26	BCS RPC P Branch on DSU Parity Correct	2
2516P26(K=1)	BCS VCK Branch on Valid Character, Document Handler K	2
2516P27	BCS BNE P Branch on No Error	2
2516P27(K=2)	BCS VCK Branch on Valid Character, Document Handler K	2
2516P30(K=1)	BCS SKC Branch on Document Handler K Correct	2
2516P31	BCS FAC P Branch on No Error - Any File	2
2516P31(K=2)	BCS SKC Branch on Document Handler K Correct	2
2516P32(File 0)	BCS FKC P Branch on File K, No Unit Error	2
2516P32(K=1)	*BCS NQK Branch on Document TCD Not Correct, Document Handler K	2
2516P33(File 1)	BCS FKC P Branch on File K, No Unit Error	2
2516P33(K=2)	*BCS NQK Branch on Document TCD Not Correct, Document Handler K	2
2516P34(File 2)	BCS FKC P Branch on File K, No Unit Error	2
2516P35(File 3)	BCS FKC P Branch on File K, No Unit Error	2
2516PCC	BCS XXX P Branch on Controller Selector	2
25MMMMM TTNNNNN	RTS M T (blank) N Read Tape Special Binary Mode	2
2600000	BRU Y X Branch Unconditionally	1
2700000	STO Y X Store Operand Address	3
3000000	FLD Y Load Auxiliary Arithmetic Unit	72 usec
30YYYYY 01XXXXX	WFL Y X N (WPL) Write Format Line	2

* This instruction is an optional feature

Mnemonic	Octal	Word Times
3100002	MAQ A Move AX to QX	49. 5 usec
3100010	SET NFLPOINT Set Normalized Floating-Point Mode	49. 5 usec
31YYYYY	FAD Y AAU Add	Min. 162 usec Max. 709 usec
3200002	LQA A Load QX From AX	49. 5 usec
3200010	SET UFLPOINT Set Unnormalized Floating-Point Mode	49. 5 usec
32YYYYY	FSU Y AAU Subtract	Min. 162 usec Max. 709 usec
3300000	FST Y Store Auxiliary Arithmetic Unit	72 usec
3500002	XAQ A Exchange AX and QX	117 usec

Mnemonic	Octal	Word Times
3500010	SET FIXPOINT Set Fixed-Point Mode	49. 5 usec
35MMMMM TTNNNNN	RBS M T (blank) N Read Backward Special Binary	2
35YYYYY	FMP Y AAU Multiply	Min. 297 usec Max. 1062 usec
3600002	LAQ A Load AX From QX	49. 5 usec
36YYYYY	FDV Y AAU Divide	Min.814.5 usec Max.1095 usec
3700000 00MMMMM	WRF N F (blank) M Write on MRADS Unit F	2
3701000 00MMMMM	WRD N F (blank) M Write on DSU F	2

APPENDIX C. ALPHABETIC LIST OF GE-225 INSTRUCTIONS

Mnemonic		Octal	Word Times
ADD	Y X	0100000	2
	*Decimal Add		2
ADD	Y X	0100000	2
	Add		
ADO		2504032	3
	Add One		
ADO		2504032	3
	*Add One Decimal		
ALF	(Pseudo)		
	Alphanumeric		
ANQ	K	2511400	2 to 12
	Shift A into N and Q		
BAR	BAN 7	2516720	2
	Branch on AAU Not Ready		
BAR	BAR 7	2514720	2
	Branch on AAU Ready		
BAR	BER 7	2514727	2
	Branch on AAU Error		
BAR	BMI 7	2514721	2
	Branch on AAU Minus		
BAR	BNE 7	2516727	2
	Branch on AAU No Error		
BAR	BNO 7	2516723	2
	Branch on AAU No Overflow		
BAR	BNU 7	2516724	2
	Branch on AAU No Underflow		
BAR	BNZ 7	2516722	2
	Branch on AAU Not Zero		
BAR	BOV 7	2514723	2
	Branch on AAU Overflow		
BAR	BPL 7	2516721	2
	Branch on AAU Plus		
BAR	BUF 7	2514724	2
	Branch on AAU Underflow		
BAR	BZE 7	2514722	2
	Branch on AAU Zero		

