CPB-374

GE-235 CENTRAL PROCESSOR REFERENCE MANUAL

GENERAL ELECTRIC
COMPUTER DEPARTMENT

GE-235

CENTRAL PROCESSOR

REFERENCE MANUAL

CPB-374

MARCH 1964

GENERAL ELECTRIC
COMPUTER DEPARTMENT

PREFACE

The GE-235 Central Processor Programming and Operating Manual has been compiled as a single source factual information guide to programming and operating the central processor of General Electric's 235 system. The material has been arranged in three basic sections:

1. Components and operation of the entire GE-235 system that personnel in both the operating and programming fields should know
2. Programming the central processor
3. Specific functions of operating the GE-235 central processor.

While the manual has been primarily arranged for the convenience of GE-235 system operators and programmers, the material covered makes it useful as an instruction guide.

A broader discussion of the system is contained in the GE-235 System Manual (CPB-267).

Complete information on programming and operating the many peripheral subsystems which are available with the GE-235 system is covered in individual manuals for each of these peripherals. These manuals are available from General Electric Computer Department offices.

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

CONTENTS

Page

SECTION B. PROGRAMMING CONSIDERATIONS

Page

SECTION C. OPERATING THE GE-235 SYSTEM

ILLUSTRATIONS

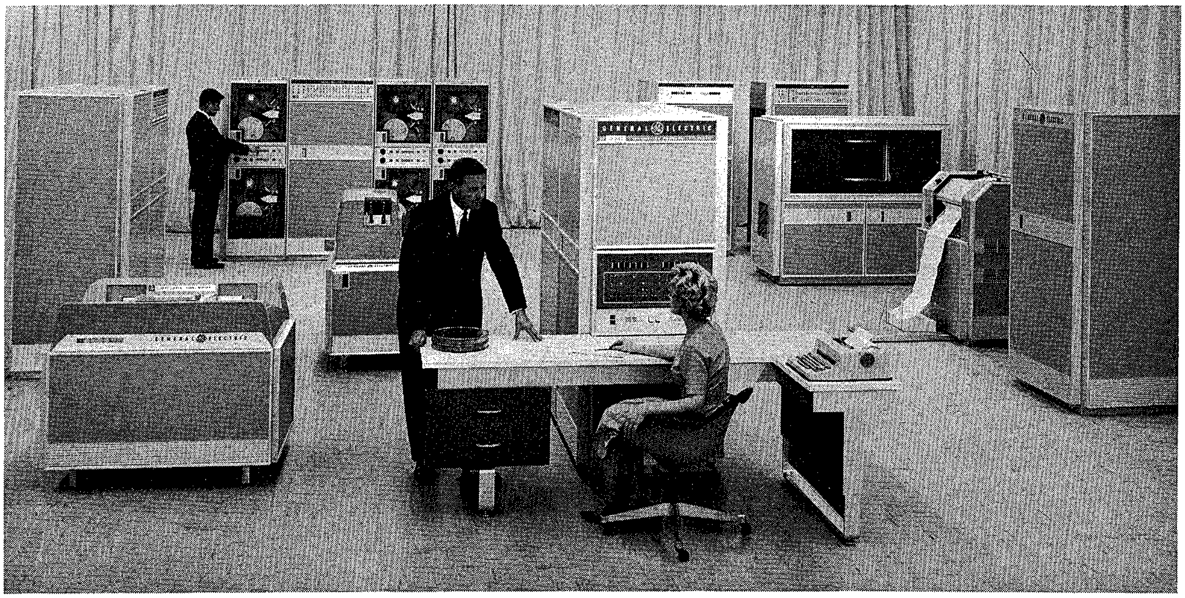
Figure Page

Figure Page

Table

SECTION A

THE GE-235 INFORMATION PROCESSING SYSTEM



The GE-235 Information Processing System

I. GENERAL ELECTRIC'S COMPATIBLE SYSTEMS

The GE-235 is a powerful new fast and versatile member of General Electric's Compatibles/200 information processing systems. Like the other members of this family--the GE-200, GE-215 and GE-225--the GE-235 is complemented by a full range of powerful programming tools. Together, the equipment and programming provide a fully integrated approach to the total information processing needs of business and finance, government, scientific and engineering applications.

The basic design philosophy of the new GE-235 system is an extension of that of the GE-215 and GE-225, which have been proven fast, accurate, highly reliable and economical in highly divergent fields. The GE-235 system is thus upward compatible with the GE-215 and GE-225 in programming, coding and logic. As a result, hundreds of programs, tested and proved on the other compatible systems, can immediately be processed on a GE-235 system having the same configuration. Only in the relatively few programs containing timing loops, minor changes may be necessary.

The modular design of the GE-235 system provides flexibility in meeting data processing requirements for a wide range of applications. A 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, and the features of the central processor, are determined largely by the desired applications. The principal features of the GE-235 peripheral subsystems are covered briefly on the following pages.

Input data can be read from the following system components:

- Punched Card reader
- Perforated Tape reader
- Magnetic Tape unit
- Console Typewriter Keyboard
- Magnetically-encoded printing (MICR) on paper documents
- Disc Storage Unit
- DATANET data communications units.

Output data can be written on the following system components:

- **Card Punch for tabulating cards**
- **Perforated Tape punch**
- **Magnetic Tape unit**
- **Disc Storage Unit**
- **Printed Reports on High-Speed Printer**
- **Console Typewriter printer**
- **DATANET data communications units**

In addition, both alphabetic and numeric data can be received or produced by the GE-235 system, either locally, or over long distances from the central processor, using peripheral data transmission equipment.

II. SYSTEM COMPONENTS

The GE-235 is a solid-state, single-address computer that operates under both stored program and operator control. It features a priority control system that permits the input and output peripheral subsystems to operate concurrently. This results in a much faster system than one in which operations are handled consecutively.

CENTRAL PROCESSOR

GE-235 Central Processor

The central processor and console is the heart of the GE-235 system. It has the following features and operating characteristics:

- 6 microsecond instruction execution time
- Magnetic core memories of 4096, 8192, or 16,384 words are available
- 20 and 40-bit word lengths - plus error-checking bits
- Index words are used for automatic instruction modification
- Accomodates alphabetic, numeric or binary information
- Parity check is made on all words stored in memory
- Up to 10 input/output peripheral subsystems can be operated concurrently with computations
- Complete read-write-compute simultaneity achieved by a priority control system for all input/output peripheral subsystems
- Over 300 instructions provide programming flexibility
- Total retrieve and execute times for typical operations are:

  Test and Branch - 12 microseconds
  Branch - 6 microseconds
  Add or Subtract - 12 microseconds
  Multiply - 30 microseconds minimum, 138 microseconds maximum
  Divide - 156 microseconds minimum, 174 microseconds maximum

- Input/Output Typewriter - Transmits input data to the central processor while printing the data.
- Control Console - Provides for complete operator control and communication with the system. Also displays contents of the significant registers and provides control signals to the operator for the effective monitoring of system operations.
- Floating Point Arithmetic capability is attained through an optional Auxiliary Arithmetic Unit.
- Other Standard Options to the GE-235 central processor include:
  - Single Access Controller Selector Channels - A data transfer and control module that grants peripheral subsystem--other than the punched card and perforated tape equipment--access to memory on the basis of their data rate, permitting simultaneous operation of peripherals and computation.
  - Dual Access Controller Selector Channels - Doubles memory access available to input/output equipment when processing requirements increase.

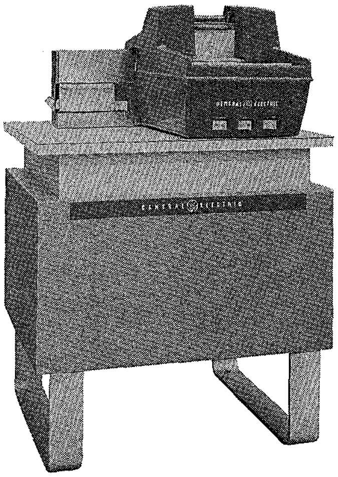
- Automatic Program Interrupt - As peripherals complete a function and are ready for next operation, the main program is interrupted and an executive program may issue new instructions to make maximum use of peripherals.
- Decimal Package - Three-way Compare, Decimal Arithmetic on BCD data without conversion to binary, and 31 Additional Index Groups with 93 additional address modification words.
- Move Command - A block of data can be transferred from one area of memory to another with one instruction.
- Real Time Clock - A 24-hour clock used for timing and logging operations in increments of 1/6 of a second under program control.

Detailed descriptions of these features and their operation are covered in subsequent sections of this manual.

GE-235 PUNCHED CARD SUBSYSTEMS

The GE-235 Information Processing System features two types of card readers and card punch units with which to process information via tabulating cards. The reader and punch units for a given system are usually selected on the basis of desired operating speed.

Card Readers


GE-235 400-CPM Card Reader

GE-235 1000-CPM Card Reader

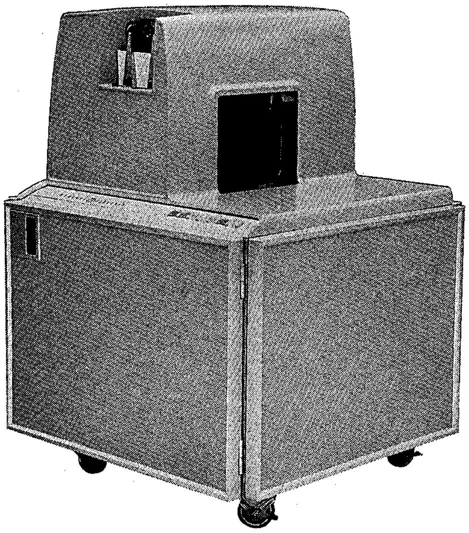
The following features apply to both models of card reader:

- Punched cards may be read in alphanumeric (binary coded decimal or Hollerith), 10-row binary, and 12-row binary formats.
- Card readers can operate simultaneously with central processor computation and other input/output operations.
- Cards are read serially, column by column, by photocells, providing reliable and accurate card reading either continuously or on demand.
- Read both alphanumeric and binary punched cards randomly intermixed.
- Input and output card bins have the following capacities:

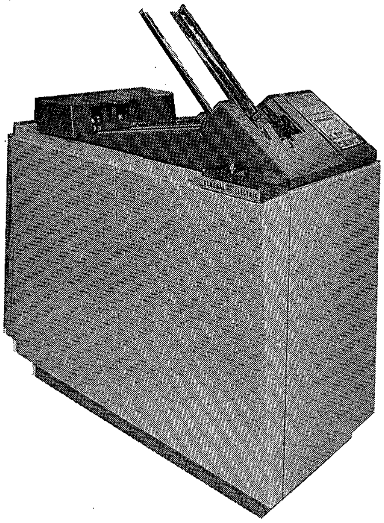
400 card-per-minute reader	-	600 cards per bin
1000 card-per-minute reader	-	2000 cards per bin

- Checking features (both readers): Card timing and synchronization, optical equipment operating, card feed error, card read error, and invalid characters. In addition, the 1000-card-per-minute reader performs input hopper empty, output stacker full, end-of-file test, and card jam error checks.
- Both readers will read at variable speeds (non synchronous).

Card Punches



GE-235 100-CPM Card Punch



GE-235 300-CPM Card Punch

The card punches with the GE-235 system have these features:

- Cards are punched in decimal (Hollerith), 10-row binary, or 12-row binary modes.
- Punching may occur simultaneously with computation and other input/output peripheral operations.
- Cards are punched in parallel, row by row, with all 80 columns of each row being punched simultaneously.

Both card punch models include card feed and card positioning checks. In the 100-card-per-minute punch, checks are made for incorrect double punches and blank columns.

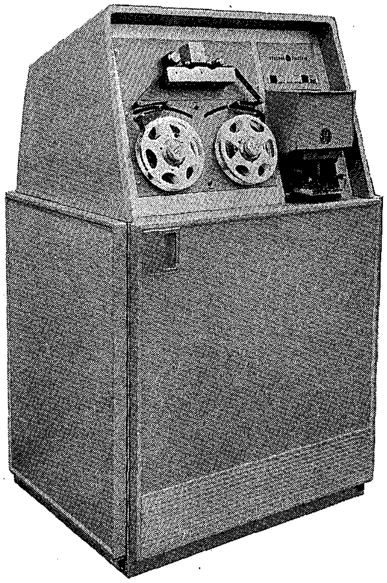
In the 300-card-per-minute punch, accuracy is checked by a read-after-punch hole count.

The 100-card-per-minute punch can also be operated off-line for gang punching of cards.

Input and output card bins have the following capacities:

100-card-per-minute punch	- 800 cards in each bin
300-card-per-minute punch	- 3000 cards in each bin

GE-235 PERFORATED TAPE SUBSYSTEM



GE-235 Perforated Tape Subsystem

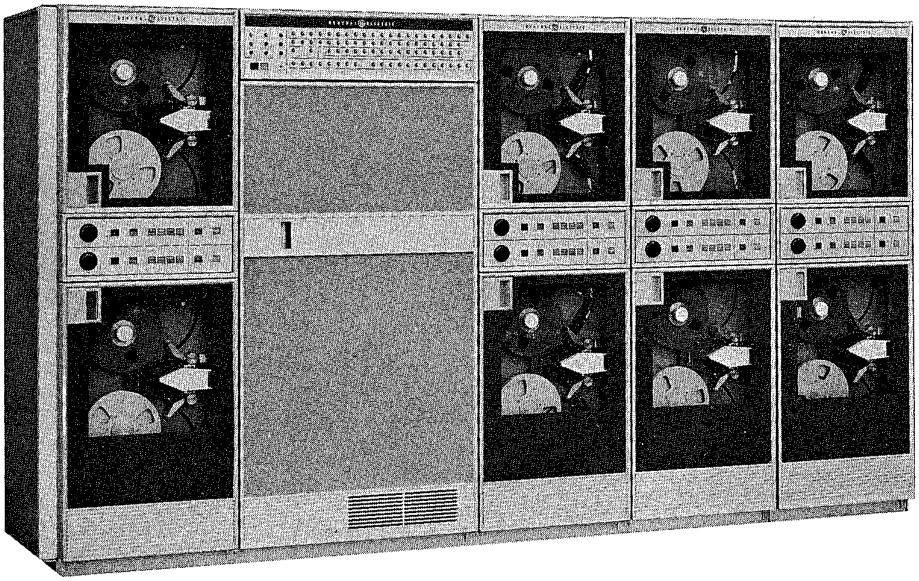
The GE-235 system includes equipment for reading and punching perforated tape. Combinations include a reader, punch and spooler mechanism in one cabinet; a reader and punch; a reader and spooler; a reader only; or, a punch only, depending on the system selected. Perforated tape is usually generated as a by product of transactions recorded on tape-punching typewriters, billing machines, remote terminal links and calculators, and data communications systems.

The subsystem features:

- Reading and punching of 5, or 6-, 7- 2nd 8-level tapes under program control
- Simultaneous operation with computation and other input/output operations
- Reading at either of two speeds--

  250 characters per second (10 characters per inch at 25 inches per second)
  1000 characters per second (10 characters per inch at 100 inches per second)
- Punching at 110 characters per second (10 characters per inch)
- Parity error checking when reading or punching 7- or 8-level tape

GE-235 MAGNETIC TAPE SUBSYSTEM



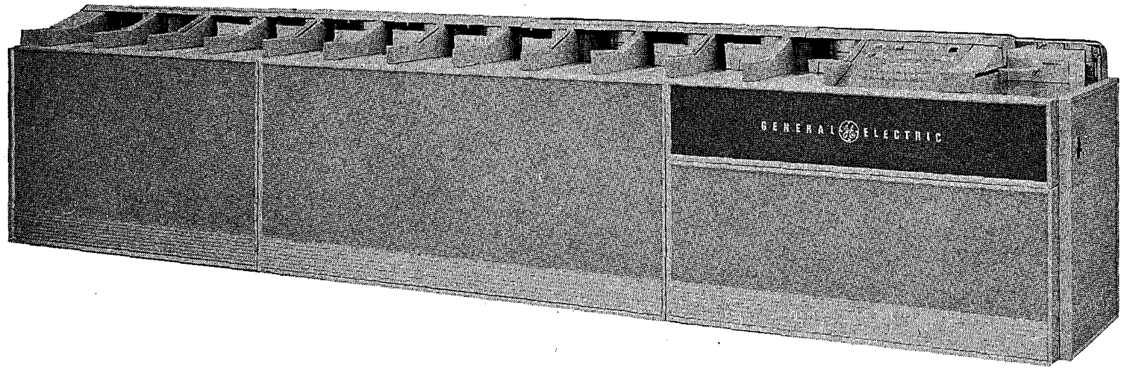
GE-235 Magnetic Tape Subsystem

The GE-235 Magnetic Tape Subsystem provides a fast method for transmission of extensive files and other large quantities of information into and out of the GE-235 central processor. It is comprised of two sections: magnetic tape controllers, and magnetic tape handlers. Principal features include:

- Each controller can direct the operations of up to 8 tape handlers
- Up to 7 controllers (with a total of 56 tape handlers) can be connected to the GE-235 system
- Controller selects the tape handler and provides complete control and error checking
- Handlers read from and write on tape at 75 inches per second (ips); rewind at 150 ips.
- Two character densities are available--

  low density - 200 characters per inch (15,000 characters per second)
  high density - 556 characters per inch (41,600 characters per second)
- Standard tape is 1/2 inch wide, 1.5 mils thick, Mylar, up to 2400 feet in length on 10 1/2 inch diameter reels. Millions of characters may be recorded on a single reel of tape.
- Magnetic tape operations occur simultaneously with computation and other input-output peripheral operations.
- Error checking features include subsystem ready, read after write check, and end-of-file, end-of-tape, tape parity error checks, echo check, and buffer check.

GE-235 12-POCKET DOCUMENT HANDLER SUBSYSTEM

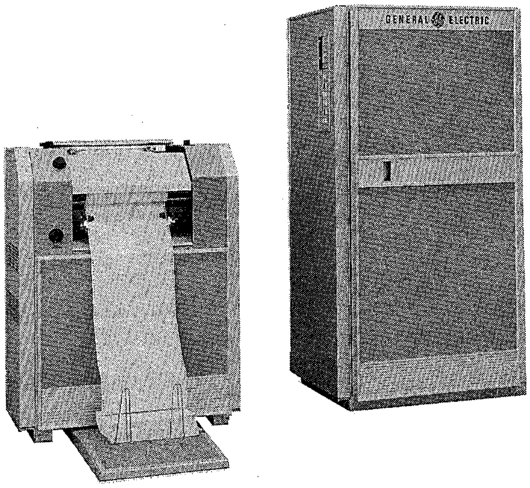


GE-235 12-Pocket High-Speed Document Handler

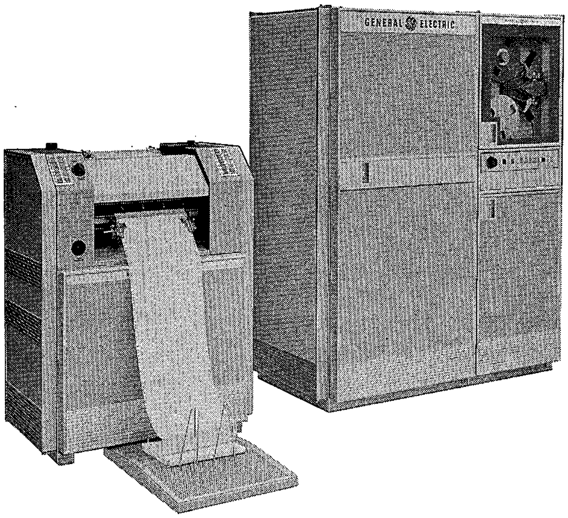
The GE 12-pocket document handler makes possible the high-speed sorting and reading of paper documents--such as bank checks--encoded with a special set of characters printed in magnetic ink (called MICR printing). This encoded information can be read directly into the memory of the GE-235 system, while at the same time the documents are sorted into any combination of the 12 pockets in a predetermined order. Or, documents can be sorted without reading the information into memory. This subsystem can:

- Read the special E13b font MICR characters which are also readable by the human eye
- Read and sort up to 1200 magnetically encoded documents per minute
- Sort up to 2400 documents per minute when two sorters are operated from each document handler controller
- Sort documents in a wide variety of formats, and ranging in size from 2-1/2 by 5-1/4 inches, to 3-3/4 by 8-3/4 inches
- Handle defaced and multilated documents
- Hold up to 2500 documents in the input feed hopper, and up to 1500 documents in each of the 12 sort pockets
- Checking features include checks for parity generation, cue characters, long characters, multiple reading, missing digit detection, and an optional transposition digit check.

GE-235 HIGH-SPEED PRINTER SUBSYSTEMS



GE-235 On-Line High-Speed Printer



GE-235 Off/On-Line High-Speed Printer

The high-speed printer subsystems permit the rapid printing of reports and other output type information from the GE-235 system. They are used where volumns of information must be transformed into a permanent visual record. Two models are available: One an on-line printer that operates under computer program control; the other operates off or on line and can print large-volume reports from information on magnetic tape, while operating independent of computer control.

The on-line printer has the following features:

- Prints 900 lines per minute
- Prints 120 alphanumeric characters per line (160 optional)
- Prints in Open Gothic type style, spaced 10 characters per inch horizontally, and 6 lines per inch vertically
- Provides automatic editing of printed format, and buffering of information from computer
- Paper width may range from 3-1/2 to 19-1/2 inches
- Paper length may be up to 22 inches per page in fan-folded, continuous-form type
- Printer makes up to five copies, depending on paper weight
- Total of 50 characters are provided as standard
- Printer can advance paper vertically at 25 inches per second

Checking features include a parity check on each word received from memory, end of paper detection, inoperative print hammer drive detection, buffer overflow check, and print cycle check.

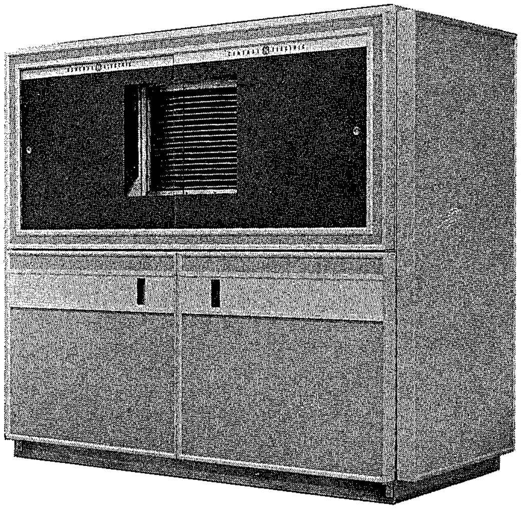
The off/on line high speed printer model, also has the same standard features, plus the following:

- Controller unit contains its own magnetic tape handler for off-line printing of data recorded on tape
- A 1024-character memory provides data storage and buffering
- Any one of up to 7 intermixed report formats written on magnetic tape may be selected for printing by push button control

Optional features available include:

- Choice of both 6 and 10 vertical lines per inch. This option gives the printers symmetrical plotting capability
- FORTRAN character set
- Choice of both 6 and 8 vertical lines per inch

GE-235 DISC STORAGE UNIT (DSU) SUBSYSTEM



GE-235 Disc Storage Unit

The Disc Storage Unit subsystem provides high-speed storage for large amounts of data so that any specific record can be immediately retrieved for processing. It thus greatly extends the memory capability of the GE-235 system. Since data does not have to be stored in any specific sequence, no sorting is required for retrieval.

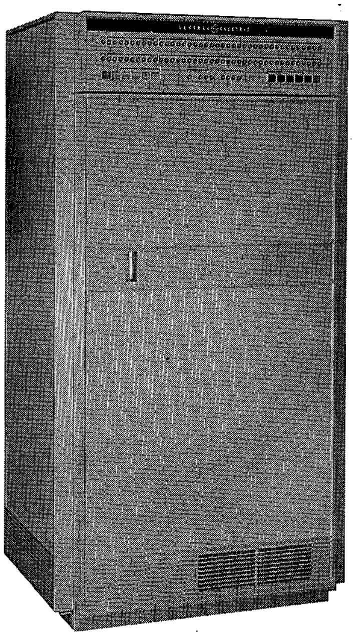
The subsystem consists of an electronics unit, a controller unit, and a disc file unit. One controller will handle from one to four disc file units. Features and characteristics of the subsystem include:

- Each file unit will store up to 34.4 million numeric digits, or 18.8 million alphabetic characters
- Data is stored on 16 magnetic-coated, rotating discs
- Data is written on, or read from each disc with 8 read/write heads
- Data is stored in 64-word groups called frames (records)
- One instruction sequence from the central processor can result in the reading or writing of from 1 to 16 frames (64 to 1024 words)
- Data can be stored in the disc file at an average rate of 62,500 characters per second
- Heads can move from one read/write position to another in an average of less than 200 milliseconds.

- Average frame latency time is 26 milliseconds.

- Controller performs error checks when data is read from, or written on discs. Two-way error detection permits the program to locate an error and correct it. Controller also performs a read-after-write comparison without program effort.

GE-235 AUXILIARY ARITHMETIC UNIT (AAU)



GE-235 Auxiliary Arithmetic Unit

The addition of the Auxiliary Arithmetic Unit to the GE-235 system extends its arithmetic capability to include floating point and double precision arithmetic. It is not a peripheral device; it is an extension of the basic arithmetic unit of the Central Processor. All peripheral operations may occur concurrently with AAU operations. The AAU:

- Operates on-line to do floating point and double word length calculations

- Circuits process floating point arithmetic at much higher speeds than those attained using programming techniques

- Provides 3 modes of calculations--

  Unnormalized floating point
  Normalized floating point
  Double Precision Fixed Point

- Provides the following operation times on floating point normalized calculations--

Add or Subtract	- 24 to 36 microseconds
Multiply	- 30 to 54 microseconds
Divide	- 72 to 78 microseconds

The Auxiliary Arithmetic Unit provides GE-235 users in the scientific and engineering fields with an invaluable tool.

GENERAL ELECTRIC DATANET* DATA COMMUNICATIONS EQUIPMENT

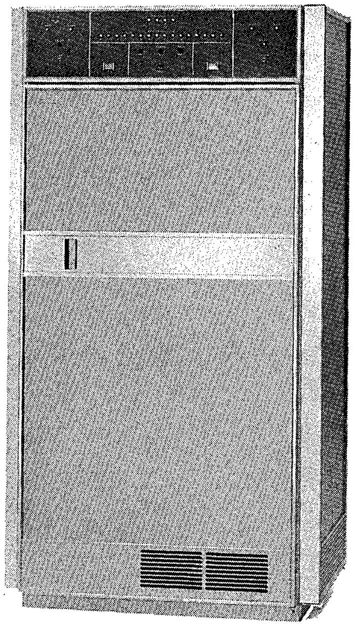
A broad selection of data communications equipment enables the GE-235 system to automatically receive and process information originated at locations remote from the computer center. Also, information can be automatically sent (replies, results, etc.) to the remote locations.

Information from remote locations can be handled through data accumulation systems, Teletype equipment, or from perforated tape generated as a byproduct of business machines.

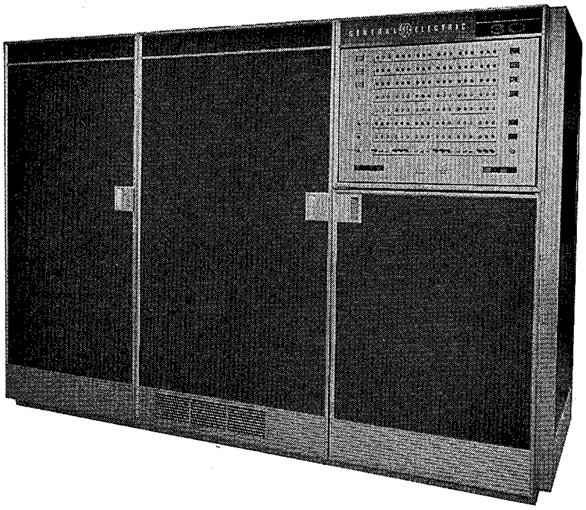
Special data communications devices are available to process information from special peripherals, such as sensors, readers, and analog devices. These DATANET systems and devices have a wide range of applications in business, scientific and manufacturing fields where large volumes of data must be exchanged over distances. Detailed information on each of these General Electric DATANET systems is given in individual manuals.

*Trademark of General Electric Company

The DATANET equipments which are compatible with the GE-235 system are:



GE-235 DATANET-15



GE DATANET-30 Data Communications Processor

DATANET-15 Data Transmission Controller--An on-line controller which accommodates up to 15 communication lines to let the GE-235 communicate with remote input/output stations. It accepts start-stop asynchronous data in 5, 6, 7, or 8-level codes at transfer rates from 60 to 2400 bits per second.

DATANET-30 Data Communications Processor--A stored program data communications processor designed for real-time systems. It may be operated as a separate system, or with the GE-235 system. It provides magnetic core memory of 4096, 8192, or 16,384-word sizes, with 7-microsecond access time. Operates with its own optional disc storage unit, magnetic tape and receive parallel unit subsystems.

DATANET-60 High-Speed Buffered Channel--A special buffered channel that permits non-standard peripherals to be connected to the GE-235 system. It transfers up to 165,000 characters of data per second, and has its own data channel to memory for operations concurrent with computation and operation of other peripherals in the GE-235 system.

DATANET-61 Data Exchange Controller--Permits the GE-235 computer to exchange data with non-standard peripherals. With its two parallel buffer units, the DATANET-61 can receive data in one buffer, and send data by the other. Output data can be used to control a process by having the signals operate switches, valves, or change a dial setting. Transfers up to 54,000 characters of data per second combined in both channels.

DATANET-62 Data Communications Channel--A high-speed buffered channel to the GE-235 system which gives up to 32 lines access to the core memory on a sequentially-scanned basis. The DATANET-62 can be located remotely from the computer, and have access to the GE-235 memory through its own data channel. Transfers up to 54,000 characters of data per second through one or 32 lines.

DATANET-63 Data Exchange Controller--Allows many non-standard peripherals to send data to the GE-235 system. High-speed data sources can go directly on line and data reduction and data processing can be done concurrently while data is being accumulated. The DATANET-63 has 8 channels which may be split in any combination of input/output totaling eight. Each channel is capable of handling multiple lines or signals in parallel. Transfer rate is up to 165,000 characters per second combined transfer rate in one or all channels.

DATANET-90 Data Communications Terminal--An on-line tape-to-computer terminal that permits a centralized computer to read tape from a remote magnetic tape handler via a broadband communication channel. It handles standard magnetic tape formats and transfers up to 90,000 characters of data per second.

DATANET-91 Data Communications Terminal--An off-line, high-speed magnetic tape-to-magnetic tape terminal that permits high-speed data transmission without using valuable computer time. Separated computer sites can exchange data to equalize the work loads between computers. Permits exchange of data at speeds up to 100,000 characters per second.

DATANET-600 Data Communication Perforated Tape Terminal--A bi-directional perforated tape terminal unit for locations remote from a GE-235 system or a DATANET-30. It transmits and receives data on perforated tape over telephone circuits at speeds up to 50 characters per second. The DATANET-600 provides full error detection and tape editing features to assure error-free data transmission.

DATANET-601 Data Communication Perforated Tape Terminal--A modified version of the DATANET-600 for use as an interface between the remote stations and a DATANET-15 controller at a GE-235 system.

DATANET-3100 Shoptrol System--A factory monitoring and data collection system which provides complete and rapid production control information for increased operating efficiency. The Shoptrol System outputs can be converted to data processing inputs to provide business managers with current data for effective operating decisions.

DATANET-3101 Data Accumulation System--Provides a fast and accurate means of accumulating and recording on perforated tape all pertinent information in manufacturing and distribution operations. Data can then be used as input to a GE-235 system, resulting in reports which supply management with timely information about all phases of operating a business.

These DATANET equipments are typical of General Electric's growing and versatile line of compatible accumulating and transmission systems.

III. FUNCTIONS OF THE BASIC GE-235 SYSTEM COMPONENTS

The basic GE-235 system comprises the central processor, the operator's console, and the input/output typewriter. Two equipment rack cabinets bolted together house the central processor and certain other system circuitry. (A third cabinet is included when an auxiliary arithmetic unit is part of the system.) The operator's console is mounted on the side of the first cabinet. The input/output typewriter rests on the right of the operator's table, which is conveniently placed in front of the console.

CENTRAL PROCESSOR

The central processor of a GE-235 system is made up of the primary storage unit; the arithmetic unit; registers, or temporary storage devices; and the circuitry that binds all these into a functioning whole. The primary storage unit, or memory, holds the data to be processed and the instructions for all processing operations. The arithmetic unit and its associated registers perform the logical and mathematical operations. And the control registers and their associated circuitry control all processor and peripheral operations. Figure III-1 shows schematically the major functional parts of the central processor.

Storage

MEMORY ORGANIZATION

The memory module in a GE-235 central processor is an array of magnetic cores. An instruction or data word is stored in a group of these cores known as a "location." The cores in the location correspond to the bits in the word, and the polarity of each core is made magnetically either + or - to represent the 0 or 1 value of its matching bit.

When an instruction or data word is transferred from memory, the process destroys the stored information. But circuitry is provided to rewrite the word immediately in its location, since it may be needed again many times during the course of the program. A word can be erased from memory only by replacing it with another word.

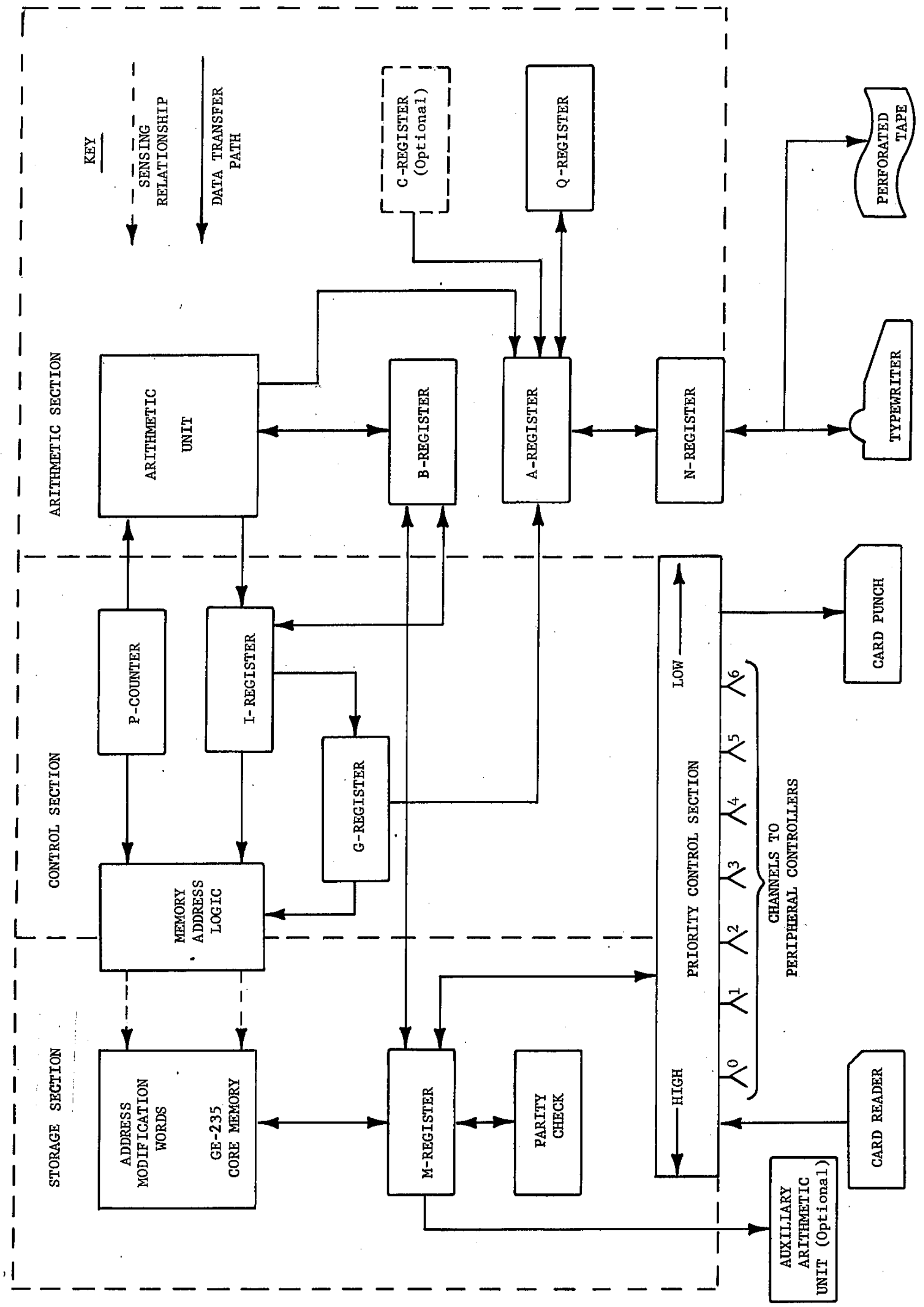


Figure III-1. Functional Block Diagram Showing Major Relationships Within the GE-235 Central Processor

Each memory location has a specific address. Thus, through properly coded instructions, words may be stored in and retrieved from a known location. Addresses are numbered from 0000 through 4095 or 8191 for the two basic memory sizes. An additional 8192 words are available as an option. The special techniques for addressing this "upper bank" of memory are discussed in Chapter VIII.

MEMORY ALLOCATION

Efficient programming requires orderly use of memory. Assigning certain areas for specific uses will reduce programming time and help eliminate errors. A typical memory allocation scheme is shown in Figure X-8. It should be emphasized that except for certain locations reserved for special functions (to be discussed later), allocations are arbitrary; any arrangement that produces the desired results at a particular installation is satisfactory.

The designation of locations 0000 through 0003 as an address modification word group is required because these locations have unique properties and special instructions are provided for their use. Locations 0000 through 0003 may be used as counter words, and locations 0001 through 0003 may also be used as modification words for automatic address modification. Their special functions are described in Chapter VIII. Thirty-one additional 4-word modification groups in memory locations 0004 through 0127 are available as part of a package option.

M-REGISTER AND PARITY CHECKING

Though the M-register is not part of memory, it is closely associated with it. All information transferred to or from memory must pass through the M-register; therefore, it is an integral part of the storage function. Its action is automatic, and it is not under the control of either the program or the operator's console.

The M-register has 21 bit positions -- 20 for the bits of the instruction or data word being transferred and 1 for a parity bit. When a word is written into memory, the parity check circuits count the 1-bits in the word while it is held in the M-register. If the count is even, a 1-bit is placed in the 21st bit position; if the count is odd, no parity bit is needed. Then the entire 21 bits are stored in the designated memory location.

A parity check is performed automatically as a word is read out 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 pickup) has occurred; and the MEM PARITY alarm indicator on the operator's console will light. In addition, depending upon the position of the STOP ON PARITY ALARM switch on the console, the processor may halt or a programmed branch for remedial action may occur.

Arithmetic

Arithmetic operations are performed by the arithmetic unit and its associated registers, which hold the initial quantities and intermediate and final results. The arithmetic registers are also used for shifting and other data manipulations related to decision-making and calculating.

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.

B-REGISTER

The B-register is a 20-bit register that acts as a buffer between the M-register and the arithmetic and control circuits. Outputs are supplied from it to the I-register and the arithmetic unit. Its operation is automatic, and it is not accessible through program or console instructions. During arithmetic operations it contains:

1. The addend for addition
2. The subtrahend for subtraction
3. The multiplicand for multiplication
4. The divisor for division.

A-REGISTER

The 20-bit A-register is the most frequently used register 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:

1. The augend during and the sum after addition

2. The minuend during and the result after subtraction

3. The most significant half of the product after multiplication

4. The most significant half of the dividend before and the quotient after division

5. The most significant half of a word during the execution of all double-length word instructions

6. A word transferred from, or to be transferred to, memory

7. The word on which extraction is performed during the execution of the EXT instruction (see Chapter VIII for an explanation of this instruction)

8. The word to be shifted (or its most significant half) during various shift instructions

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

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

Manual access to the A-register is provided by 20 switches on the operator's console.

Q-REGISTER

The Q-register is a 20-bit register that acts with the A-register to form a double-length word accumulator (38 bits plus two sign bits) 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 transferred into the Q-register. This register holds:

1. The least significant half of the augend before and the least significant half of the sum after double-length addition

2. The least significant half of the minuend before and the least significant half of the result after double-length subtraction

3. The multiplier before and the least significant half of the result after multiplication

   The least significant half of the dividend before and the remainder after division

4. The least significant half of the word during the execution of double-length word instructions

5. The least significant half of information to be shifted during double-length shift instructions.

N-REGISTER

The N-register is a 6-bit register used as a single-character buffer between the central processor and (1) the console typewriter, (2) the perforated tape reader, and (3) the perforated tape punch. This action permits input/output operations with these units to occur simultaneously with other processor operations. Information is transferred serially between the N-register and A-register by shift instructions.

C-REGISTER (optional)

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 either by the system or by external equipment. 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 system. 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 off line or by a simple conversion routine. (Instructions and convcrsion procedures are discussed in Chapter VIII.)

Control

The control portion of the central processor comprises the priority control logic (see discussion in Chapter IV), the internal sequence logic, the memory address logic, and the associated registers. Together they control all internal and input/output operations. The functions of the control registers are discussed individually below.

I-REGISTER

The I-register, or instruction register, contains all 20 bits of the word during the execution of an instruction. Initially, however, instructions are read from memory and set into the B-register. Bits 0-6 (operation code and address modification--see Chapter V for word formats) are transferred to the I-register for decoding. At the same time, bits 7-19 (operand address, if the word is a memory-reference instruction) are routed to the arithmetic unit. If bits 5 and 6 indicate automatic address modification, the contents of bits 5-19 of the indicated modification word are added to the quantity in the arithmetic unit; and the sum is transferred to bit positions 5-19 in the I-register. If no address modification is indicated or if address modification does not apply to the instruction format used, the unmodified quantity is set into the I-register.

G-REGISTER

The G-register is a 5-bit register that contains in binary the number of the optional address modification word group currently being used. Its contents are displayed on the operator's console as INDEX GROUP. (See “Using Address Modification Words” in Chapter VIII.)

P-COUNTER

The P-counter is a 15-bit location counter (or register) that contains the memory address of the next instruction to be executed. The contents of the P-counter are incremented by 1 before the execution of an instruction, so that the P-counter indicates the address of the next instruction in sequence. An SPB (store P and branch) or BRU (branch unconditionally) instruction, however, suspends this action. As a result of these instructions, the contents of the P-counter is changed to the address specified in the instruction, and normal incrementing by 1 is not performed.

The contents of the P-counter (that is, the address of the instruction currently in the I-register) are displayed on the operator's console.

Functional Integration: The Operating Cycle

In a GE-235 system not only the data to be processed but the instructions for processing are stored in memory. In such a stored program system processing, once begun, continues automatically by repetition of the operating cycle until the program has been completely executed, as long as power is applied to the system.