*Optional Instruction

Mnemonic		Octal	Word Times
BCN		2516006	2
	Branch on Card Reader Not Ready		
BCR		2514006	2
	Branch on Card Reader Ready		
BCS	BAA P	2514P21	2
	Branch on Any Alert		
BCS	BEF P	2514P21	2
	Branch on End of File		
BCS	BER P	2514P27	2
	Branch on Error		
BCS	BET P	2514P22	2
	Branch on End of Tape		
BCS	BIC P	2516P25	2
	Branch on Input/Output Buffer Correct		
BCS	BIO P	2514P25	2
	Branch on Input/Output Buffer Error		
BCS	BME P	2514P26	2
	Branch on Mod 3 or 4 Error		
BCS	BNA P	2516P21	2
	Branch on Printer No Alert		
BCS	BNE P	2516P27	2
	Branch on No Error		
BCS	BNF P	2516P21	2
	Branch on No End of File		
BCS	BNM P	2516P26	2
	Branch on No Mod 3 or 4 Error		
BCS	BNO P	2516P23	2
	Branch on No Printer Buffer Overflow		
BCS	BNP P	2516P22	2
	Branch if Printer Not Out of Paper		
BCS	BNR P	2516P23	2
	Branch on No Tape Rewinding		

Mnemonic	Octal	Word Times
BCS BNS P Branch on No Printer Slew Alert	2516P24	2
BCS BNT P Branch on No End of Tape	2516P22	2
BCS BOP P Branch on Printer Out of Paper	2514P22	2
BCS BOV P Branch on Printer Buffer Overflow	2514P23	2
BCS BPC P Branch on Tape Parity Correct	2516P24	2
BCS BPE P Branch on Tape Parity Error	2514P24	2
BCS BPN P Branch on Printer Not Ready	2516P20	2
BCS BPR P Branch on Printer Ready	2514P20	2
BCS BRN P Branch on DSU Controller Not Ready	2516P20	2
BCS BRR P Branch on DSU Controller Ready	2514P20	2
BCS BRW P Branch on Tape Rewinding	2514P23	2
BCS BSA P Branch on Printer Slew Alert	2514P24	2
BCS BTN P Branch on Tape Controller Not Ready	2516P20	2
BCS BTR P Branch On Tape Controller Ready	2514P20	2
*BCS DQK Branch on Document TCD Correct, Document Handler K.	2514P32(K=1) or 2514P33(K=2)	2

* This instruction is an optional feature

Mnemonic	Octal	Word Times
BCS FAC P Branch on No Error - Any File	2516P31	2
BCS FAE P Branch on Error - On Any File	2514P31	2
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
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
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
BCS FKR P Branch on File K Ready	2514P21(File 0) or 2514P22(File 1) or 2514P23(File 2) or 2514P24(File 3)	2
BCS FSK P Branch on Feeding, Document Handler K	2514P24(K=1) or 2514P25(K=2)	2
BCS ICK P Branch on Invalid Character, Document Handler K	2514P26(K=1) or 2514P27(K=2)	2
BCS NFK P Branch on Not Feeding, Document Handler K	2516P24(K=1) or 2516P25(K=2)	2
BCS NPK P Branch on No Pocket Decision, Document Handler K	2514P22(K=1) or 2514P23(K=2)	2
*BCS NQK Branch on Document TCD Not Correct, Document Handler K	2516P32(K=1) or 2516P33(K=2)	2