Coordination of the storage, arithmetic, and control functions within the operating cycle is maintained by a crystal-controlled timer. During eight pulses from this device a word can be:

1. Extracted from memory
2. Transferred to the proper register
3. Restored in memory.

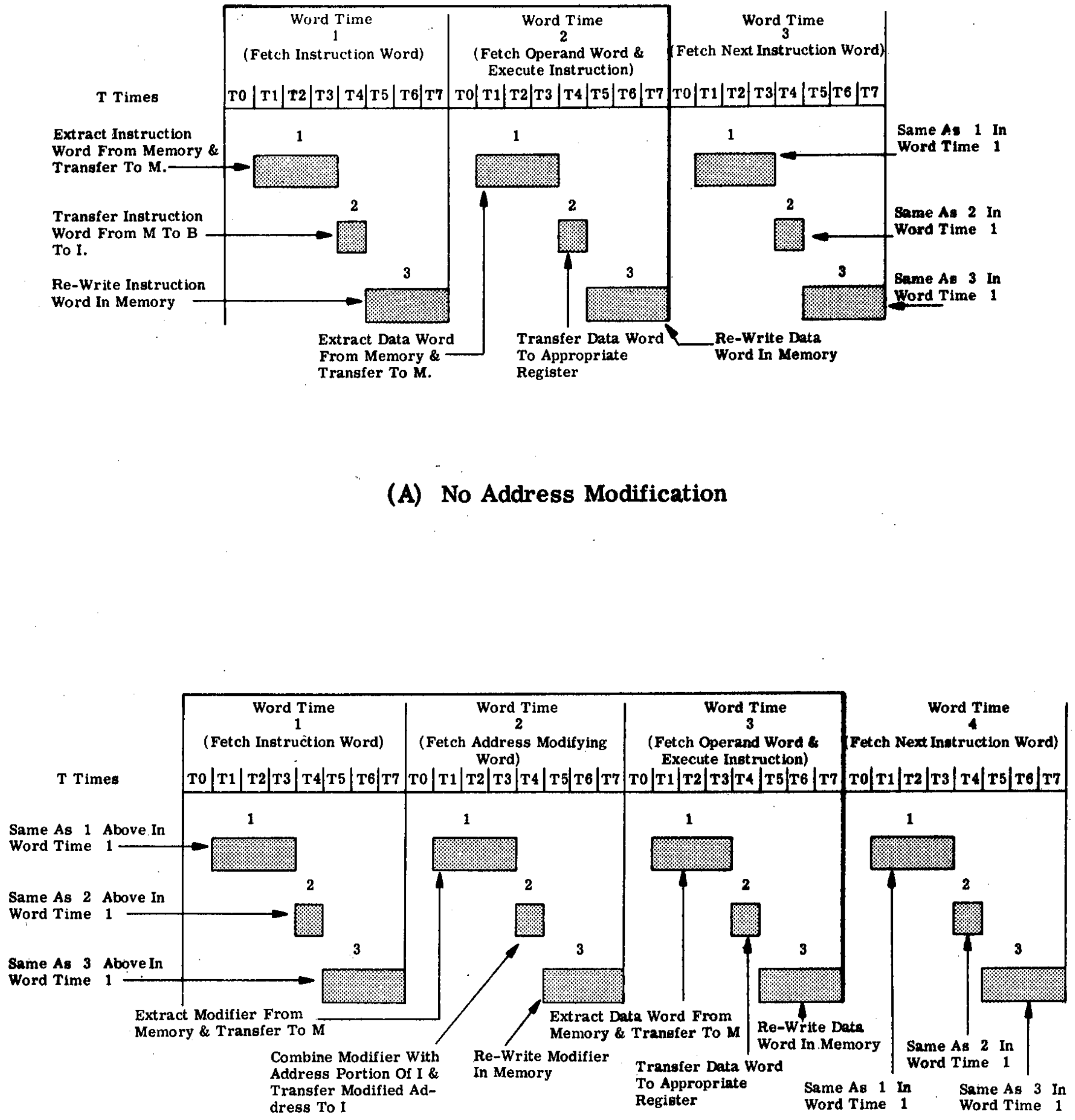
This basic eight-pulse interval is known as a "word time" (also called "memory cycle"). Each of the three operations is performed on the word as a unit; all bits are operated on at the same time, or in parallel.

An operating cycle of the central processor is made up of 1 or more word times, depending on the instruction being executed. (Most instructions require at least 2 word times for execution. Only a few, like BRU, require just 1 word time.) The number of word times for execution of each instruction is included in its individual description (see Chapter VIII, for example). A word time, however, is only a means for expressing the relative duration of various operations within the same system. The actual time represented by a word time in the GE-235 is 6 microseconds.

Instructions are normally executed in sequence under the control of the P-counter. However, decision results, priority control, or automatic program interrupt may break the sequence. In addition, the data called for by succeeding instructions may not be stored in adjacent locations. But these variations have no effect on the execution time for processor operations, since the core memory is a random access memory. The word time for all locations is the same, and locations may be addressed in any order.

OPERATING CYCLE PHASES

In executing an instruction that requires memory access, one word time is required to fetch the instruction and another to fetch the operand and perform the operation specified (see Figure III-2A, word times 1 and 2). The internal sequence logic controls the performance of the operating cycle; and, upon completion of one cycle, immediately initiates the next (see Figure III-2A, word time 3).


T Times
Word Time 1 (Fetch Instruction Word)
Word Time 2 (Fetch Operand Word & Execute Instruction)
Word Time 3 (Fetch Next Instruction Word)
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
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
T Times
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)
Same As 1 Above In Word Time 1
Same As 2 Above In Word Time 1
Same As 3 Above In 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
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

(A) No Address Modification

(B) With Address Modification

Figure III-2. Basic Operating Cycle

The basic operating cycle, then, requires two word times, which represent the two basic phases of the cycle: the instruction phase and the execution phase, respectively. For certain instructions either of these phases, or both, may take more than one word time. The extra word times are automatically provided by the internal sequence logic when necessary.

Instruction Phase

The instruction phase serves to:

1. Locate the instruction in memory and transfer it to the I-register
2. Modify the operand address, if necessary
3. Locate the data in memory as specified by the instruction operand address
4. Establish control circuits for execution of the instruction.

Figure III-2A, word time 1, shows in detail the operations performed on the instruction word during this phase in terms of the eight pulse intervals (T times) making up one word time.

Instructions with address modification require two word times for the instruction phase of the operating cycle allotted to them (see Figure III-2B, word times 1 and 2).

A flow chart of the instruction phase is shown in Figure III-3A. First, an instruction is read from memory. The P-counter is then incremented (usually by 1), so that it contains the address of the next instruction. Finally, any address modification necessary takes place (see the discussion under "I-Register" in this chapter for details).

Execution Phase

During the execution phase (see Figure III-2A, word time 2, and B, word time 3), 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. The execution phase for some instructions--like those for multiply, divide, shift, and double-length words--requires more than one word time. As previously mentioned, the additional word times are automatically made available by the internal sequence logic.

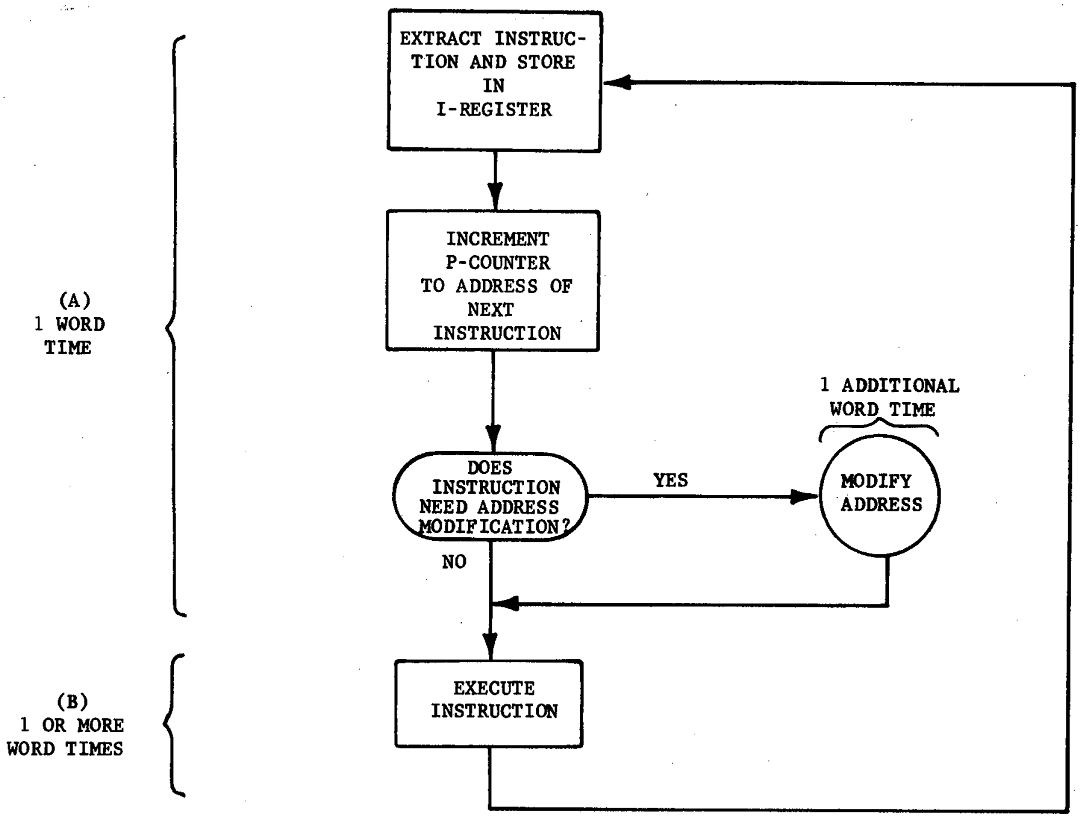

(A)
1 WORD
TIME
EXTRACT INSTRUCTION AND STORE IN I-REGISTER
INCREMENT P-COUNTER TO ADDRESS OF NEXT INSTRUCTION
1 ADDITIONAL WORD TIME
DOES INSTRUCTION NEED ADDRESS MODIFICATION?
YES
MODIFY ADDRESS
NO
(B)
1 OR MORE
WORD TIMES
EXECUTE INSTRUCTION

Figure III-3. Operating Cycle: (A) Instruction Phase and (B) Execution Phase

DETAILED ANALYSIS OF THE OPERATING CYCLE

Three different kinds of memory access are required to execute instructions. One requires access to memory under control of the P-counter, another involves control by an index word, and the third is controlled by the I-register. The type of access permitted during any word time is governed by the following internal sequence logic:

1. AMP - logic used to address memory from the P-counter
2. AMX - logic used to address memory from one of the address modification words
3. AMI - logic used to address memory from the I-register.

Figure III-4 is a flow chart depicting the operations performed by the central processor while executing an instruction. This diagram illustrates the nature of the operations and tests performed during one complete operating cycle:

1. Extraction of the instruction from memory (AMP)
2. Modification of the address portion of the instruction, if required (AMX)
3. The subsequent execution of the operation (AMI, GIS, or AMX).

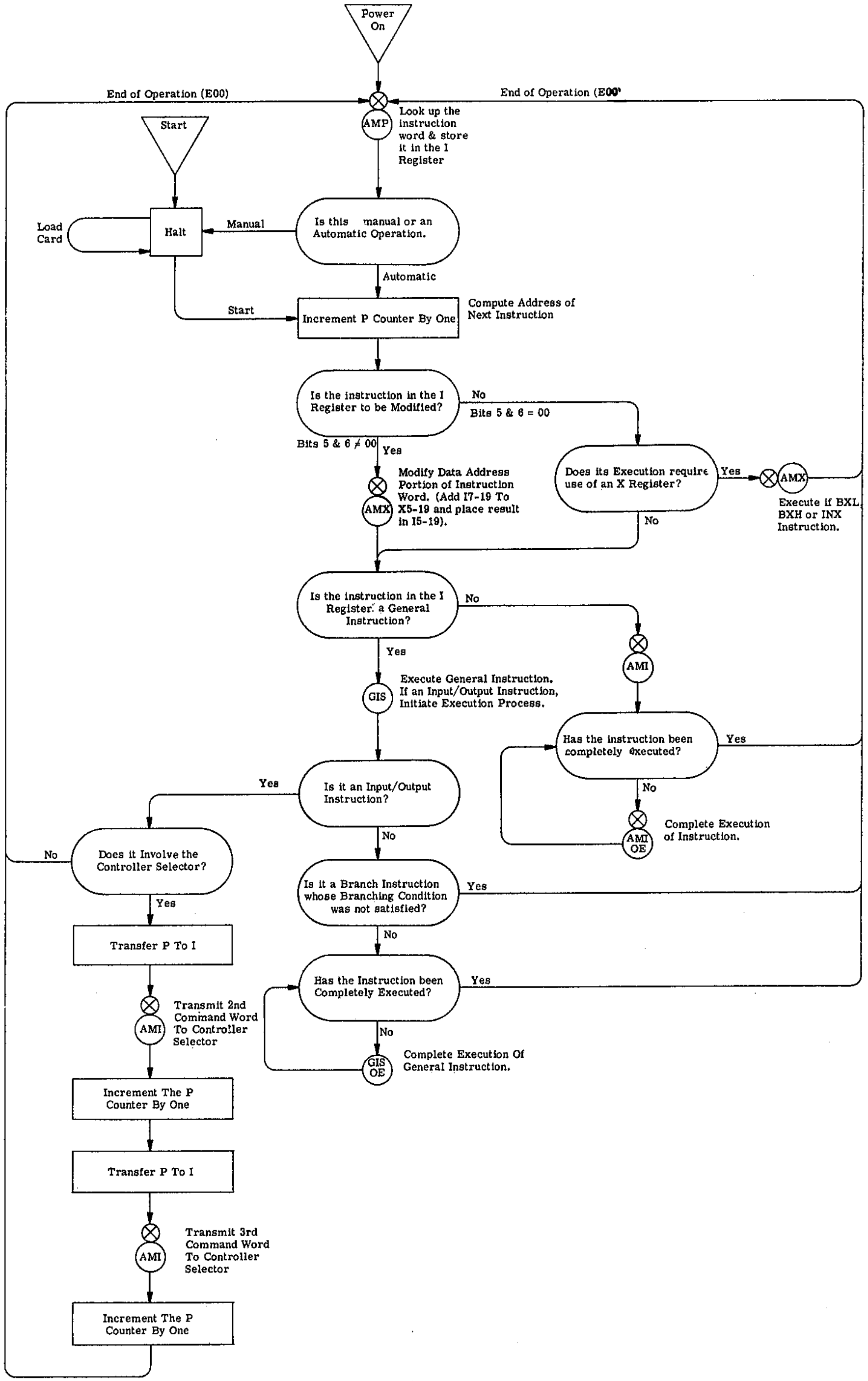

Power On
End of Operation (E00)
End of Operation (E00)
AMP
Look up the instruction word & store it in the I Register
Start
Load Card
Halt
Manual
Is this manual or an Automatic Operation.
Automatic
Start
Increment P Counter By One
Compute Address of Next Instruction
Is the instruction in the I Register 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
Yes
AMI
Execute General Instruction. If an Input/Output Instruction, Initiate Execution Process.
GIS
Has the instruction been completely executed?
Yes
No
AMI OE
Complete Execution of Instruction.
Yes
Is it an Input/Output Instruction?
No
Does it Involve the Controller Selector?
Yes
Is it a Branch Instruction whose Branching Condition was not satisfied?
Yes
No
Transfer P To I
Has the Instruction been Completely Executed?
Yes
No
AMI
Transmit 2nd Command Word To Controller Selector
GIS OE
Complete Execution Of General Instruction.
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 III-4. Flow Chart Showing Detailed Analysis of Operating Cycle

GIS (general instruction sequencing) is internal sequence logic that controls the execution sequence during all general instructions.

The 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 III-4 require some explanation. Each circle containing alphabetic characters represents an operation requiring one word time. The abbreviations correspond to the controlling circuits in the internal sequence logic. Each smaller circle containing an X indicates that the operation involves memory access during the associated word time. Operation extender (OE) directs the sequence control logic to allow additional word times for execution when necessary.

For a manual start, the first instruction is assumed to be already 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 address modification word, the next operation is an AMX access cycle. Otherwise, the next cycle is either a basic AMI cycle or a general GIS cycle. Format I instructions (see Chapter V) 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 synchronized AMI cycles (for input/output operations involving the controller selector).

Completion of the operating cycle 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 AUTO/MAN switch and the STOP ON PARITY ALARM switch.

If the AUTO/MAN switch is in the MAN position, the processor halts. Otherwise, processing of the next instruction is initiated, unless the STOP ON PARITY ALARM switch is in the STOP position and a memory parity error has occurred during one or more of the word times of the previous operating 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.

OPERATOR'S CONSOLE

The switches on the operator's console provide a means for manual control of processing. This includes control of such operations as the initial reading-in of a program, starting program execution, and interrupting the program for checking and other purposes. The console indicators display the status of important system components and the contents of key central processor registers. The functions of these console switches and indicators are discussed in detail in the following paragraphs. See Figure III-5 for all references to console controls and indicators. (An additional row of indicators appears above the standard console on systems equipped with the optional AAU. These are illustrated and discussed in the manual for the AAU.)

Alarm Indicators

The nine alarm indicators at the top left of the console signal the occurrence of certain error conditions during system operation.

PRIORITY: The yellow priority alarm indicator will light under any of the following conditions:

1. The AUTO/MAN switch is in the MAN position
2. The central processor registers have been denied access to memory
3. The STOP ON PARITY ALARM/NORM switch is in the STOP ON PARITY ALARM position and a memory parity error is detected
4. Any echo alarm condition has occurred.

PARITY: The two red parity alarm indicators signal detection of parity error. When the AUTO/MAN switch is in the MAN position, the parity alarm indicators may be turned off and the alarm circuits reset by depressing the RESET ALARM switch. The same result may be obtained when the AUTO/MAN switch is in the AUTO position by remedial action included in the program.

MEM: The M-register parity alarm indicator will light if the memory readout circuits detect a parity error. If the STOP ON PARITY ALARM/NORM switch is in the STOP ON PARITY ALARM position, detection of an M-register parity error will halt the central processor.

N REG-I/O: The appropriate section of the N-register and priority control parity alarm indicator will light if the checking circuits associated with the perforated tape reader or a priority control peripheral detect a parity error.

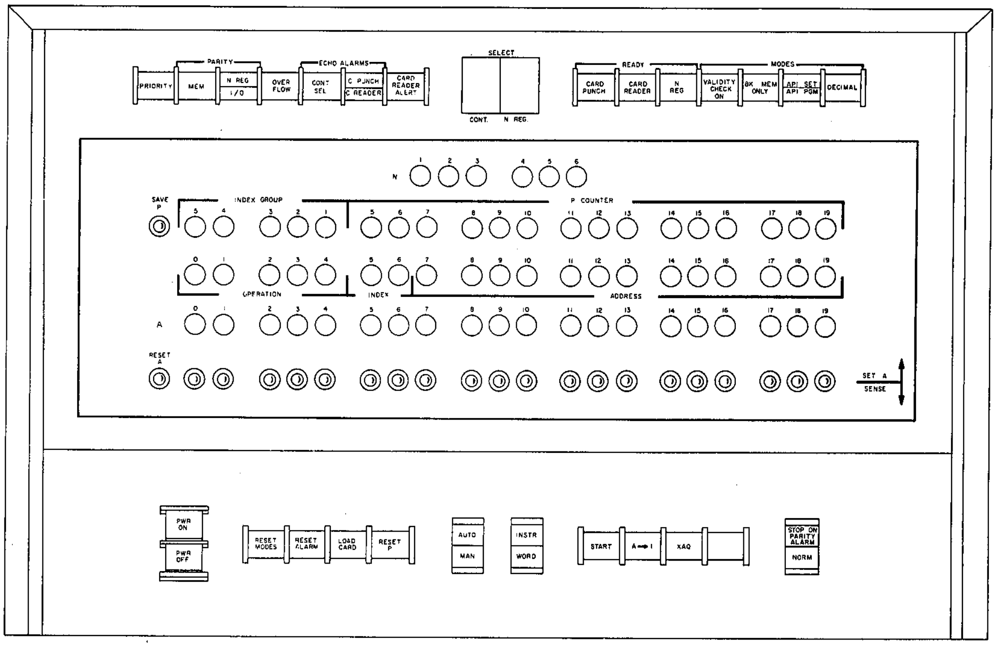


Figure III-5. GE-235 Operator's Console

OVERFLOW: The red overflow alarm indicator will light under either of the following conditions:

1. The numerical capacity of the A-register has been exceeded
2. A 1-bit has been shifted out of the A-register by a shift-left operation.

Detection of an overflow condition does not halt the central processor.

ECHO ALARMS: The two red echo alarm indicators signal the inability of a controller to respond when addressed.

CONT SEL: The controller selector (priority control) echo alarm indicator will light when one of the controllers on a priority control channel is addressed for an input/output operation under any of the following conditions:

1. The controller is in off-line status
2. The controller is malfunctioning
3. The controller power is off

4. Tne controller is in not-ready status
5. The channel addressed is not occupied
6. The controller is given an illegal instruction.

C PUNCH/C READER: The appropriate section of the card punch/card reader echo alarm indicator will light when either of the card equipment controllers is addressed under either of the following conditions:

1. The controller is malfunctioning
2. The controller is in not-ready status.

An echo alarm condition will halt the central processor and turn on the PRIORITY indicator.

CARD READER ALERT: The red card reader alert alarm indicator will light if any of the following conditions occur during execution of a read card instruction:

1. The reader input hopper is empty
2. The reader power is off
3. A card is not positioned on the reader's sensing platform
4. A card jam occurs
5. A feed error occurs during execution of a read instruction (400 cpm reader)
6. A phantom feed is detected (1000 cpm reader).

The central processor will not halt under a card reader alert alarm condition.

Ready Indicators

The three green indicators in the top row just to the right of center are ready indicators. They will light when the system components that they monitor are ready for input/output operations.

CARD PUNCH: The card punch ready indicator will be lighted when the punch is in ready status. It will not be lighted if any of the following conditions occur:

1. A punch cycle is in process
2. The stacker is full
3. The input hopper is empty
4. A card is misfed
5. A card jam is detected
6. The chad box is improperly seated (100 cpm punch only).

If a punch instruction is given when the punch is in not-ready status, the C PUNCH and PRIORITY alarm indicators will light and the central processor will halt.

CARD READER: The card reader ready indicator will be lighted when the reader is in ready status. It will not be lighted if any of the following conditions occur:

1. A read cycle is in process
2. The input hopper is empty
3. A card is misfed
4. A card jam is detected.

If a read instruction is given when the reader is in not-ready status, the C READER echo alarm and PRIORITY alarm indicators will light and the central processor will halt.

N REG: The N-register ready indicator will be lighted when the register is in ready status. It will not be lighted if the N-register is currently being used for data transfer by the perforated tape equipment or the input/output typewriter. If an attempt is made to shift data from the A-register to the N-register when the latter is in not-ready status, the data will be shifted out of the A-register and lost. There will be no alarm indication on the console. If an attempt is made to shift data from the N-register to the A-register when the former is in not-ready status, the A-register will shift but the N-register will not. There will be no alarm indication on the console.

Mode Indicators

The four indicators in the top row to the right of the ready indicators are mode indicators. They display the status of certain optional system components.

VALIDITY CHECK ON: The white BCD validity check mode indicating switch may be depressed to enable or disable the BCD validity check mode for the card readers. When the mode is enabled the indicator will light.

8K MEM ONLY: The white 8k memory only mode indicating switch may be depressed to limit a 16k memory to operation with only the lower bank of memory. When operating in this mode the indicator is lighted. The mode may be disabled by depressing the lighted switch.

API SET/API PGM: The automatic program interrupt mode indicator shows the status of the optional API function. If the interrupt mode has been set by execution of a PST instruction, the yellow API SET upper section of the indicator will light. During the intervals when the main program has relinquished system control to a priority program, the white API PGM lower section of the indicator will light.

DECIMAL: The white decimal mode indicator shows the status of the optional decimal operating mode for the central processor. When the mode is enabled the indicator will light.

Select Display Indicators

The two display indicators in the center of the top row are the select display indicators. They show which peripherals are currently logically connected to the central processor.

CONT: The controller alphanumeric display indicator will show which peripheral controller the central processor currently has access to for testing or the issuing of an instruction. The display code is as follows:

1. CR indicates that the card reader controller is either being tested for ready status by a BCN or BCR instruction or being given an RCB, RCF, RCD, RCM or HCR instruction.
2. CP indicates that the card punch controller is either being tested for ready status by a BPN or BPR instruction or being given a WCB, WCD, or WCF instruction.
3. A numeral from 0 through 6 (denoting priority control channel) indicates which peripheral controller is either being tested for ready status by a BCS instruction or being given as SEL instruction.
4. AAU indicates that the AAU is being tested.
5. N indicates that the N-register is being tested for ready status.

N REG: The N-register alphabetic display indicator will show in the following manner which peripheral is currently logically connected to the N-register for input or output:

1. KON indicates that the typewriter is logically connected for input to the N-register
2. TON indicates that the typewriter is logically connected for output from the N-register
3. RON indicates that the perforated tape reader is logically connected for input to the N-register
4. PON indicates that the perforated tape punch is logically connected for output from the N-register.

Register Display Indicators and Associated Switches

The four rows of circular display indicators in the center of the console are register display indicators. Each indicator represents one binary digit: when lighted, a 1-bit; when not lighted, a 0-bit. The indicators are arranged in groups of three for ease in reading the octal equivalents of the binary numbers displayed.

N: The six N-register display indicators are centered just below the SELECT indicators. They show the current contents of the N-register.

INDEX GROUP: The five address modification word group display indicators are at the left of the second line of register display indicators and show the contents of the G-register. This register contains in binary form the number of the address modification word group (0 through 31) currently selected by the program being run. Groups 1 through 31 are an optional system component; therefore, in a system without this option these display indicators always remain unlighted to indicate selection of group 0.

P COUNTER: The P-counter display indicators are to the right of the INDEX GROUP display indicators. They are numbered to correspond to the 15 bits of a GE-235 word that they represent. These indicators display the address of the instruction currently in the I-register.

SAVE P: The save P-counter toggle switch is at the left of the INDEX GROUP display indicators. This single-throw switch is normally in the center (off) position; when it is placed in the down (on) position it prevents the P-counter from incrementing. (Its use permits return to the program at the step where it has been halted to make a correction manually.)

I: The I-register display indicators occupy the third row of register display indicators. They are numbered and labeled to correspond to the bit positions and divisions of a GE-235 instruction word and display the current contents of the I-register---that is, the next instruction to be executed, which remains displayed until it has been fully executed.

A: The A-register display indicators are just below the I-indicators and are numbered in the same manner. They display the contents of the A-register (see also the discussion of the XAQ switch).

A-Register Input Switches: The 20 A-register input switches are three-position toggle switches located just below and in vertical alignment with their corresponding display indicators. These switches are normally in the center (off) position. They are spring-loaded when moved to the up (SET A) position and will return automatically to the center position. The up position of the switches is effective only during manual system operation. When the system is being operated manually, moving an A-register input switch to the up position will enter a 1-bit in the corresponding position in the A-register and light the appropriate indicator.

The down (SENSE) position of the switches is used during automatic system operation. Moving an A-register input switch to the down position will cause a 1-bit to be entered in the corresponding position of the A-register when a subsequent programmed RCS (Read Control Switches) instruction is executed.

RESET A: The reset-A pushbutton switch is to the left of the A-register input switches. It is effective only during manual system operation. Depressing this switch clears the contents of the A-register to zero and turns off all A-register display indicators.

Control Switches

The switches grouped below the A-register input switches give the operator manual control over certain central processor and system functions.

PWR ON: Depressing the yellow power-on indicating switch turns on all central processor ac and dc power. At the same time it automatically performs the following:

1. Resets all alarms
2. Sets the P-counter to zero
3. Selects address modification word group 0
4. Sets the arithmetic mode to binary
5. Disables API
6. Sets the N-register to zero and places it in ready status
7. Logically connects the typewriter for output from the N-register (TON condition)
8. Resets the BCD remembered carry function.

POWR OFF: Depressing the white power-off pushbutton switch turns off all power to the central processor.

RESET MODES: The white reset modes pushbutton switch is effective only when the AUTO/MAN switch is in the MAN position. Depressing the switch causes the following:

1. Address modification word group 0 is selected
2. The arithmetic mode is set to binary
3. API is disabled
4. The N-register is set to zero and placed in ready status
5. The typewriter is logically connected for output from the N-register (TON condition)
6. The BCD remembered carry function is reset.

RESET ALARM: The white reset alarm pushbutton switch is effective only when the AUTO/MAN switch is in the MAN position. Depressing the switch resets all existing alarm indications. That is, it turns off all alarm indicators and resets the alarm circuitry so that the central processor may continue operation. It does not eliminate the conditions that have caused the alarm indications. (The operator should remember that using the RESET ALARM switch when not authorized to do so by the programmer can damage the program.)

LOAD CARD: The white load card pushbutton switch is effective only when the AUTO/MAN switch is in the MAN position. Depressing the switch causes the card reader to go through one load-and-read cycle. The contents of the card are read into memory in 10-row binary mode, beginning at location 0000. (The LOAD CARD switch is most often used by the operator to load the first card into memory during program startup operations.)

RESET P: The white reset-P pushbutton switch is effective only when the AUTO/MAN switch is in the MAN position. Depressing the switch sets the P-counter to zero. (The RESET P switch is most often used to cause the first instruction in a program to access memory location 0000. It is normally depressed just before placing the AUTO/MAN switch in the AUTO position at the beginning of program operation.)

AUTO/MAN: The white automatic/manual rocker switch controls the mode of system operation. If the AUTO portion of the switch is depressed and then the START switch is depressed, continuous processing under program control is initiated. If the MAN portion of the switch is depressed while the system is operating in this automatic mode, processing halts after execution of the instruction currently in the I-register (unless the INSTR/WORD switch is in the WORD position, when it will halt at the end of the current word time).

If the MAN portion of the AUTO/MAN switch is depressed, then each depression of the START switch will cause execution of one program instruction. When the system is in the manual mode, the PRIORITY indicator will remain lighted continuously. The following console controls function only in the manual mode:

1. RESET A
2. A-register input switches (up position)
3. RESET MODES
4. RESET ALARM
5. LOAD CARD
6. RESET P
7. INSTR/WORD
8. A→I
9. XAQ

INSTR/WORD: The white instruction/word rocker switch is effective only when the AUTO/MAN switch is in the MAN position. When the INSTR portion of the switch is depressed, one program instruction will be executed each time the START switch is depressed. When the WORD portion of the switch is depressed, one word-time portion of the current central processor operation will occur each time the START switch is depressed. The WORD position of the switch is intended for maintenance use only.

START: The function of the yellow start pushbutton switch is fully described above in the discussion of the AUTO/MAN and INSTR/WORD switches.

A→I: The white A-to-I pushbutton switch is effective only when the AUTO/MAN switch is in the MAN position. Depressing this switch transfers the contents of the A-register to the I-register. The contents of the A-register remain unchanged. (The A-to-I switch may be used to load an instruction manually into the I-register or to correct an instruction already there.)

XAQ: The white exchange-A-and-Q pushbutton switch is effective only when the AUTO/MAN switch is in the MAN position. Depressing this switch causes an exchange of the contents of the A and Q registers. (Having depressed the XAQ switch, the operator may then clear and correct the contents of the Q-register by using the RESET A switch and the A-register input switches, while saving the contents of the A-register.)

BLANK SWITCH: The blank pushbutton switch to the right of XAQ does not function on present standard GE-235 systems.

STOP ON PARITY ALARM/NORM: The white stop-on-parity-alarm/normal rocker switch determines the response of the central processor to the detection of a parity error. When the STOP ON PARITY ALARM portion of the switch is depressed, processing halts each time an M-register parity error (error in readout from memory) is detected; and the PRIORITY indicator and MEM parity alarm indicator will light. When the NORM portion of the switch is depressed, detection of a parity error will cause the MEM parity alarm indicator to light; but processing will continue without interruption. (The setting of the STOP ON PARITY ALARM/NORM switch will be specified in the programmer's instructions. Which setting is used will depend on whether or not the program includes remedial action for parity errors.)

CONSOLE TYPEWRITER

Relation to System

The GE-235 console electric typewriter (see Figure III-6) is mounted on the operator's console table and is connected directly to the central processor through the N-register. All external circuitry for the typewriter is located in the central processor.

The console typewriter may be used for either output or input. As an output device it prints one character at a time, which it receives from the N-register under program control. Though it may be used for more extensive output in the absence of a high-speed printer, it is generally used as a means for the central processor to communicate with the operator by printing short messages that serve as a log of program performance. If a program has been prepared with this function in mind, the typewriter can do such things as record the identity of the program and list tape labels and operator instructions. The operator may also make additions to the log by manually operating the machine as a standard typewriter.

As an input device the typewriter may be used in debugging operations or as a means for making manual entries when requested by a running program. In this latter application a single character at a time is entered into the N-register by keyboard operation. Then under program control each character is shifted to the A-register and subsequently processed.

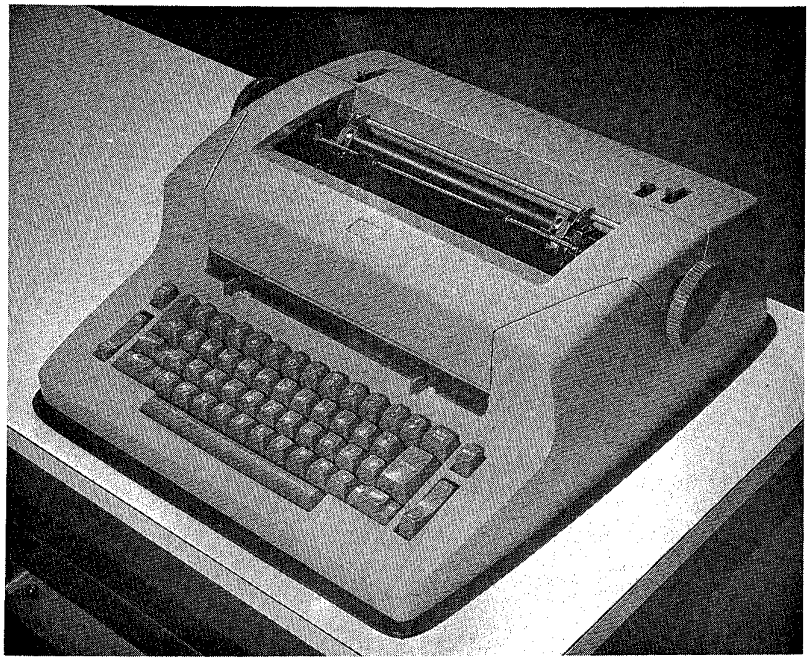


Figure III-6. GE-235 Console Typewriter

Characteristics

The console typewriter prints (1) the letters of the alphabet (upper case only), (2) arabic numerals, and (3) 21 symbols and punctuation marks under program control at the maximum rate of 15 characters per second. It also performs six common typewriter functions in conjunction with printing: indexing (paper advance), carrier return, spacing, print red, print black, and tabulation; an IGNORE (no operation) function may also be programmed.

The typewriter keyboard has 44 data keys, a space bar, and nine functional control keys (see Figure III-7). Twenty of the data keys have upshift significance, and a shift key must be depressed for the typewriter to transmit the proper code for these characters as input to the N-register. The upshift is automatically accomplished when these characters are programmed for output from the N-register to the typewriter.

Depressing any of the function keys causes their associated action to occur at the typewriter but does not cause a code to be sent to the N-register. Depression of the space bar causes the associated action at the typewriter and also transmits the corresponding code to the N-register.

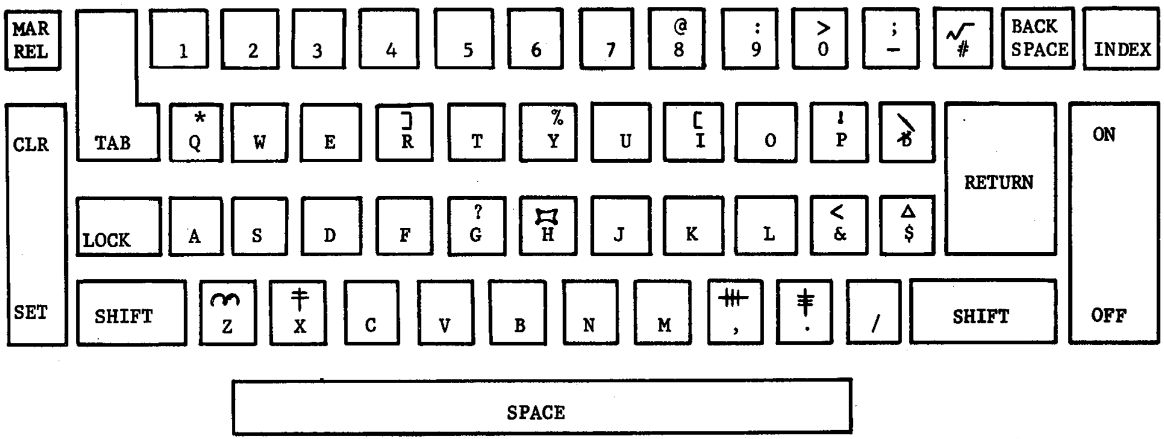


Figure III-7. Console Typewriter Keyboard Layout

A few of the typewriter's characteristics deserve special mention. Chief among these is the special method of printing. All type characters are embossed on the surface of a single, removable, globe-shaped typing element that is fixed to a mechanism called the carrier. The carrier performs the function of the carriage in a conventional typewriter by moving from left to right past the stationary platen and returning to the left margin upon depression of the RETURN key. As each character key is struck, the typing element rotates about its vertical axis to the proper position and then tilts toward the platen to make the character. Another special feature that should be noted is the interchangeable ribbon cartridge, which is attached to the carrier along with the typing element. For paper handling, standard sprocket feed is provided for use with multi-fanfold forms, 8.5 inches wide. A minimum of two legible carbon copies may be produced.

IV. SYSTEM CONTROL OPERATIONS

Three system control features make possible the outstanding flexibility and speed of a GE-235 system. Foremost is the priority control logic; in conjunction with the peripheral controllers, it allows concurrent operation of the central processor and the input/output peripherals. Second is the optional automatic program interrupt (API), which allows concurrent processing of multiple programs. Third is the peripheral switch control unit. This device allows use of the same peripherals by two otherwise independent compatible General Electric Information Processing Systems.

CONCURRENT OPERATION OF SYSTEM COMPONENTS

In a GE-235 system up to 10 input/output devices of various types may operate concurrently with the central processor. Because the operating cycles of the concurrently operating peripherals and the central processor are of different lengths, the cycles overlap to a great extent and occur simultaneously. For example, a magnetic tape handler may be reading tape, the central processor may be executing an instruction, and the high-speed printer may be printing a line--all at the same time.

The greater the amount of this simultaneous, independent operation of system components, the less the total run time; and, thus, the closer the approach to an ideal system. An efficient balance between simultaneous operation and shared access to common circuitry is provided automatically in the GE-235 by two significant design features:

1. The use of separate controllers to control peripheral operating cycles and address memory independently after initiating instructions have been received.

2. The use of priority control logic to determine which system component is to have access to memory during any particular cycle.

Peripheral Controllers

CONTROL OF INDEPENDENT OPERATION

Each input/output peripheral except the console typewriter is independently controlled by a controller. One controller, however, may control more than one device. For example, a magnetic tape controller may control, one at a time, as many as eight tape handlers. Some controllers are physically separate units; others are incorporated in the peripheral cabinets. The punched card equipment controllers are located in the central processor.

Upon receiving instructions, a controller initiates the operating cycle of its particular peripheral and performs independently the control functions necessary to complete the cycle. Thus, the central processor is relieved of these duties and may continue with subsequent steps in the program. The specific duties of a controller during the operating cycle vary, depending upon the requirements of the peripheral it is designed to control.

CONTROLLER BUFFERING FUNCTION

All GE-235 peripheral controllers contain registers (buffers)that temporarily store data being transferred between the peripheral and memory. In performing this function such a register helps to adjust for the difference between the data transfer rate of the peripheral and the rate at which its controller is granted memory access. This so-called buffering action enables the peripheral to receive or transmit data through the register at its own transfer rate, making its operation relatively independent.

Part of the timing adjustment made by a buffer may be in changing the manner of data transfer from serial to parallel, or vice versa. For example, data is transferred serially from memory to the 80-bit buffer register in the card punch controller. Then, when the buffer is full, a row of 80 punches is made at the same time in parallel on a card. Similarly, data is read from magnetic tape one character at a time and assembled into a word within the tape controller for parallel transfer into memory.

OTHER CONTROLLER FUNCTIONS

As has been shown, a controller plays an active role in controlling independent operation of peripherals and a passive one in buffering. The following are additional controller functions:

1. Translating the code used by a peripheral into the code used internally by the central processor, or vice versa.

2. Making error checks and setting appropriate indicators.

3. Upon interrogation by the central processor, signaling whether it is ready or not to receive or transmit data.

4. Signaling a request for memory access to the priority control logic.

Priority Control

All peripherals and the central processor registers access memory through the M-register. Since, with concurrently operating components, access to memory may be requested by more than one component during the same cycle, some provision must be made to select the component that is to be allowed access. This is accomplished automatically in the GE-235 by the priority control logic. (See Figure IV-1.)

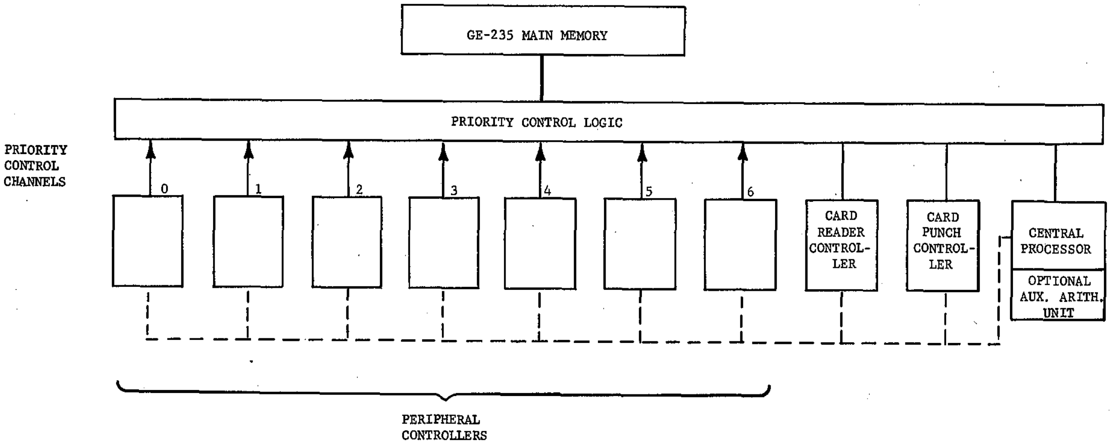


Figure IV-1. GE-235 Priority Control Logic

The relationship of the central processor and the card reader and card punch controllers to the priority control logic is fixed. Other peripheral controllers are attached to the priority control logic through a priority control channel.

The seven priority control channels are optional features for attaching peripheral devices to the priority control logic. A controller can be attached to each of the channels, which are numbered from 0 through 6, representing their relative order of priority. Channel 0 has the highest priority and channel 6 the lowest. Which peripheral to attach to a particular channel is at the discretion of the user. Thus a wide variety of system configurations is possible; and an existing system may easily be expanded, updated, or adapted to new uses.

For the greatest degree of simultaneous operation, however, channel assignment must take into account both (1) the maximum rate at which a peripheral may request memory access and (2) its relative ability to wait for a later access without loss of data when it is denied access in favor of a channel with a higher priority. A magnetic tape controller, for example, should generally have a higher priority than a printer controller. Once a tape controller initiates a tape handler read or write operation, the controller must have ready access to memory for optimum data transfer. The printer, on the other hand, does not complete its operating cycle until it has had several accesses to memory; and it can afford to wait for several cycles between accesses without loss of data or speed. Recommended channel assignments are shown in Table IV-1.

Channel	Controller
0	Disc storage unit (or magnetic tape, if no disc storage unit is used)
1	Magnetic tape (or second disc storage unit, if used)
2	Magnetic tape
3	Magnetic tape
4	Document handler, DATANET, or third high-speed printer
5	Second DATANET, document handler, or high-speed printer
6	High-speed printer

Table IV-1. Recommended Controller Selector Channel Assignments

With the standard priority control channels a maximum throughput of 55,000 words per second can be achieved. However, when a combination of high-speed, continuous-data-transfer devices--such as magnetic tapes and disc storage units--is used, the number of concurrent operations becomes limited. The standard priority control, for example, will accommodate no more than one disc storage unit controller for concurrent operations. The disc storage unit controller may be used in combination with a single 14 kc tape controller; but, if two 41 kc tape controllers are used, the disc storage unit controller cannot be used concurrently.

An optional dual access priority control, however, will allow for a maximum throughput of 110,000 words per second. And, with this feature, four 41 kc tape controllers or two disc storage unit controllers with two 41 kc tape controllers can operate concurrently.

Results of Time Sharing

The system configurations shown in Table IV-2 illustrate the time sharing made possible through concurrent operation of system components. The compute time available as shown in the examples is the minimum. The percentage may range upward from that shown, since the execution of many instructions does not require access to memory.

	I/O operation	Percent of total time
1	Read cards at 900 cpm	0.7
	Read magnetic tape at 15 kc (500 character record)	2.1
	Print at 900 lines per minute	0.7
	Total I/O time (%)	3.5
	% of total time left for computing:	96.5
	I/O operation	Percent of total time
2	Read cards at 300 cpm	0.2
	Disc storage unit (read and write cycle, 200 milliseconds)	0.4
	Print at 300 lines per minute	0.3
	Total I/O time (%)	0.9
	% of total time left for computing:	99.1
	I/O operation	Percent of total time
3	Read cards at 900 cpm	0.7
	Read magnetic tape at 41 kc (500 character record)	4.2
	Write magnetic tape at 41 kc	4.2
	Print 900 lines per minute	0.7
	Total I/O time (%)	9.8
	% of total time left for computing:	90.2

Table IV-2. Typical Controller Selector I/O Configurations

AUTOMATIC PROGRAM INTERRUPT (API)

General Description

Adding the optional API (automatic program interrupt) circuitry to a GE-235 system allows more efficient use of peripherals through simultaneous processing of unrelated programs. A main program and any number of subprograms (called "priority programs") can be accommodated. However, the API Executive Routine is limited to four priority programs. The priority programs are of two types: (1) peripheral-to-peripheral runs-- such as magnetic tape-to-printer, magnetic tape-to-card, and card-to-magnetic tape--and (2) remote inquiry station programs. Only one of the second type may be used, either alone or in combination with one or more of the first type.

When the API feature is functioning, the main program is interrupted whenever a peripheral governed by one of the priority programs signals that it is in ready status. The priority program is then automatically executed (usually with the aid of the API Executive Routine), and control is returned to the main program until another interrupt signal occurs.

API SWITCHES ON CONTROLLERS

Any input/output peripheral except the typewriter, perforated tape reader, and perforated tape punch can signal the central processor that it has finished one operation and is ready to begin another. The generation of the signal depends on the manual setting of a switch on the controller for each device. If the switch is ON, the signal will be generated and will cause an interrupt (if the system is in the interrupt mode); if the switch is OFF, the signal will not be generated and no interrupt will be caused by that controller under any condition.

SYSTEM OPERATING MODES WITH API

The system operating modes when equipped with API are as follows:

1. Noninterrupt mode-In this mode only the main program is processed and cannot be interrupted by a signal from a peripheral device. Whenever power is initially applied to the GE-235 system, it is in the noninterrupt mode.

2. Interrupt mode (API mode indicator on console shows API SET—In this mode the system is processing the main program, but it will be interrupted whenever there is a signal from a peripheral controller.

3. Priority mode (API mode indicator on console shows API PGM)—In this mode the system is processing a priority program as a result of an interruption of the main program by a signal from one of the peripherals.

Functioning of API

For API to function two conditions must be satisfied:

1. The system must be placed in the interrupt mode by execution of a SET PST (Set Priority Start) instruction.

2. The controllers that have been designated to cause an interrupt for a priority program must have their API manual switches ON.

If only the first condition is satisfied, the system stands ready to change to priority mode; but it will not receive any signal to do so. If only the second condition is satisfied, the interrupt signals will be generated; but no interrupt will occur. Instead, the system will remain in the noninterrupt mode; and any interrupt signal generated will be remembered until a SET PST is executed. At that time the automatic interrupt will follow immediately.

When API is functioning, the system enters the priority mode upon receiving an interrupt signal. However, the interrupt occurs only after the operating cycle for the instruction presently being executed has been completed. Also, after a test like BZE, for example, the interrupt will not occur until the central processor has made the decision called for. And an interrupt cannot occur between a BRU and the transfer to the location specified by this instruction. Thus, a loop such as BRU* or any sequence of BRU instructions cannot be interrupted.

The following actions occur automatically when interrupt becomes effective:

1. The central processor selects address modification group 32 (locations 0128 through 0131). This group is available only on systems with the API feature and can be used only as prescribed for API.

2. The contents of the P-counter (address of the next instruction in the main program are stored in bit positions 5-19 of word 1 of modification group 32 (location 0129). An indication of certain conditions existing at the time of interrupt are stored in bit positions 0-4 of the same word, as follows:

Bit position	1-Bit Indicates
0	KON in effect
1	TON in effect
2	BCD remembered carry function on
3	DEMODE in effect
4	OVERFLOW indicator on

   This record in bit positions 0-4 enables the programmer to restore the conditions existing at the time of interrupt.

3. Program control is transferred to location 0132.

4. SET PBK (Set Priority Break) is automatically executed (that is, it is executed by the API circuitry, not by programmed instruction).

Any number of SPB instructions may be executed while in the priority mode. They will refer to modification group 32, since only this group is available in the priority mode. The programmer should keep in mind that the address of the next instruction in the main program is stored in location 0129 (word 1 of group 32) and must not be destroyed.

Since the system has been placed in the noninterrupt mode by the automatically executed SET PBK instruction, the operations of the priority program cannot be interrupted. Signals from other priority peripheral devices that occur during this time are remembered and become effective whenever a SET PST followed by an indexed BRU is again executed. The peripheral being serviced in the priority mode cannot cause another interrupt until it completes its operating cycle and signals that it is again in ready status.

When the priority program operations have been completed, the following instructions must be executed for return to the main program:

1. SET PST: This instruction is required whether it is desired to return to the main program in the interrupt mode or not. If the programmer wishes to return to the main program in the noninterrupt mode, the SET PST must be followed by a SET PBK.

2. BRU (modified): This instruction should (if standard programming conventions are being followed) indicate branching to location 0000, modified by the contents of location 0129, (BRU 0,1) thus setting the P-counter to the address of the next instruction in the main program and causing a return to the modification group the main program was previously operating in. This is always the final step in the sequence for returning to the main program.

Any other modified BRU following the SET PST will cause exit from the priority mode but, of course, will not return the system to the main program. It is permissible to program any number of instructions between the SET PST and the modified BRU.

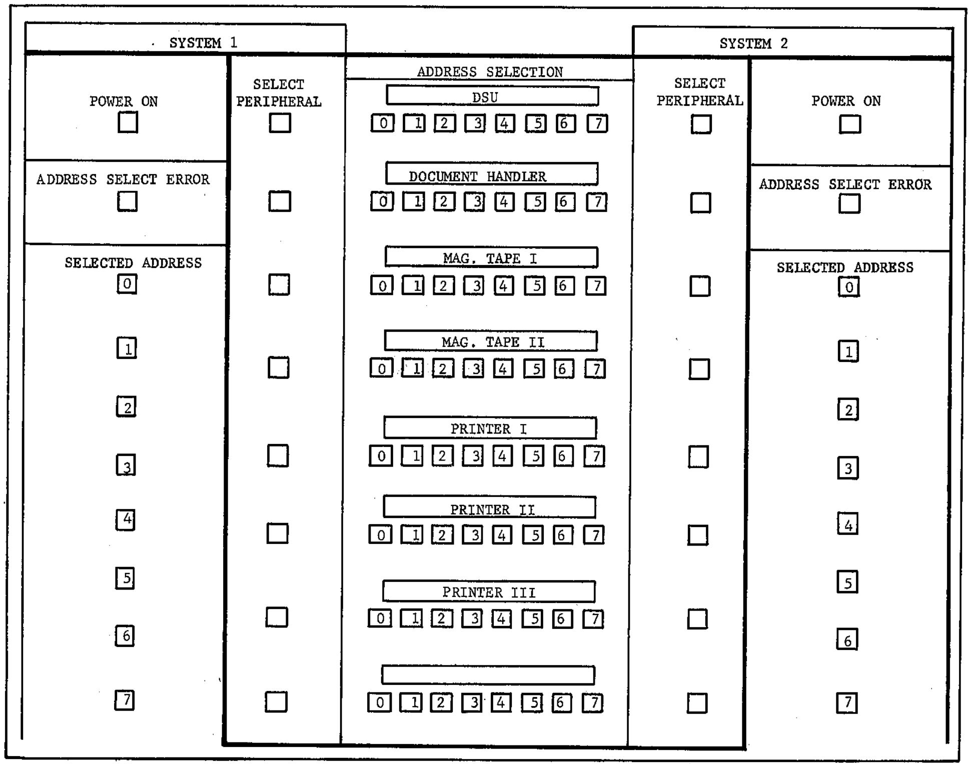


Figure IV-2. Peripheral Switch Control Unit Console

INTERSYSTEM SWITCHING OF CONTROLLER SELECTOR PERIPHERALS

The peripheral switch control unit is an optional GE-235 system feature that makes it possible to switch as many as seven priority control peripherals between the GE-235 and another compatible General Electric Information Processing System. This capability permits optimum use of the peripherals for multiple-system operation.

Switching between systems is accomplished by a peripheral switch unit at each peripheral. These units are powered and actuated by the peripheral switch control unit, which is also connected to each of the central processors. The controls on the peripheral switch control unit console perform two functions: (1) switch each controller to either system, as desired, and (2) assign the selected priority to each controller (see Figure IV-4). Both systems must be halted manually before peripheral switching is possible. The appropriate ADDRESS SELECT ERROR indicator will light if more than one peripheral is assigned the same priority on one system.

Two SELECT PERIPHERAL pushbuttons are associated with each peripheral for assigning it to one or the other system. The eight ADDRESS SELECTION pushbuttons for each peripheral are used for determining its priority control channel assignment, which is then displayed in the vertical SELECTED ADDRESS column. (Channel 7 of ADDRESS SELECTION is not used with a GE-235 system.) A power on-off indicator for each system is also provided. (Full details for the operation of this optional subsystem are given in a separate reference manual.)

V. MACHINE LANGUAGE

Information processing systems generally use a numeric system in a special form known as the machine language. In order to efficiently operate and program the GE-235 system, persons who perform these functions at a computer center should possess a general knowledge of numbering systems.

NUMBERING SYSTEMS

In addition to the familiar decimal numbering system, based on the numerals 0 through 9, the other numeric systems used in computer operations are:

1. Octal numbers, based on the numerals 0 through 7
2. Binary numbers, based on the numerals 0 and 1.

The operator and programmer should understand the principles of converting numbers from one system to another. The reasons for this are simply:

1. The GE-235 system holds and manipulates data in binary notation
2. The operator and programmer generally function most effectively when working with numbers expressed in decimal form
3. Because neither decimal nor binary numerals are satisfactory as a common language between programmer and computer, an intermediate numbering system (octal) is generally used for these conversions.

A general review of the three numbering systems and the methods of conversion among them is covered in the "References" section of this manual. There are, of course, several texts on the subject of computer arithmetic which may be used for a more extensive review.

DATA WORDS

In the GE-235 system, the word (or basic unit of information) consists of 20 binary digits. These words are stored in core memory storage locations, each of which is individually addressable. A word can represent:

1. An instruction
2. A binary data word or number
3. A binary-coded-decimal word (for expressing either alphabetic or numeric characters)
4. Or, any pattern of 20 bits if the programmer so desires.

The 20 bit positions of the GE-235 system word are depicted in Figure V-1. 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.

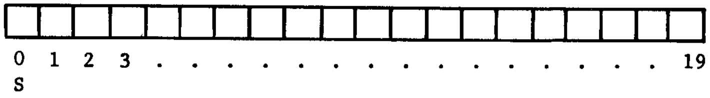


Figure V-1. Basic GE-235 Word

Binary Data Words

When a word is interpreted by the GE-235 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

0 0 0 0 1 0 4

(in octal)

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

In octal: 3777673_8

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

In octal: 3777674_8

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.

GE-235 Octal Notation

In programming debugging and patching, octal notation is used mainly for three reasons:

1. Octal notation provides the programmer with a more meaningful presentation than does binary
2. The GE-235 systems provide printed outputs (during General Assembly Program) and memory dumps in octal notation
3. Octal notation can easily be converted to binary or decimal. On the other hand, binary is difficult to read or write, or convert to decimal notation.

Conversion of GE-235 words from binary to octal or octal to binary is a simple mechanical procedure.

Given the 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.

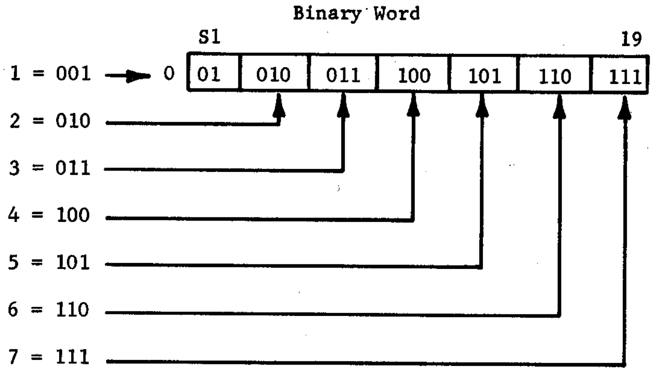
Evaluate each group and write the equivalent octal digit:

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

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-235 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.

Double Length Binary Words

The GE-235 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^{37}															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

In octal: 0000007_8

			2^{16}		2^{14}						2^{11}			2^{8}					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

In octal: 0567723_8

The portion of the double word with the higher binary position factors (2^{19} to 2^{37}) is stored in the first of the adjacent memory locations. The next sequential location contains the portion of the word with the lower (2^0 to 2^{18}) binary position factors. 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-235 to include normalized and unnormalized floating-point operations. Representation of floating-point numbers is discussed in the publication, GE-235 Auxiliary Arithmetic Unit Reference Manual (CPB-329).

GE-235 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-235 can process binary-coded decimal (BCD) or alphanumeric data. The BCD code uses six bit positions to express up to 64 character configurations, including all alphanumeric and special characters.

The 6-bit code consists of two groups:

ZONE GROUP		NUMERIC GROUP			
B	A	8	4	2	1
□	□	□	□	□	□

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-235 characters and their equivalent BCD codes are shown in the "References" section.

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

	B	A	8	4	2	1	Octal Representation
1	0	0	0	0	0	1	01
5	0	0	0	1	0	1	05
9	0	0	1	0	0	1	11
A	0	1	0	0	0	1	21
N	1	0	0	1	0	1	45
R	1	0	1	0	0	1	51
/	1	1	0	0	0	1	61
Z	1	1	1	0	0	1	71
$	1	0	1	0	1	1	53

The two higher order bit positions (S and 1) do not normally contain data, but can be used for program and printer control purposes discussed later. A representative GE-235 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					

In octal: 0220602_8

Double length BCD words are possible to express alphanumerics consisting of six characters. The basic BCD word of three numeric characters can be expanded with optional instructions to permit BCD arithmetic operations with BCD numbers of any practical length. 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 automatically set to zero.

Examples of BCD quantities:

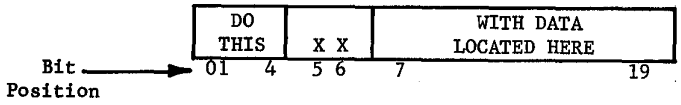
Decimal	BCD word(s)		Octal
+ 10	+ 0 1 0		0000100
+ 989	+ 9 8 9		0111011
- 10	- 9 9 0		2111100
- 989	- 0 1 1		2000101
+ 87649	+ 0 8 7	+ 6 4 9	
- 87649	- 9 1 2	- 3 5 1	

INSTRUCTION WORDS

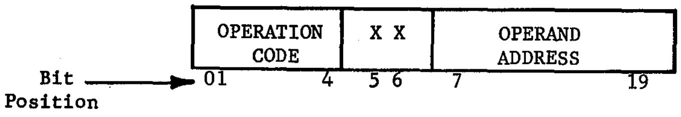
In the GE-235 system, instructions are expressed as single address words having 20 binary bits. Instructions are executed sequentially except for branching operations. The reading of the next instruction from memory occurs after the execution of the current instruction.

Instruction Word Formats

Instruction words are used to perform information processing operations within the central processor. Information flow to and from the peripheral units also is controlled by these instructions. The basic format of the instruction word is:



Or, in computer terminology:



In the above examples, the first five bits (0 to 4) indicate the operation to be performed.

Bits 5 and 6 provide for automatic modification of memory addresses by stipulating whether an index word is to modify the operand address. Automatic address modification is covered in Chapter VIII of this manual.

Bits 7 through 19 designate the operand address; that is, the memory location where the data to be operated on is stored. A number of instructions in the GE-235 repertoire have no operand address; this permits bits 5 and 6, and 7 through 19 to be used for other purposes.

Since the computer programmer deals primarily in octal notation, it is easy to express the basic GE-235 instruction format in this form. Simply follow the conversion from binary to octal notation described under "Number System" in the "Reference" section of this manual.

Instruction Categories

Instructions are divided into two general categories; those which are performed with no access to memory; and those which do require access to information stored in memory. Within these categories are several more specific instruction formats, as follows:

General Instructions (Require no access to memory):

- Data Transfers between registers
- Input/Output Operations
- Most Internal Test-and-Branch Instructions
- Shift Left or Right

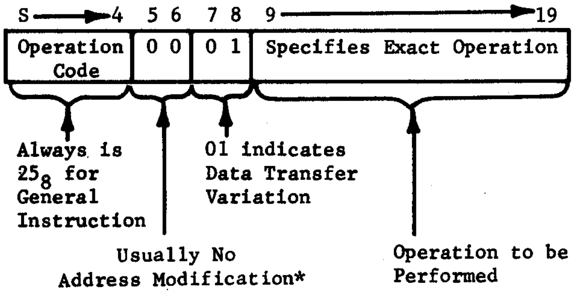
Instructions Requiring Access to Memory:

- Arithmetic
- Memory Transfers
- Certain Test-and-Branch Instructions
- Address Modifications

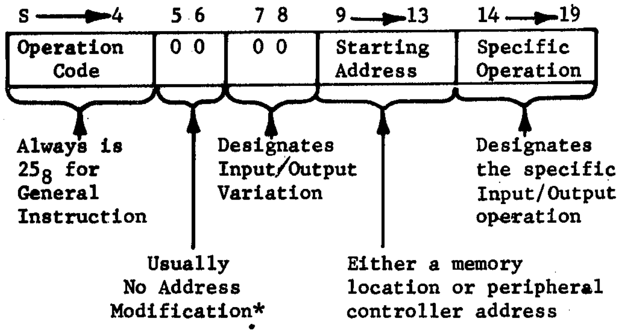
Detailed descriptions of these categories are covered in subsequent sections of this manual.

While the basic instruction format is used for some instructions, several variations of this format are necessary for execution of many other instructions. An example of each variation follows.

Data Transfer Format. It is used for instructions involving full word transfers between arithmetic registers and the arithmetic unit. They assume this format:

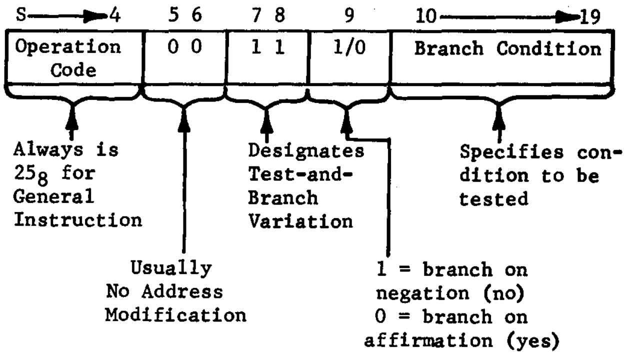

S → 4
5 6
7 8
9 → 19
Operation Code
0 0
0 1
Specifies Exact Operation
Always is 25_8 for General Instruction
01 indicates Data Transfer Variation
Usually No Address Modification*
Operation to be Performed

Input/Output Format. This 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:

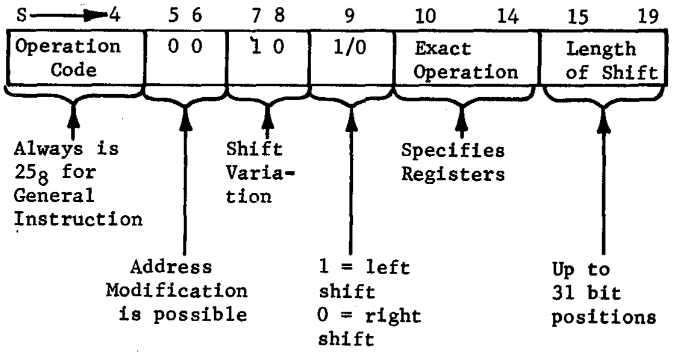

S → 4
5 6
7 8
9 → 13
14 → 19
Operation Code
0 0
0 0
Starting Address
Specific Operation
Always is 25_8 for General Instruction
Designates Input/Output Variation
Designates the specific Input/Output operation
Usually No Address Modification*
Either a memory location or peripheral controller address

* Address modification is possible.

Test-and-Branch Format. It is used for instructions that provide for breaking the normal sequence of instruction execution. These instructions are identified by 25_8 in bit positions S through 4, and 1-bits in positions 7 and 8. The test conditions 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 25_8 for General Instruction
Usually No Address Modification
Designates Test-and-Branch Variation
1 = branch on negation (no)
0 = branch on affirmation (yes)
Specifies condition to be tested

Shift Left or Right Format. Only shift instructions are written in this format. 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:



The specific bit patterns for the foregoing instruction formats can be found by converting the octal equivalent of the instructions given in the following sections to binary notation.

VI. PROGRAMMING AIDS

A large and ever-growing library of programming aids has been developed for the GE-235 system. These programs have been thoroughly tested and proved on two other experienced systems--the GE-215 and GE-225--with which the GE-235 is compatible. A brief review of programming aids is presented here for programming and operating staffs.

These aids are divided into the following categories:

Compilers
Assemblers
Simulators and Generators
Subroutines
Service Routines
Specialized Programs

GECOM, The General Compiler

The key program among compilers is GEneral COMpiler (GECOM), a programming technique which accepts a wide range of familiar languages and translates them into a program of machine instructions, ready to run on the GE-235. Among the programming languages included in GECOM are:

COBOL - COmmon Business Oriented Language, so close to English syntax that it can easily be read and understood by management, systems and accounting personnel.

TABSOL - TABular SOlution Language, the language of decision making, can solve an unwieldy number of sequential decisions without involving extensive data file processing.

ALGOL - ALGOrithmic Language, a language for algebraic expressions which simplifies programming mathematical problems.

GECOM REPORT WRITER - An extension of GECOM language that simplifies the programming of business reports. Provides a complete editing function for reports generated on the high-speed printer subsystem.

In addition to GECOM, General Electric has other versatile and timesaving compilers for the GE-235 system. They are:

WIZ - A very fast, one-pass algebraic compiler, used for engineering, scientific and other complex mathematical problems. It accepts a program written in easily-understood algebraic format.

FORTRAN II - FORmula TRANslator - Essentially a routine with which a source program written in the language of FORTRAN II is assembled into a program ready to use with the GE-235 system.

Assemblers:

The General Assembly Program is the key to all GE-235 compiling systems. It allows the programmer to write his own program in symbolic code rather than the absolute code of the computer. Instructions are written using mnemonic codes carefully chosen to provide significance to the user. Memory addresses can be written in symbolic notation. The programmer can call on a variety of subroutines to perform functions required by the program through a system of pseudo-instructions, thus further simplifying the assembly.

Operating System:

BRIDGE II - A magnetic tape maintenance and run sequence program, providing the following functions:

Run collection and sequencing from punched cards or magnetic tape
Tape correction of binary or symbolic programs
Dating of magnetic tape, using either date created or current date
Provision of run-to-run linkage
Provision of altering run sequencing
Combining of runs with subroutine or relocatable sections
Provision for loading priority programs for use with Automatic Program Interrupt.

The functions of Bridge II are directed by control cards that establish the run sequence for run execution. Use of Bridge II reduces overall processing and program editing time.

Simulators and Generators:

Forward Sort/Merge Generator - Produces tailored programs to efficiently sort and merge data files. It has extensive options that allow for use of user-oriented General Assembly Program coding to attain complete flexibility in data entry, format and selection.

Card Program Generator - Provides a smooth conversion from tabulating card equipment to the GE-235 by generating report programs from sets of input parameter cards. Another version of the program is available for magnetic tape input.

IBM-650 Simulator - Accepts IBM-650 system programs and data as input. It selects and executes the required routines to simulate 650 computer commands, and produces the same results and outputs as the 650 system.

LGP-30 Simulator - Executes the LGP-30 system instructions in the GE-235 and produces essentially the same results and outputs as obtained from the LGP-30.

Specialized Programs:

BANKPAC - A series of specialized programs tailored to the needs of commercial banks. The series features programs covering demand deposit accounting, installment loans, savings accounts, transit items, and personal trusts. Other BANKPAC routines do updating and maintaining files, issuing reports, making customer statements, and many other normal banking functions.

MATHPAC - A package of programs for scientific and engineering applications, including solution of simultaneous linear equations, matrix algebra, linear programming, Bessel and Gamma functions, multiple linear regression, roots of a polynomial, and least squares polynomial fit, in addition to numerous other mathematical routines.

Electric Utility Routines - A series of routines tailored to the needs of individual utilities, designed to compute load flow, optimal loading, load duration, gas flow and pressure, etc.

GE-235/CPM - Adapts a major network analysis technique to the GE-235. Analyzes complex projects, such as new product introduction, large construction projects and assembly line planning. These may consist of as many as 2100 activities and 1000 events, and can be analyzed in a matter of minutes.

TRIM-Rules for Inventory Management, is a comprehensive inventory control simulator which permits advance planning and control of inventories for reduced investment and higher efficiency.

Assembly Line Balancing - Provides production control management with a systematic method for analyzing the most efficient and profitable man-work element relationship automatically, to reduce cycle and idle time, and increase production and labor efficiency.

Parts Explosion Program - A method for automating the engineering, manufacturing and inventory control paperwork which always accompanies the production of increasingly diversified products.

Text Searching System - Permits automatic storage and retrieval of information from texts through three programs which convert texts into a form suitable for searching, compiles a program to search the texts for the information requested, and executes the compiled programs to search converted texts and announce the results of the search.

Subroutines: Designed to enter, handle, manipulate or move information within the GE-235 system memory. Some of the important subroutines accomplish the following:

Conversion of data from one number base to another (octal, binary, BCD)

Word replacement

Input/output for punched cards,magnetic tape, disc storage unit, perforated tape, typewriter, floating-point and double precision fixed point arithmetic

Solve scientific problems with mathematical routines described previously under MATHPAC.

Service Routines: The main functions of service routines are to assist in debugging programs and in simplifying operating procedures. Service routines prepared in symbolic and/or object program form are available to perform the following tasks:

Reset memory
Dump memory to cards, magnetic tape, perforated tape, or printer
Load programs into memory
Trace programs
Compare, correct, and print out contents of tape
Correct cards
Scan memory
Convert, analyze, and relativize card decks
Reproduce cards or print out contents.

SECTION B. PROGRAMMING CONSIDERATIONS

VII. PROGRAMMING THE GE-235 SYSTEM

Symbolic Programming

Programs for the GE-235 information processing system are written in symbolic coding. The programmer is thus able to write instructions in symbolic notation, rather than the absolute numeric code language of the computer. This relieves the programmer of much time-consuming clerical detail.

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-235 system. These mnemonics 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-235 performs algebraic addition.

The General Assembly Programming System is comprised of two parts:

1. The symbolic language used by the programmer in coding the source program.

2. The General Assembly Program that processes the source (or symbolic) program into a ready-to-execute machine language (or object) program.

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

1. 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-235 instructions.

2. Mnemonic operation codes corresponding to the more than 300 machine instructions of the GE-235 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.

Some additional features of the General Assembly Program are:

1. Memory addresses may be assigned either by using decimal or octal numeric notation, or by using symbolic notation, whichever results in maximum convenience to the programmer.

2. Additions to, and/or deletions from the coding, are easily made when the program is being written.

3. During assembly, either of the following output formats may be selected:

   a. Absolute numeric coding which is executed only in a predetermined fixed area of the computer memory.

   b. Relocatable numeric coding which is executable in any area of the computer memory. Its ultimate position in memory is decided by its loader routine.

4. Many types of clerical or language errors are detected and listed, thus reducing the amount of machine debugging time needed in new programs.

5. A listing of the assembled program is printed out, including error indications, symbolic listings, and assigned memory addresses.

General Assembly Program II

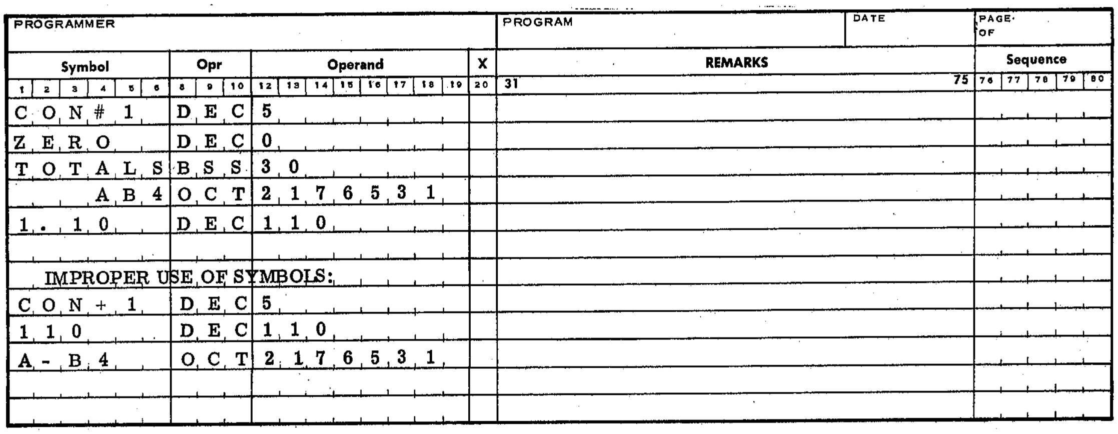
The General Assembly Program has been constantly improved to provide more features and greater flexibility. In the latest version--General Assembly Program II--several new operation codes and pseudo-operation codes have been added. The major aspects of General Assembly Program II, including the pseudo-instructions and the assembly directions, are described in this chapter. All GE-235 central processor instructions are covered later in this manual. The instructions for each peripheral subsystem are covered in the separate manuals for each subsystem.

Coding Sheet

The General Assembly Program coding sheet, illustrated in Figure VII-1, is divided into 6 fields designated from left to right as:

1. Symbol
2. Operation (Opr)
3. Operand
4. Index (abbreviated as "X")
5. Remarks
6. Sequence

Each of these fields indicates to the assembly program that certain operations are to be performed. The following simple rules should insure that correct results are obtained.



	PROGRAMMER				PROGRAM	DATE	PAGE OF
	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	CON#1	DEC	5				
2	ZERO	DEC	0				
3	TOTALS	BSS	30				
4	AB4	OCT	2176531				
5	1. 10	DEC	110				
6							
7	IMPROPER USE OF SYMBOLS:						
8	CON+1	DEC	5				
9	110	DEC	110				
10	A-B4	OCT	2176531				
11							
12							

Figure VII-1. Symbol Fields on General Assembly Program Coding Sheet

Symbol Field

The Symbol field is an address. The following rules apply to its use:

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

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

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

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

The assembly program assigns any Symbol field entry, along with its associated information, a specific memory location. Thus, the programmer need not know the actual computer address, but can refer to the symbolic address when the information is needed. Figure VII-1 shows both proper and improper use of symbols. The symbols shown are for illustrative purposes only.

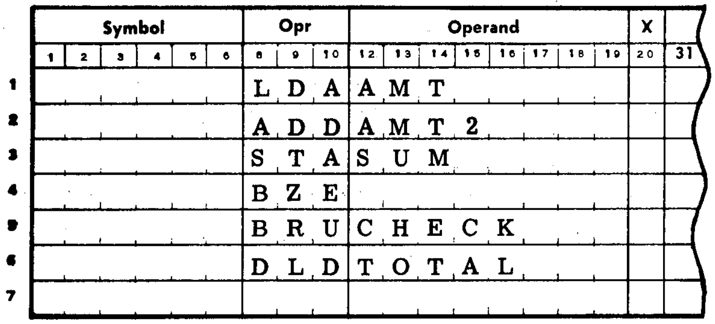
Entries 8 and 10, CON+1 and A-B4, are improper entries in the Symbol field because they violate rule two, above.

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

Operation Field

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

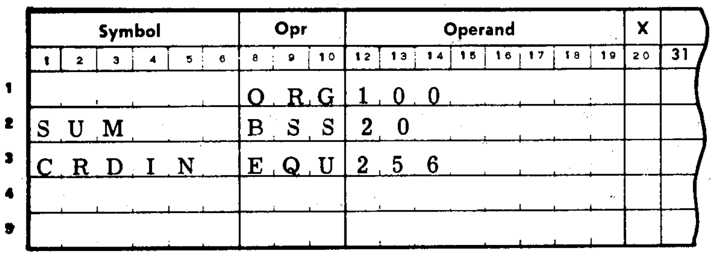
An instruction line contains a mnemonic indicating the desired computer operation. Usually this line is handled as one computer machine operation for each instruction mnemonic. Typical instruction lines are shown by Figure VII-2.



	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		LDA	AMT		
2		ADD	AMT2		
3		STA	SUM		
4		BZE			
5		BRU	CHECK		
6		DLD	TOTAL		
7					

Figure VII-2. Typical Instruction Lines

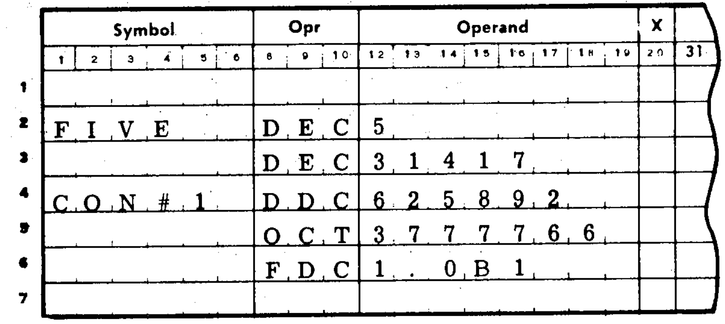
An assembly control line is interpreted and used by the program for internal assembly operations and does not become part of the assembled program. However, a control line in certain applications can cause additional words to be reserved in the assembled program. Figure VII-3 illustrates typical control lines.



	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		ORG	100		
2	SUM	BSS	20		
3	CRDIN	EQU	256		
4					
5					

Figure VII-3. Assembly Control Lines

Constant lines indicate to the assembly program the type of constant required by the user. The program then assembles the constant in the correct form. Figure VII-4 contains constant lines.



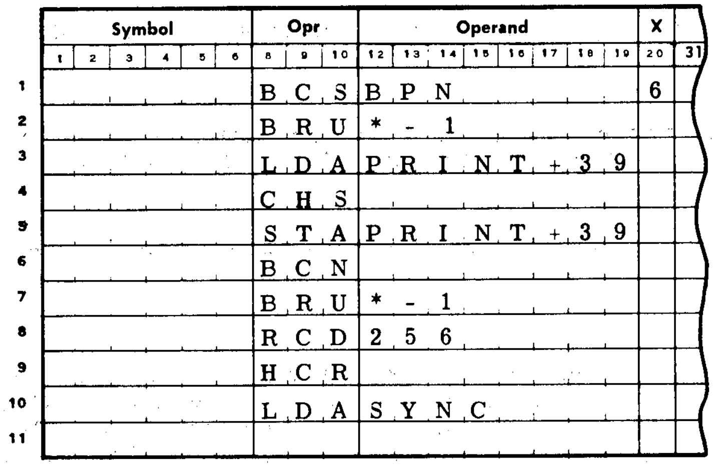
	Symbol	Opr	Operand	X
1				
2	FIVE	DEC	5	
3		DEC	31417	
4	CON#1	DDC	625892	
5		OCT	3777766	
6		FDC	1.0B1	
7				

Figure VII-4. Constant Lines

Operand Field

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

1. A mnemonic specifying an operation to be performed by the computer, as shown by line 1 of Figure VII-5.

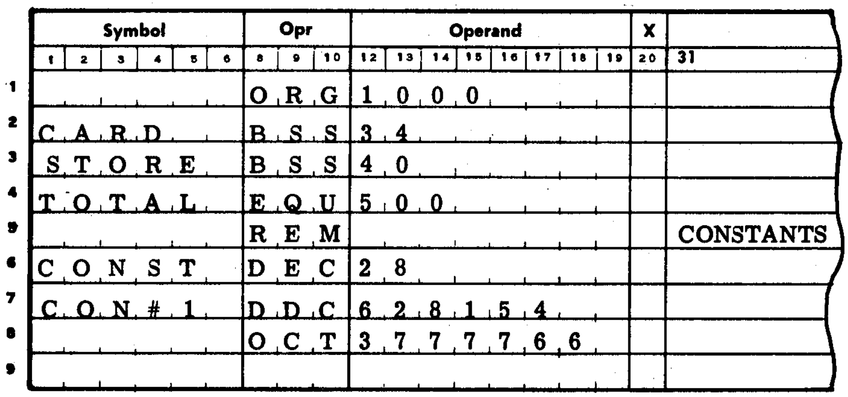


	Symbol	Opr	Operand	X
1		BCS	BPN	6
2		BRU	*-1	
3		LDA	PRINT+39	
4		CHS		
5		STA	PRINT+39	
6		BCN		
7		BRU	*-1	
8		RCD	256	
9		HCR		
10		LDA	SYNC	
11				

Figure VII-5. Coding Sheet Illustrating Operand Use by Instruction Lines

2. A decimal number which is the address portion of a computer instruction. This decimal address will be converted to binary by the General Assembly Program. For an example, see line 8 of Figure VII-5.

3. A symbol representing some address or number within the program. This symbol in conjunction with plus (+) or minus (-) can be used for relative addressing. The combination of symbol and sum (+) or difference (-) must not exceed eight (8) characters. Also an asterisk (*) can be used to denote reference to the line itself. Figure VII-5 contains illustrative examples. The asterisk and relative addressing involving arithmetic expressions can be used to reduce the total number of program symbols used, if necessary. Additional uses of the asterisk symbol are covered on page VII-34.



	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		ORG	1000		
2	CARD	BSS	34		
3	STORE	BSS	40		
4	TOTAL	EQU	500		
5		REM			CONSTANTS
6	CONST	DEC	28		
7	CON#1	DDC	628154		
8		OCT	3777766		
9					

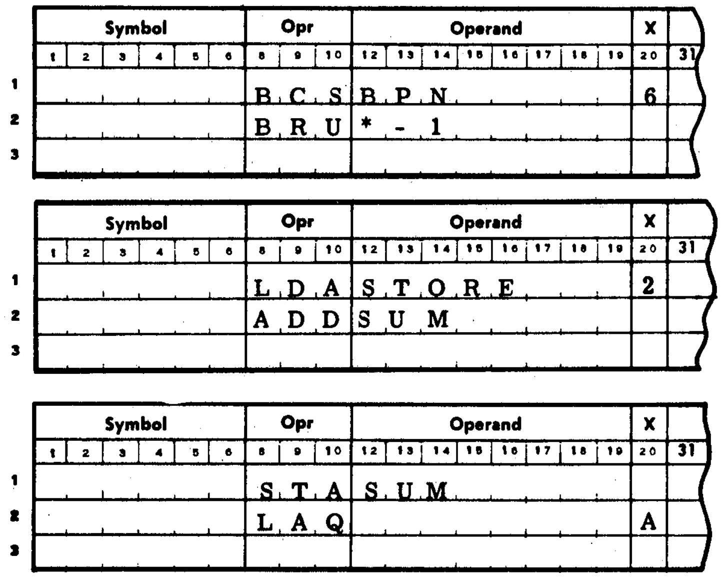
Figure VII-6. Examples of Operand Field in Assembly Control and Constant Lines

When an assembly control line is specified, the Operand field of the line contains information required by the General Assembly Program. Figure VII-6 contains illustrative examples.

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

X Field

The X field (column 20) specifies address modification before execution of the assembled computer instruction. Or, it may provide additional information to the assembly program, such as peripheral channel or tape handler number. If the X field is part of an instruction line, it may be blank or contain either a number or an alphabetic. Figure VII-7 shows examples of how the X field is used. Address modification and use of X-words are explained in detail in Chapter VIII. An "A" is used in the X field for certain auxiliary arithmetic unit instructions.



	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		BCS	BPN	6	
2		BRU	* - 1		
3					

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		LDA	STORE	2	
2		ADD	SUM		
3					

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		STA	SUM		
2		LAQ		A	
3					

Figure VII-7. Three Examples of X Field Assembly Coding

Remarks Field

The Remarks field is a helpful programming aid in that it can be used by the coder to explain or describe the actions of each program line. The Remarks field does not require memory locations within the assembled program, nor does it affect the assembly process. The Remarks field should be used extensively by programmers for adequate documentation. Refer to Figure VII-8 for examples.

Sequence

The Sequence field specifies the order of the lines to be assembled. This is strictly a programmer's convenience and is checked only when specifically indicated by use of an SEQ pseudo-instruction (On magnetic tape General Assembly Program only). Cards for the source program should be sequenced to prevent accidental mixing going unnoticed. Normally, the General Assembly Program does not check the card sequence; it does show the sequence on the object program listing. Figure VII-8 shows sequenced lines.

Properly sequenced decks of cards can be sorted using off-line tabulating card equipment to recreate a mixed program deck.

	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		REM			SUBROUTINE TO CHECK VOID DATE	05
2		LDA	VOID		VOID DATE DAY/MONTH/YEAR	10
3		SUB	CON#1		VOID CONSTANT	15
4		BNZ	OK		DATA NOT VOID SO PROCESS	20
5		SPB	CLOSE	1	CLOSE FILE DATA VOID	25
6						

Figure VII-8. Remarks and Sequence Entries

HOW THE GENERAL ASSEMBLY PROGRAM WORKS

The General Assembly Program produces an object program by successively processing the symbolic coding of the source program. The appropriate section of the source program is passed through the GE-235 in three runs. The output produced by each pass forms part of the input for the next pass. A typical flow chart diagram of a General Assembly Program is shown in Figure VII-9. Note that the outputs from both Pass 0 and Pass 1 are used as inputs to Pass 2.

The program can produce as its final output an object program in any of the following media, or combinations of them:

1. A program deck of punched cards in binary or octal notation.
2. A program magnetic tape.
3. A program perforated tape.
4. A printed listing of the object program.

Each of the three passes performs certain functions which are designed to provide the programmer with maximum information concerning the object programs.

Pass 0

Pass 0 accomplishes 3 operations:

1. It prepares the source program for subsequent use in other assembly passes by packing the symbolic input program deck, four instructions to one card, as shown by Figure VII-10. This packed deck is produced in the form of punched cards or placed on either perforated tape or magnetic tape, depending on whether a punched card,perforated tape, or magnetic tape system is used. The assembly program creates Pass 0 sequence numbers for its own use (2000 series equals packed symbolic numbers), as shown in the upper right-hand corner of the three cards in Figure VII-10.