Mnemonic		Octal	Word Times
BCS	PDK P	2516P22(K=1) or 2516P23(K=2)	2
	Branch on Pocket Decision, Document Handler K		
BCS	RPC P	2516P26	2
	Branch on DSU Parity Correct		
BCS	RPE P	2514P26	2
	Branch on DSU Parity Error		
BCS	SKC	2516P30(K=1) or 2516P31(K=2)	2
	Branch on Document Handler K Correct		
BCS	SKE	2514P30(K=1) or 2514P31(K=2)	2
	Branch on Any Error, Document Handler K		
BCS	SKN P	2516P20(K=1) or 2516P21(K=2)	2
	Branch on Document Handler K Not Ready		
BCS	SKR P	2514P20(K=1) or 2514P21(K=2)	2
	Branch on Document Handler K Ready		
BCS	VCK	2516P26(K=1) or 2516P27(K=2)	2
	Branch on Valid Character, Document Handler K		
BCS	XXX P	2514PCC or 2516PCC	2
	Branch on Controller Selector		
BEV		2516000	2
	Branch on Even		
BKW	T	1600000 TT00000	2
	Backspace and Position Write Head		
BMI		2514001	2
	Branch on Minus		
BNN		2516005	2
	Branch on N Register Not Ready		
BNO		2516003	2
	Branch on No Overflow		
BNR		2514005	2
	Branch on N Register Ready		
BNZ		2516002	2
	Branch on Non-Zero		
BOD		2514000	2
	Branch on Odd		
BOV		2514003	2
	Branch on Overflow		
BPC		2516004	2
	Branch on Parity Correct		
BPE		2514004	2
	Branch on Parity Error		
BPL		2516001	2
	Branch on Plus		
BPN		2516007	2
	Branch on Card Punch Not Ready		
BPR		2514007	2
	Branch on Card Punch Ready		
BRU	Y X	2600000	1
	Branch Unconditionally		
BSS(Pseudo)			
	Block Started by Symbol		
BXH	K X	0500000	3
	Branch if X is Higher Than or Equal To		
BXL	K X	0400000	3
	Branch If X is Less Than		
BZE		2514002	2
	Branch on Zero		
*CAB	Y X	2100000	2 to 4
	Compare and Branch		
CHS		2504040	2
	Change Sign of A Register		
CPL		2504502	3
	Complement A		
DAD	Y X	1100000	3
	Double Decimal Add		3

*** This instruction is an optional feature.**

Mnemonic			Octal	Word Times
DAD	Y	X	1100000	3
	Double Length Add			
* DCB	Y	X	2200000	2 to 6
	Double Compare and Branch			
DDC (Pseudo)				
	Double Length Decimal			
DEC (Pseudo)				
	Decimal			
DLD	Y	X	1000000	3
	Double Length Load			
DNO	K		2513200	2 to 12
	Double Length Normalize			
DST	Y	X	1300000	3
	Double Length Store			
DSU	Y	X	1200000	5
	* Double Decimal Subtract			
DSU	Y	X	1200000	5
	Double Length Subtract			
DVD	Y	X	1600000	26 to 29
	Divide			
EJT (Pseudo)				
	Eject Printer Paper			
END(Pseudo)				
	End of Program			
EQO (Pseudo)				
	Equals Octal			
EQU (Pseudo)				
	Equals			
ERB	N		0520000(N=1)	2
			0000000	
		or	1120000(N=2)	
			0000000	
	End Read Busy			
EXT	Y	X	2000000	3
	Extract			
FAD	Y		31YYYYY	Min.162 usec
				Max.709 usec
	AAU Add			
FDC (Pseudo)				
	Floating Point Decimal			
FDV	Y		36YYYYY	Min. 814. 5 usec
				Max.1095 usec
	AAU Divide			
FLD	Y		3000000	72 usec
	Load Auxiliary Arithmetic Unit			
FMP	Y		35YYYYY	Min. 297 usec
				Max. 1062 usec
	AAU Multiply			
FST	Y		3300000	72 usec
	Store Auxiliary Arithmetic Unit			
FSU	Y		32YYYYY	Min. 162 usec
				Max. 709 usec
	AAU Subtract			
HCR			2500004	2
	Halt Card Reader			
HLT	M	N	0500000(N=1)	2
		or	1100000(N=2)	
	Halt Continuous Feeding			
HPT			2500016	2
	Halt Paper Tape Reader			
INX	K	X	1400000	3
	Increment X			
*LAC			2504202	3
	Load A Register from C Register			
LAQ			2504001	3
	Load A from Q			
LAQ	A		3600002	49. 5 usec
	Load AX from QX			
*LCA			2504210	3
	Load C Register from A Register			
LDA	Y	X	0000000	2
	Load A Register			