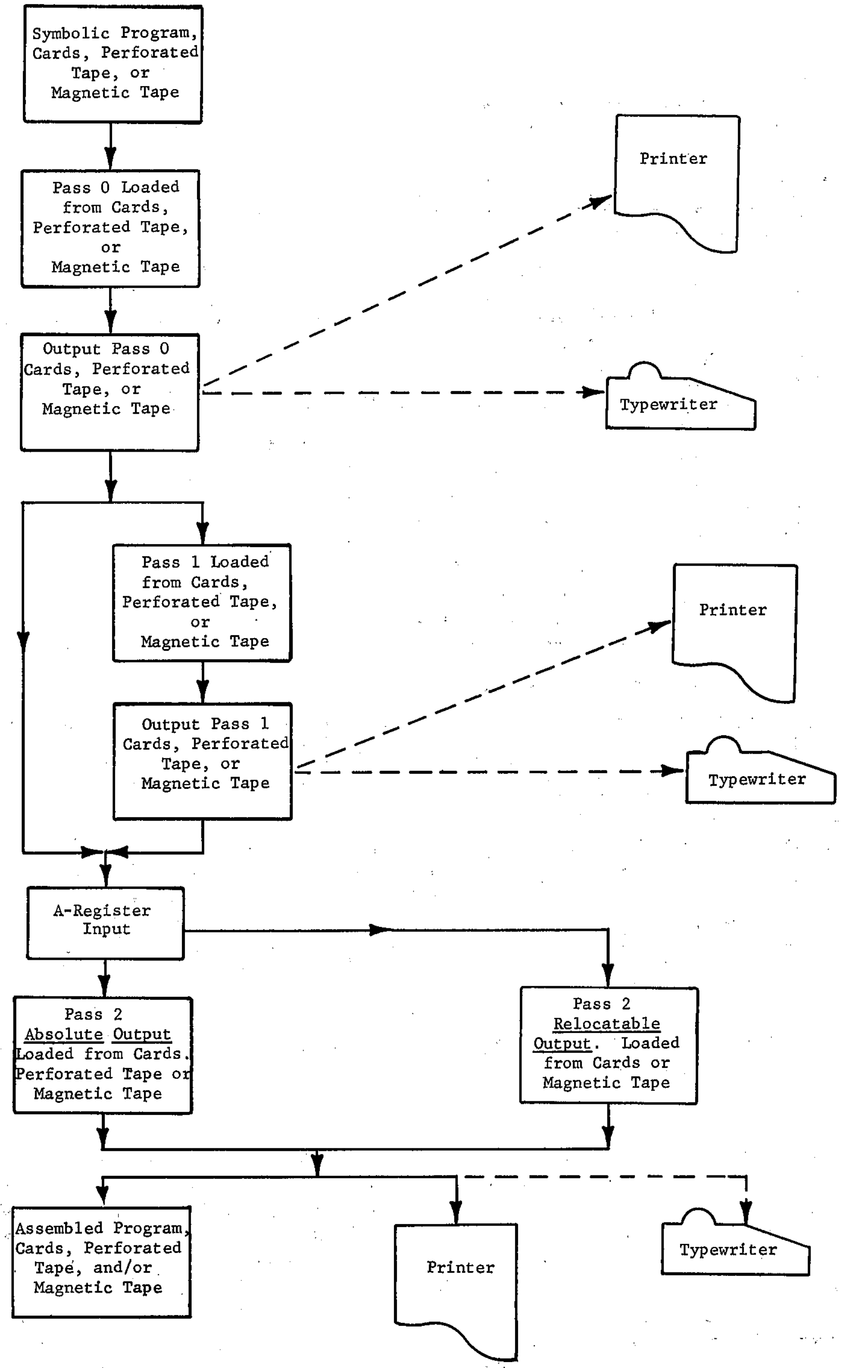
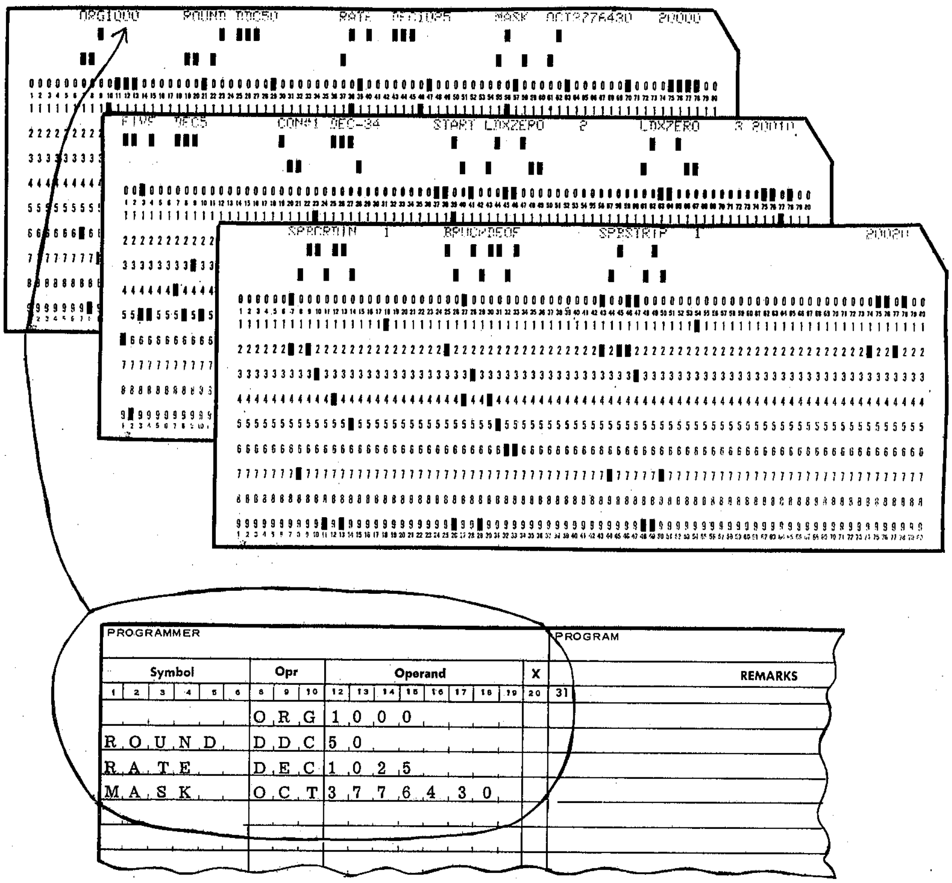

Symbolic Program, Cards, Perforated Tape, or Magnetic Tape
Pass 0 Loaded from Cards, Perforated Tape, or Magnetic Tape
Printer
Output Pass 0 Cards, Perforated Tape, or Magnetic Tape
Typewriter
Pass 1 Loaded from Cards, Perforated Tape, or Magnetic Tape
Printer
Output Pass 1 Cards, Perforated Tape, or Magnetic Tape
Typewriter
A-Register Input
Pass 2 Absolute Output Loaded from Cards. Perforated Tape or Magnetic Tape
Pass 2 Relocatable Output. Loaded from Cards or Magnetic Tape
Assembled Program, Cards, Perforated Tape, and/or Magnetic Tape
Printer
Typewriter

Figure VII-9. Flow Diagram of General Assembly Program II

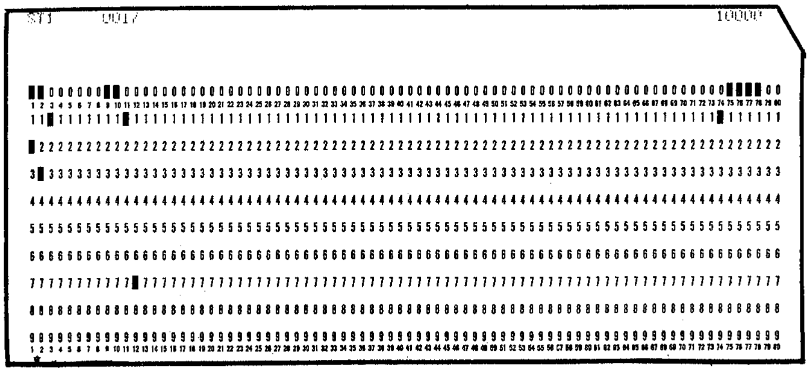

ORG1000 ROUND DDC50 RATE DEC1025 MASK OCT3776430 20000
FIVE DEC5 CONST DEC-34 START LDXZERO 2 LDXZERO 3 20010
SPBCRDIN 1 BPUCHDEOF SPBSTRIP 1 20020

PROGRAMMER | PROGRAM

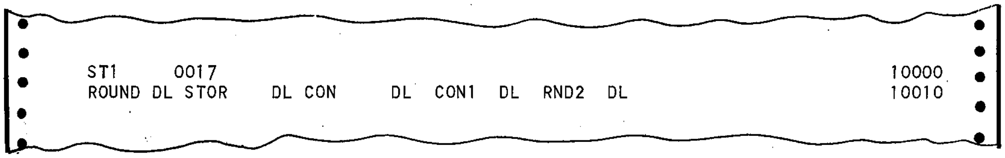
Symbol	Opr	Operand	X	REMARKS
	ORG	1000		
ROUND	DDC	50		
RATE	DEC	1025		
MASK	OCT	3776430		

Figure VII-10. Coding Sheet for First Card of Pass 0, Packed Card Deck. First Three Cards are Illustrated

2. A table of special symbols is formed containing all symbolic operands appearing in the card input/output, double length, floating-point, or document handler instructions. This symbol table, referred to as symbol table 1 (ST 1), can be recorded on magnetic tape, perforated tape, or punched cards. Figure VII-11 is a symbol table 1 card and a high-speed printer listing of the table as it appears on tape. Columns 1 through 3 on on the symbol card contain the identifying letters ST1. In columns 9 through 12, octal 0017 (15_{10}) is a control number used by the assembly program. It equals 3 times the number of symbols in ST1 (5). Columns 74 through 78 contain the assembly program-created sequence number of the card in the Pass 0 packed deck. For any succeeding pass, the cards are arranged according to sequence numbers, i.e., ST1 cards before packed symbol cards, and ST2 cards before both of these. In the printer listing, the letters DL (DL equals Double Length) between ROUND and STOR is a special category symbol.



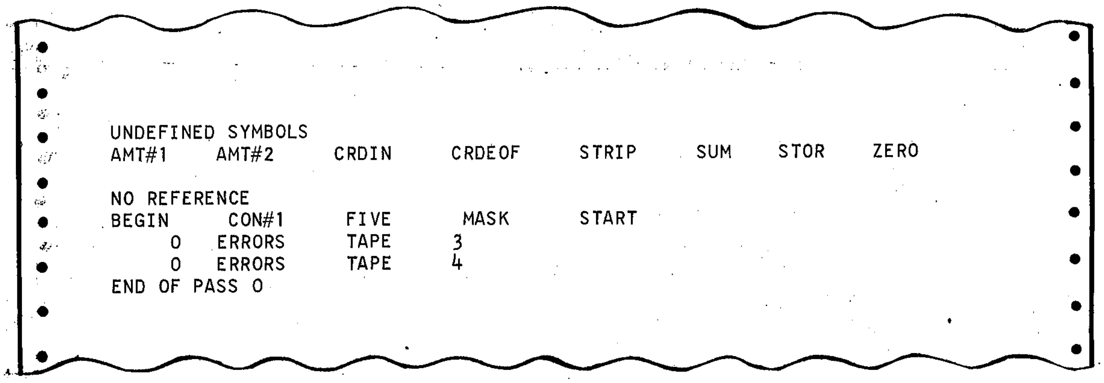
ST1 Card



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

Figure VII-11. Special Symbol Table as Produced by a Punched Card System and Listed on the High-Speed Printer by a Magnetic Tape Assembly Program

3. Pass 0 checks all symbolic names in the input deck for no reference, undefined, or multiple-defined usage. If any one of these conditions exist, Pass 0 provides a listing on the high-speed printer. Or, if desired, the listing can be printed on the console typewriter. Figure VII-12 is a sample printout by Pass 0 on the high-speed printer showing these symbolic names.



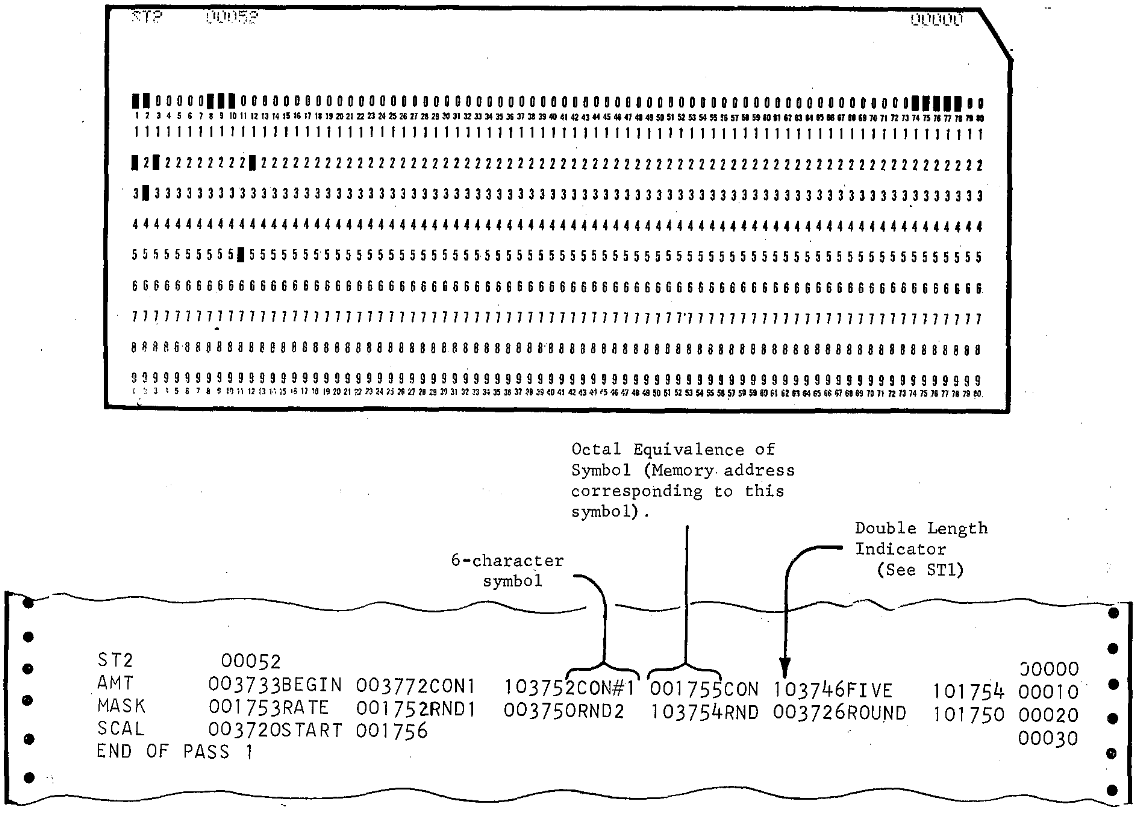
```
UNDEFINED SYMBOLS
AMT#1     AMT#2      CRDIN      CRDEOF     STRIP      SUM      STOR      ZERO

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

Figure VII-12. Printer Listing of Symbol Errors, Pass 0

Pass 1

Pass 1 uses the output of Pass 0 to assign memory locations to all symbols in the input source deck. It forms a sorted table of these symbols and the numeric values assigned. Symbol table 2 (ST2) is listed by the high-speed printer, written on magnetic tape, punched on perforated tape, or punched into cards. If no printer is available the console typewriter is used. Figure VII-13 shows a listing of symbol table 2 and a representative ST2 card.

In columns 8 through 12 of the card and the first line of the listing, the numerals 00052 are an octal control number (42_{10}) equal to 3 times the number of symbols in ST2 (14). In columns 74 through 78 of the card, and at the right-hand side of the listing, are the assembly-program-created Pass 1 sequence numbers (00000 series equals ST2). Other symbols on the printer listing are explained in Figure VII-13.

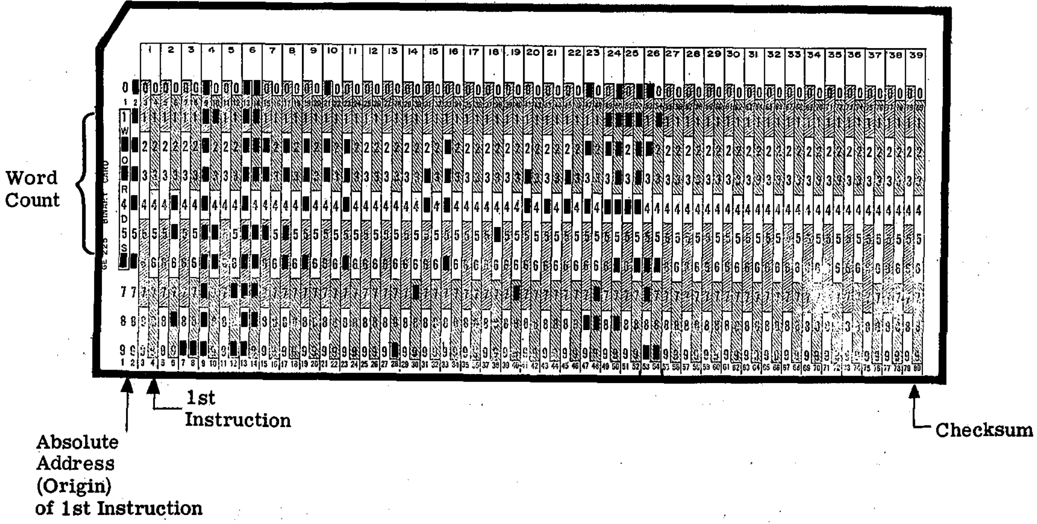

ST2
00052
00000
Octal Equivalence of Symbol (Memory address corresponding to this symbol).
Double Length Indicator (See ST1)
6-character symbol

```
ST2         00052                                                          ɔ0000
AMT         003733BEGIN 003772CON1   103752CON#1 001755CON 103746FIVE    101754 00010
MASK        001753RATE  001752RND1   003750RND2  103754RND 003726ROUND   101750 00020
SCAL        003720START 001756                                                 00030
END OF PASS 1
```

Figure VII-13. ST2 Card and Printer Listing of Symbol Table 2 of Pass 1

Pass 2

The outputs of Pass 0 and Pass 1 are used for input to the final pass of the General Assembly Program. Pass 2 does the complete assembly of each instruction as specified by the symbolic input program. The output of Pass 2 is the assembly listing, with indicated errors, and the object program itself, on cards, paper tape, or magnetic tape. Figure VII-14 shows (a) a typical object program binary card and (b) the printed assembly listing of a program of coded instructions. Pass 2 of General Assembly Program II may be relocatable (Pass 2R) or absolute (Pass 2A).



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

Figure VII-14. Assembly Listing and Object Program Binary Card (Absolute) from Pass 2

PSEUDO-INSTRUCTIONS

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

Pseudo-instructions are not executed by the computer, but are used to generate constants, control the assembly, and provide information on the program listing.

A listing of the constant-line General Assembly Program pseudo-instructions and the type of information assembled for each is given in Figure VII-15.

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

Figure VII-15. Pseudo-Instructions for Constant Lines

Pseudo-Instructions Used for Constant Lines

ALF ALPHANUMERIC

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

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

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1					
2	T Y P E	A L F	P L A		
3		A L F	N T		
4		A L F	C O D		
5		A L F	E N		
6		A L F	O T		
7		A L F	I N		
8		A L F	T A B		
9		A L F	L E		
10					

Appears in Memory
0474321
0456360
0234624
0256045
0466360
0314560
0632122
0432560

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

Example:

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

Appears in Memory
0604743
0606045

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

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

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

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

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

Appears in Memory
3567674
3565576
3565552

MAL MULTIPLE ALPHANUMERIC

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

	Symbol	Opr	Operand	X	REMARKS
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1	TYPE	MAL	08		PLANT CODE NOT IN TABLE
2					

This data is placed in memory as follows:

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

PAL MULTIPLE ALPHANUMERIC FOR PRINTER WITH PRINT LINE INDICATOR

This pseudo-instruction is similar to the MAL instruction with the exception of entering a minus sign in the last word of the alphanumeric data. (The minus sign signifies end-of-line during high-speed printer operation.)

Example:

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

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

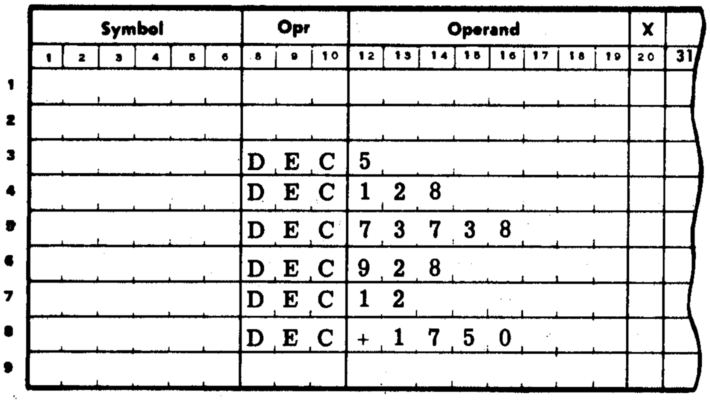
	Data	Memory
1.	PLA	0474321
2.	NT	0456360
3.	COD	0234624
4.	E	2256060

Note: Pseudo-instructions MAL and PAL cannot be used if a symbolic General Assembly Program is entered from magnetic tape during the assembly operation.

DEC DECIMAL

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

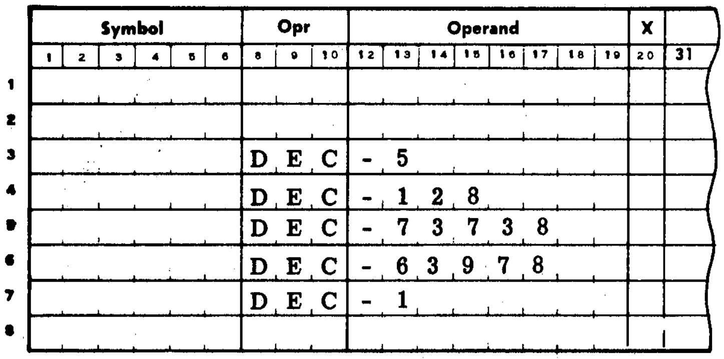
Examples of plus numbers:



	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1					
2					
3		DEC	5		
4		DEC	128		
5		DEC	73738		
6		DEC	928		
7		DEC	12		
8		DEC	+1750		
9					

Appears in Memory
0000005
0000200
0220012
0001640
0000014
0003326

Examples of minus numbers:



	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1					
2					
3		DEC	-5		
4		DEC	-128		
5		DEC	-73738		
6		DEC	-63978		
7		DEC	-1		
8					

Appears in Memory
3777773
3777600
3557766
3603026
3777777

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

Examples using the B character:

	Symbol	Opr	Operand	X		Appears in Memory
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31	
1						
2						
3		DEC	5B16			0000050
4		DEC	-5B16			3777730
5		DEC	73738B18			0440024
6		DEC	-4B3			3000000
7		DEC	+4B3			1000000
8						

The characters "." and "E" can be used to specify decimal scales, or decimal exponents. Normally, the . indicates a mixed number (see lines 3 through 6 in the following example), while E specifies the power of 10 by which the constant is multiplied. For example, E-2 indicates 10^{-2} and E2 indicates 10^{2}. If the character B is not used with . or E, only the integral portion of the number will be converted by the program. If the number of characters used to specify a decimal constant exceeds 8, the operation field of the next line is left blank and the constant is continued in the operand field, using two lines for one entry (See lines 7 and 8, 9 and 10, 11 and 12). Only one binary scale and one decimal scale can be indicated by a single decimal constant.

	Symbol	Opr	Operand	X		Appears in Memory
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31	
1						
2						
3		DEC	5.52			0000005
4		DEC	5.52B18			0000013
5		DEC	.52B18			0000001
6		DEC	5.5B18E2			0002114
7		DEC	.73738E2		}	0000000
8			B18			
9		DEC	-.5625B1		}	
10			0			3777340
11		DEC	-.1875B1		}	3777640
12			0			
13		DEC	.3217B0			0511327
14						

In using the DEC instruction it is possible to express the decimal constant in the Operand field in symbolic notation. However, unless there is some specific advantage, doing this results in having 'to add an EQU instruction immediately following the DEC that is otherwise not required.

DDC — DOUBLE LENGTH DECIMAL

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

Example of DDC:

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1					
2					
3		DDC	12		
4					
5		DDC	786432		
6					
7		DDC	-24		
8					
9		DDC	.0796896		
10			7928B2		
11		DDC	24B36		
12					
13		DDC	1.570796		
14			31847B2		
15		DDC	1018B0E-		
16			10		
17		DDC	152527B0		
18			E-10		
19					

Appears in Memory

{ 0000000
0000014

{ 0000001
1000000

{ 3777777
3777750

{ 0024315
0127545

{ 0000000
0000140

{ 0622077
0651010

{ 0000000
0066516

{ 0000007
1774566

FDC — FLOATING POINT DECIMAL

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

A detailed description of floating-point decimal formats and operations appears in the General Electric reference manual which covers the auxiliary arithmetic unit subsystem.

Examples of FDC:

	Symbol	Opr	Operand	X	31	Appears in Memory
1						
2						
3		FDC	1B1			0006000
4						0000000
5		FDC	1.442695			0006705
6			041B1			0507312
7						
8		FDC	3.321928			0013244
9			094B2			1517022
10		FDC	10B4			0022400
11						0000000
12						
13		FDC	1F40			Both give same
14			and			results:
15		FDC	1E40B133			1027530
16						0430221
17						Result unnor-
18						malized:
19		FDC	1E40B134			1031654
20						0214110

OCT — OCTAL

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

Example of OCT:

	Symbol	Opr	Operand	X	31
1					
2					
3		OCT	2777766		
4		OCT	377		
5		OCT	-377		
6		OCT	27600		
7		OCT	3777746		
8		OCT	-1777746		
9					

Appears in Memory
2777766
0000377
2000377
0027600
3777746
3777746

Z OCTAL OPERATION CODE

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

Sample Coding:

	Symbol	Opr	Operand	X	REMARKS
1		Z00	TEMP	2	EQUIVALENT TO LDA TEMP 2
2		Z04	-100	2	EQUIVALENT TO BXL 100 2
3		Z20	0		EQUIVALENT TO WORD 1 OF RWD
4					

Assembly Listing:

```
Memory            Memory
Location          Contents
 (Octal)           (Octal)          Assembly Program

01764             0000000           TEMP  DDC  0
01765             0000000
∫ ∫                ∫ ∫               ∫ ∫
00770             0041764                 Z00 TEMP  2
00771             0457634                 Z04 -100  2
00772             2000000                 Z20 0
```

Control Line Pseudo-Instructions

Assembly control instructions provide information which controls the internal operations of the General Assembly Program, although this information does not become part of the assembled program. In certain cases, additional words may be reserved in, or added to the assembled program. Figure VII-16 is a list of control instructions.

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

Figure VII-16. Assembly Control Pseudo-Instructions

SBR SUBROUTINE CALL

This pseudo-instruction can be used only if the General Assembly Program is called from the master assembly tape. An SBR instruction is used to instruct Pass 0 to obtain the specified subroutine from the master program tape. An error indication results if the specified subroutine is not present. The library subroutines furnished for the GE-235 are in symbolic form, and not in binary format.

The subroutine calls are saved until the user's END card is encountered. The specified subroutines are then placed behind the user's coding.

Note: The subroutines are assigned memory locations following the last instruction of the user's coding. If the user desires, the subroutines can be placed in reserved memory locations, as shown in the following examples.

The SBR pseudo-instruction cannot be used if a symbolic General Assembly Program is entered from magnetic tape during the assembly operation.

Example:

	Symbol	Opr	Operand	X	REMARKS
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		SBR	STRIP		BINARY - BCD CONVERSION ROUTINE
2		END	START		
3					

Assembly Listing:

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

This subroutine can also be called in the following manner:

Assembly Coding:

Symbol	Opr	Operand	X	
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
	LDA	TEMP		
	ADD	TOTAL		
	SPB	NPRIBD	1	
NPRIBD	SBR			
ZERO	DDC	0		
	END	START		

In both of these examples, the STRIP and NPRIBD subroutines are assigned memory locations following the last instruction (ZERO DDC0) of the coding.

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

Example:

Symbol	Opr	Operand	X	REMARKS
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
SUBR#1	BSS	76		RESERVE 76 LOCATIONS FOR ROUTINE
	LDA	TEMP		
	((			
	((			
	))			
	SPB	NPRIBD	1	
	SBR	NPRIBD		
	((			
	))			
	BRU	*		
	ORG	SUBR#1		
	END	START		

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

ORG ORIGIN

This pseudo-instruction controls the memory assignments performed by the General Assembly Program. When an ORG instruction is encountered, the program uses the contents of the Operand field to reset an internal counter in the assembly program, referred to as the Memory Allocation Register (MAR). Normally, the MAR is increased by one for each instruction encountered.

If the operand is a decimal, it is converted to binary by the program before being used. If the operand is symbolic, the symbols (S) must be predefined before being used. A symbol is "defined" by placing its name in the Symbol field (Columns 1 to 6) once, and only once, in a given program. The General Assembly Program ignores all but the Operand field on an ORG instruction. When no ORG is used, the program assigns an origin of memory location 00000.

Examples of ORG Pseudo-Instructions:

	Symbol	Opr	Operand	X	REMARKS
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		ORG	128		THE MAR IS SET TO 200 (128_{10})
2					AND NEXT INSTRUCTION STARTS
3					AT AN OCTAL 200
4					

	Symbol	Opr	Operand	X	REMARKS
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		ORG	BEGIN		THE SYMBOL "BEGIN" MUST BE PRE-
2					DEFINED AND THE MAR IS SET TO
3					THE ASSIGNED VALUE. IF BEGIN IS
4					NOT PREDEFINED THE MAR IS SET
5					TO ZERO
6					

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

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		ORG	1000		
2	TWO	DEC	2		
3	TEN	DEC	10		
4					

LOC LOCATION IN OCTAL

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

Example of LOC:

	Symbol	Opr	Operand	X	REMARKS
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		LOC	1750		MAR IS SET TO 3326 (1750_{10})
2	TWO	DEC	2		
3	TEN	DEC	10		
4					

BSS — BLOCK STARTED BY SYMBOL

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

Examples of BSS:

	Symbol	Opr	Operand	X	REMARKS
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		ORG	0256		
2	CRD	BSS	28		MAR IS INCREASED BY 28
3	PRINT	BSS	40		MAR IS INCREASED BY 40
4	INDEX	BSS	3		MAR IS INCREASED BY 3
5	STORE	BSS	SAVE		SAVE MUST BE PREDEFINED
6					

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

A negative decimal operand can be used to reduce the MAR, in effect equating a block of memory to another block already defined.

TCD — TRANSFER PUNCHED CARD

A TCD generates an instruction that will cause the loader to transfer control to the location specified by the Operand field. The operand may be decimal or symbolic. A TCD may be used as often as necessary in a source program because this instruction does not affect the Memory Allocation Register. A symbol in the operand must be predefined.

The TCD should not be used in place of an END instruction at the end of a source program. Information on the use of END follows.

Examples of TCD:

	Symbol	Opr	Operand	X	REMARKS
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		TCD	2900		
2		TCD	START#1		START #1 MUST BE PREDEFINED
3					

EQU EQUALS

This instruction equates a new symbol to some memory location already known to the assembly program. The operand (decimal or symbolic) indicates the specific memory location to be used. This instruction does not affect the MAR. Thus it may be used as often as necessary and at any point within the source or symbolic program without disturbing the memory assignment sequence of the assembly program. If the operand is symbolic, the symbol must be predefined. A decimal operand is converted to binary before being utilized.

Example of EQU:

	Symbol	Opr	Operand	X	REMARKS
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1	CRD	EQU	256		
2	AREA	EQU	CRD		CRD MUST HAVE BEEN PREDEFINED
3	AREA2	EQU	CRD+40		
4					

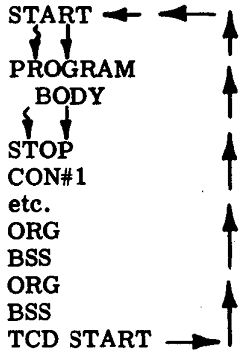
EQO EQUALS OCTAL

The EQO instruction is the same as the EQU except that the operand field must be in octal form. The assembly program ignores leading zeros in the operand.

Example of EQO:

	Symbol	Opr	Operand	X	REMARKS
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1	CRD	EQO	400		
2	AREA	EQU	CRD		CRD MUST HAVE BEEN PREDEFINED
3					

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



END — END OF PROGRAM

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

The TCD instruction previously described should be used where a transfer control card is to be generated before the end of the assembly program.

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

Examples of END Instructions:

	Symbol	Opr	Operand	X	REMARKS
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		END	1000		
2		END	START		START MUST BE PREDEFINED
3		END	CONST+3		CONST MUST BE PREDEFINED
4					

The END instruction of line 1 above would result in the punching of a transfer card as shown in Figure VII-17.

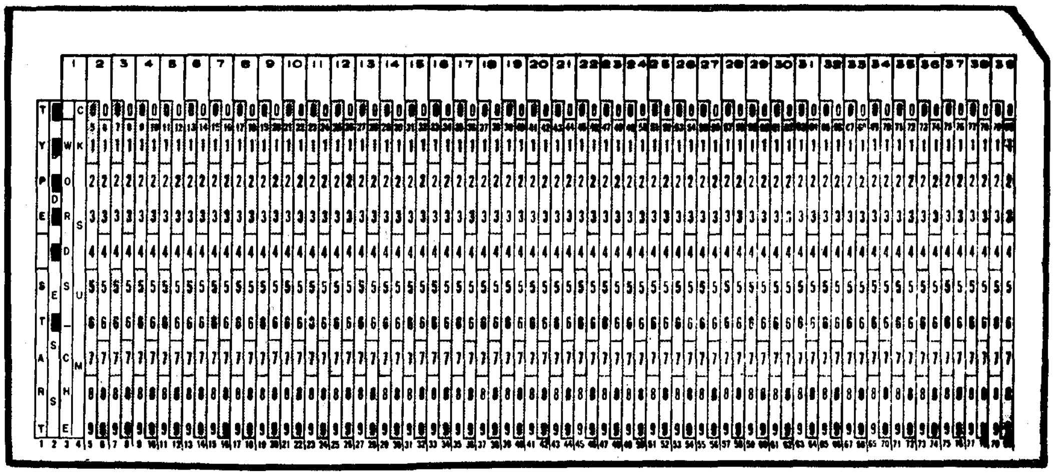


Figure VII-17. Transfer Card Generated by END Instruction of the General Assembly Program

REM REMARKS

The REM mnemonic in the Operand field is used by the assembly program to provide additional information on the object program listing. The entire card image, columns 1 through 80, is printed.

Example of REM:

	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		R E M			SUBROUTINE TO CHECK VOID DATE	V 0 0 5
2						

ADDITIONAL PSEUDO-INSTRUCTIONS

General Assembly Program II provides the following additional pseudo-instructions available only to relocatable assemblies.

These are assemblies in which the coding is written in relative locations. The block of locations can thus be shifted as necessary according to the requirements of the program being assembled.

EJT EJECT PRINTER PAPER

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

Assembly Coding:

	Symbol	Opr	Operand	X	REMARKS
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		D E C	6 2 1 5		
2		R E M			END ADJUSTMENT CONSTANTS
3		E J T			CONTINUE LISTING ON NEXT PAGE
4					

SEQ — CHECK SOURCE PROGRAM CARD SEQUENCE NUMBERS

Except for listing, sequence numbers are normally ignored by the assembly program. However, if the SEQ instruction is used, the assembly program checks to see that the card sequence numbers are in ascending order (Tape General Assembly Program only). Blanks are ignored. Numbers less than, or equal to, the preceding number are errors and are flagged with an S in the right margin of the listing. However, the assembly process continues.

Assembly Coding:

	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		SEQ			CHECK SEQUENCE NOS.	05
2		LDX	ZERO	2		10
3		DLD	TEMP	2		15
4						

NAM — PRINT NAME OR TITLE ON EACH PAGE

A page number is always printed at the top of each page of listing. When NAM is used with a name or title, which would appear in the Remarks section of the coding sheet (columns 31 to 75), the name is printed at the top of each page. The name may be changed by issuing a new NAM instruction. The name can contain a maximum of 45 characters.

Assembly Coding:

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		NAM			ROUTINE TITLE
2		ORG	1000		
3	ZERO	DEC	0		
4					

NLS — NO LIST

The General Assembly Program II normally lists the object program on the high-speed printer. The NLS pseudo-instruction is used to eliminate the printer listing if it is not needed.

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

Example:

Symbol	Opr	Operand	X	REMARKS
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
	NLS			ELIMINATE LISTING OF
PUN	DDC	0		NPRIBD SUBROUTINE
PUN1	BSS	4		
PUN2	BSS	2		
NPRIBD	EQU	PUN+12		
	LST			RESUME LISTING
TEMP	BSS	20		

RELATIVE ADDRESSING

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

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

Example 1: Assembly Coding:

	Symbol	Opr	Operand	X	REMARKS
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1	AMT	EQU	200		THIS INSTRUCTION (EQUATES THE SYMBOL
2					AMT TO MEMORY LOCATION 200 (DECIMAL)
3		LDA	AMT+6		THIS INSTRUCTION ILLUSTRATES RELATIVE
4					ADDRESSING . + 6 REFERS TO MEMORY
5					LOCATION 206
6		LDA	AMT-2		AMT - 2 IS MEMORY LOCATION 198
7					

Since the General Assembly Program has relative addressing capabilities, it will assign the correct memory addresses to AMT +6 and AMT -2.

Example 2: The starting locations of subdivisions within a program can be determined by relative addresses in ORG instructions.

Coding:

	Symbol	Opr	Operand	X	REMARKS
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1	CON	EQU	500		
2		ORG	CON		THIS ORG WILL START AT MEMORY
3					LOCATION 500 AS ESTABLISHED BY
4					USE OF EQU FOR THE SYMBOL. CON
5		ORG	CON + 50		STARTING MEMORY LOCATION HERE
6					IS 550 DUE TO RELATIVE ADDRESSING

A convenient method of relative addressing that reduces the number of symbols required is the use of the asterisk (*) character. An asterisk in the operand field of an instruction is interpreted by the assembly program as the present contents of the MAR; or, normally, the address of the instruction itself.

Example of Use of Asterisk:

	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	76
1		BCN			IF THESE INSTRUCTIONS START AT MEMORY	
2		BRU	* - 1		LOCATION 1000, THE ASTERISK IN THE OPERAND	
3		RCD	CRD		FIELD OF THE SECOND INSTRUCTION IS INTER-	
4		HCR			PRETED AS BEING 1001. THE ADDRESS	
5		LDA	* + 6		IN THE OPERAND FIELD IS 1001 - 1 or 1000.	
6		BMI				
7		BRU	*		THE INSTRUCTION ON LINE 7 CAUSES THE COM-	
8		SUB	CON 4		PUTER TO ENTER A CONTINUOUS LOOP SINCE	
9		BRU	* + 8		THE ASTERISK IS INTERPRETED AS THE	
10					ADDRESS OF THE INSTRUCTION ITSELF. THE	
11					MACHINE THEN EXECUTES THE SAME INSTRUC-	
12					TION CONTINUOUSLY.	
13						

The use of the asterisk in the previous example (line 7) is equivalent to writing the same symbol in both the Symbol and Operand fields of the same line (i. e., STOP, BRU, STOP).

Multiple relative addressing is permitted by which combinations of symbols, numbers, and asterisks can be added or subtracted in any given Operand field. The order of symbols, numbers, and asterisks in the operand is not restricted.

Examples of Multiple Relative Addressing:

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		BXL	SUM-B+8	2	
2		BSS	*+B-C+4		
3		BRU	9+4+3+GO		
4		LDA	8+*+*		
5		STA	128+5+14		
6					

Line 1 illustrates a convenient way to define the size of a memory area to be manipulated under the control of a BXL instruction. The area is 8 locations larger than the memory area between the sum and B.

Line 5 leaves to the assembly program simple arithmetic that often causes error, and documents the programmer's intention to store a quantity in the 14th location of the second field of a card image starting at location 128.

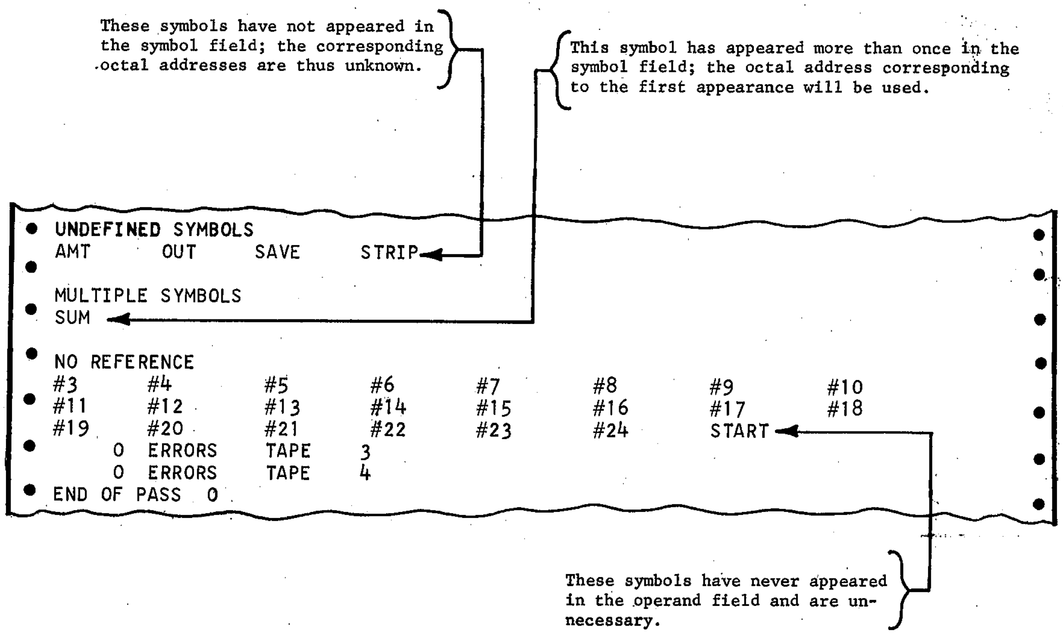
The other lines singly illustrate valid combinations about which the programmer should know, no matter how infrequently they are used.

GENERAL ASSEMBLY PROGRAM-DETECTED CODING ERRORS

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

Pass O

Pass 0 of the assembly program provides a listing of undefined symbols, multiple symbols, no reference, and, if a tape assembly program is used, an indication of any tape errors. Figure VII-18 is a sample printout from Pass 0.



These symbols have not appeared in the symbol field; the corresponding octal addresses are thus unknown.

This symbol has appeared more than once in the symbol field; the octal address corresponding to the first appearance will be used.

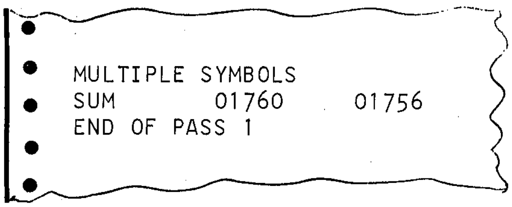
```
UNDEFINED SYMBOLS
AMT        OUT        SAVE       STRIP

MULTIPLE SYMBOLS
SUM

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

These symbols have never appeared in the operand field and are unnecessary.

Figure VII-18. High-Speed Printer Listing from Pass 0 of Assembly Program

Pass 1

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



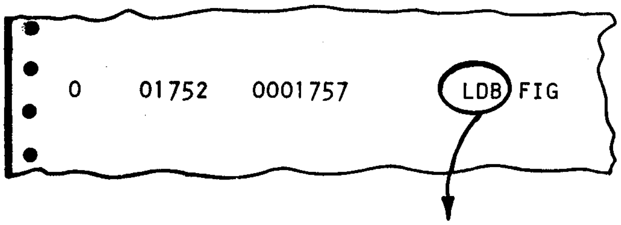
```
MULTIPLE SYMBOLS
SUM          01760       01756
END OF PASS  1
```

Pass 2

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

Code	Meaning
O	Illegal Mnemonic Operation. The mnemonic is unknown to the General Assembly Program. The program generates a 00 octal operation code as a substitute.

Example: Printout from Pass 2

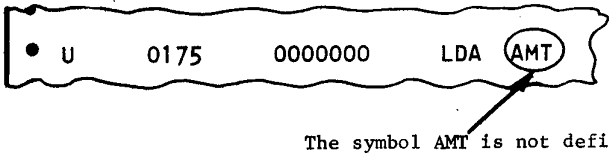


```
0      01752     0001757          LDB FIG
```

The OPR field contains an illegal operation.

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

Example: Printout from Pass 2

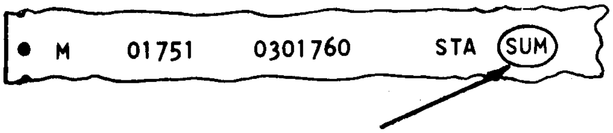


```
U      0175      0000000        LDA AMT
```

The symbol AMT is not defined in the symbol field.

Code	Meaning
M	Multiple-Defined Symbol. A symbolic name in this line appears in the Symbol field more than once in the program. The symbol in question is given an octal address corresponding to the address of the instruction in which it first appeared in the Symbol field.

Example: From Pass 2



```
M      01751      0301760        STA SUM
```

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

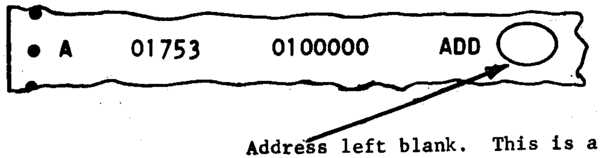
Code	Meaning
A	Error or Suspected Error in the Operand Address.

1. Blank Operand field in a line normally requiring an address.

2. An entry in the Operand field of a line which normally should be blank.

3. The numeric value of the operand does not meet the requirements of the line in which it was used.

Example: From Pass 2



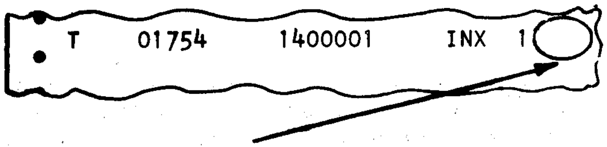
```
A     01753     0100000     ADD
```

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

Code	Meaning
T	Error or Suspected Error in X-Field.

1. **The X field is blank** in a line normally requiring an entry.
2. **The X field contains** an entry in an instruction line which ordinarily does not require address modification.
3. The numeric value of the entry in the X field violates the requirements of the line in which it appears.

Example: From Pass 2



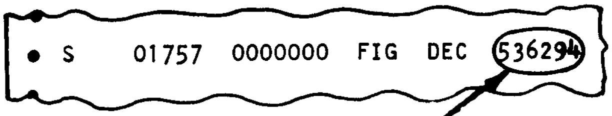
```
T     01754     1400001     INX  1
```

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

Code	Meaning
S	Scale Factors in DEC, DDC, FDC.

1. The specified binary and decimal scales are incompatible.
2. Two decimal or binary scales have been specified in the constant line.
3. The specified or implied binary scale causes the constant to exceed one binary word (overflow).

Example: **From Pass 2**



```
• S     01757  0000000  FIG  DEC  536294
```

The constant in the Operand field is larger than the scale permits (assumed by the assembly program to be binary 19). This constant must be double length.

OPERATING INSTRUCTIONS

Operating instructions for the General Assembly Program II depend on the GE-235 system configuration, and whether a punched card, perforated tape, or magnetic tape system is used for assembly. The specifications of the input/output media can be changed during assembly, as long as the output from one pass of the assembly is acceptable as input to the subsequent passes. Thus, the operating instructions and console switch settings on the central processor vary with different GE-235 system configurations.

Since the General Assembly Program II is constantly being improved, the detailed operating instructions are not contained in this manual.

Instead, a separate publication (GE-225 General Assembly Program II; publication Number CD225F1.006/007) covers this information. It is furnished with programs of punched card decks, or tapes, from the GE Computer Department Program Library and contains details on the assembly operation, systems tapes, modifications to the assembly program, relocatable object programs, and perforated tape assembly.

VIII. CENTRAL PROCESSOR OPERATIONS

CONTENTS SUMMARY AND FORMAT EXPLANATION

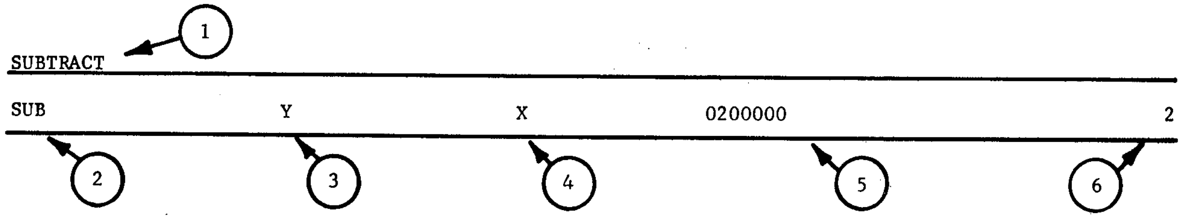
In this chapter are presented the instruction repertoire and programming procedures for operations that involve the central processor alone. The discussion is organized under seven main divisions:

1. Arithmetic (binary and decimal, with discussions of overflow, scaling, and rounding)
2. Data transfer (memory transfers, arithmetic register transfers, and register modification)
3. Shifting
4. Internal branching (unconditional branching and test-and-branch)
5. Using address modification words
6. Programming 16K memory systems
7. Sample coding.

Information about a particular central processor instruction may be located in this chapter by searching under the appropriate category or by consulting the alphabetical list of General Assembly Program instructions in the Reference Section at the end of the manual for the exact page number.

Programming information about the other system components treated in this manual is included in Chapter IX. Programming information about optional peripheral equipment is included in the separate subsystem reference manuals.

Each instruction discussed in this manual is introduced by a heading in the standard format shown in the example below. An explanatory key follows the example.



1. Name of instruction (operation to be performed)

2. General Assembly Program mnemonic operation code

3. Operand field symbol:

   a. Y indicates that the operand field in this instruction is occupied by the address of a memory location where data is to be found or stored. (In General Assembly Program language, the address may be either numeric or symbolic.)

   b. In some instructions the symbol K appears in this position. This signifies that the operand field in this instruction is the data itself (called "immediate data"); the nature of this data is specified in the description of the instruction.

   c. If no symbol appears in this position, the coding for this part of the instruction is fixed and does not refer to a memory location or other data.

4. The X symbol indicates that the address in the operand field of the instruction can be automatically modified by using address modification words. Omission of the symbol indicates the instruction cannot be modified in this way.

5. Representation of the operation code of the instruction in octal notation. (Addresses and immediate data are represented by zeros.)

6. Number of word times required to read the instruction out of memory and execute it (one GE-235 word time is equal to 6 microseconds).

Immediately below the heading appears a description of the action resulting from execution of the instruction. This is followed by one or more examples showing typical uses of the instruction and how they are coded for the General Assembly Program.

Certain conventions in format and usage have been followed in the descriptions and examples for a more efficient presentation. These are listed and explained below:

1. All descriptions and examples are in terms of the General Assembly Program detailed in Chapter VII.

2. Octal notation is used whenever applicable, both for convenience and for familiarizing programmers with this method of expression. It may be assumed that, unless otherwise noted, the following two types of numbers are octal:

   7-digit numbers (which usually represent the 20-digit binary contents of a register or memory location)

   5-digit numbers (which are used to represent memory location addresses)

   Where there is a possibility of confusing decimal and octal notation, subscripts or labels are used.

3. In binary or octal notation a negative value for a number is not indicated by placing a minus sign in front of it but by expressing it as the 2's complement of its positive value, since negative numbers are represented in this manner in the central processor.

4. Registers are referred to by the appropriate capital letter. For example, "transferred from A" means "transferred from the A-register."

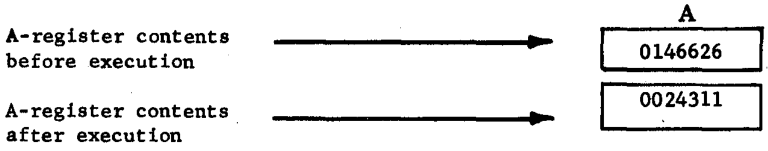
5. The capital letters Y and X are used to refer to memory locations and address modification words, respectively, as discussed in the explanation of the headings. For example, "transferred to Y" means "transferred to the memory location whose address is Y."

6. Such statements as "If the address of Y is an even number..." apply equally to Y and to any address modifications of Y that are also even numbers.

7. Such statements as "The contents of Y are loaded into A" or "Y is loaded into A" mean that the full contents of the one is loaded into the other. That is, the sign bit plus the bits in positions 1 - 19 have been transferred: Y(S, 1-19) is loaded into A(S, 1-19). When less than the full contents is referred to, the limitation will be noted. For example, Y(1-19) refers to the contents of location Y except for the sign bit.

8. Wherever possible, symbolic addressing is used in the sample coding illustrating the example.

9. The contents of the affected registers before and after execution of the instruction are displayed in each example in the following manner:


A
A-register contents before execution
0146626
A-register contents after execution
0024311

10. If a question mark is shown as register contents, it indicates that the contents of the register are unknown or immaterial or both, so far as the instruction being executed is concerned.

ARITHMETIC

Binary Mode Instructions

ADD

ADD	Y	X	0100000	2

The contents of Y are algebraically added to the contents of A. The result is placed in A. Y is unchanged. Overflow is possible.

EX 1: Add the positive number 0122315 in symbolic location AMT#2 to the positive number 0146626, which has previously been loaded into A.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		ADD	AMT#2		
2					

A

0146626

0271143

EX 2: Add the negative number 3655463 in symbolic location AMT#3+1 to the positive number 0146626, which is already in A.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		ADD	AMT#3+1		
2					

A

0146626

0024311

SUBTRACT

SUB	Y	X	0200000	2

The contents of Y are algebraically subtracted from the contents of A. The result is placed in A. Y is unchanged. Overflow is possible.

EX 1: Subtract the positive number 0122315 in symbolic location AMT#2 from the positive number 0146626, which has been previously loaded into A.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		SUB	AMT#2		
2					

A
0146626
0024311

EX 2: Subtract the positive number 0177615 in symbolic location AMT#3 from the smaller positive number 0146626, which has been previously loaded into A.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		SUB	AMT#3		
2					

A
0146626
3747011

DOUBLE-LENGTH ADD

DAD Y X 1100000 3

If the address of Y is even, the contents of Y (S,1-19) and Y+1 (1-19) are algebraically added to the contents of A (S,1-19) and Q (1-19). However, if the address of 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 Q is set to agree with that of A. Y and Y+1 are unchanged. Overflow is possible.

EX 1: Add the positive number 0000001 1104677 in symbolic locations AMT#7 and AMT#7+1 to the positive number 0000001 0003734, which has been previously loaded into A and Q. AMT#7 is an even-numbered location.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		DAD	AMT#7		
2					

A	Q
0000001	0003734
0000002	1110633

EX 2: Add the positive number 0000001 1104677 in symbolic locations AMT#7 and AMT#7+1 to the negative number 3777776 3774044, which has been previously loaded into A and Q. AMT#7 is an even-numbered location.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		DAD	AMT # 7		
2					

A	Q
3777776	3774044
0000000	1100743

EX 3: Add the negative number 3777776 3145660 in symbolic locations AMT#9 and AMT#9+1 to the negative number 3777776 3774044, which has been previously loaded into A and Q. AMT#9 is an even-numbered location.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		DAD	AMT # 9		
2					

A	Q
3777776	3774044
3777775	3141724

EX 4: Add the positive number 1104677 0001144 in symbolic locations AMT#7+1 and AMT#7+2 to the positive number 0000001 0003734, which has been previously loaded into A and Q. AMT#7+1 is an odd-numbered location; thus the contents of AMT#7+1 are added to the contents of A and also to the contents of Q. The contents of AMT#7+2 are ignored.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		DAD	AMT # 7 + 1		
2					

A	Q
0000001	0003734
1104700	1110633

DOUBLE-LENGTH SUBTRACT

DSU Y X 1200000 3

If the address of Y is even, the contents of Y (S, 1-19) and Y+1 (1-19) are algebraically subtracted from the contents of A (S, 1- 9) and Q (1-19). However, if the address of 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.

EX 1: Subtract the positive number 0000001 0003734 in symbolic locations AMT#6 (even) and AMT#6+1 from the positive number 0000001 1104677, which has been previously loaded into A and Q.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		D S U	A M T # 6		
2					

A	Q
0000001	1104677
0000000	1100743

EX 2: Subtract the positive number 1104677 0001144 in symbolic locations AMT#6+1 (odd) and AMT#6+2 from the positive number 1104677 1104700, which has been previously loaded into A and Q.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		D S U	A M T # 6 + 1		
2					

A	Q
1104677	1104700
0000000	0000001

ADD ONE

ADO 2504032 2

A positive 1 is added algebraically to A (bit position 19). If the capacity of A is exceeded, overflow occurs.

EX 1: Add +1 to the positive number 0146626, which has been previously loaded into A.

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

A
0146626
0146627

EX 2: Add +1 to the negative number 3655463, which has been previously loaded into A.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		ADO			
2					

A

3655463

3655462

SUBTRACT ONE

SBO 2504112 2

A positive 1 is algebraically subtracted from the contents of A (bit position 19). If the capacity of A is exceeded, overflow occurs.

EX 1: Subtract +1 from the positive number 0177615, which has been previously loaded into A.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		SBO			
2					

A

0177615

0177614

EX 2: Subtract +1 from the negative number 3600163, which has been previously loaded into A.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		SBO			
2					

A

3600163

3600164

MULTIPLY

MPY Y X 1500000 5-23

The overflow indicator on the operator's console is automatically turned off before execution of this instruction.

The contents of Y are algebraically multiplied by the contents of Q. The product is placed in 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.

EX 1: Multipy the positive number 0146626 in symbolic location AMT#1 by the positive number 0122315 in Q. The A-register contains zeros.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		M P Y	A M T # 1		
2					

A	Q
0000000	0122315
0010213	0134436

EX 2: Multiply the positive number 0146626 in symbolic location AMT#1 by the positive number 1460716 in Q. The A-register contains the positive number 0112103.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		M P Y	A M T # 1		
2					

A	Q
0112103	1460716
0122001	1754367

DIVIDE

DVD Y X 1600000 26-29

The overflow indicator on the operator's console is automatically turned off before execution of this instruction.

The contents of A(S, 1-19) and Q(1-19) are algebraically divided by the contents of Y. The quotient is placed in A(S, 1-19); the remainder is placed in Q(S, 1-19). The sign of the remainder is the sign of the dividend.

The dividend must be right-adjusted in A and Q for proper scaling. For proper division the absolute magnitude of the divisor (Y) must be greater than the absolute magnitude of that portion of the dividend appearing in A. (The leading 1-bits of a negative number, which is in 2's complement form, are recognized as such by the processor and are not considered as part of any actual quantity in A.)

A divide check is made by the central processor before execution of this instruction. If the proper conditions for division have not been satisfied, the overflow indicator is turned on; and control is transferred to the next instruction in sequence. A quotient will be produced by such a division operation, but its accuracy is unreliable.

EX 1: Divide the positive number 1777674 in Q by the positive number 0146626 in symbolic location AMT#1. The A-register contains zeros.

	Symbol	Opr	Operand
	1 2 3 4 5 6	8 9 10	12 13 14 15 16
1		DVD	AMT#1
2			

A	Q	AMT#1
0000000	1777674	0146626
0000011	0142566	0146626

As can be seen from the description of the DVD instruction given above, care must be taken in preparing the registers for its execution. This is especially true when dealing with dividends that may be either positive or negative. If the dividend is first loaded into A and then transferred to Q by executing an MAQ instruction, the quotient resulting from the following division will be incorrect. This happens because the processor considers A(S, 1-19) and Q(1-19) together as the dividend. With a single-word negative dividend, then, A should be filled with leading 1-bits, since the number is in 2's complement form. However, execution of an MAQ always places zeros in A. This is the proper bit configuration for a positive single-word dividend but not for a negative one.

A suggested procedure for placing a single-word dividend in Q with the proper bit configuration in A regardless of the sign of the dividend is shown below:

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		LDA	DVND		
2		SRD	19		
3					

The dividend, assumed to be in symbolic location DVND, is first loaded into A. When the Shift Right Double instruction is executed, A(1-19) is shifted to Q(1-19); and A(1-19) is filled with either 1-bits or 0-bits, depending on the sign of DVND. This procedure, of course, may also be followed in cases where the dividend is known always to be negative.

In general, an XAQ instruction should not be used to transfer a single-word dividend from A to Q. With this instruction, the contents of Q are transferred to A and become part of the dividend. Since the contents of Q are either unknown or are not usually the configuration of 1-bits or 0-bits required, the quotient resulting from the subsequent DVD instruction will be incorrect.

Decimal Mode

In business applications, data to be processed is often recorded externally in a decimal or BCD (binary-code decimal) format. To use such data in a binary processor often requires conversion of data from decimal or BCD to binary, computation in binary mode, and subsequent reconversion to decimal or BCD format for external use.

The decimal arithmetic optional feature (part of the optional group which includes additional modification word groups and the three-way compare instruction) provides the central processor 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.

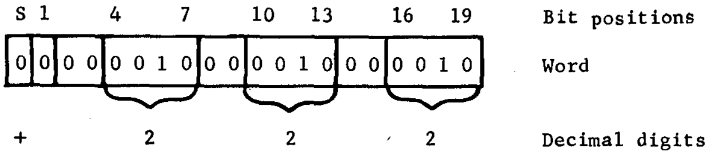
A system with the decimal arithmetic feature normally operates in the binary mode. Operation is shifted to the decimal mode only by executing a SET DECMODE instruction and can be returned to the binary mode by executing a SET BINMODE instruction. Initial depression of the PWR ON switch on the control console automatically sets the system in the binary mode.

The decimal arithmetic feature modifies the execution of the following existing binary arithmetic instructions:

1.	Add	ADD
2.	Subtract	SUB
3.	Add One	ADO
4.	Subtract One	SBO
5.	Double Add	DAD
6.	Double Subtract	DSU.

All other instructions are unaffected and continue to be executed in the normal binary mode. Address modification is performed in binary regardless of the mode set.

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



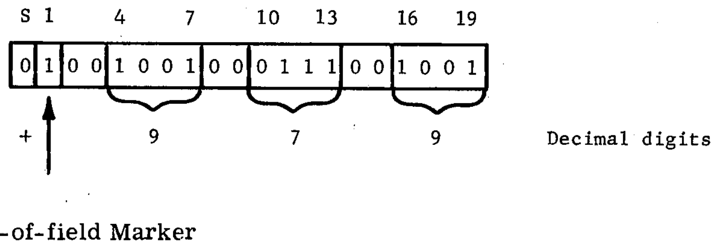
Bit positions 4 - 7, 10 - 13, and 16 - 19 are used to express decimal digits in standard BCD format. Decimal quantities greater than 999 are expressed by using two or more 20-bit words. The sign of the decimal number is in the S-bit position of the word containing the most significant decimal digit; 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, 14 and 15) contain 0-bits and do not enter into arithmetic operations.

END-OF-FIELD MARKERS

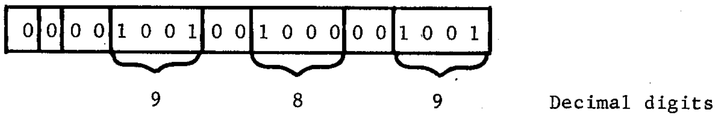
A 1-bit must be placed in bit position 1 in the word containing the most-significant (highest-order) digits of a decimal number. This bit serves as a marker to define the end of the decimal field.

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

Memory Location Y

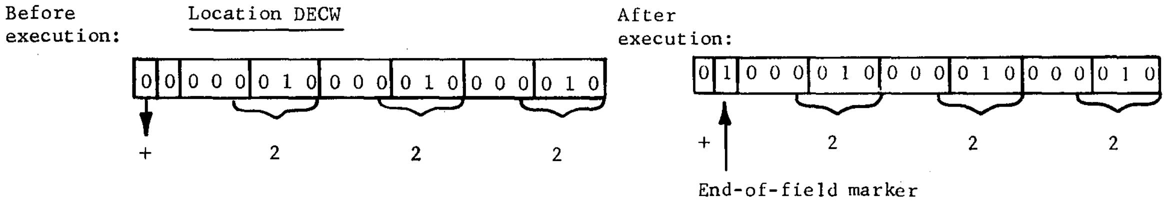


Memory Location Y+1



Prior to arithmetic operations the programmer should set the end-of-field markers he desires by coding which places a 1-bit into bit position 1 of the most-significant word of the numbers he wishes to mark. A typical method of accomplishing this bit insertion is shown on the following page.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1	MILL	OCT	1000000		
2		LDA	MILL		
3		ORY	DECW		
4					



The OCT 1000000 places the end-of-field marker constant in storage; LDA MILL and ORY DECW insert a 1-bit into bit position 1 of DECW.

The end-of-field marker affects the disposition of carries generated during arithmetic operations, since the word containing the marker is eventually loaded into the A-register for addition. A carry-out of the most significant digit position is remembered if the A-register does not contain an end-of-field marker prior to addition. The carry is remembered either until the next decimal arithmetic instruction is executed, the RESET MODES switch is depressed, or a SET BINMODE is executed. If the marker is present, a carry-out of the most significant digit position causes overflow, reversing the sign of the most significant word of the decimal number and turning on the overflow indicator (see later discussion of overflow in this chapter).

When it is desired to use end-of-field markers, a marker is not essential for both quantities involved in the 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 marked and the field in the A-register is not, the carry continues to be remembered and causes the next addition to be incorrect. If both fields are marked, the effect is the same as if only the A-register were marked. A marker in the A-register field automatically generates an end-of-field marker for the result field.

NEGATIVE DECIMAL NUMBERS

Negative decimal numbers must be expressed in the 10's complement form before decimal operations. The 0's complement is formed automatically by subtracting the decimal number from a decimal zero (preceded by an end-of-field marker in bit position 1) while in the decimal mode. Thus, the decimal number -222222 would be converted to -777,778 before being used in arithmetic operations. Negative results of decimal operations are also in the 10's complement form.

One method of converting decimal quantities to 10's complement form is shown below:

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1	MILL	OCT	1000000		
2		OCT	0000000		
3		DLD	MILL		
4		DSU	NEGD		
5		DST	COMP		
6					

	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

(End-of-field marker not shown.)

PROGRAMMING DECIMAL OPERATIONS WITH VARIABLE LENGTH FIELDS

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1	ABLE	BSS	4		
2	BAKER	BSS	4		
3	CHAS	BSS	4		
4	⌇	⌇	⌇		
18		SET	DECMØDE		
19		DLD	ABLE + 2		
20		DAD	BAKER + 2		
21		DST	CHAS + 2		
22		DLD	ABLE		
23		DAD	BAKER		
24		DST	CHAS		
25		BØV	ØVFLØ		

The foregoing example illustrates General Assembly Program coding for decimal arithmetic operations with fields of varying length. It is assumed that the programmer knows that the maximum length of the fields he will encounter will be 12 digits. It is also assumed that somewhere between steps 3 and 18 of the program coding he has loaded data in locations ABLE and BAKER and has set an end-of-field marker in the first word of location ABLE.

The numbers of the steps in the explanation of the coding correspond to the numbered steps on the coding sheet.

1. With this pseudo-instruction, a 4-word (12-decimal-digit) space is reserved in the object program for the number that will be placed in A during the addition.

2. Space is reserved for the second number to be used in the addition.

3. Space is reserved for storing the results of the addition.

18. The system is placed in the decimal mode.

19. The 6 least significant digits of ABLE are loaded into A and Q.

20. The 6 least significant digits of BAKER are added to the contents of A and Q.

21. The sum of the least significant digits of ABLE and BAKER is stored in the 6 least significant decimal positions of the 4-word location CHAS.

22, 23, and 24. These steps are similar to steps 19, 20, and 21, except that they deal with the 6 most significant digits. (The sum stored will automatically have the end-of-field marker set, since it has been set on the quantity in A.)

25. This instruction in the General Assembly Program language will cause a BOV followed by a BRU OVFLO to be entered in the object program. Since an end-of-field marker has been placed in the first word of location ABLE, an addition operation that produces a 13-digit sum will cause an overflow condition and a branch to the overflow routine beginning at OVFLO. If the end-of-field marker had not been set, the carry from a 13-digit sum would be remembered; and the BOV would not cause a branch, since no overflow condition would occur.

DECIMAL MODE INSTRUCTIONS

Mode Control Instructions

SET DECIMAL MODE		
SET DECMODE	2506011	2

Causes the subsequent arithmetic instructions ADD, DAD, SUB, DSU, ADO, and SBO to be executed in the decimal mode. No other instructions are affected.

SET BINARY MODE

SET BINMODE	2506012	2

Causes the subsequent arithmetic instructions ADD, DAD, SUB, DSU, ADO, and SBO to be executed in the binary mode and resets the BCD remembered carry function. No other instructions are affected.

Decimal Arithmetic Instructions

In the representations of registers and memory locations accompanying the examples in this discussion, the contents are shown in BCD; it is assumed that end-of-field markers are present, though not shown. It is also assumed in all examples that the central processor is already operating in the decimal mode.

DECIMAL ADD

ADD	Y	X	0100000	2

The contents of Y (S, 4-7, 10-13, and 16-19) are algebraically added to the contents of A (S, 4-7, 10-13, and 16-19). The result is placed in A (S, 4-7, 10-13, and 16-19). Bit positions 2, 3, 8, 9, 14, and 15 of A are set to zero.

EX 1: Add the quantity +333 in symbolic location INCR to +444, which has been previously loaded into A.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		ADD	INCR		
2					

A	INCR
+ 4 4 4	+ 3 3 3
+ 7 7 7	+ 3 3 3

EX 2: Add the quantity -333 in symbolic location NEGN to +444, which has been previously loaded into A.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		ADD	NEGN		
2					

A	NEGN
+ 4 4 4	- 6 6 7
+ 1 1 1	- 6 6 7

DECIMAL SUBTRACT

SUB	Y	X	0200000	2

The contents of Y (S, 4-7, 10-13, and 16-19) are algebraically subtracted from the contents of A (S, 4-7, 10-13, and 16-19). The result is placed in A (S, 4-7, 10-13, and 16-19).

EX 1: Subtract the quantity +333 in symbolic location DECR from +444, which has been previously loaded into A.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		SUB	DECR		
2					

A	DECR
+ 4 4 4	+ 3 3 3
+ 1 1 1	+ 3 3 3

EX 2: Subtract the quantity -333 in symbolic location NEGN from +444, which has been previously loaded into A. Assume that the 10's complement of -333 has already been formed.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		SUB	NEGN		
2					

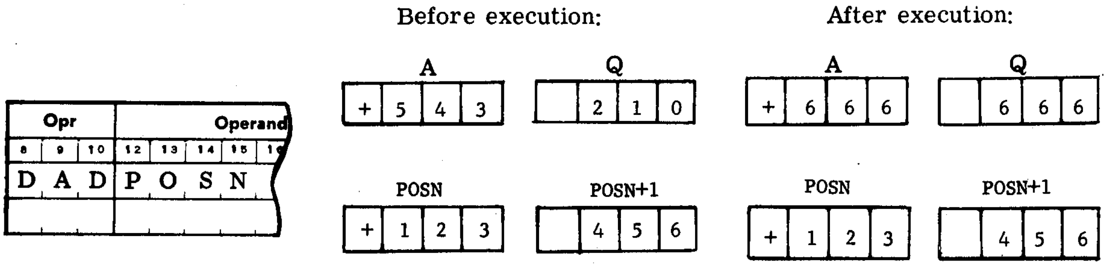
A	NEGN
+ 4 4 4	- 6 6 7
+ 7 7 7	- 6 6 7

DOUBLE DECIMAL ADD

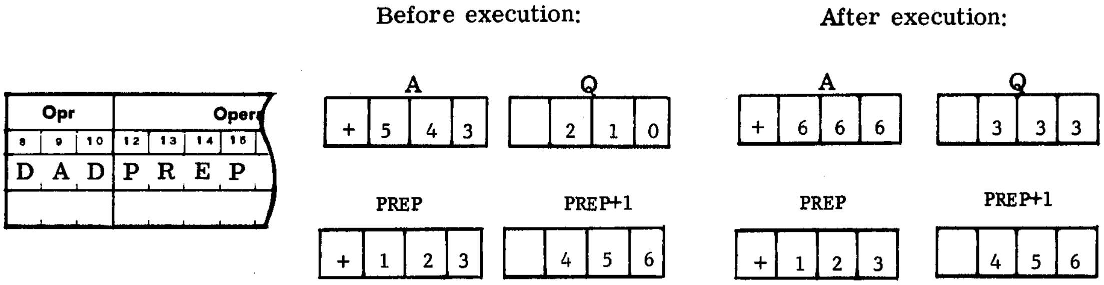
DAD	Y	X	1100000

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 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 A (S, 4-7, 10-13, and 16-19) and to Q (4-7, 10-13, and 16-19). The result is placed in A and Q.

EX 1: Add the quantity +123456 in symbolic locations POSN and POSN+1 to the quantity +543210, which has been previously loaded into A and Q. Assume that POSN is an even-numbered location.



EX 2: Add the quantity +123456 in symbolic locations PREP and PREP+1 to the quantity +543210, which has been previously loaded into A and Q. Assume the PREP is an odd-numbered location.



DOUBLE DECIMAL SUBTRACT

DSU Y X 1200000 3

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 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 A (S, 4-7, 10-13, and 16-19) and from Q (4-7, 10-13, and 16-19). The result is placed in A and Q.

EX 1: Subtract the quantity +123456 in symbolic locations DECR and DECR+1 from the quantity +543210, which has been previously loaded into A and Q. Assume that DECR is an even-numbered location.

Opr			Oper			
8	9	10	12	13	14	15
D	S	U	D	E	C	R

Before execution:

A				Q			
+	5	4	3		2	1	0

DECR				DECR+1			
+	1	2	3		4	5	6

After execution:

A				Q			
+	4	1	9		7	5	4

DECR				DECR+1			
+	1	2	3		4	5	6

EX 2: Subtract the quantity +123456 in symbolic locations NEGR and NEGR+1 from the quantity +543210, which has been previously loaded into A and Q. Assume that NEGR is an odd-numbered location.

Opr			Oper			
8	9	10	12	13	14	15
D	S	U	N	E	G	R

Before execution:

A				Q			
+	5	4	3		2	1	0

NEGR				NEGR+1			
+	1	2	3		4	5	6

After execution:

A				Q			
+	4	2	0		0	8	7

NEGR				NEGR+1			
+	1	2	3		4	5	6

ADD ONE DECIMAL

ADO 2504032 2

A positive 1 is algebraically added to the contents of the A register (4-7, 10-13, and 16-19). If the capacity of A is exceeded, overflow occurs. The BCD remembered carry function is inoperative with this instruction. This instruction operates properly only on decimal quantities of three digits or less.

EX: Add a +1 to the quantity +832 in A.

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

A			
+	8	3	2
+	8	3	3

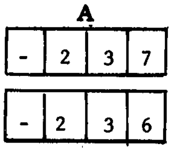
SUBTRACT ONE DECIMAL

SBO 2504112 2

A positive 1 is subtracted algebraically from the contents of A (4 - 7, 10 - 13, and 16 - 19). If the capacity of A is exceeded, overflow occurs. The BCD remembered carry function is inoperative with this instruction. This instruction operates properly only on decimal quantities of three digits or less.

EX: Subtract a +1 from the quantity -763 in A. Assume that the 10's complement of -763 has already been formed.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		S B O			
2					



SAMPLE LISTING OF DECIMAL OPERATIONS

The General Assembly Program listing below illustrates the fundamentals of performing arithmetic operations in the decimal mode:

```
          01751          ORG   1001
01751   1000000     MILL OCT   1000000
01752   0020202       A1 ALF   222
01753   0020202          ALF   222
01754   0040404       B1 ALF   444
01755   0040404          ALF   444
01756   0001751    START LDA   MILL
01757   2301752          ORY   A1
01760   2506011          SET   DECMODE
01761   1001752          DLD   A1
01762   1101754          DAD   B1
01763   1301604          DST   0900
01764   2506012          SET   BINMODE
```

1. Location 01750 contains the end-of-field marker to be inserted in the two BCD numbers before addition.

2. Locations 01756 and 01757 contain the instructions for inserting end-of-field markers in the BCD numbers.

3. The instruction in location 01761 converts the central processor to decimal mode prior to the addition.

4. The instruction in location 01765 restores the central processor to binary mode.

The memory printout below shows that locations 01752 and 01754 contain end-of-field markers in the words containing the most significant digits. Location 01604 contains the sum, which also is automatically marked.

Memory Printout

```
                                          01604 & 01605
                             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

                                          01752             01754
                                            &                 &
                                          01753             01755
```

Overflow

The overflow indicator on the operator's console will light whenever:

1. The numerical capacity of the A-register is exceeded.
2. An attempt is made to negate the largest negative number.
3. A 1-bit is shifted out of the A-register by a shift-left operation.

The first two types of overflow are discussed below, as they are the types that occur during arithmetic operations. The third type is covered in the appropriate place in this chapter in the discussion of shift operations.

Arithmetic overflow may be illustrated by considering two examples that involve operations with the largest positive number and the largest negative number that the A-register can hold. The largest positive number is the binary equivalent of $+524,287_{10}$, as shown below:

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	$+524,287_{10}$

Adding any positive number except zero to this quantity will result in a number that cannot be represented by 19 binary digits and will thus cause a 1-bit to appear in bit position 0 (the sign bit). This condition, in which the numerically significant capacity of the A-register is exceeded, is called arithmetic overflow.

Overflow (or "underflow," as it is sometimes called for negative numbers) also occurs if the capacity of the A-register is exceeded in a negative direction. The largest negative number that the A-register can hold is $-542,288_{10}$; it is shown below as it would appear when properly loaded into memory as the 2's complement of $+524,288_{10}$.

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	$-524,288_{10}$

Though this quantity may also be read as "negative zero," it is never so considered by the central processor.

Adding any negative number to the quantity above will result in a negative number that cannot be represented by 20 binary digits. Thus, a 0-bit will appear in the sign bit position, causing overflow. (Also, as already mentioned at the beginning of this discussion, an attempt to negate this quantity with a NEG instruction will cause overflow.)

The two foregoing examples illustrate the conditions for arithmetic overflow: the positive or negative capacity of the A-register is exceeded; and the overflow causes a change in the sign bit, resulting in an overflow indication on the console. The specific conditions for overflow during each type of arithmetic operation are discussed and illustrated in the following paragraphs.

ADDITION OVERFLOW

The overflow indicator will light whenever the sum of two positive numbers exceeds the numerical capacity of the A-register and changes the sign bit, as already shown by the first example, above. It also will light whenever the sum of two negative numbers reverses the sign bit. No overflow indication is possible when two numbers of unlike signs are added, since their sum is always less than the larger of the two numbers and cannot exceed the capacity of the A-register. The single bit that overflows as the result of addition operations is not lost and may be recovered by a programmed routine.

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

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		A D D	A M T # 1		
2					

A

1777674

2146522

EX 2: Add the contents of symbolic location AMT#2 (2000000) to -1, which has previously been loaded into A.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		A D D	A M T # 2		
2					

A

3777777

1777777

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. Thus, both are examples of overflow.

SUBTRACTION OVERFLOW

To perform subtraction the central processor adds the 2's complement of the subtrahend to the contents of the A-register. Therefore the rule for addition overflow applies equally to subtraction overflow: whenever the operation results in a carry, which changes the value of the sign bit, the overflow indicator will light. Again, as with addition overflow, the programmer may write a routine to recover the single bit that overflows.

EX: Subtract the negative number in symbolic location AMT#3 (3600163) from the positive number 1777674, which has previously been loaded into the A-register.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		SUB	AMT#3		
2					

A
1777674
2177511

MULTIPLICATION OVERFLOW

Overflow indication occurs in multiplication when there is an attempt to multiply the maximum negative number by the maximum negative number.

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

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		MPY	AMT#7		
2					

A	Q
0000000	2000000
2000000	2000000

It is possible to cause overflow when the MPY instruction is used to evaluate expressions of the form AB+C (that is, when the A-register is not set to zero before multiplication). In this type of operation overflow will occur only when multiplying the maximum negative number by the maximum with a positive number in A. If the number in A is negative, overflow is impossible.

DIVISION OVERFLOW

For proper division the magnitude of the divisor must be greater than the magnitude of that portion of the dividend appearing in A. Otherwise, the quotient appearing in A as a result of the operation is unreliable. A test is made by the central processor for the proper conditions before the division is executed. If the test shows an improper condition, the overflow indicator is turned on; and control is transferred to the next instruction in sequence.

EX: Divide the positive number 0100457 0312711, which has been previously double-loaded into A and Q, by 0047040 in symbolic location WRDS.

Opr	Operand	X	
8 9 10	12 13 14 15 16 17 18 19	20	31
D V D	W R D S		

A	Q	WRDS
0100457	0312711	0047040
?	?	0047040

Scaling

Before the binary equivalents of decimal numbers can be correctly added or subtracted in the central processor, the number of places to the right of the decimal point in both numbers must be the same. For example, to add 3.0 to 4.16, the binary equivalent of the first number must first be arranged to correspond to 3.00. Moving the decimal point either right or left to produce this alignment is called "scaling".

As a further example, suppose it is desired to add the following two decimal numbers:

$$\begin{array}{rr} & 24.4 \\ + & \underline{13.25} \\ & 37.65 \end{array} \quad \text{(desired sum)}$$

These quantities would normally appear in memory without decimal points--that is, as the binary equivalents of $+244_{10}$ and $+1325_{10}$:

0 1	2 3 4	5 6 7	8 9 10	11 12 13	14 15 16	17 18 19	
0 0	0 0 0	0 0 0	0 0 0	0 1 1	1 1 0	1 0 0	244
0 0	0 0 0	0 0 0	0 1 0	1 0 0	1 0 1	1 0 1	1325
0 0	0 0 0	0 0 0	0 1 1	0 0 0	1 0 0	0 0 1	1569

Without scaling, adding them would result in the binary equivalent of 1569_{10}, as shown above. To obtain the correct sum it would be necessary to scale the quantity 244 the equivalent of one decimal place to the left to attain the correct alignment of the digits:

$$\begin{array}{r} 2440 \\ \underline{1325} \end{array}$$

assumed decimal point

Scaling 244 one decimal position left is the same as multiplying it by 10_{10}. The binary equivalent of this multiplication may be performed as follows much more rapidly than by using the MPY instruction (assuming 244_{10} to be in symbolic location ARG1):

	Symbol 1 2 3 4 5 6	Opr 8 9 10	Operand 12 13 14 15 16 17 18 19	X 20	31
1		LDA	ARG1		
2		SLA	2		
3		ADD	ARG1		
4		SLA	1		

Step 1: Loads A with binary equivalent of 244.

Step 2: Shifts contents of A two places left (equivalent to 4 x 244, since each binary place represents a power of 2).

Step 3: Adds 244 to the contents of A: 244+(4 x 244)=(5 x 244).

Step 4: Shifts contents of A one place left (that is, multiplies contents by 2): 2(5 x 244)= 10 x 244.

With the quantity 244 properly scaled to 2440, the binary addition will then produce the correct sum:

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	2440
0 0	0 0 0	0 0 0	0 1 0	1 0 0	1 0 1	1 0 1	1325
0 0	0 0 0	0 0 0	1 1 1	0 1 0	1 1 0	1 0 1	3765

Whenever possible, the most efficient procedure is to scale constants while coding the source program for the General Assembly Program (see the discussion of the DEC pseudo-instruction in Chapter VII):

1. Constants may be entered using the minimum possible binary scale:

	Symbol 1 2 3 4 5 6	Opr 8 9 10	Operand 12 13 14 15 16 17 18 19	X 20	31
1		DEC	24.4B5		
2		DEC	13.25B5		
3					

2. Constants may be scaled decimally:

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		DEC	2440		
2		DEC	1325		
3					

(Binary scale of 19 is assumed by the General Assembly Program unless otherwise indicated by the coding.)

Data used as input to the object program can be scaled in a similar manner with the conversion and scaling subroutines available from the GE-235 Program Library. The desired scales are given in the calling sequences, providing the programmer with a convenient means for handling scaled-number arithmetic.

Data derived during processing that is to be used for subsequent calculations may be scaled by multiplying or dividing by the appropriate power of 10, in the manner already demonstrated.

Rounding Off

After a calculation has been completed, it is sometimes necessary to "round off" the results to a less-precise whole or mixed number having a specified number of places to the right or left of the decimal point. In decimal notation a number is rounded off by adding 5 to the decimal position to the right of the rounding-off point, performing any carries, and then discarding any digits to the right of the rounding-off point. Two examples are shown below:

```
456.                    45.312
  5.                      .005
461. = 460              45.317 = 45.31
```

The quantity 456, when rounded off to the nearest-ten value, thus becomes 460. The quantity 45.312, when rounded off to the nearest hundredth, becomes 45.31. Note that, after rounding off a whole number, the appropriate number of zeros must be added to preserve the original position of the remaining digits relative to the decimal point.

Since calculations within the central processor are primarily performed with binary numbers, the binary equivalent of the rounding-off factor (5) is entered in memory as a constant. It is then added at the appropriate point to any number to be rounded off, and the excess 1-bits to the right of the rounding-off point can be deleted through dividing by a power of 10.

To illustrate rounding-off for central processor calculations, assume that the decimal quantity 10.76 is to be rounded off to the nearest tenth. By scaling the stored rounding-off factor to the proper position to represent 0.05 relative to 10.76 and adding, the desired result is obtained, as shown below:

0 1	2 3 4	5 6 7	8 9 10	11 12 13	14 15 16	17 18 19	
0 0	0 0 0	0 0 0	0 1 0	0 0 0	1 1 0	1 0 0	10.76_{10}
0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	1 0 1	$+ 0.05_{10}$
0 0	0 0 0	0 0 0	0 1 0	0 0 0	1 1 1	0 0 1	10.81_{10}

The rounding is then completed by dividing by 10:

0 0	0 0 0	0 0 0	0 0 0	0 0 1	1 0 1	1 0 0	=	10.8_{10}

DATA TRANSFER

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

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 when otherwise noted.

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

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

Memory Transfers

LOAD A				
LDA	Y	X	0000000	2

The contents of Y replace the contents of A. Y is unchanged.

EX 1: Load A with the contents of symbolic location AMT#1, which contains the positive number 0146626. The A-register initially contains zeros.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		L D A	A M T # 1		
2					

A
0000000
0146626

EX 2: Load A with the contents of symbolic location AMT#5, which contains the negative number 3655463. The A-register initially contains 0122315.

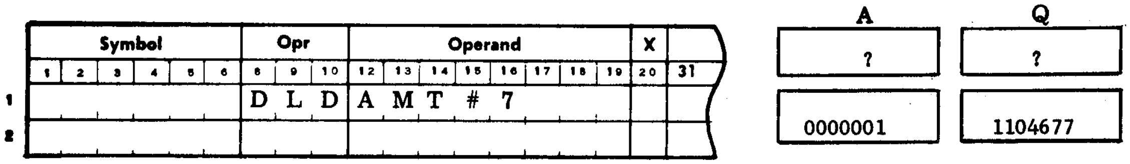
	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		L D A	A M T # 5		
2					

A
0122315
3655463

DOUBLE-LENGTH LOAD				
DLD	Y	X	1000000	3

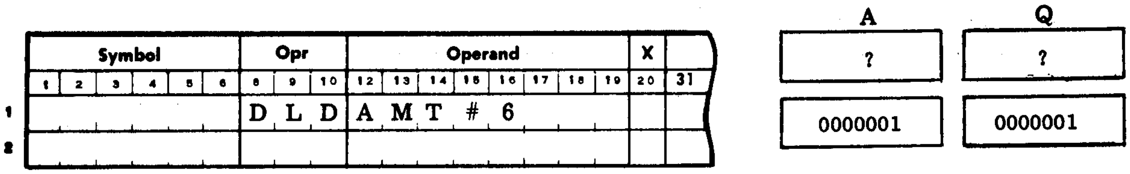
If the address of Y is even, the contents of Y and Y+1 replace the contents of A and Q. If the address of Y is odd, the contents of Y replace the contents of A and Q. Y and Y+1 are unchanged.

EX 1: Load A and Q with the positive number 0000001 1104677 in symbolic locations AMT#7 (even) and AMT#7+1.



	Symbol	Opr	Operand	X	31
1		DLD	AMT#7		
2					

EX 2: Load A and Q with the positive number 0000001 0003734 in symbolic locations AMT#6 (odd) and AMT#6+1.



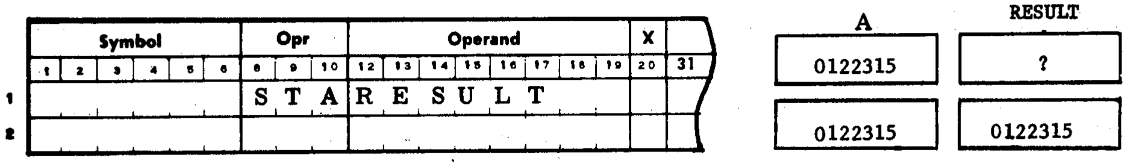
	Symbol	Opr	Operand	X	31
1		DLD	AMT#6		
2					

STORE A

STA Y X 0300000 2

The contents of A replace the contents of Y. The contents of A are unchanged.

EX 1: Store the A-register contents 0122315 in symbolic location RESULT.



	Symbol	Opr	Operand	X	31
1		STA	RESULT		
2					

EX 2: Store the A-register contents 3600163 in symbolic location OUTPUT. OUTPUT initially contains 3655463.

	Symbol	Opr	Operand	X	31
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		STA	OUTPUT		
2					

A	OUTPUT
3600163	3655463
3600163	3600163

DOUBLE-LENGTH STORE

DST Y X 1300000 3

If the address of Y is even, the contents of A and Q replace the contents of Y and Y+1. If the address of Y is odd, the contents of Q replace the contents of Y. The contents of A and Q are unchanged.

EX 1: Store A- and Q-register contents, 0000001 1104677, in symbolic locations AMT#8 (even) and AMT#8+1.

Opr	Operand
8 9 10	12 13 14 15 16 17 18 19
DST	AMT#8

A	Q	AMT#8	AMT#8+1
0000001	1104677	?	?
0000001	1104677	0000001	1104677

EX 2: Store A- and Q-register contents, 0000001 0003734, in symbolic locations AMT#7 (odd) and AMT#7+1.

Opr	Operand
8 9 10	12 13 14 15 16 17 18 19
DST	AMT#7

A	Q	AMT#7	AMT#7+1
0000001	0003734	?	?
0000001	0003734	0003734	?

STORE OPERAND ADDRESS

STO	Y	X	2700000	2

The contents of A (7-19) replace the contents of Y (7-19). A (S, 1-19) and Y (S, 1-6) are unchanged.

EX: Store the operand address that is in the A-register, 17777, in symbolic location TAX#1, which initially contains 0001667, an LDA instruction.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		S T O	T A X # 1		
2					

A	TAX#1
1117777	0001667
1117777	0017777

OR A INTO Y

ORY	Y	X	2300000	2

Corresponding bit positions of memory location Y are set with 1-bits for every bit position of A containing a 1-bit. The contents of A and other bit positions of Y remain unchanged.

EX 1: OR A into Y, with A containing 1641374 and Y (symbolic location $OUT) containing 0013711.

Before execution (binary):

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	1	0	1	1	1	1	1	0	0	1	0	0	1	$OUT
0	1	1	1	0	1	0	0	0	0	1	0	1	1	1	1	1	1	0	0	A

Opr	Operand	X
8 9 10	12 13 14 15 16 17 18 19	20
O R Y	$ O U T	

After execution:

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	1	0	1	1	1	1	1	1	1	1	1	0	1	$OUT
	↑	↑	↑		↑									↑	↑		↑			CHANGE
0	1	1	1	0	1	0	0	0	0	1	0	1	1	1	1	1	1	0	0	A

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

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		ORY	PRICE		
2					

A BCD			PRICE		
$	0	0	0	5	6
$	0	0	$	5	6

OR G INTO A

OGA	2500111	2

Corresponding bit positions of A (11-15) are set with 1-bits for every bit position in G (1-5) containing a 1-bit. The contents of G and A (S, 1-10, 16-19) are unchanged.