*** This instruction is an optional feature.**

Mnemonic		Octal	Word Times
LDO		2504022	3
	Load One into A Register		
LDX	Y X	0600000	3
	Load X		
LDZ		2504002	3
	Load Zero into A Register		
LMO		2504102	3
	Load Minus One into A Register		
LOC (Pseudo)			
	Location in Octal		
LQA		2504004	3
	Load Q from A		
LQA	A	3200002	49. 5 usec
	Load QX From AX		
LST (Pseudo)			
	List		
MAL (Pseudo)			
	Multiple Alphanumeric		
MAQ		2504006	3
	Move A to Q		
MAQ	A	3100002	49. 5 usec
	Move AX to QX		
*MOV	Y	2400000	4 + 2N
	Move		
MPY	Y X	1500000	9 to 23
	Multiply		
NAL (Pseudo)			
	Negative Alphanumeric		
NAM (Pseudo)			
	Print Name or Title on Each Page		
NAQ	K	2511100	2 to 12
	Shift N, A, and Q Right		
NEG		2504522	3
	Negate A		
NLS (Pseudo)			
	No List.		

* This instruction is an optional feature.

Mnemonic		Octal	Word Times
NOP		2504012	3
	No Operation		
NOR K		2513000	3 to 12
	Normalize the A Register		
OCT (Pseudo)			
	Octal		
OFF		2500005	2
	Power Off (Direct I/O Devices)		
ORG (Pseudo)			
	Origin		
ORY	Y X	2300000	3
	Or A into Y		
PAL (Pseudo)			
	Multiple Alphanumeric for Printer with Print Line Indicator		
PKT	X N	0460000(N=1) or 1060000(N=2)	2
	Pocket Select		
PON		2500015	2
	Punch On		
PRF	F	2500000	2
OCT	(MRADS Address)	MMMMMMM	
	Position DSU File		
RAW	N F	1202000	2
(blank)	zero	0000000	
	Read After Write Check		
RBB	M T	15MMMMM	2
(blank)	N	TTNNNNN	
	Read Backward Binary		
RBD	M T	14MMMMM	2
(blank)	N	TTNNNNN	
	Read Backward Decimal		
RBS	M T	35MMMMM	2
(blank)	N	TTNNNNN	
	Read Backward Special Binary		

Mnemonic			Octal	Word Times
RCB			250YY01	2
	Read Cards Binary			
RCD	Y		250YY00	2
	Read Cards Decimal			
RCF	Y		250YY10	2
	Read Cards Full			
RCM	Y		250YY12	2
	Read Cards Mixed			
RCS			2500011	2
	Read Control Switches			
RDC	M	N	0440000(N=1)	2
		or	1040000(N=2)	
	Read Document Continuously			
REM (Pseudo)				
	Remarks			
RON			2500014	2
	Paper Tape Reader On			
RPT			2500006	2
	Read Paper Tape			
RRD	N	F	1201000	2
(blank)	M		00MMMMM	
	Read from DSU F			
RRF	N	F	1200000	2
(blank)	M		00MMMMM	
	Read from DSU F			
RSD	M	N	0420000(N=1)	2
		or	1020000(N=2)	
	Read Document Single			
RTB	M	T	05MMMMM	2
(blank)	N		TTNNNNN	
	Read Tape Binary			
RTD	M	T	04MMMMM	2
(blank)	N		TTNNNNN	
	Read Tape Decimal			
RTS	M	T	25MMMMM	2
(blank)	N		TTNNNNN	
	Read Tape Special Binary Mode			
RWD		T	2000000	2
			TT00000	
	Rewind			