EX: Execute a subroutine beginning at location SUBR1, using X-group 22, and return to the main program and the original X-group. Assume that the contents of P have been stored (by an SPB instruction) in X-word 1 of the original X-group. The explanation following the coding, below, is numbered to correspond to the steps in the coding.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1	SXG0	SXG	0		
2	SUBR1	LDA	SXG0		
3		ØGA			
4		STA	EXIT		
5		SXG	22		
	EXIT	DEC	0		
		BRU	1	1	

1. Stored constant for use in the following step.

2. Beginning of the subroutine. The A-register is loaded with the constant from step 1.

3. The contents of the G-register (which contains the number of the current X-group) are set into the proper place in SXG instruction.

4. The SXG instruction is stored in location EXIT, where it will be executed at the end of the subroutine in order to return the program to the original X-group.

5. X-group 22 is selected for use during the subroutine.

The subroutine follows step 5. Then, the instruction placed in location EXIT restores the original X-group; and the BRU returns control to the main program at the instruction following the SPB that transferred control to location SUBR1.

EXTRACT

EXT	Y	X	2000000	2

For each 1-bit in Y a 0-bit is placed in the corresponding bit position of A. If bit positions in Y contain 0-bits, the corresponding bit positions in A are unchanged. Y is not affected.

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

Opr	Operand
8 9 10	12 13 14 15 16 17 18 19
E X T	M Ø D

Before execution (binary):

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 1 1	1 0 1	0 1 1	MOD
1 0	1 0 0	1 1 0	1 0 1	0 1 1	0 0 1	1 1 1	A

After 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 1 1	1 0 1	0 1 1	MOD CHANGE
		↓	↓	↓ ↓	↓	↓ ↓	
1 0	1 0 0	1 0 0	0 0 1	0 0 0	0 0 0	1 0 0	A

EX 2: Delete the dollar sign ($) from the BCD word $89 in A, prepáratory to storing the word into memory. Assume symbolic location STRIP contains the BCD word $00.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		E X T	S T R I P		
2					

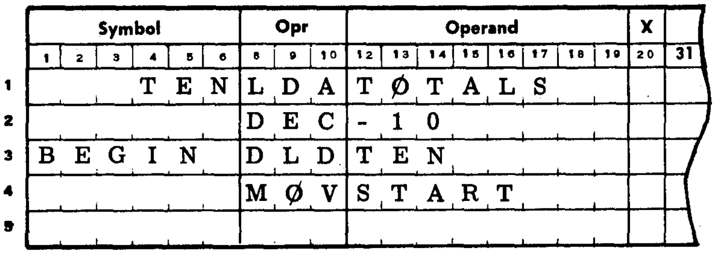
A BCD			STRIP		
$	8	9	$	0	0
0	8	9	$	0	0

MOVE (This instruction is an optional feature.)

MOV	Y	2400000	4+2N

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 Q must contain the 2's complement of the number of words to be moved. The contents of the P-counter are stored automatically in modification word 0 (bits 5 through 19). The time required to execute this command is 4+2N word times, where N is the number of words to be moved. After execution, A is set to 0's; and Q contains the 2's complement of the number of words moved. This instruction cannot be automatically modified.

EX: 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 the General Assembly Program has assigned the symbolic location START to actual address 00261 and TOTALS to actual address 02260. Assume that the number of words to be moved has previously been loaded into Q in 2's complement form.



	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1	TEN	LDA	TØTALS		
2		DEC	-10		
3	BEGIN	DLD	TEN		
4		MØV	START		
5					

Word times necessary to move this block of information:
4 + 2N = 4 + 2(10) = 24 word times.

Before execution:

Memory

Octal Address	Contents
00261	0123456
00262	0246531
00263	1234567
00264	0765432
00265	0135764
00266	2345670
00267	1001234
00270	0132456
00271	2147765
00272	1777777
02260	?
02261	?
02262	?
02263	?
02264	?
02265	?
02266	?
02267	?
02270	?
02271	?

Registers

A	Q
0002260	3777766

After execution:

Memory

Octal Address	Contents
00261	0123456
00262	0246531
00263	1234567
00264	0765432
00265	0135764
00266	2345670
00267	1001234
00270	0132456
00271	2147765
00272	1777777
02260	0123456
02261	0246531
02262	1234567
02263	0765432
02264	0135764
02265	2345670
02266	1001234
02267	0132456
02270	2147765
02271	1777777

Registers

A	Q
0000000	3777766

Arithmetic Register Transfers

LOAD A FROM Q

LAQ 2504001 2

The contents of Q replace the contents of A. Q is unchanged.

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

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

A	Q
1234567	3654321
3654321	3654321

No operand address is required. Automatic modification will change the nature of the instruction.

LOAD Q FROM A

LQA	2504004	2

The contents of A replace the contents of Q. A is unchanged.

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

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

A	Q
1117776	2465317
1117776	1117776

No operand address is required. Automatic modification will change the nature of the instruction.

MOVE A TO Q

MAQ	2504006	2

The contents of A replace the contents of Q. Zeros replace the contents of A.

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

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

A	Q
1333444	3777777
0000000	1333444

No operand address is required. Automatic modification will change the nature of the instruction.

EXCHANGE A AND Q

XAQ 2504005 2

The contents of A and Q are interchanged.

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

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

A	Q
1234567	1777777
1777777	1234567

No operand address is needed. Automatic modification will change the nature of the instruction.

LOAD A FROM C-REGISTER (This instruction is an optional feature.)

LAC 2504202 2

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

EX: Load A from C. Assume that C contains the binary equivalent of 1 hour ($21,600_{10}$ sixths of a second).

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

A	C
?	0052140
0052140	0052140

(See explanation of real time clock operation after next instruction, below.)

LOAD C-REGISTER FROM A (This instruction is an optional feature.)

LCA	2504210	2

The contents of the C-register are replaced by the contents of A (1-19). The sign of A is ignored. The contents of A are unchanged.

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

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

A	C
0772200	?
0772200	0772200

The C-register operates as a binary counter that is incremented by 1 every sixth of 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 A, makes the contents of C available to the stored program or to the console operator.

Subroutines are required for program translation of the C-register contents to hours, minutes, seconds, and sixths of a second and for conversion to BCD for print-out through the console typewriter.

A simple, straightline subroutine is shown below to illustrate how translation into hours, minutes, seconds, and sixth of a seconds could be done. In actual practice, a more sophisticated approach involving modification words and controlled looping would be more efficient.

EX: Convert the C-register contents 1205701 to hours, minutes, seconds, and sixth-seconds. Assume symbolic locations CON1 through CON3 contain conversion constants as follows:

Symbolic Location	General Assembly Program Coding	Contents	Remarks
CONl	DEC 21600	0052140	Hours factor
CON2	DEC 360	0000550	Minutes factor
CON3	DEC 6	0000006	Seconds factor

Opr	Operand	X	REMARKS
8 9 10	12 13 14 15 16 17 18 19	20	31
LAC			TRANSFER TIME TO A
MAQ			TRANSFER TIME TO Q FOR DVD
DVD	CON1		COMPUTE HOURS
STA	HOURS		
LDZ			CLEAR A
DVD	CON2		COMPUTE MINUTES
STA	MINS		
LDZ			
DVD	CON3		COMPUTE SECONDS
STA	SECS		
XAQ			TRANSFER SIXTH-SECONDS TO A-REG
STA	SXTHS		

Standard binary to BCD conversion routines could then be used on the binary contents of symbolic locations HRS, MINS, SECS, and SXTHS and the results printed out on the typewriter as decimal information.

Initial Register Contents:

C	A	Q
1205701	?	?

Registers Affected by Each Instruction:

Coding	Registers	
LAC	1205701	?
MAQ	0000000	1205701
DVD CON1	0000017	0015041
STA HOURS	0000017	0015041
LDZ	0000000	0015041
DVD CON2	0000022	0000321
STA MINS	0000022	0000321
LDZ	0000000	0000321
DVD CON3	0000042	0000005
STA SECS	0000042	0000005
XAQ	0000005	0000042
STA SXTHS	0000005	0000042

Memory Contents After Execution:

Symbolic Location	Contents
HOURS	0000017
MINS	0000022
SECS	0000042
SXTHS	0000005

Register Modification

LOAD ZERO INTO A

LDZ	2504002	2

The contents of A are replaced by zeros.

EX: Load zero into A, or replace the existing contents of A, 3777777, with zeros.

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

A

3777777

0000000

No operand address is needed. Automatic modification will change the nature of the instruction.

LOAD ONE INTO A

LDO	2504022	2

A 1-bit is placed in bit position 19 of A; all other bit positions (S, 1-18) are set to 0-bits.

EX: Load one (1) into A. Assume that A initially contains 3777777.

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

A

3777777

0000001

No operand address is needed. Automatic modification will change the nature of the instruction.

LOAD MINUS ONE INTO A

LMO 2504102 2

The contents of A are replaced by 1-bits, giving the octal configuration 3777777.

EX: Load minus one (-1) into A. Assume that A initially contains 1357642.

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

A
1357642
3777777

No operand address is needed. Automatic modification will change the nature of the instruction.

COMPLEMENT A

CPL 2504502 2

Each bit position in A is inverted; each 1-bit is replaced by a 0-bit and each 0-bit is replaced by a 1-bit, forming the 1's complement of the original number.

EX: Complement A. Assume that A contains 1234567.

Opr	Operand	X	
8 9 10	12 13 14 15 16 17 18 19	20	31
C P L			

A

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 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

No operand address is needed. Automatic modification will change the nature of the instruction.

NEGATE A

NEG 2504522 2

The 2's complement of the contents of A replaces the contents of A. An attempt to negate the maximum negative number causes overflow.

EX: Negate A-register contents 0000101.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		N E G			
2					

A

0000101

3777677

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 nature of the instruction.

CHANGE SIGN OF A

CHS 2504040 2

The sign bit of A is changed. Bit positions 1-19 of A are unchanged.

EX: Change the sign of A. Assume that A contains 1357642.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		C H S			
2					

A

1357642

3357642

No operand address is needed. Automatic modification will change the nature of the instruction.

NO OPERATION

NOP 2504012 2

Zero is added to the contents of the A-register.

EX: No operation, or add zero to the contents 1234567 of the A-register.

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

A
1234567
1234567

This instruction is useful in reserving space in a program for later insertion of an instruction. (A BRU*+1 will also serve this purpose and has the advantage of requiring only 1 word time for execution. A NOP, however, is more readily recognizable in program documentation.) No operand address is needed. Automatic modification will change the nature of the instruction.

SHIFTING

Serial (bit-by-bit) movement of data within a register or between registers is known as shifting. All shift instructions involve the A-register, and some also involve the N- or Q-register, or both.

Shifting is used to transfer data between the A-register and N-register peripherals, to scale quantities before and after arithmetic operations, to recover from overflow conditions, and to perform simple multiplication and division. Bit positions 15-19 in a shift instruction indicate the number of places to be shifted. Thus, no instruction can specify a shift of more than 31 bit positions, since this is the maximum number that can be expressed with five binary digits. The number of word times for execution of a shift instruction varies from 2 to 6, depending upon the length of the shift. A shift of 4 bit positions requires 2 word times; each additional shift of up to 3 positions takes another word time.

If a modification word is used to control the number of bits to be shifted, care should be taken always to keep 0-bits in positions 5-15 of the modification word. Otherwise automatic modification of a shift instruction may cause a change in the nature of the instruction.

Shift overflow from the A-register occurs when a 1-bit is shifted left out of bit position 1. The overflow bit is not shifted into bit position 0 but is shifted out of the register and lost. This type of overflow can be recovered through a programmed routine for restoring the lost bit by OR-ing it back into the word.

SHIFT RIGHT A

SRA	K	X	2510000	2-6

The contents of A (1-19) are shifted right K places, where K≤ 31. 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.

EX 1: Shift right 3 bit positions the positive number 1234567, previously loaded into A.

Opr	Operand	X	
8 9 10	12 13 14 15 16 17 18 19	20	31
S R A	3		

A

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

Octal

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

EX 2: Shift right 7 bit positions the negative number 3765432, previously loaded into A.

Opr	Operand	X	
8 9 10	12 13 14 15 16 17 18 19	20	31
S R A	7		

A

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

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

EX 3: Divide by 8 the positive number 1612350, previously loaded into A.

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

A
1612350
0161235

(Shifting right three places is equivalent to dividing the number by 2^3, or 8.)

SHIFT LEFT A

SLA K X 2512000 2-6

The contents of A (1-19) are shifted left K places, where K≤31. Vacated bit positions of A are filled with 0-bits. If a nonzero bit is shifted out of position 1, overflow occurs and the bit is lost. The sign of A is unchanged.

EX 1: Shift left 2 bit positions the positive number 1234567, previously loaded into A.

Opr	Operand	X	
8 9 10	12 13 14 15 16 17 18 19	20	31
S L A	2		

A

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

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
0		5	1	6	2	7	0

EX 2: Shift left 5 places the negative number 2036361, previously loaded into A.

Opr	Operand	X	
8 9 10	12 13 14 15 16 17 18 19	20	31
S L A	5		

A

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

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

EX 3: Multiply by 4 the positive number 0002470, previously loaded into A.

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

A
0002470
0012340

(Shifting left two places is equivalent to multiplying the number by 2^2, or 4.)

SHIFT RIGHT DOUBLE

SRD K X 2511000 2-6

The contents of A (1-19) and Q (1-19) together are shifted K places to the right, where K≤31. 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.

EX 1: Shift right double 2 octal positions the contents of A and Q. A contains 1234567; Q contains 3654321.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		S R D	6		
2					

A	Q
1234567	3654321
0012345	1576543

EX 2: Shift right double 2 bit positions the contents of A and Q.

Opr	Operand	X	
8 9 10	12 13 14 15 16 17 18 19	20	31
S R D	2		

A	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
Q	0 0	0 1 1	0 1 0	1 1 0	1 1 0	0 1 1	0 1 0

A	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
Q	0 0	1 0 0	1 1 0	1 0 1	1 0 1	1 0 0	1 1 0

SHIFT LEFT DOUBLE

SLD	K	X	2512200	2-6

The contents of A and Q (1-19) together are shifted K places to the left, where K≤31. Bits shifted out of Q (1) shift into A (19). The vacated positions of Q are filled with 0-bits. If a nonzero 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.

EX: Shift left double 4 bit positions the contents of A and Q.

Opr	Operand	X	
8 9 10	12 13 14 15 16 17 18 19	20	31
S L D	4		

A

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

Q

A

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

Q

SHIFT CIRCULAR A

SCA	K	X	2510040	2-6

The contents of A (1-19) are shifted right K places in a circular fashion, where K≤31; that is, bits shifted out of position 19 are inserted in position 1, replacing bits as they are shifted right out of position 1. The sign of A is unchanged.

EX: Shift circular A-register contents 8 bit positions.

Opr	Operand	X	
8 9 10	12 13 14 15 16 17 18 19	20	31
S C A	8		

A

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

SHIFT CIRCULAR DOUBLE

SCD . K X 2511200 2-6

The contents of A and Q (1-19) together are shifted K places to the right in a circular fashion, where K≤31. 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.

EX: Shift circular double 4 bit positions the contents of A and Q.

Opr	Operand	X	31
8 9 10	12 13 14 15 16 17 18 19	20	31
S C D	4		

A

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

Q

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

SHIFT A AND N RIGHT

SAN K X 2510400 2-6

The contents of A (1-19) and N (1-6) together are shifted K places to the right, where K≤31. Bits shifted out of A (19) shift into N (1). N must be ready before this instruction can be properly executed. (See BNN and BNR instructions.)

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.

EX: Shift A and N right 6 bit positions.

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

A	N
1460430	?
0014604	30

SHIFT N AND A RIGHT

SNA	K	X	2510100	2-6

The contents of N (1-6) and A (1-19) together are shifted K places to the right, where K≤31. 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. N must be ready before this instruction can be properly executed. (See BNN and BNR instructions.)

EX: Shift N and A right 6 bit positions.

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

A	N
0014604	30
1400146	0

SHIFT A INTO N AND Q

ANQ	K	X	2511400	2-6

The contents of A (1-19) are shifted K places to the right into both registers N and Q, where K≤31. 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. N must be ready before this instruction can be properly executed. (See BNN and BNR instructions.)

EX: Shift A into N and Q 6 bit positions. Assume Q has been zeroed.

A	Q	N
1460430	0000000	?
0014604	0600000	30

SHIFT N, A, AND Q RIGHT

NAQ	K	X	2511100	2-6

The contents of N (1-6), A (1-19), and Q (1-19) together are shifted K places to the right, where K≤31. 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. N must be ready before this instruction can be properly executed. (See BNN and BNR instructions.)

EX: Shift N, A, and Q right 6 bit positions. Assume Q has been zeroed.

A	Q	N
0014604	0000000	30
1400146	0100000	0

NORMALIZE A

NOR	K	X	2513000	3-7

The effect of this instruction depends upon the value of K, the sign of A, and R (number of leading zeros or 1-bits in A), where K≤31.

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 00000. Vacated positions of A are filled with 0-bits.

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 are shifted left K places; 0-bits replace the contents of memory location 00000 (15-19); bit positions S, 1-14 of 00000 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, R equals the number of leading 1-bits of A (1-19); all other conditions are the same as when the sign of A is plus. The sign of A is unchanged.

The overflow alarm indicator will not be affected by normalizing a negative number.

EX 1: Normalize A, which contains 0001234 (9 leading 0-bits) to 10 bit positions (K = 10, R = 9).

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		N O R	1 0		
2					

A	00000
0001234	00000??
1234000	0000001

EX 2: Normalize the A-register, which contains 0012345 (6 leading 0-bits) to 5 bit positions (K = 5, R = 6).

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		N O R	5		
2					

A	00000
0012345	00000??
0516240	0000000

EX 3: Normalize A, which contains the negative number 3776542 (9 leading 1-bits) to 6 bit positions (K = 6, R = 9).

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		N O R	6		
2					

A	00000
3776542	00000??
3654200	0000000

The NOR instruction is used primarily in normalizing A for interpretive programming of floating-point arithmetic operations. (See the separate AAU manual for details.)

NOR can be automatically modified; however, the length of a shift after modification must not exceed 31 places.

DOUBLE-LENGTH NORMALIZE

DNO K X 2513200 3-7

If the sign of A is plus, and the number of leading 0-bits (R) of A (1-19) is less than the constant (K), then the contents of A (1-19) and Q (1-19) are shifted left R places. The difference K-R replaces the contents of location 00000 (15-19).

If R is greater than or equal to K, then the contents of A (1-19) and Q (1-19) are shifted left K places; 0-bits replace the contents of memory location 00000 (15-19). Bit positions S, 1-14 of location 00000 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, R equals the number of leading 1-bits of A (1-19); all other conditions are the same as when the sign of A is plus. The sign of A is unchanged.

The overflow alarm indicator will not be affected by double-length normalizing a negative number.

EX 1: Double-length normalize A and Q, which contain 0001234 0076543 (8 leading 0-bits) to 6 bit positions (K = 6, R = 9).

Opr			Operand								X
8	9	10	12	13	14	15	16	17	18	19	20
D	N	O	6								

A	Q	A	Q
0001234	0076543	0123403	1654300
00000		00000	
00000??		0000000	

EX 2: Double-length normalize A and Q, which contain 0001777 0000177 (9 leading 0-bits) to 15 bit positions (K = 15, R = 9).

Opr			Operand								X
8	9	10	12	13	14	15	16	17	18	19	20
D	N	O	1	5							

A	Q	A	Q
0001777	0000177	1777000	0177000
00000		00000	
00000??		0000006	

INTERNAL BRANCHING

Branch instructions, which provide decision-making capability, fall into two categories: internal branch instructions (described in this section) and input/output branch instructions (described in the separate peripheral subsystem reference manuals). The internal branch instructions are further subdivided into two groups -- unconditional branch instructions and test-and-branch instructions -- for the following discussion.

Unconditional Branching

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 symbolic addresses, when coded for the General Assembly Program.

BRANCH UNCONDITIONALLY

BRU Y X 2600000 1

Control is transferred to the instruction at Y by setting the address of Y into P. If this instruction is modified automatically, all 15 bits of P are altered by the sum of bits 7-19 of Y and bits 5-19 of the specified X-word. If there is no modification, then only 13 bits of P are altered.

EX: Branch unconditionally to the symbolic location STORE. Assume that STORE has been assigned the address 01766 and that the BRU instruction is located at 00460.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1	A S K 4 6 0	B R U	S T O R E		
2					

P

00461
01766

Note that, before execution, P has already been stepped to the address of the next sequential instruction. BRU modifies P to transfer control to the instruction in location 01766.

STORE P AND BRANCH

SPB Y X 0700000 2

The address of the SPB instruction (held in bits 5-19 of P) replaces the contents of bit positions 5-19 of the specified modification word. Bits 0-4 of the modification word are automatically set to zero. Control is then transferred to the instruction held in Y. That is, the contents of P are replaced by Y.

EX: Store P and branch. Store the location of the SPB instruction, 02676, in X-word 3 and branch to the instruction held in symbolic location RERUN, assigned to 00500.

	Symbol	Opr	Operand	X	31
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1	OUT267	SPB	RERUN	3	
2					

P	X-3
02676	???????
00500	0002676

SPB cannot be automatically modified, because bit positions 5 and 6 are used to specify the X-word to receive the SPB memory location.

Test and Branch

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 (the condition exists), the central processor executes the next sequential instruction; if the test is false (the condition does not exist), the central processor skips the next instruction and executes the second sequential instruction.

EX:

	Symbol	Opr	Operand	X	31
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		BOV			
2		BRU	OVFLOW		
3		BRU	CONTIN		
4					

The BOV (Branch on Overflow) instruction causes a check to be made for an overflow condition. If the overflow indicator is on (the tested-for condition exists), the next sequential instruction is executed; and the program branches to the symbolic address OVFLOW. If the overflow indicator is not on (the tested-for condition does not exist), the program skips the BRU to OVFLOW and executes the second sequential instruction, which branches to CONTIN.

Registers are unchanged by testing; tested indicators may or may not change, depending upon the type of 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 nonbranch instructions; if the condition tested is false, the P counter is increased by two, thereby skipping an instruction.

In absolute coding a test-and-branch instruction has no operand address. Therefore, it must be followed by a BRU instruction specifying the transfer address. However, in General Assembly Program coding an operand may be specified in the test-and-branch instruction, so that, for example, BOV OVFLO will produce the same results at a BOV followed by a BRU OVFLO.

For convenience, relative and symbolic addressing is also permitted with test-and-branch instructions as illustrated in the examples following three of the instruction descriptions. An example using several of the test-and-branch instructions together is also included.

The test-and-branch instructions BXL and BXH are in the following section, "Using Address Modification Words," since they apply specifically to X-words.

BRANCH ON OVERFLOW

BOV	2514003	2

The OVERFLOW indicator is tested for the on condition. If on, it is automatically turned off.

BRANCH ON NO OVERFLOW

BNO	2516003	2

The OVERFLOW indicator is tested for the off condition. If overflow has occurred, the indicator is automatically turned off.

BRANCH ON PLUS

BPL	2516101	2

The A-register is tested for a plus sign (0) in the sign-bit position.

BRANCH ON MINUS

BMI	2514001	2

The A-register is tested for a minus sign in the sign-bit position.

BRANCH ON ODD

BOD	2504000	2

The A-register is tested for an odd value (bit position 19 in A contains a 1-bit for all odd values).

BRANCH ON EVEN

BEV	2516000	2

The A-register is tested for an even value (bit position 19 in A contains a 0-bit for all even values).

BRANCH ON ZERO

BZE	2514002	2

The A-register contents are tested for 0-bits in all positions.

BRANCH ON NONZERO

BNZ	2516002	2

The A-register contents are tested for 1-bits in any positions.

BRANCH ON PARITY ERROR

BPE	2514004	2

The PARITY alarm indicators are tested for the on condition. If any parity error has occurred, the lighted indicator is automatically turned off.

If the console STOP ON PARITY ALARM/NORM switch is in the STOP ON PARITY ALARM position and a memory parity error occurs, the MEM PARITY alarm indicator turns on, and the central processor halts. If the switch is in the NORM position, a memory parity error turns on the MEM PARITY alarm indicator but processing continues. This permits programmed interrogation of the MEM PARITY alarm indicator with a BPE or BPC (see below) instruction and optional branching to a corrective routine instead of a program halt.

If either an I/O or N-register parity error occurs, the appropriated half of the split indicator turns on; but the processor does not halt. These indicators may be tested and reset regardless of the STOP ON PARITY ALARM/NORM switch position.

BRANCH ON PARITY CORRECT

BPC	2516004	2

The PARITY alarm indicators are tested for the off condition. If all PARITY alarm indicators are off, the condition tested for is true. If any parity error has occurred the lighted PARITY alarm indicator is turned off automatically. (See BPE, above.)

EX: Store all even positive numbers (not including zero) in symbolic location RESULT; branch to locations not in the main sequence for negative numbers, zeros, and odd positive numbers. This example illustrates the use of several test-and-branch instructions together. The comments following the coding are numbered to correspond to the numbers of the steps of the coding. Assume that the quantity to be tested has previously been loaded into A.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1	TEST	BMI			
2		BRU	MINUS		
3		BZE			
4		BRU	ZERØ		
5		BØD			
6		BRU	ØDD		
7		STA	RESULT		
8					

1. If the number in A is negative, the next sequential instruction is executed, as in normal processing. If the number in A is positive, P is incremented by 1, and the instruction in step 3 is executed next.

2. Unconditional branch to location MINUS, if the number in A is negative. MINUS is the address of a location not in the main sequence at this point. It may be the beginning of a subroutine or an earlier or later instruction in the main sequence.

3. The positive number in A is tested to determine whether or not it is zero, since the sign of zero is + (a 0-bit) and it is considered a positive number by test-and-branch instructions. (These instructions also consider zero to be an even number and do not differentiate between it and other even numbers.) If the number in A is zero, step 4 will be executed next; if the number is not zero, step 5 will be executed next.

4. Unconditional branch to location ZERO.

5. The positive nonzero number in A is tested to determine whether it is odd. If the number is odd, step 6 will be executed next; if the number is even, step 7 will be executed next.

6. Unconditional branch to location ODD.

7. A positive nonzero even number is stored in location RESULT.

COMPARE AND BRANCH (This instruction is an optional feature.)

CAB	Y	X	2100000	2-3

The contents of A are compared algebraically with the contents of 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. (See comment after DCB description, below.)

EX: Compare the contents of symbolic location TEST with the contents of A. If TEST is greater than A, go to MORE for the next instruction. If TEST equals A, go to EQUALS. If TEST is less than A, go to LESS. Assume CAB is in 00123.

Symbol	Opr	Operand	X	
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
ANS	CAB	TEST		
	BRU	MORE		
	BRU	EQUALS		
LESS	ADD	34		

	P		
Before execution:	00123	=	ANS
After execution:			
Y > A	00124	=	ANS+1
Y = A	00125	=	ANS+2
Y < A	00126	=	LESS

DOUBLE COMPARE AND BRANCH (This instruction is an optional feature.)

DCB	Y	X	2200000	2-5

The contents of A and Q are compared algebraically with the contents of 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 next instruction is skipped and the second sequential instruction is executed. If the contents of Y and Y+1 are less than the contents of A and Q, the next two instructions are skipped and the third sequential instruction is executed. 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.

Both the DCB and the CAB instructions provide a "three-way compare" capability. CAB provides single-length word comparisons, while DCB compares double-length words. In both instructions the effect on P is similar:

Y (or Y and Y+1)>A (or A and Q)	P unchanged
Y (or Y and Y+1) = A (or A and Q)	Step P+1
Y (or Y and Y+1)<A (or A and Q)	Step P+2.

The next two instructions in sequence are skipped, and the third sequential instruction is executed next. That is, P is incremented by 3 instead of the normal 1. (With a GE-225 system, this instruction is equivalent to NOP, and the following instruction is executed. With a GE-215 system, a WAI causes the second sequential instruction to be executed next.)

EX 1: A program that may be used with all GE-Compatibles/200 systems requires an additional routine when used with either a GE-225 or 215 system.

	Symbol	Opr	Operand	X	31
1		WAI			
2		BRU	225SYS		
3		BRU	215SYS		
4	UTAB	LDA	EXL#1		
5					

Execution of the WAI allows branching to the special routines for the GE-225 and 215. If a GE-235 system is used, the program continues immediately from location UTAB.

EX 2: A program that may be used with all Compatibles/200 systems requires a different constant at location TCONST for each of the three systems. Assume location ZERO contains 0000000. The explanation following the coding, below, is numbered to correspond to the steps in the program.

	Symbol	Opr	Operand	X	31
1		LDX	ZERØ	1	
2		WAI			
3		INX	1	1	
4		INX	1	1	
5		LDA	TABLE	1	
6		STA	TCØNST		
	TCØNST	DEC	0		
	TABLE	DEC	45000		
			8000		
			15000		

1. Current X-word 1 is zeroed.

2. The program branches to step 3, 4, or 5, depending on the system in use.

3. If the system is a GE-225, this instruction is executed. X-word 1 then contains 0000001.

4. (a) If the system is a GE-225, this step is executed after step 3; and X-word 1 then contains 0000002. (b) If the system is a GE-215, this step is executed after step 2; and X-word 1 then contains 0000001.

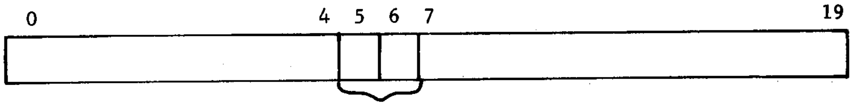
5. (a) If the system is a GE-225, this step is executed after step 4; and A is loaded with the contents of location TABLE + 2. (b) If the system is a GE-215, this step is executed after step 4; and A is loaded with the contents of location TABLE + 1. (c) If the system is a GE-235, this step is executed immediately after step 2; and A is loaded with the contents of location TABLE.

6. The proper constant to match the system is used and is stored in location TCONST for later reference. This and the preceding steps would be executed only once, at the beginning of the program.

USING ADDRESS MODIFICATION WORDS

Memory locations 00000 through 00003 have special properties and, together, are known as address modification word group 0. They may be used individually as program counters by making provision for incrementing their contents by a constant (with the INX instruction) and testing their contents with either of two special instructions (BXL or BXH).

In addition, locations 00001 through 00003 can be used as address modification words. The symbol X is frequently used in this manual when referring to these locations as address modification words. When applicable, bit positions 5 and 6 of an instruction word specify which X-word is being used for address modification, as shown below:



00 no address modification
01 modification word 00001
10 modification word 00002
11 modification word 00003

If an instruction containing an operand address also specifies an X location in bit positions 5 and 6, the contents of the X-word are added by the central processor to the operand address to give the effective (modified) address. If the instruction is then executed, it will operate on the data stored in the location at the effective address. The original instruction in storage is unchanged. The code 00 is effective in addressing location 00000 only in the X-word instructions (INX, BXL, BXH, LDX, and STX) included in this section; as indicated in the diagram above, it does not function in address modification.

Address modification words are also used in addressing the upper bank of memory in a system with a 16k memory. (See the following section in this chapter, "Programming 16K Memory Systems.")

Thirty-one additional address modification word groups, in locations 00004-00171 (0004_{10}-0127_{10}), are available as part of an optional package. The first location in each group may be used in the same manner as location 00000, and the three following locations have the same properties as locations 00001-00003.

Any one of the groups (0-31) may be selected by using the SXG (SELECT X-GROUP) instruction. All address modification operations are then addressed to this group until another SXG instruction is executed. (The number of group presently in use is displayed on the operator's console.) Once a group is selected, its members are addressed by the same bit configurations in positions 5 and 6 of an instruction word as are used with address modification word group 0. For example, in group 1 (locations 00004-00007) location 00006 is the second modification word and is addressed by placing a 1-bit and a 0-bit in positions 5 and 6, respectively.

None of the address modification instructions whose descriptions follow can themselves be automatically modified. Since they concern operations that deal directly with address modification words, bits 5 and 6 are used to specify the X-word involved in the operation and are unavailable for indicating modification.

INCREMENT X

INX	K	X	1400000	3

This instruction adds the number K (in bit positions 7-19 of the I-register) to the contents of the specified X-word (5-19). The result replaces the contents of X (5-19); any carry from bit position 5 is dropped.

EX 1: Increment X-word 00002, which contains 0001000, by 1.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		I N X	1	2	
2					

X-2

0001000

0001001

The INX instruction should not be used to decrement an X-word used as a modification word (as in INX -2, 1). This operation brings about the desired result in bit positions 7-19; but it generates a carry into bit position 6. The carry will affect a later use of the word for address modification, as this operation involves bits 5-19.

The INX instruction should also not be used to zero a modification word, as the required increment will also cause a carry into bit position 6. The best procedure for zeroing a modification word is to use the LDX instruction, addressing a memory location Y known to be loaded with zeros.

BRANCH IF X IS HIGHER THAN OR EQUAL TO

BXH	K	X	0500000	2

If the contents of X (7-19) are greater than or equal to the constant K, where $K \leq 8191_{10}$ (in absolute coding, K must be in 2's complement form), the next sequential instruction is executed; if less than K, the second sequential instruction is executed. X is unchanged. (Though this is a test-and-branch instruction, it has been included here because of its special application.)

EX 1: Branch if X is higher than or equal to 4. Assume that X-word 2 which contains 0000006, is to be used. Assume the BXH is in actual memory location 00163.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		BXH	4	2	
2	FICA	BRU	0777		
3		STA	TEMP		
4					
5					

P-Counter Contents in Octal and Symbolic

00163	=	FICA-1
00164	=	FICA

EX 2: Branch if X is higher than or equal to 4. Assume that X-word 2, which now contains 0000003, is to be used. Assume that BXH is in actual memory location 00163.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		BXH	4	2	
2	FICA	BRU	0777		
3		STA	TEMP		
4					
5					
6					

P-Counter Contents in Octal and Symbolic

00163	=	FICA-1
00165	=	FICA+1

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 to select the desired group.

BRANCH IF X IS LESS THAN

BXL	K	X	0400000	2

If the contents of X (7 - 19) are less than the constant K, where K 8101$_{10}$ (in absolute coding, K must be in 2's complement form), the next sequential instruction is executed; if greater than or equal to K, the second sequential instruction is executed. X is unchanged. (Though this is a test-and-branch instruction it has been included here because of its special application.)

EX 1: Branch if X is lower than 5. Assume that X-word 3, which contains 0000006, is to be used. Assume that BXL is in actual location 00014.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		BXL	5	3	
2	MOD	BRU	1411		
3		STA	TEMP		
4					
5					
6					
7					

P-Counter Contents in Octal and Symbolic

00014	=	MOD-1
00016	=	MOD+1

EX 2: Branch if X is lower than 5. Assume that X-word 3, which contains 00002, is to be used. Assume that BXL is in actual location 00014.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		BXL	5	3	
2	MOD	BRU	1411		
3		STA	TEMP		
4					

P-Counter Contents in Octal and Symbolic

00014	=	MOD-1
00015	=	MOD

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 to select the desired group.

LOAD X

LDX Y X 0600000 3

The contents of Y are loaded into X. Y is not affected.

EX: Load X with the contents of symbolic location SET1. Use X-word 3. Assume SET1 contains 0000001.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		LDX	SET1	3	
2					

SET1	X-3
0000001	?
0000001	0000001

This instruction is useful in initializing an X-word.

STORE X

STX Y X 1700000 3

The contents of X are stored in Y. X is not affected.

EX: Store X-word 2 contents in symbolic location RESET. Assume the X-word contains 0135746.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		STX	RESET	2	
2					

RESET	X-Word 2
?	0135746
0135746	0135746

SELECT X GROUP (This instruction is an optional feature.)

SXG Y 2506YY3 2

The address modification word group (0-31) specified by Y (in bit positions 11-15 of the instruction; position 16 always contains a 1-bit) is selected and remains selected until another SXG instruction is given. That is, after a given group is selected, all instructions referencing an X-word will refer to one of the words within the selected group.

EX: Select X-group 2, so that subsequent instructions containing X modification coding (bit positions 5 and 6) will refer to memory locations 00010 through 00013.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		SXG	2		
2					

After execution of the SXG instruction shown above, subsequent instructions containing 01, 10, or 11 in bit positions 5 and 6 will reference memory location 00011, 00012, or 00013, respectively, until another SXG instruction selects another modification word group. X-word instructions (INX, BXL, BXH, LDX, SPB, and STX) containing 00, 01, 10, or 11 will reference memory location 00010, 00011, 00012, or 00013.

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-word coding of the instruction in question to the result. For example, assume that an STA instruction specifies X-word 3 (coding 11_2) and that a previous SXG instruction selected X-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

An optional 16,384-word (16k) memory is available for GE-235 systems. This larger memory is considered for programming purposes to be divided into two banks of 8192 (8k) words each: the lower bank (locations 00000-08191$_{10}$) and the upper bank (locations 08192$_{10}$-16,383$_{10}$). Operation is limited to the lower bank if the 8K MEM ONLY indicating switch on the operator's console is lighted. Depressing the switch to turn off the indicator removes the limitation.

In the following discussion of 16k memory systems all references to addresses and location or register contents will be in decimal notation unless otherwise noted. If 5-digit or 6-digit numbers are used to signify register or location contents, the reference is to the operand portion of the contents only (bits 7-19 or 5-19, respectively).

Accessing Data Stored in the Upper Bank

The 13 bit positions (7-19) of the standard instruction word allotted for the operand address can accommodate only the binary equivalents of the addresses for locations 00000-08191. Thus, special procedures involving address modification are necessary for access to upper memory.

In automatic address modification, the address decoding network first determines what modification word (if any) is to be used by examining the coding of bit positions 5 and 6 of the instruction word held in the I-register. Then bits 7-19 of the instruction and bits 5-19 of the selected modification word are sent to the arithmetic unit and are added. Their sum then replaces bits 5-19 of the instruction word.

Since no location in an 8k memory requires more than 13 bits for its address, bits 5 and 6 in the modification word are always of the value 00 when operating with a standard memory or the lower bank of a 16k memory. Therefore, the modified address reset into the instruction word in the I-register still has effectively only 13 bits in the operand address. (It should be noted that the zeros in bit positions 5 and 6 replace any 1-bits in the original coding of these positions. The automatic modification coding is thus erased, once it has performed its function in initiating the modification; and bit positions 5 and 6 become available, though are not necessarily used, for part of the operand address.)

However, any location in upper memory requires 14 bits for its address--that is, bit position 6 must always contain a 1-bit to address a location above 08191:

0				4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
					0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	08191
					0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	08192

This 1-bit must be added during the process of address modification. It cannot appear in the original coding of the instruction, since a 1-bit in position 6 has, at that time, reference only to a modification word.

EX: An example of using address modification to access the upper bank appears below:

Symbol	Opr	Operand	X	
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
UPBNK	DEC	8192		
	LDX	UPBNK	2	
	LDA	6	2	

The binary equivalent of 08192 is first stored as a constant in symbolic location UPBANK and then transferred to the current X-word 2. (It is necessary that UPBNK is a lower-bank location; otherwise, an LDX instruction cannot access it, as will be seen later in the discussion.) The X-word may then be used whenever required to modify the operand address of an instruction in the manner shown. It is only necessary in the original coding to make the operand address of the instruction to be modified equal to the upper-bank address desired minus 08192. In this example, the desired effective address is 08198 (08198 - 08192 = 6).

Transferring Program Control to the Upper Bank

It has just been shown how to access data in the upper bank, regardless of the bank in which the instruction calling for the data has been stored. The following discussion is concerned with transferring control of the program to instructions that have been stored in the upper bank, regardless of the bank in which the data they access is stored.

If a 16k memory has not been limited by activation of the 8K MEM ONLY indicating switch, the P-counter (which is a 15-bit register, equivalent to word bits 5-19) automatically advances from 08191 to 08192 and beyond in normal sequence. The only other way to shift program control to the upper bank is to use a suitably indexed BRU instruction to set a 1-bit in position 6 of the P-counter. Once program control has been transferred to the upper bank, it remains there until another indexed BRU is executed. Intervening unindexed BRU's do not transfer control to the other bank.

EX 1: Change program control from the lower bank to location 12000. Assume that X-word 2 contains the constant 08192 and that P contains 01756.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		BRU	3808	2	
2					

P	X-2
01756	08192
12000	08192

The P-counter will now access location 12000 (03808+01892) for the next instruction and will continue accessing instructions in sequence (12001, 12002, etc.) without a further modification.

EX 2: Control is already established in the upper bank at location 12250. Transfer control to location 12000.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		BRU	3808		
2					

P
12250
12000

Execution of instructions in the upper bank continues. Since an unmodified BRU affects only bits 7-19 in P, the 1-bit remains in position 6. This 1-bit is equal to 08192. When bits 7-19 are replaced with the binary equivalent of 03808 upon execution of the BRU, the result is equal to 08192 + 3808, or 12000--which is the address of the next location to be accessed for an instruction. No X-word is involved.

EX 3: Change program control to lower bank from upper bank. Assume X-word 2 contains 00000 and that program control is at location 12250.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		BRU	3808	2	
2					

P
12250
03808

The next location accessed by P will be 03808 in the lower bank. Control will remain in the lower bank until another indexed BRU. The 0 in bit position 6 in X-word 2 replaces the 1-bit in position 6 in P, since 00000 is first added to 03808 and the resulting 15 bits are then placed in P.

Special Restrictions on the Use of the SPB, LDX and STX, and STO Instructions

SPB INSTRUCTION

An SPB instruction can be used in the upper bank to refer to an upper-bank subroutine. However, an SPB instruction in the upper bank cannot be used to refer directly to a subroutine in the lower bank; a modified BRU instruction must be used in conjunction with it. Likewise, using an SPB instruction in the lower bank to refer to a subroutine in the upper bank requires an accompanying modified BRU. The reason is that only 13 bits (position 7-19) are available for addressing in the SPB instruction.

EX 1: Assume X-word 2 contains 08192 and that SPB is stored in location 01750. Program control is in the lower bank. Use an SPB and BRU to transfer control to a subroutine in the upper bank.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		SPB	UPPER	1	
2	UPPER	BRU	3808	2	
3					

In step 1, address 01750 is stored in X-word 1; and control is transferred to symbolic location UPPER (that is, step 2). In step 2, the contents of X-word 2 (5-19) are added to the operand (7-19) of the BRU instruction and placed in the P-counter (5-19): 08192 + 03808 = 12000. Thus control is transferred from location 01750 in the lower bank to a subroutine beginning at location 12000 in the upper bank.

To return to the main program in the lower bank after execution of the subroutine beginning at location 12000, another modified BRU must be executed:

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		BRU	2	1	
2					

This instruction caused the contents of X-word 1 (5-19), previously stored, to be added to the operand (7-19) of the BRU and placed in P (5-19): 01750 + 00002 = 01752. This is the address of the instruction in the main program following the BRU in symbolic location UPPER (shown in the coding at the beginning of this example). Thus, control is returned to the lower bank.

EX 2: Assume program control is in the upper bank at the SPB stored in location 12225. Execute an SPB, branching to another upper-bank location.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		SPB	2000	1	
2					

The P-counter (5-19) before execution will contain 12225. Since this is an upper-bank address, bit position 6 will contain a 1-bit. Upon execution of the SPB, the quantity 12225 will be stored in X-word 1 of the current group; and 02000 will be set in bits 7-19 of P. Bit position 6 will not be affected, making the effective address 08192 + 02000 = 10192.

EX 3: Assume X-word 2 contains 00000 and that program control is at the SPB stored in location 12000. Use an SPB and BRU to transfer control to a subroutine in the lower bank.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		SPB	* + 1	1	
2		BRU	4000	2	
3					

In step 1 the address 12000 is stored in the current X-word 1 (5-19); and control is transferred to the next step. In step 2 the contents of X-word 2 (5-19) are added to the operand (7-19) of the BRU and placed in P (5-19): 00000 + 04000 = 04000. This operation replaces the 1-bit in position 6 of P with a 0-bit, transferring control to a subroutine beginning at location 04000 in the lower bank.

To return to the main program in the upper bank after execution of the subroutine beginning at location 04000, another modified BRU must be executed:

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		B R U	2	1	
2					

This instruction causes the contents of X-word 1 (5-19), previously stored, to be added to the operand (7-19) of the BRU and placed in P (5-19): 12000 + 00002 = 12002. This is the address of the instruction in the main program following the BRU shown in the coding at the beginning of this example. Thus, a 1-bit is restored to position 6 in P; and control is returned to the upper bank.

LDX AND STX INSTRUCTIONS

The contents of X-words are normally loaded and stored by using LDX and STX instructions. These instructions cannot be automatically modified, since they concern operations directly affecting an X-word and bits 5 and 6 are already in use to designate the X-word to be operated on. Therefore LDX and STX always refer to a location in the lower bank, though they may, of course, be executed from the upper bank, as shown in the following example.

EX: Assume that program control is in the upper bank at the LDX in location 12250 and that location 06500 contains 00000. Execute the LDX.

Opr	Operand	X
8 9 10	12 13 14 15 16 17 18 19	20
L D X	6 5 0 0	2

X-2	P	X-2	P
?	12250	00000	12251

06500	06500
00000	00000

The instruction is executed from the upper bank, but it refers to a location in the lower bank. Note that it would be impossible to modify it to refer to the corresponding location in the upper bank.

STO INSTRUCTION

The STO instruction is used for address replacement by storing bits 7-19 of the contents of the A-register in the selected memory location. It cannot be used to store addresses that are in the upper bank, as this would require transfer of bits 6-19 of the contents of A. The following example exhibits this limitation.

EX: Store an upper-bank address in location 09010, which contains 00000. The desired address is already loaded in A. Assume X-word 2 contains 08192.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		STO	818	2	
2					

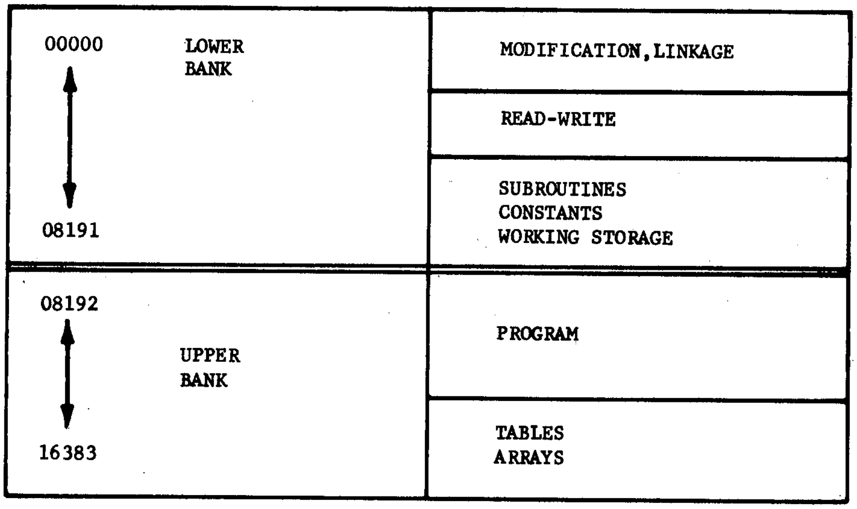
A	09010
12170	00000
12170	03978

Since only bits 7-19 (representing 12180 =03978) have been transferred from A to location 09010, the stored operand address is not correct.

Memory Allocation

The proper allocation of various portions of memory becomes particularly important with 16k memory systems, so that address modification operations may be minimized.

The diagram below illustrates an efficient and economical layout:



Subroutine Design

Packaged routines available from the GE Computer Department are designed for storage in the lower bank. It is also advisable that routines designed to run at an individual installation be constructed in this way. Since, in general, subroutines contain their own constants and working storage areas, all references to these locations would have to be address-modified in an upper-bank subroutine. In addition the LDX and STX instructions could not be used; and these are essential for effective use of the optional index groups.

When an existing subroutine written for an 8k memory is used with a 16k system, the following cautions should be observed, so that the routine is compatible with the larger memory:

1. Some older subroutines make use of negative address modification -- for example, INX -1,1. This practice generates a 1-bit carry into bit position 6 in the I-register. Such programming does not perform properly on 16k memory systems. (See the discussion accompanying the description of the INX instruction.)

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 bank.

3. Subroutine call sequence 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 call sequence arguments to the lower bank.

4. Many older routines, and even basic card formats, assumed a 13-bit operand address field. Since a 16K memory requires fourteen bits for the operand address field, changes may be necessary, depending on the routine.

Summary of Instruction Characteristics when Used to Address a 16K Memory

The chart below summarizes the characteristics of the various types of instructions when they are used to address a 16k memory. All those that require an extended description have been treated elsewhere in this discussion.

Instructions	Behavior
1. MOV and controller instructions	1. No modification is possible, but any memory location may be accessed with the 15-bit direct address. When these are applied along with other instructions to data in the upper bank, they are unindexed and the other instructions indexed to access the same data.
2. General instructions	2. The operand address is restricted or nonexistent, independent of memory size.
3. Indexed BRU	3. The only means for transferring control from one bank to the other. 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. STO	6. Stores only 13-bit operands. Otherwise like 7, below.
7. All others	7. Unindexed instructions access locations in the lower memory bank; indexed instructions may access any location via automatic address modification.

Programming Central Processor Operations

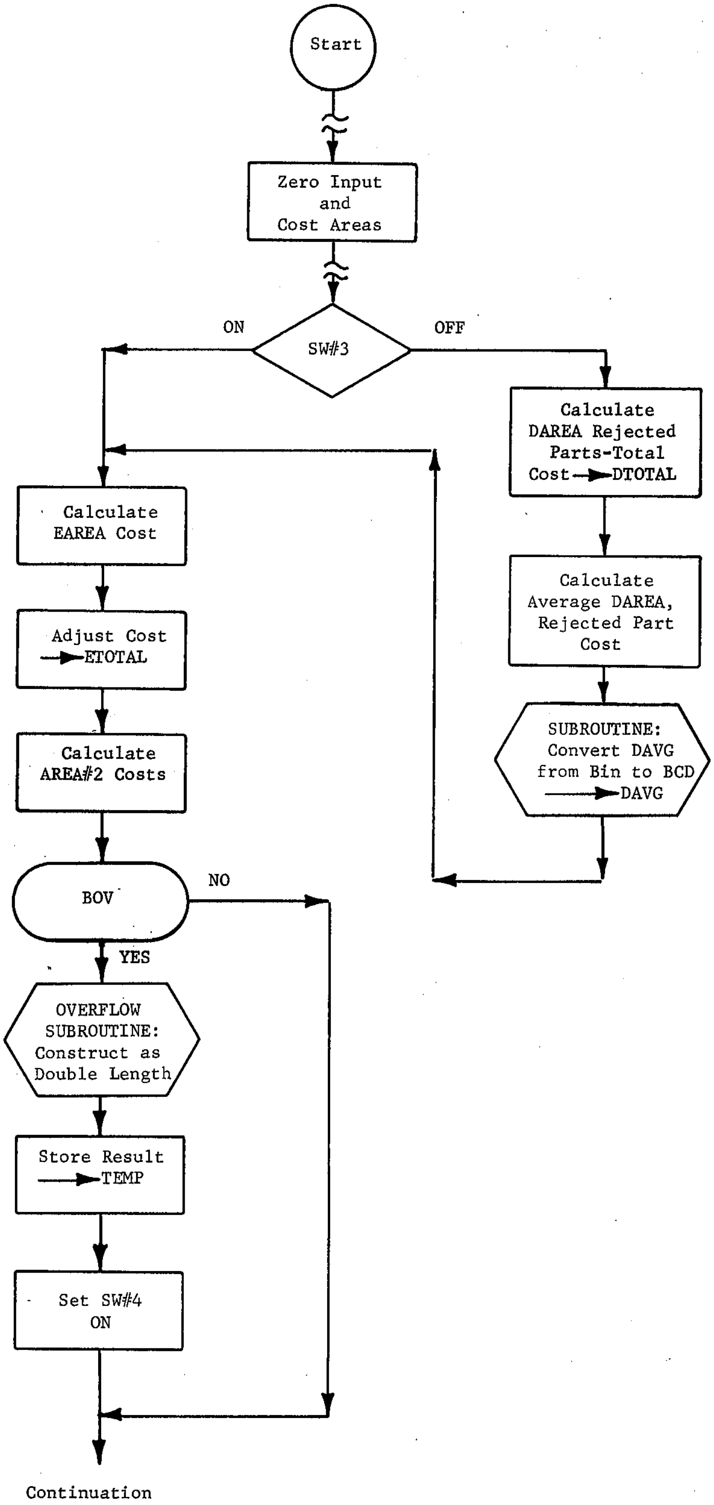
Figure VIII-1 illustrates a portion of the flow charting for a cost program for rejected parts. Coding sheets corresponding to that portion of the flow chart are shown in Figures VIII-2 through VIII-5. The coding shown was chosen to illustrate typical usage of central processor instructions rather than to show methods for programming specific problems.

In Figure VIII-2, lines 2 through 10 initialize the input and cost areas by storing zeros in the affected locations. Note the use of X-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 VIII-3, 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 3 transfers control to BYPASS (line 3, Figure VIII-4), DAREA calculations are skipped, and EAREA calculations are made.

Line 20 of Figure VIII-3 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 X-word 1. The final instruction of the NPRIBD subroutine is a BRU 0001, modified by X-word 1, which returns control to the instruction following the SPB.

Following the EAREA parts calculation in Figure VIII-4 is a test for overflow. If an overflow condition exists, line 11 causes the control location to be stored in X-word 1; and control then transfers to OVRFLO (line 2 of Figure VIII-5). After overflow recovery the BRU 0002, modified by X-word 1, returns to the main routine (line 13, Figure VIII-4).


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

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
	ORG	1000		MAIN PROGRAM ORIGIN	1000
START	LDX	ZERO	2	ZERO INDEX WORD TWO	1005
	DLD	ZERO			1010
	DST	APART	2	ZERO INPUT AREAS	1015
	INX	2	2		1020
	BXL	200	2		1025
	BRU	*-3			1030
	LDX	ZERO	2		1035
	DST	ACOST	2	ZERO COST AREAS	1040
	INX	2	2		1045
	ʃ ʃ				

Figure VIII-2. RPC Program - Initialization

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
	LDX	ZERO	2	ZERO INDEX WORD TWO	1250
	LDA	SW#3		SWITCH NO. 3	1255
	BNZ	BYPASS		SKIP DAREA COST	1260
	LDA	#DPART		NUMBER DAREA INDIVIDUAL PARTS	1265
	NEG			CONVERT TO TWO'S COMPLEMENT FORM	1270
	STO	LOOPD		SET UP NUMBER TIMES THRU LOOP	1275
DCALC	LDA	DPART	2	NUMBER OF EACH PART REJECTED	1280
	MAQ				1285
	MPY	DCOST	2	COST PER REJECTED PART	1290
	XAQ				1295
	ADD	DTOTAL			1300
	STA	DTOTAL		TOTAL COST DAREA REJECTED PARTS	1305
	INX	1	2		1310
LOOPD	BXL	0	2		1315
	BRU	DCALC			1320
	MAQ				1325
	DVD	#DPART		CALCULATE AVERAGE DAREA COST	1330
	DAD	ADJUST		ADJUST $	1335
	MAQ				1340
	SPB	NPRIBD	1	BIN-BCD CONVERSION ROUTINE	1345
	STO	*+3			1350
	ADO				1355
	STO	*+3			1360
	LDA	0		AVERAGE COST DAREA REJECTS	1365
	STA	DAVG			1370

Figure VIII-3. RPC Program - DPARTS Calculations

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 VIII-4. RPC Program - EPARTS Calculations and Constants

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 VIII-5. RPC Program - OVRFLO Routine

IX. ADDITIONAL INSTRUCTION REPERTOIRE

The additional instructions and programming for controlling direct input/output operations in the GE-235 central processor are described in this chapter. These operations include the priority control instructions and the input and output functions of the console typewriter, the operator's console, and the API (automatic program interrupt). The functions of these units and system control of operations have already been covered in Chapters III and IV.

CONTROLLER SELECTOR INSTRUCTIONS

Input/output operations of peripherals connected to priority control channels are accomplished by a programmed sequence of instructions. The specific contents of the instructions vary with the particular type of peripheral addressed. Only the generalized forms of the two instructions that refer specifically to the priority control are given here. The specific forms are given in the individual programming discussion in the separate subsystem reference manuals.

BRANCH ON CONTROLLER SELECTOR

BCS	XXX	P	2514PCC/2516P2C	4-6

The peripheral connected to priority control channel P is tested for the condition (CC) indicated by a mnemonic placed in the operand address field identified by XXX. When the priority control is ready, the channel to which the desired peripheral controller is connected must be selected by an SEL (Select) instruction. (This instruction may be automatically modified if provision is made in absolute coding. General Assembly Program coding does not allow for modification.)

SELECT

SEL	P	X	2500P20	4-6

The peripheral connected to priority control channel P is selected for the operation indicated by a following 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 clears controller error conditions, except those requiring manual intervention.

If this instruction is to be automatically modified, care should be taken that the modification affects only bit positions 11-13. Otherwise the nature of the instruction may be changed.

In programming for priority control peripheral operations the priority control should first be tested by a BCS instruction to determine if it is in a ready state before issuing an instruction to perform an operation. Attempted execution by the processor of an SEL instruction when the priority control is busy will halt the processor and turn on the CONT SEL echo alarm and PRIORITY alarm indicators.

Every peripheral connected to the priority control 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 but by the selected controller. Therefore when the SEL is in the I-register, the P-counter will hold the address of the third sequential instruction.

The contents of the two words following the SEL instruction are governed by the operation desired and by the peripheral equipment to be used. Specific details for programming these peripheral operations are given in the instructions for programming each peripheral.

CONSOLE TYPEWRITER PROGRAMMING AND INSTRUCTIONS

The GE-235 console typewriter can be used by the programmer to type out messages concerning conditions within a program and also to instruct the operator as to program needs. Using the typewriter for operation control can help reduce human errors.

Typical messages on program conditions are:

```
1.        0   ERRORS        TAPE  3
          0   ERRORS        TAPE  4
    END OF PASS 0

2.  END OF JOB

3.  EOF P  1T      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 WONT READ

2.  REMOVE 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.

Programming the Typewriter as an Output Device

Use of the typewriter as an output device requires that six-bit character codes be transmitted to the typewriter via the N-register. All typewriter character and function codes are shown in the "References" section of this manual.

To cause the typewriter to print data held in the A-register, the following sequence of operations must be programmed:

1. A TON (Typewriter On) instruction must be given, if the typewriter is not already on.
2. The N-register must be tested for ready status. (NOTE: When the N-register is "ready" it is logically connected to the A-register. When it is "not ready" it is logically connected to a peripheral.)
3. When the N-register becomes ready, one character may be shifted from the A-register to the N-register. If the shift is attempted when the N-register is in not-ready status, the data will be shifted out of the A-register and lost.
4. A TYP (Type) instruction will then cause the character in the N-register to be transmitted to the typewriter, where the character will be typed or a typewriter function performed, if the character is a typewriter control code.

Steps 2, 3, and 4 must be repeated in sequence for each character to be typed. The TON condition is disabled only by execution of an OFF (N-Register Logical Connections Off) instruction or a KON (Keyboard On), PON (Perforated Tape Punch On), RON (Perforated Tape Reader On) instruction. It should be noted that the typewriter is placed in the TON condition automatically by activation of either the PWR or the RESET MODES console switch, as well as by execution of a TON instruction.

It is a GE-215/225/235 programming convention to program different types of messages to originate at different points on the typing line:

1. General messages: at left margin.
2. Messages pertaining to input: at first tab (10 spaces).
3. Messages pertaining to output: at second tab (20 spaces).

Error messages are generally programmed to be printed in red.

When the carrier reaches the end of a line, it will automatically return to the beginning of the next line. Ordinary indexing to the next line need not be programmed, as it is accomplished automatically with each carrier return. Also, special programming is not necessary to print the characters or perform the functions appearing in the upshift position on the typewriter keys.

Programming the Typewriter as an Input Device

When typewriter input to the central processor is desired in debugging or as part of a program, a routine should be provided to perform all operations necessary to transfer input data from the N-register and process it in the desired manner as each character is typed.

An initial KON (Keyboard On) instruction is necessary before typing can begin. A testing and shifting sequence must be programmed in the same manner as for N-register output to the typewriter. (If a shift is attempted when the N-register is in not-ready status, the A-register will shift but the N-register will not.) The KON condition will remain in effect until an OFF instruction or a TON, PON, or RON instruction is given.

Typewriter Instructions

(Examples of typewriter instructions are shown later in the sample program.)

TYPEWRITER ON

TON	2500007	2

Logically connects the N-register to the typewriter as an output device and logically disconnects the perforated tape equipment from the N-register.

Sets the N-register ready. Failure to program a TON before a TYP (or a series of TYP instructions) will cause the N-register to become and remain not ready.

To disable the TON condition it is necessary to execute an OFF, PON, RON, or KON instruction.

KEYBOARD ON

KON	2500013	2

Logically connects the N-register to the typewriter as an input device and logically disconnects the perforated tape equipment from the N-register.

Clears N-register and sets it not ready. Then each time the N-register is loaded from the typewriter, it becomes ready. When the contents of the N-register are shifted into the A-register, the N-register becomes not ready again.

To disable the KON condition it is necessary to execute an OFF, TON, PON, or RON instruction.

TYPE

TYP 2500006 2

Types the 6-bit character held in the N-register. The contents of the N-register are unchanged.

The N-register must be in the ready status (brought about by an earlier execution of TON) to execute this instruction properly. During execution of a TYP the N-register becomes not ready and returns to ready again after the typing of the character is completed. If the N-register is not ready when a TYP is given, the character will not be typed; and the next instruction will be executed without any indication of program malfunction.

N-REGISTER LOGICAL CONNECTIONS OFF

OFF 2500005 2

Logically disconnects the typewriter (and the perforated tape equipment) from the N-register.

After an OFF is executed, a subsequent TON or KON is required to connect the typewriter logically to the N-register again.

BRANCH ON N-REGISTER NOT READY

BNN 2516005 2

If the N-register is logically connected to a peripheral (for the purpose of executing an instruction involving input or output)--that is, if the N-register is not ready--the next sequential instruction is executed. If the N-register is ready, the second sequential instruction is executed.

BRANCH ON N-REGISTER READY

BNR 2514005 2

If the N-register is logically connected to the A-register and is available for input to or output from the A-register--that is, if the N-register is ready--the next sequential instruction is executed. If the N-register is not ready, the second sequential instruction is executed.

Either BNN or BNR is used to ensure that the N-register is in the proper status to receive or transmit data codes during typewriter operations. The choice depends upon the particular programming situation.

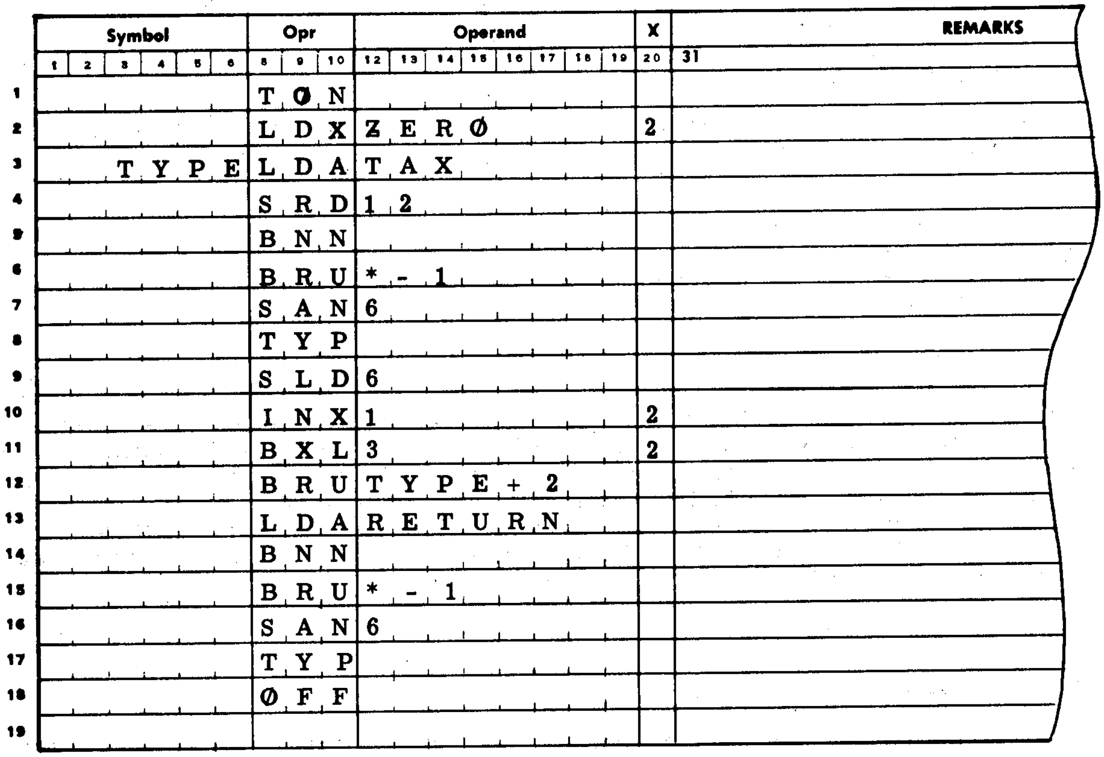
Sample Coding for Typewriter Operations

Prepared routines are available to assist programmers in coding for the typewriter. These routines provide for such operations as single- or multiple-word output, choice of red or black printout, tabulation, and insertion of punctuation.

To illustrate the use of instructions related to typewriter operations, a simple example of coding is shown in Figure IX-1. It is assumed that the programmer wishes to type a three-character message. Proper coding for the message is stored in symbolic location TAX and 37_8 (carrier return) is stored in symbolic location RETURN.

The following list of comments on the example is numbered to correspond with the numbering of the steps in the program:

1. The typewriter is logically connected to the N-register.
2. This step prepares index word 2 for use as a character counter by loading it with zeros. (It is assumed that symbolic location ZERO has previously been loaded with zeros.)
3. The three-character (18-bit) message is loaded into the A-register.
4. The second and third characters are shifted into the Q-register to place the first character in the proper position in the A-register for transfer to the N-register.
5. If the N-register is not ready, this step and the following one form a loop until it is ready. If the N-register is ready, processing continues with step 7.
6. See step 5.
7. The first character is shifted from the A-register to the N-register.
8. The first character is typed.
9. The second character is shifted from the Q-register to the proper position in the A-register.
10. Index word 2 is incremented by 1 to indicate that the first character has been typed.
11. If the count in index word 2 is less than 3, the following step is executed. If the count is 3, step 3 is executed.
12. The program loops back to step 5 to begin the sequence for printing the next character.
13. The program branches from step 11 to this step after the printing of the third character. The A-register is loaded with 37_8, the code for typewriter carrier return. (If the programmer did not desire to disconnect the typewriter logically from the N-register or return the carrier to the left margin and index to the next line, this and the following steps would be omitted; and the main program would continue from this point.)



	Symbol	Opr	Operand	X	REMARKS
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		TØN			
2		LDX	ZERØ	2	
3	TYPE	LDA	TAX		
4		SRD	12		
5		BNN			
6		BRU	*-1		
7		SAN	6		
8		TYP			
9		SLD	6		
10		INX	1	2	
11		BXL	3	2	
12		BRU	TYPE+2		
13		LDA	RETURN		
14		BNN			
15		BRU	*-1		
16		SAN	6		
17		TYP			
18		ØFF			
19					

Figure IX-1. Sample Coding for Typewriter Operation

14. Same function as step 5.

15. Same function as step 6.

16. Shifts carrier return code to N-register.

17. Returns typewriter carrier to left margin and indexes to next line.

18. Logically disconnects typewriter (and perforated tape equipment) from N-register.

THE PROGRAMMER AND THE OPERATOR'S CONSOLE

Instructions to Operator

The programmer should specify console switch settings in the run book for each program. The settings of the following switches are of particular importance:

1. VALIDITY CHECK ON
2. 8K MEM ONLY
3. STOP ON PARITY ALARM/NORM
4. A-register input switches

The first two switches listed above are for selecting operating modes available as options for a GE-235 system. The programmer's instructions for setting them depend on whether the system being used is equipped with the options or not and on the characteristics of the particular program. The setting of the STOP ON PARITY ALARM/NORM switch depends on whether or not the program includes remedial action for parity errors. If the program calls for operator intervention through the use of an RCS instruction, the setting of certain A-register input switches in the down position must be specified, as discussed below.

Operator's Console Instruction

Certain variations may be required in running a particular program. For example, the program may require tape output on one run and both tape and high-speed printer output on another. The flexibility required may be provided by furnishing the program with alternate paths that can be selected by instructions through the console, using an RCS instruction to interrogate the A-register input switches.

READ CONTROL SWITCHES

RCS	250011	2

Each of the 20 A-register input switches is examined. If a switch is in the down position, a 1-bit is entered (OR-ed) into the corresponding position in the A-register. If a switch is not in the down position, no alteration is made in the corresponding position in the A-register.

The position of the switches has no effect during automatic system operation except at the time the RCS is in the I-register. Thus, it is possible to preset the switches, making a halt unnecessary. In most applications the A-register should be cleared to zero before the execution of an RCS instruction.

EX: It is assumed that the programmer wishes to furnish an operand for a BRU instruction from an external source. The operator is to enter the information through the A-register input switches. The following coding would bring about the desired result. The explanation of the coding is numbered to correspond to the step numbers on the coding sheet.

	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		LDA	ZERØ		
2		RCS			
3		BPL			
4		BRU	* - 3		
5		STØ	INSTR		
6	INSTR	BRU	0		
7					

1. Zero the A-register.

2. Read the control switches. (This step is not significant until the operator has finished setting the input switches, as is explained below.)

3. Branch if the A-register contents are positive. Since the A-register has been zeroed by step 1, this step will be followed by step 4, until the operator enters a 1-bit (minus sign) in position 0 through the input switch on the console.

4. Branch to step 1. This instruction establishes a loop, which is not broken until the operator toggles input switch 0. Therefore, he may take the time necessary to set up the desired operand by means of the input switches and check to be sure that it is correct. Then he can place input switch 0 in the down position, causing the loop to be broken and steps 5 and 6 to be executed.

5. Store the operand in the A-register in the operand bit positions of the BRU instruction of step 6. This step becomes effective only after the operator has set input switch 0, as it will then be executed immediately after step 3.

6. The program will now branch to the location whose address has been set into the A-register by the operator and by the latest execution of the RCS instruction of step 2.

PROGRAMMING THE API

API Instructions

Two instructions are unique to the API feature and control the mode of system operation. One causes API to become effective; the other disables it. Their detailed descriptions follow on the next page.

SET PRIORITY START

SET PST	2506015	2

Places the system in the interrupt mode. After execution of this instruction, API remains in effect until an interrupt occurs and the system switches to priority mode. Since this instruction must be given before API can become effective, a programmer not wishing to use the interrupt feature can simply avoid executing a SET PST.

SET PRIORITY BREAK

SET PBK	2506016	2

Disables API. This instruction causes the system to leave the interrupt mode (initiated by a SET PST) and enter the noninterrupt mode. It remains in effect until another SET PST is given. A SET PBK instruction is automatically executed whenever the RESET MODES switch is pressed.

API Executive Routine

The API Executive Routine (CD225J4.000R) is a programming aid for reducing the housekeeping functions and the checks necessary for efficient use of GE-235 systems equipped with the API feature. Any configuration of priority program peripherals may be used with the executive routine except one including a document handler, the perforated tape reader, the perforated tape punch, or the console typewriter. Simultaneous processing of unrelated programs with API is possible without the executive routine, but it is recommended for all such applications. A GE-215/225/235 Automatic Priority Interrupt Executive Specifications Manual is available for detailed information. The contents of the manual comprise (1) a general description, (2) hardware requirements, (3) definitions, (4) system use--main program description, priority program description, remote inquiry program description and loading procedures, (5) conventions, (6) timing, (7) flow charts, and (8) a listing of the routine.

The executive routine performs the following functions:

1. Starts and ends all programs being executed under its control.
2. Saves the A- and Q-register contents, the contents of location 0000, and any overflow indication when a main program is interrupted by a priority program.
3. Determines which priority peripherals are in ready status and executes the appropriate priority program.
4. Restores the A and Q registers, the contents of location 0000, and any overflow condition before returning control to the main program.

Special API Programming Techniques

A main program to be used with API should be inspected carefully to determine whether damage would result from an interrupt at any particular points in the program. A SET PBK must be executed before entering any portion of the program that must not be interrupted, and a SET PST must be executed upon exiting. Main program operations that should not be interrupted are:

1. Perforated tape operations
2. Typewriter operations (if any of the priority programs make use of the N-register)
3. BCD arithmetic operations
4. All file subroutines for tape movement
5. Any other operation in which information could be lost if an interrupt occured before it was completed: for example, a SET PBK must be given before an RCD (Read Card) to prevent continuous card reading resulting from an interrupt between the RCD and the following HCR (Halt Card Read).

Care should be taken, however, not to overdo operating in the noninterrupt mode, or the effectiveness of API will be unnecessarily reduced.

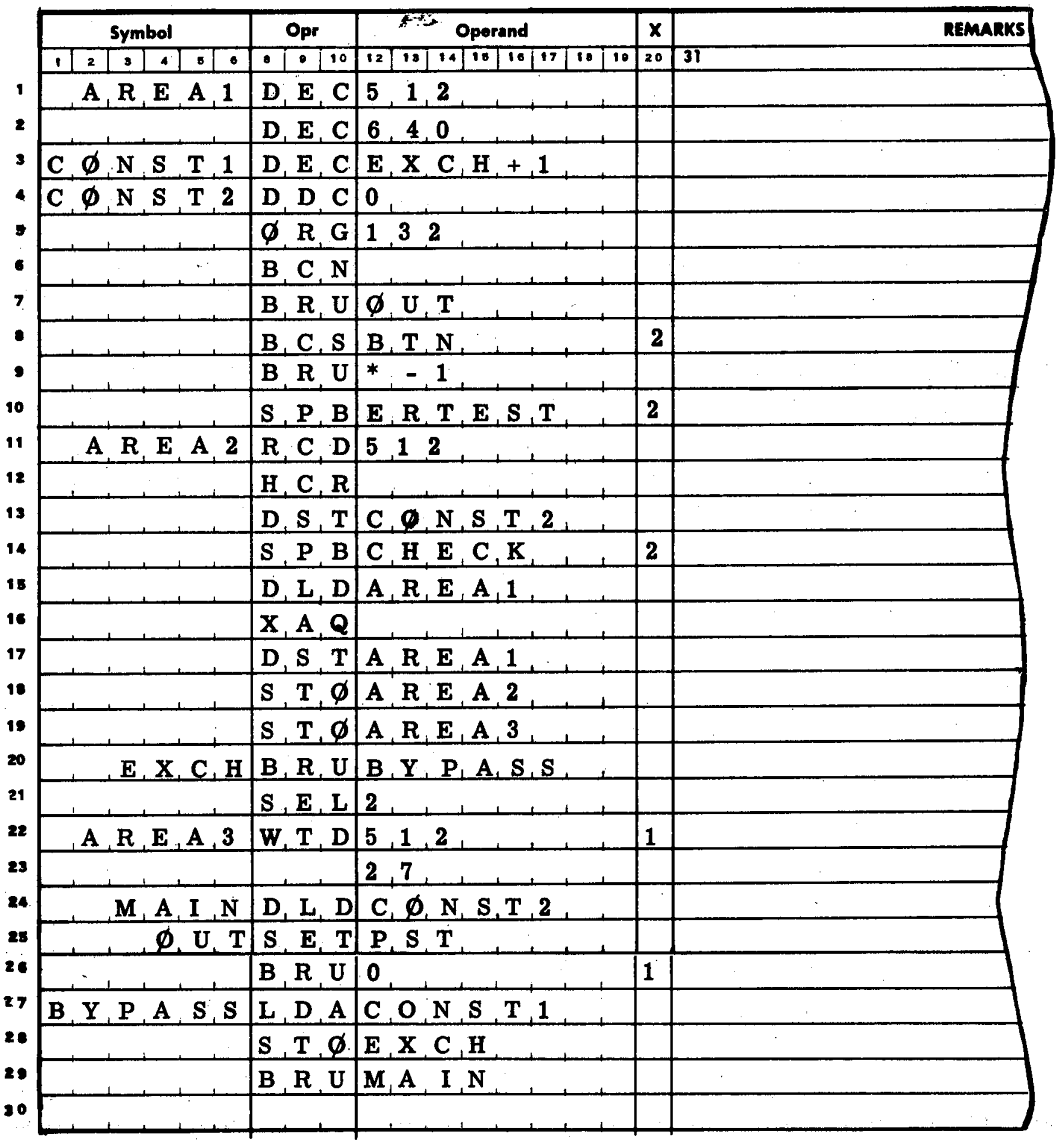
Proper coding of the priority programs is important to the success of API operation. If too much time is consumed in servicing a priority peripheral, another may become ready and cause another interrupt immediately upon return to the main program. The same situation may result from the use of too many peripherals. Execution of the main program may then be delayed until some of the priority programs have been completed, thus reducing the degree of simultaneous processing.

Sample Priority Program Coding

For the example of priority program coding shown in Figure IX-2 it has been assumed that a card-to-magnetic tape conversion and an independent processing operation are to be run concurrently in a GE-235 system equipped with API. The card-to-tape conversion is the priority program.

Ordinarily, the API Executive Routine would be used; it would relieve the programmer of the necessity to code some of the steps shown in the example and take care of some of the programming for the entire operation. For example, it would automatically store the A- and Q-register contents on leaving the main program and load them again at the end of the priority program operation. And it would take care of reading out the information from the last card onto the tape. (This is not provided for by the program shown in Figure IX-2. It is assumed that it is taken care of in the coding for the entire operation.)

Since the example is concerned with only one priority program, it is further assumed that the manual API switch for the card reader is the only one turned to ON. Thus, the card reader will be the only peripheral signaling for an interrupt.



	Symbol	Opr	Operand	X	REMARKS
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1	AREA1	DEC	512		
2		DEC	640		
3	CØNST1	DEC	EXCH+1		
4	CØNST2	DDC	0		
5		ØRG	132		
6		BCN			
7		BRU	ØUT		
8		BCS	BTN	2	
9		BRU	* -1		
10		SPB	ERTEST	2	
11	AREA2	RCD	512		
12		HCR			
13		DST	CØNST2		
14		SPB	CHECK	2	
15		DLD	AREA1		
16		XAQ			
17		DST	AREA1		
18		STØ	AREA2		
19		STØ	AREA3		
20	EXCH	BRU	BYPASS		
21		SEL	2		
22	AREA3	WTD	512	1	
23			27		
24	MAIN	DLD	CØNST2		
25	ØUT	SET	PST		
26		BRU	0	1	
27	BYPASS	LDA	CONST1		
28		STØ	EXCH		
29		BRU	MAIN		
30					

Figure IX-2. Example of Priority Program Coding

The numbering of the following list of comments on the priority program corresponds to the steps in the program (see Figure IX-2):

1. First read/write area.

2. Second read/write area.

3. Transfer location (refers to SEL). Eliminates writing bypass after first pass.

4. Storage area for contents of A-and Q-registers from main program.

5. Origin for priority program (Location 0132 must be specified, as API automatically branches to it for first instruction after interrupt.)

6. Test for card reader not ready. (This test is actually not needed, since no other peripheral could be causing the signal and the signal would not have been given unless the card reader was ready. However, if more than one priority program is being processed, such a test must be made. It would be followed by tests for ready status on the other priority peripherals capable of signaling, to determine the origin of the signal.)

7. Exit if card reader not ready. (This step appears only because step 8 requires it. It would never be executed in this program, because the card reader would always be ready at this point.)

8. Test for tape controller not ready. (Makes certain that next card will not be read into an area still being written from.)

9. Delay until tape controller ready.

10. Branch to error check of last write operation.

11. Read card into memory, beginning at location 0512.

12. Halt card reader.

13. Store contents of A- and Q-registers from main program.

14. Branch to error check of card reading operation.

15. Load addresses of beginning locations for read/write areas.

16. Reverse order of read/write area beginning addresses. (This procedure allows use of the areas alternately for reading and writing.)

17. Store read/write addresses again as constants.

18. Set up new reading area for next pass by operand modification of step 11.

19. Set up writing area. (Present writing area, after the first pass, is the same as next reading area. Thus, the same operand is stored in this step and in step 18.)

20. Bypass writing operation on first pass.

21. Select priority control channel 2 (to which tape controller is attached).

22. Write tape in decimal mode from locations beginning at 0512 onto tape 1.

23. Write 27 words.

24. Load contents of A and Q registers from main program in preparation for exit from priority program.

25. Set automatic priority interrupt ON (first step in standard exit from priority program).

26. Branch to location 0000 as modified by word 1 of index group 32--that is, to the address of next instruction in the main program, which was set in the P-counter at the time of interrupt (second and final step in standard exit from priority program).

27. Load address of unconditional branch that bypasses tape reading on first pass. (This and the following steps would be executed only on the first pass.)

28. Nullify branch for all passes after the first by operand modification of step 17.

29. Branch to preparation to exit to main program.

X. GENERAL PROGRAMMING PRACTICES

This chapter contains suggestions of a general nature on programming practices. Users of the GE-235 system should find them helpful in establishing and maintaining effective and efficient programming operations.

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:

1. Run number and title
2. Name of programmer, date completed, and date of last modification
3. A concise description of what the run is to accomplish
4. A write up containing all internal and external controls pertinent to the program, including:

   a. A completed operator instruction or run form that contains the following:

      Average run time and procedure to follow if established time limit is exceeded
      Console switch settings and brief description of each
      Error and special procedure loops with brief explanation of each
      Tape controller and input and output tape handler numbers

Identification and disposition of tapes
Rerun and restart procedure
All peripheral device set-ups and channel designation

b. Completed description and Layout Forms for all input and output

c. Memory allocation layout

Mark input and output areas
Program and subroutine areas
Working storage areas identifying each location used
If overlays are used, identify areas in which it occurs

5. Run diagram and flow chart:

An up-to-date run diagram and an accurate flow chart should be in the run book. Assembly coding reference points should be marked or identified on the flow chart. This provides reference between the operating program and the flow chart providing for easier program corrections.

6. 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.

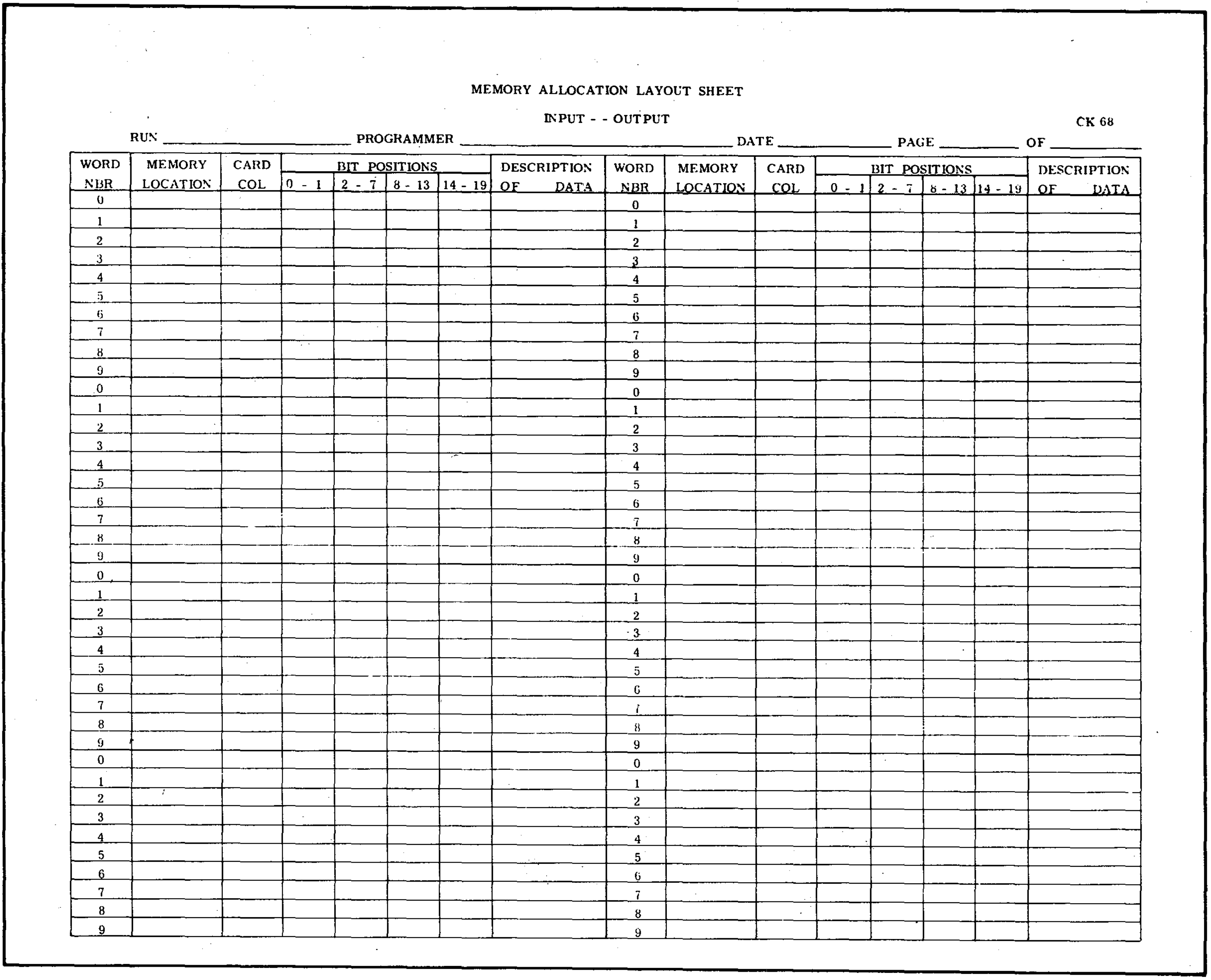
7. General Assembly Program listing:

The General Assembly Program listing can be included in the run book. If the 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.

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 for documentation are shown in Figures X-1 through X-5. Included are memory allocation and layout forms, punched card layout sheets, and a magnetic tape record layout sheet.



MEMORY ALLOCATION LAYOUT SHEET

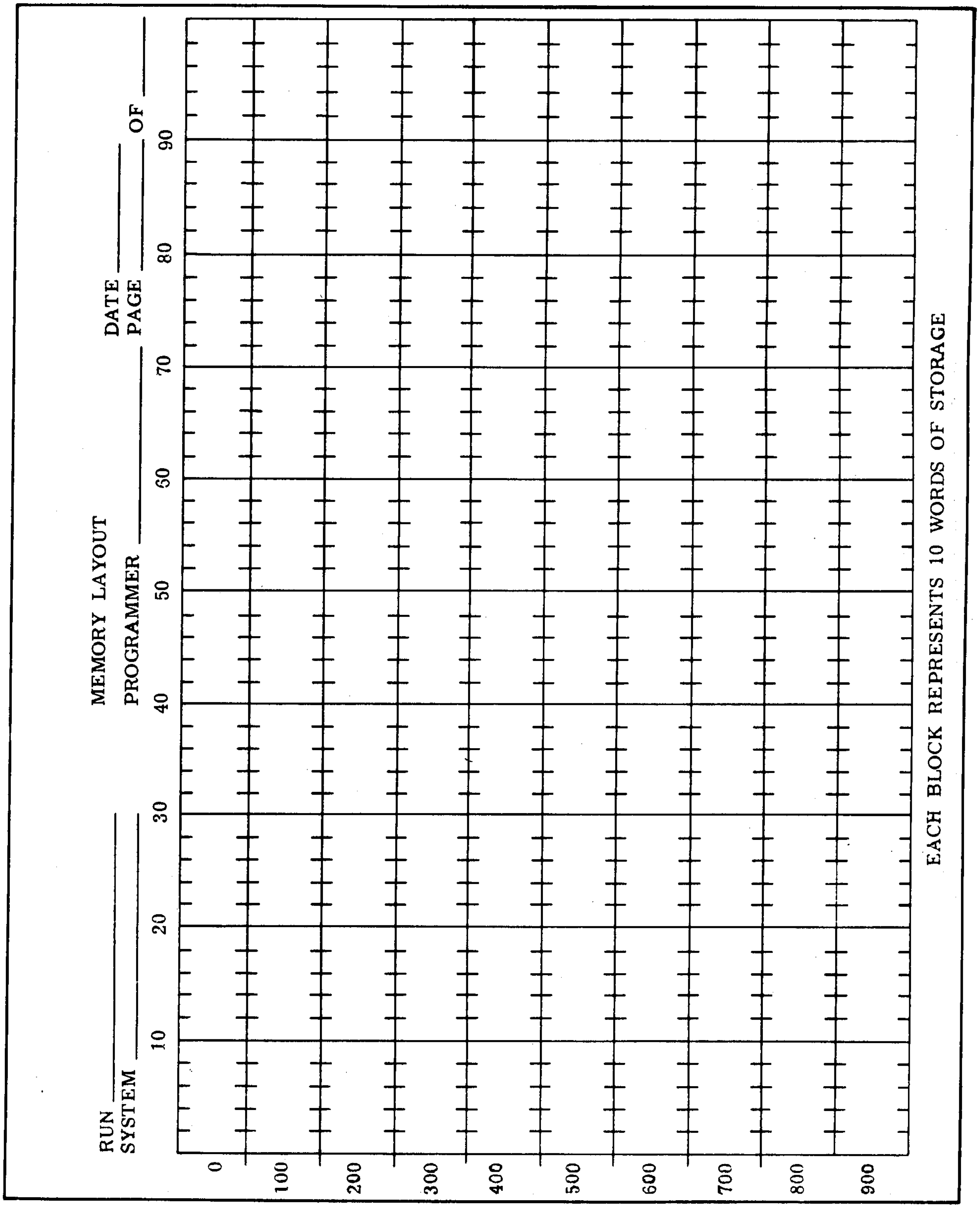
INPUT - - OUTPUT

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 X-1. Memory Allocation Layout Sheet



MEMORY LAYOUT

RUN ____________ PROGRAMMER ____________ DATE ____________

SYSTEM ____________ PAGE ______ OF ______

EACH BLOCK REPRESENTS 10 WORDS OF STORAGE

10
20
30
40
50
60
70
80
90
0
100
200
300
400
500
600
700
800
900

Figure X-2. Memory Layout Sheet

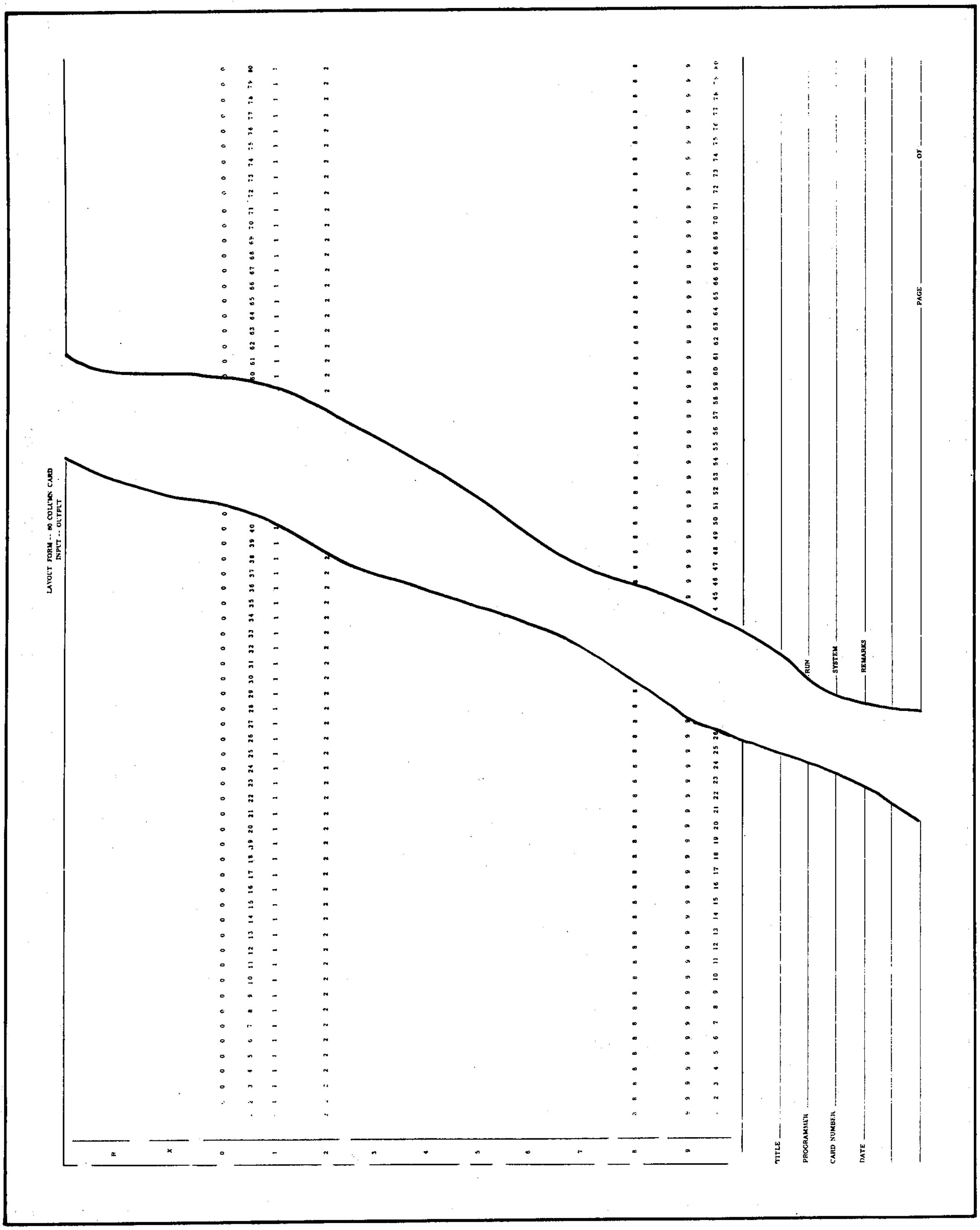
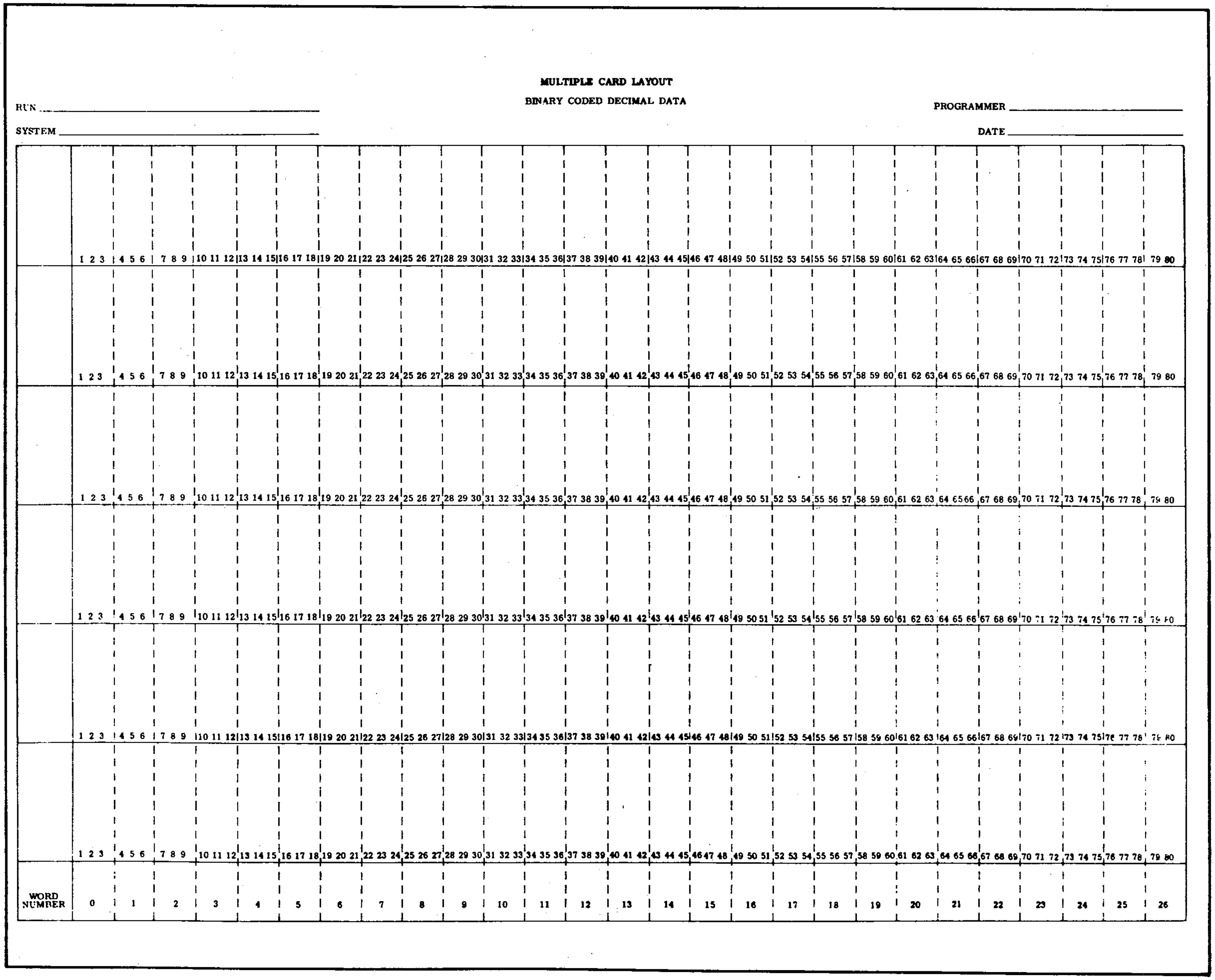


Figure X-3. 80-Column Card Layout Form



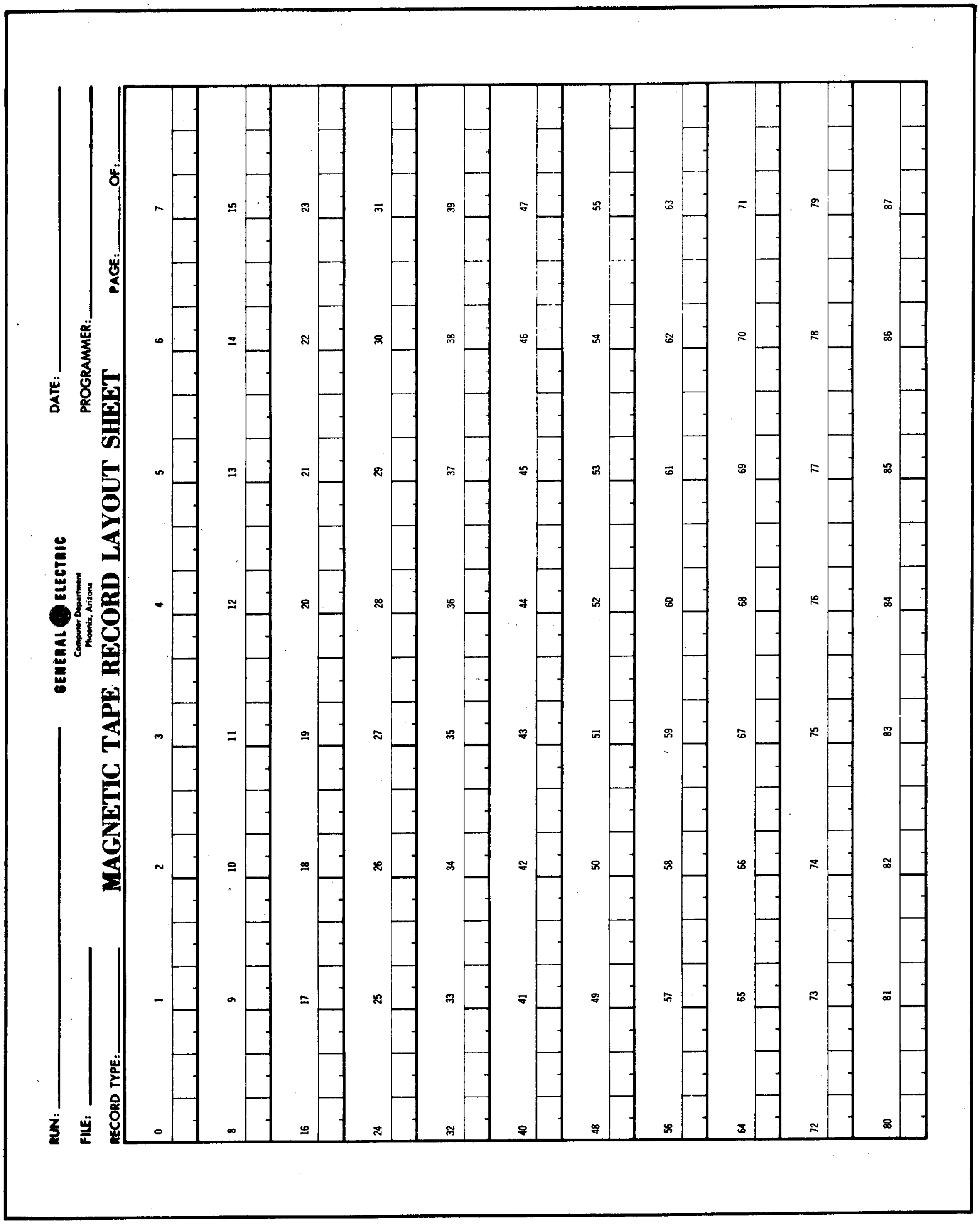
MULTIPLE CARD LAYOUT
BINARY CODED DECIMAL DATA

RUN ______________________ PROGRAMMER ______________________

SYSTEM ______________________ 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 X-4. BCD Multiple Card Layout Sheet



GENERAL ELECTRIC
Computer Department
Phoenix, Arizona

MAGNETIC TAPE RECORD LAYOUT SHEET

RUN: ______ DATE: ______

FILE: ______ PROGRAMMER: ______

RECORD TYPE: ______ 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 X-5. Magnetic Tape Record Layout Sheet

FLOW CHARTS

The flow chart is essentially the "road map" of a computer program. On it, the flow of data and processing are traced from the start of a program to its end. Flow charts are essential to the planning of a new program or routine; and, are valuable aids when new programs are being debugged.

There are two general types of flow charts:

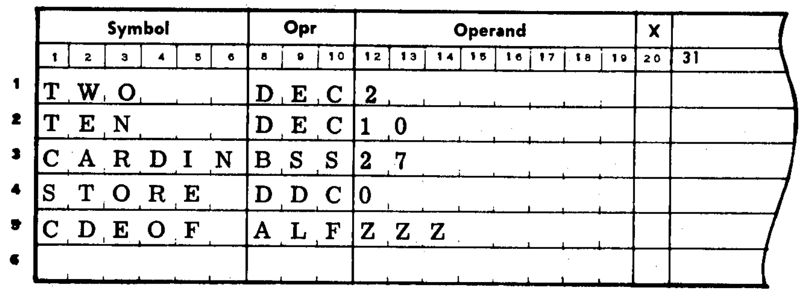
1. Top Level Flow Charts - In which the overall sequence of operations in a program is planned and illustrated.

2. Detailed Flow Charts - Used to define and illustrate specific portions of an overall program or routine in detail.

Complete details on drawing flow charts and the symbols used in them are given in the Reference section of this manual (Pages A-1 to A-5).

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 X-6 illustrates typical symbols.



	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1	TWO	DEC	2		
2	TEN	DEC	10		
3	CARDIN	BSS	27		
4	STORE	DDC	0		
5	CDEOF	ALF	ZZZ		
6					

Figure X-6. Typical Symbolic Addresses

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. An example of a memory layout is shown in Figure X-7.

Decimal Location	Description	Decimal Location	Description
0000 to 0003	Index Words	0402 to 0511	Miscellaneous Constants or Working Storage
0004 to 0127	Optional Index Words and Loaders	0512 to 0539	Card Punch Area
0128 to 0169	Reserved for Automatic Program Interrupt	0540 to 0639	Reserved fpr Automatic Program Interrupt
0170 to 0255	Miscellaneous Constants or Working Storage	0640 to 0719	Printout and Format Areas
0256 to 0283	Card Read-In Area	0720 to 0839	Magnetic Tape Input and Output Areas
0284 to 0383	Miscellaneous Constants or Working Storage	0840 to 1999	Subroutines
0384 to 0401	Card Read-In Area	2000 to 8191	Main Program

Figure X-7. Table Showing a Representative Memory Allocation Within an 8192-Word Memory.

SUBROUTINE USAGE

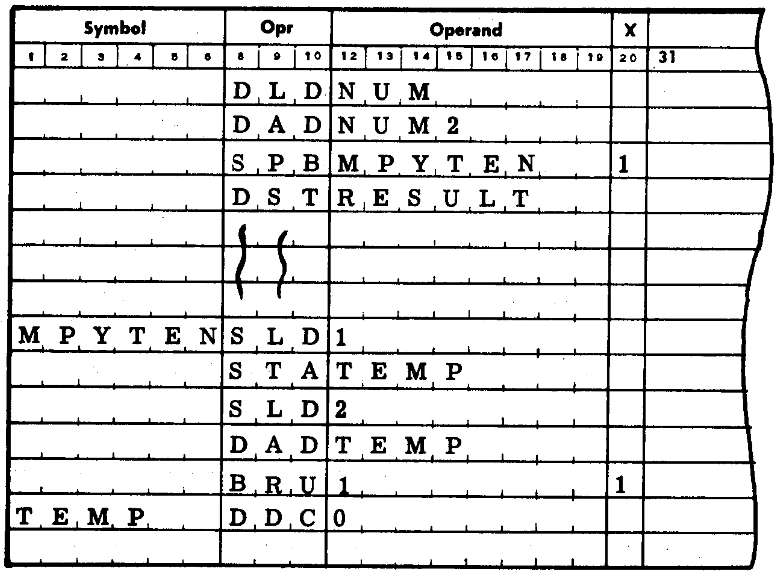
The use of subroutines can result in saving of both programming and machine running time. Subroutines can control all input and output operations and many internal operations of a program and use less memory. Normally a subroutine is a series of instructions which perform a repetitive function for the 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-235 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 X-8.

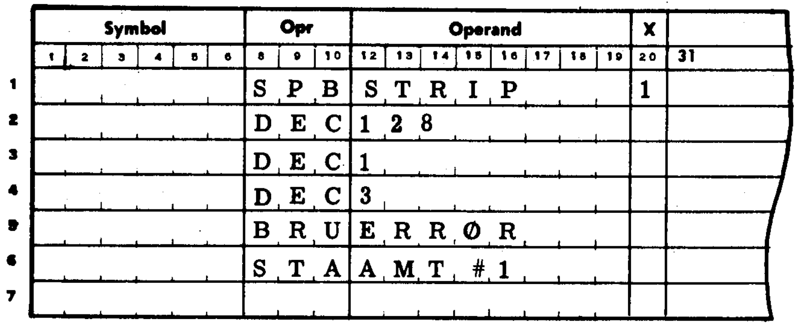


Symbol	Opr	Operand	X	
1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
	DLD	NUM		
	DAD	NUM2		
	SPB	MPYTEN	1	
	DST	RESULT		
MPYTEN	SLD	1		
	STA	TEMP		
	SLD	2		
	DAD	TEMP		
	BRU	1	1	
TEMP	DDC	0		

Figure X-8. 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 X-9.



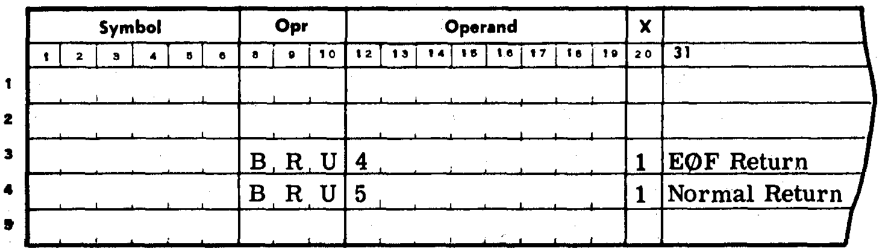
	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1		SPB	STRIP	1	
2		DEC	128		
3		DEC	1		
4		DEC	3		
5		BRU	ERRØR		
6		STA	AMT,#1		
7					

Figure X-9. Subroutine Requiring a 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

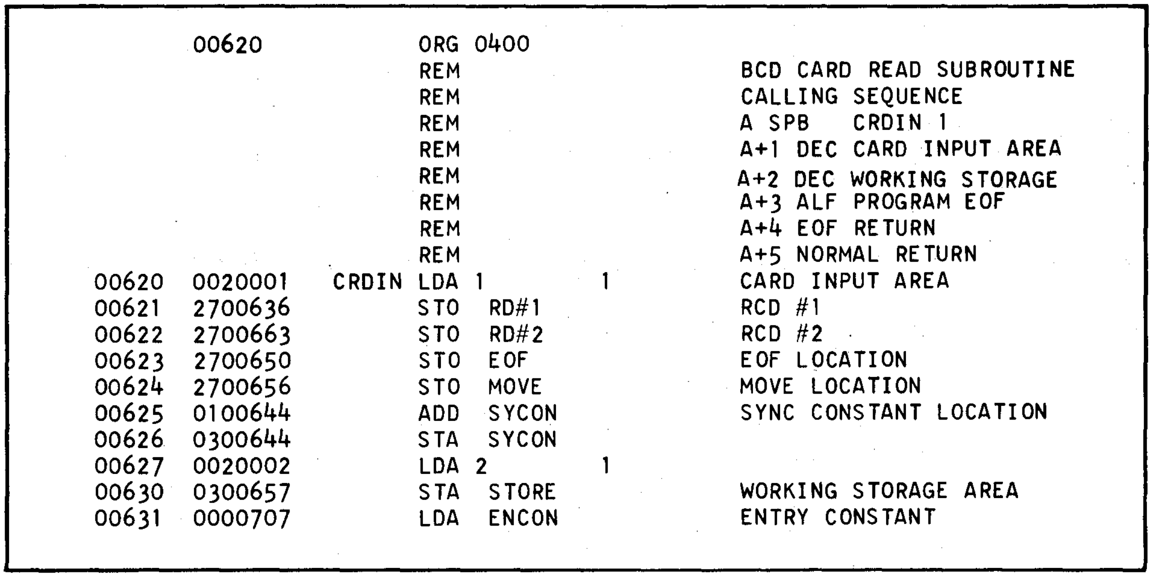
The exits from the routine are handled in the manner shown in Figure X-10.



	Symbol	Opr	Operand	X	
	1 2 3 4 5 6	8 9 10	12 13 14 15 16 17 18 19	20	31
1					
2					
3		B R U	4	1	EØF Return
4		B R U	5	1	Normal Return
5					

Figure X-10. Coding for Subroutine Calling Sequence.

A printout showing the calling sequence and the use of the parameters within the sequence is illustrated in Figure X-11.