Mnemonic			Octal	Word Times
SAN	K	X	2510400	2 to 12
	Shift A and N Right			
SBO			2504112	3
	Subtract One			
SBO			2504112	3
	*Subtract One Decimal			
SBR (Pseudo)				
	Subroutine Call			
SCA	K	X	2510040	2 to 12
	Shift Circular A Register			
SCD	K	X	2511200	2 to 12
	Shift Circular Double			
SEL	P	X	2500P20	2
	Select			
SEQ (Pseudo)				
	Check Source Program Card Sequence Numbers			
SET BINMODE			2506012	2
	Set Binary Mode			
SET DECMODE			2506011	2
	Set Decimal Mode			
SET FIXPOINT			3500010	49.5 usec
	Set Fixed-Point Mode			
SET NFLPOINT			3100010	49.5 usec
	Set Normalized Floating-Point Mode			
SET	PBK		2506016	2
	Set Automatic Priority Interrupt Off			
SET	PST		2506015	2
	Set Automatic Priority Interrupt On			
SET UFLPOINT			3200010	49.5 usec
	Set Unnormalized Floating-Point Mode			
SLA	K	X	2512000	2 to 12
	Shift Left A Register			
SLD	K	X	2512200	2 to 12
	Shift Left Double			
SLT	K		0X00000	2
			XX00000	
	Slew Paper to Tape Punch			

Mnemonic		Octal	Word Times
SLW	N	0600000 NN00000	2
	Slew Paper N Lines		
SNA	K X	2510100	2 to 12
	Shift N and A Right		
SPB	Y X	0700000	2
	Store P and Branch		
SRA	K X	2510000	2 to 12
	Shift Right A Register		
SRD	K	2511000	2 to 12
	Shift Right Double		
STA	Y X	0300000	2
	Store A		
STO	Y X	2700000	3
	Store Operand Address		
STX	Y X	1700000	3
	Store X		
SUB	Y X	0200000	3
	Subtract		
SUB	Y X	0200000	3
	*Decimal Subtract		
* SXG	Y	2506YY3	2
	Select X Register Group		
TCD(Pseudo)			
	Punch Transfer Card		
TON		2500007	2
	Typewriter On		
TYP		2500006	2
	Type		
WCB	Y	250YY03	2
	Write Card Binary		
WCD	Y	250YY02	2
	Write Card Decimal		
WCF	Y	250YY17	2
	Write Cards Full		
WEF	T	0200000 TT00000	2
	Write End of File		
WFL (WPL)	N Y X	30YYYYY 01XXXXX	2
	Write Format Line		
WPL	Y N	2000000 01YYYYY	2
	Write Print Line		
WPT		2500006	2
	Write Paper Tape		
WRD (blank)	N F M	3701000 00MMMMM	2
	Write on DSU F		
WRF (blank)	N F M	3700000 00MMMMM	2
	Write on DSU F		
WTB (blank)	M T N	03MMMMM TTNNNNN	2
	Write Tape Binary		
WTD (blank)	M T N	02MMMMM TTNNNNN	2
	Write Tape Decimal		
WTS (blank)	M T N	23MMMMM TTNNNNN	2
	Write Tape Special Binary Mode		
XAQ		2504005	3
	Exchange A and Q		
XAQ	A	3500002	117 usec
	Exchange AX and QX		
Z (Pseudo)			
	Octal Operation Code		

* This instruction is an optional feature.

Progress Is Our Most Important Product

GENERAL ELECTRIC

INFORMATION SYSTEMS DIVISION

LITHO U.S.A.