```
          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 X-11. Subroutine Calling Sequence Printout.

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.

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 the assembly any card containing punches in columns 7 and 11 will be rejected. Correct any errors found before proceeding to the program assembly.

Correcting Errors Detected By The General Assembly Program

After the symbolic program has been assembled by the program 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 assembly 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.

```
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 X-12. Printer Controller Octal Memory Dump.

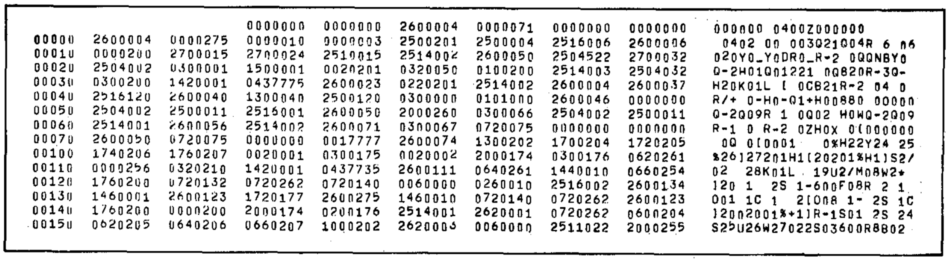
Memory Dumps

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 X-12), starting at memory location 0000 and continues until the manual clear button on the printer controller is depressed. The printer does not stop automatically when the entire memory has been dumped, but continues looping through memory until the clear button is pressed.

The octal dump that is most frequently used provides the octal memory location for each eight (8) word line of print (column 1 of Figure X-13). If the words for a line of print are identical to those of the last line printed, the line is skipped.

This saves the number of lines printed and machine time required for dumping. **This routine is a program feature and thus must be either in memory or read in from cards or tape.**



```
                        0000000 0000000 2600004 0000071 0000000 0000000   000000 0400Z000000
00000 2600004 0000275 0000010 0000003 2500201 2500004 2516006 2600006      0402 00 003021004R 6 06
00010 0000200 2700015 2700024 2510015 2514002 2600050 2504522 2700032     020Y0_Y0DR0_R-2 0QQNBY0
00020 2504002 0300001 1500001 0020201 0320050 0100200 2514003 2504032     Q-2H01Q01221 0Q820R-3Q-
00030 0300200 1420001 0437775 2600023 0220201 2514002 2600004 2600037     H20K01L [ 0CB21R-2 04 0
00040 2516120 2600040 1300040 2500120 0300000 0101000 2600046 0000000     R/+ 0-H0-Q1+H00880 00000
00050 2504002 2500011 2516001 2600050 2000260 0300066 2504002 2500011     Q-2Q09R 1 0Q02 H0WQ-2Q09
00060 2514001 2600056 2514002 2600071 0300067 0720075 0000000 0000000     R-1 0 R-2 0ZH0X 0(000000
00070 2600050 0720075 0000000 0017777 2600074 1300202 1700204 1720205      0Q 0[0001   0%H22Y24 25
00100 1740206 1760207 0020001 0300175 0020002 2000174 0300176 0620261     %26]27201H1[20201%H1]S2/
00110 0000256 0320210 1420001 0437735 2600111 0640261 1440010 0660254     02  28K01L  19U2/M08W2*
00120 1760200 0720132 0720262 0720140 0060000 0260010 2516002 2600134     ]20 1  2S 1-600F08R 2 1
00130 1460001 2600123 1720177 2600275 1460010 0720140 0720262 2600123     001 1C 1  2[008 1- 2S 1C
00140 1760200 0000200 2000174 0200176 2514001 2620001 0720262 0600204     ]2002001%+1]R-1S01 2S 24
00150 0620205 0640206 0660207 1000202 2620003 0060000 2511022 2000255     S25U26W27022S03600R8B02
```

Figure X-13. Memory Dump Printout Showing Octal and BCD Representation.

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 available to make a printout of these tapes via the high-speed printer.

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:

1. It gives the results of any program modification that may have been done
2. Programmer can check memory to see that information is correct and in the proper locations
3. It gives temporary or final results in key memory locations up to the time memory was dumped
4. 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 words 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.

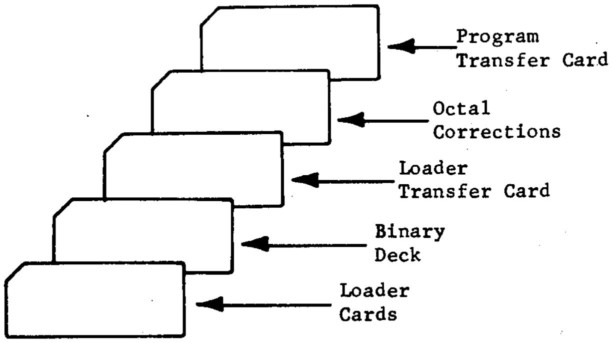
Tracing output is normally through the high-speed printer.

Loaders

When the program deck from the assembly program is in binary form, a binary loader deck is used to read the assembly program deck into the proper memory locations.

During debugging, it is best to use a binary loader which loads a binary deck 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 assembly.

To illustrate loader and correction card usage, the deck setup using octal corrections is shown below.



SECTION C: OPERATING THE GE-235 SYSTEM
XI. THE COMPUTER SYSTEM TEAM

This section of the GE-235 Central Processor Operating and Programming Manual is devoted to information which the computer operating staff needs to know during the normal course of operations. General Electric application and service engineers, and training instructors provide specific recommendations which are too specialized to cover in this manual.

THE COMPUTER SYSTEMS TEAM

The staff at a GE-235 installation normally consists of three general functions:

Management - Directs the overall operations of the data processing center. Plans and schedules the operation in the most efficient manner, and develops expansion of processing as needed.

System and Programming Analysts- Plan and write specific programs for the types of processing to be performed on the system. Test and debug programs, and prepare program run books for the operating staff.

Operators - Set up and run programs on the system, including setting up peripheral subsystems for each run. Keep appropriate records of operations and program documents and files. Inspect results obtained to insure accuracy.

The above personnel are generally employed by the organization which is running the system. Backing up this team are the General Electric application and service engineers. The application engineer provides counsel on tailoring General Electric's broad range of packaged programs to each system user's needs. He also aids the analysts in planning and writing specialized programs for the operations to be processed on the system.

The service engineer provides preventative maintenance for the system on a regularly scheduled basis in accordance withits needs. He also is available to correct any malfunctions that may occur in the equipment.

The staff associated with a GE-235 system is usually directly proportional to the size of the system. A typical small system with few peripherals may have simply an operations manager, one or more programmers--depending on the number of specialized programs to be developed--and an operator for each period of hours during which the system is operating.

The staff for a large system--one with a full range of peripheral subsystems to handle a wide range of applications--is usually organized for 16 to 24 hours of daily operation. It may consist of an overall systems manager, supervisor or manager for each shift of operations, machine room supervisor, operators for each shift, a program and file librarian, and a staff of programmers.

Generally, each system will have a staff in accordance with that organization's policies and practices. The role of the operator is stressed in this section of this manual. Information is provided about the equipment, procedures for operating the system, and related functions, such as record keeping and routine care of the equipment.

Scheduling Operations

The overall scheduling and assignment of operating time on a computer system is usually done by the management of that system. Many programs are run on a repetitive basis periodically, such as daily, weekly, monthly, etc. These major programs thus form the basic operating schedule.

The amount of time available for running other programs and eliminating errors can then be scheduled on a priority basis. Computer time remaining after subtracting the above may then be assigned on a "first-come, first-served" basis. Remaining unscheduled computer time can often be used to make test runs of new or revised programs to help the programmer eliminate errors.

The General Electric application engineer can help the computer staff plan a carefully developed schedule of operations that will result in maximum utilization, and a minimum of idle time, for the GE-235 system.

EQUIPMENT AND FACILITIES

The computer center usually has, in addition to the actual computer, office space for the staff, areas for maintenance and operating supplies, and a program library. The programming staff may or may not be located at the computer site. If not, means of communications should be provided so that operations personnel can consult with the programmers. The extent of these facilities is generally proportional to the size of the computer system. Each center develops methods and procedures for scheduling machine time, keeping logs of programs run, and other records of system performance. Samples of suggested logs and records are shown in Chapter XVI. The actual number and types of record forms may differ from these examples according to the precise needs of the individual centers.

Safety Considerations

To avoid the possibility of serious injury, the operating staff should follow basic safety practices. Operating practices have been specified with a view to permitting the operator to handle the GE-235 system equipment only to the degree that it is safe, both for himself and the equipment. The following list specifies general safety rules of importance to the operator. These, of course, supplement the general safety practices of the organization by which he is employed:

1. Do not remove covers from any equipment unless specifically instructed to do so by this operating manual, and confirmed by the service engineer.

2. Do not attempt to remove a jam from any machine while it is running.

3. Do not operate any machine with the covers off.

4. Do not use power extension cords in the equipment area.

XII. SYSTEM STARTUP AND SHUTDOWN

This section describes the general procedures for starting up and shutting down the GE-235 system. Usually all personnel concerned with operating a system should know these procedures, and especially those which apply in an emergency situation. The service engineer will acquaint these personnel with these procedures.

POWER SOURCES

Power for the GE-235 system is usually obtained from a central utilities power source through a main power distribution panel. Individual circuit breakers in this panel distribute power to the central processor and the controllers for most peripheral subsystems. Power thus reaches the blowers for cooling each cabinet, and the dc power supplies for the electronic circuitry. The console typewriter and the 400-card-per-minute card reader obtain their power through the central processor.

Circuit breakers within the central processor and each controller are normally left in the ON position by the service engineer. Operating personnel usually do not have access to these breakers.

STARTUP PROCEDURES

The GE-235 system is turned on at the beginning of daily operations. The entire system is normally left on while any part of the system is being used. Power is usually turned off when:

1. Operating personnel go off duty at systems which do not operate continuously
2. Regular or unscheduled maintenance is performed by the service engineer
3. During emergencies.

The system should never be left unattended when power is on.

The principle of turning on power to the GE-235 system is to start with the main power, then proceed through the central processor, to individual controllers, and then to peripheral equipment. Startup procedures include the following steps:

1. Turn on each switch or circuit breaker at the main power panel, starting with the switch for the central processor

2. At the central processor:

   a. Set INSTR/WORD switch to the WORD position

   b. Set AUTO/MANUAL switch to the MANUAL position

   c. Be sure blowers are operating, then press the PWR ON switch. If blowers are not heard and it is verified that the wall switch is on, notify the service engineer.

3. For each peripheral that does not have a controller, turn power on at the peripheral in any order desired. These peripherals and their power controls are:

   a. Typewriter: Press the upper end (marked ON) of the on-off switch bar at the right side of the keyboard

   b. 400 card per minute reader: Set the power switch to the ON position and the STOP/ENABLE switch to the ENABLE position

   c. 1000 card per minute reader:

      1. Set the AUTO/MANUAL switch on the computer ole to the MANUAL position

      2. Depress the power ON switch on the card reader's control and indicator panel (will glow amber)

      3. If the READ ERROR indicator on the control and indicator panel is lit, depress the READ ERROR switch.

   d. Card punches: Depress the POWER ON switch on the control and indicator panel (will glow green).

   e. Perforated tape reader/punch:

      1. Depress the POWER ON switch on the control and indicator panel (will glow red)

      2. Depress the OPERABLE switch on the control and indicator panel (will glow white).

4. For each peripheral that has a controller, insure that the blower is operating in the controller cabinet; turn on power to controller, then to the peripheral (if it has separate power switch). These peripherals and their power sources are as follows:

a. Magnetic tape subsystem:

1. Depress the POWER ON switch on the controller (will glow red)

2. Depress the POWER ON switch on the tape handler (will glow red).

b. High-Speed printers:

1. Depress the POWER ON switch on controller panel, then quickly proceed to Step 2

2. Depress the MANUAL CLEAR switch to prevent a runaway printer. This switch will turn off any alert lights that came on abnormally.

c. **Disc storage unit:** (It is assumed that the circuit breakers in each of the three units are on.)

1. Depress the POWER ON pushbutton on the electronics unit's control and indicator panel (will glow green)

2. Depress the PWR ON pushbutton on the controller's control and indicator panel (glows white)

3. Notice whether the discs in the disc file unit are turning

4. Wait for the OPERABLE light on the electronics unit's control and indicator panel to be lit (glows green). This takes about 6 minutes (The DISC ALARM indicator light goes off when the OPERABLE light comes on).

d. High-speed document handler:

1. For on-line operation, turn on power to the central processor. This turns on power to the document handler adapter (Steps 2 through 6 following apply to both on-line and off-line operation)

2. Turn the circuit breaker switch on the document handler to the ON position (up)

3. Check to see that the POWER indicator on the handler's control and indicator panel is on (will glow green). This places the document handler's electronics in a standby condition

4. Depress the MOTOR ON pushbutton on the control and indicator panel (will glow green)

5. Depress the ON-LINE, OFF-LINE MODE pushbutton on the control and indicator panel for the desired setting (The ON-LINE or OFF-LINE portions of the switch light alternately each time it is depressed)

6. The document handler is not ready for use until the FEED READY light is illuminated; this occurs after a delay of about a minute following the depression of the MOTOR ON pushbutton (Step 4).

e. Auxiliary arithmetic unit: (On systems where the AAU is attached to the central processor, the AAU control and indicator panel appears just above the console on the central processor.)

   1. Press the DC ON pushbutton on the AAU panel

   2. If a red ALERT light is on, depress the CLEAR ALERTS button on the control panel.

f. DATANET-15:

   1. Depress the POWER ON button at the lower left corner of the control panel (will glow amber)

   2. Depress the CLEAR button at the lower right corner of the panel. This resets all counters, registers, and turns on the **READY indicator** (will glow green).

The individual peripherals will now be ready to set up for on-line operation. Directions for loading input and output media, and setting up individual peripherals are contained in the manuals which cover each of these subsystems.

SHUTDOWN PROCEDURES

The procedures for turning power off and thereby shutting down the GE-235 system are the opposite of those for turning power on. Start with the switches most distant, electronically, from the main power and work toward main power. That is, turn off power to a peripheral, then to the controller, then to the central processor, and last of all, turn off the main power switches.

Peripherals may be turned off in any sequence desired. The most convenient order will depend on physical arrangement of the equipment. If perforated tape and magnetic tape are to be removed and stored, the operator may wish to start with these units. The following sections describe procedures for both normal shutdown and emergency shutdown.

Normal Shutdown

The following procedures are to be followed any time operators go off duty and whenever service engineers perform routine maintenance. These procedures are designed to save data in the central processor core memory and to prevent runaway peripherals:

1. Magnetic tape system:

   a. Set the REMOTE/LOCAL switch to LOCAL.

   b. Rewind and de-thread magnetic tapes (remove and store if desired), relieve tension on tension arms, and close all dust covers and doors on tape handlers

   c. Depress the POWER ON switch on each tape handler (light goes off)

d. Depress the POWER OFF switch on the tape controller.

2. Perforated tape reader and punch:

   a. Remove and store punched tape

   b. Depress the OPERABLE switch on control panel if either READER ON or PUNCH ON indicators are on

   c. Depress the POWER ON switch (light goes off).

3. High-speed printers:

   a. Depress the OFF LINE switch on the controller
   b. Depress the POWER OFF switch on the controller.

4. Card punch:

   a. Unload cards from input hopper
   b. Depress the MANUAL CYCLE switch until the punch is clear of cards
   c. Depress the POWER OFF switch.

5. 400 Card per minute reader: Set the power switch to the OFF position and the STOP/enable switch to the STOP position:

6. Typewriter: Press the lower end (marked OFF) of the on-off switch bar located at the right side of the keyboard.

7. 1000 card per minute reader:

   a. Depress the power OFF switch on the card reader's control and indicator panel.

8. Disc storage unit:

   a. Depress the POWER OFF pushbutton on the electronic unit's control and indicator panel

   b. Depress the PWR OFF pushbutton on the controller's control and indicator panel.

9. High-speed document handler:

   a. Depress the MOTOR OFF pushbutton on the control and indicator panel
   b. Turn the circuit breaker switch to the off position (down).

10. Auxiliary arithmetic unit:

    a. Depress the DC OFF pushbutton on the AAU control panel.

11. DATANET-15:

    a. Depress the POWER ON pushbutton at the lower left hand corner of the control panel.

12. Central processor: (after all peripherals are turned off):

    a. Put the AUTO/MANUAL switch in the MANUAL position
    b. Put the INSTR/WORD switch in the WORD position
    c. Depress the PWR OFF switch on the control panel.

13. Turn off all individual circuit breakers at the main power panel. (The master circuit breaker is normally left on.)

At the conclusion of daily operations, operating personnel will normally have all output data returned to the persons concerned, or filed, as appropriate. Also, all input material should be returned to the program library, or retained if needed for additional processing.

Emergency Shutdown

In emergencies, equipment is shut down for protection of both the equipment and the personnel in the system room. In grave emergencies, such as fire, flood, or a malfunctioning power system, the system is less likely to suffer heavy damage if the power is turned off at the main power panel. In minor emergencies, such as mechanical malfuctions, jammed tape, or shorts in equipment, the operator will normally be required to turn off power only to the equipment and controller concerned. The gravity of the emergency will govern operator action. Whenever main power fails, main power switches should be turned off to prevent damage to the system when power is restored. Chapter XVII contains more specific information on operator action under various conditions of emergency.

XIII. CENTRAL PROCESSOR OPERATIONS

This material covers the general operating procedures and practices for the GE-235 central processor, including the control console and input/output typewriter. The procedures for setup, and manual and automatic running of programs are included. While primarily written for the operating staff, programming analysts and management should find a review of this section helpful in understanding the operation of the GE-235 system.

The following chart (Figure XIII-1) lists the controls and indicators on the GE-235 operator's console in the "Control or Indicator" column. A detailed description of the function of each control and indicator is given in Chapter III of this manual, under "Operator's Console."

Functions of Operator's Console Indicators and Switches

Group	Control or Indicator	Function
Alarm Indicators	PRIORITY (yellow)	Indicates: 1. AUTO/MAN switch is in MAN position 2. Parity alarm condition 3. Central processor does not have priority 4. Card punch or card reader alarm condition
	PARITY (red) MEM	Indicates: Memory-checking circuits of processor detected parity error while processor in automatic mode
	N REG	Parity error connected with tape reader
	I/O	Parity error detected as data passes through access channels

Figure XIII-1. Operating Aid

FIGURE XIII-1 (Cont'd.)

Group	Control or Indicator	Function
Alarm Indicators (cont'd.)	OVERFLOW (red)	Indicates: 1. Capacity of the arithmetic unit exceeded 2. Illegal divide attempted 3. Data shifted left out of the A-register and lost
	CARD PUNCH (red)	Indicates card punching was attempted when card punch was not in a ready condition
	ECHO ALARMS (red) CONT SEL	Indicates: 1. Controller in off-line status 2. Controller malfunctioning 3. Controller power off 4. Controller in not-ready status 5. Channel addressed not occupied 6. Controller given illegal instruction
	C PUNCH C READER	1. Controller malfunctioning 2. Controller in not-ready status
	CARD READER ALERT (red)	Indicates: 1. Input hopper empty 2. Reader power off 3. No card on sensing platform 4. Card jam 5. Feed error (400cpm) 6. Phantom feed (1000cpm)
Ready Indicators (green)	CARD PUNCH	Indicates (when off): 1. Punch cycle in process 2. Stacker full 3. Input hopper empty 4. Misfeed 5. Card jam 6. Power off (chip box not seated)
	CARD READER	Indicates (when off): 1. Read cycle in process 2. Input hopper empty 3. Misfeed 4. Card jam

FIGURE XIII-1 (cont'd.)

Group	Control or Indicator	Function
Ready Indicators (green) (cont'd.)	N REG	Indicates (when off) that N-register is logically connected to peripheral and awaiting peripheral action
Mode Indicators	VALIDITY CHECK ON (white)	Indicates status of BCD validity check to be made on BCD cards
	8K MEM ONLY (white)	Indicates (when on) that memory addressing is modulo 8k
	API SET (yellow)	Indicates when API interrupt mode set (PST)
	API PGM (white)	Indicates when priority program running
	DECIMAL (white)	Indicates when processor in decimal mode
Select Display Indicators	CONT	Indicates which peripheral controller being accessed by processor
	N REG	Indicates N-register peripheral connection
Register Display Indicators and Associated Switches	N	Display contents of N-register
	INDEX GROUP	Display number of address modification word group selected
	P COUNTER	Display location of instruction in I-register (see text for exception)
	SAVE P	Prevents P-counter from incrementing
	I	Display current instruction

FIGURE XIII-1 (cont'd.)

Group	Control or Indicator	Function
Register Display Indicators and Associated Switches (cont'd.)	A	Display contents of A-register
	A-REGISTER INPUT SWITCHES	Feed information into the A-register while in either manual or automatic mode
	RESET A	Clears contents of A-register to zero when manual switch is engaged
Control Switches	PWR ON	Applies power to central processor and (1) resets alarms, (2) zeros P- counter, (3) selects modification group 0, (4) sets binary mode, (5) disables API, (6) zeros N-register and sets it ready, (7) executes TON, (8) resets BCD remembered carry
	PWR OFF	Turns off power to central processor
	RESET MODES	(1) selects modification group 0, (2) sets binary mode, (3) disables API, (4) zeros N-register and sets it ready, (5) executes TON, (6) resets BCD carry (effective only when MAN switch engaged)
	RESET ALARM	Clears alarm conditions(effective only when MANUAL switch engaged)
	LOAD CARD	Causes card reader to execute one RCB cycle(effective only when MAN switch engaged)
	RESET P	Resets P-counter to all zeros (effective only when MAN switch engaged)

FIGURE XIII-1 (cont'd.)

Group	Control or Indicator	Function
Control Switches (cont'd.)	AUTO/MAN	Selects automatic or manual mode of operation of central processor; MAN stops processor, turns on PRIORITY alarm
	INSTR/WORD	Selects length of cycle of processor during manual mode of operations (instruction or word at a time)
	START	Initiates execution of one cycle of operation (instruction or word); in automatic, initiates execution of program
	A → I	Transfers contents of A-register to I-register (effective only when MAN switch engaged)
	XAQ	Exchanges contents of A and Q registers (effective only when MAN switch engaged)
	STOP ON PARITY ALARM/NORM	Determines whether central processor stops when a parity error is detected upon memory readout

CONSOLE TYPEWRITER OPERATING PRACTICES

Manual Control Settings

(The numbers in parentheses in the following paragraphs refer to corresponding numbers in Figure XIII-2, where the various typewriter controls are identified.)

The console typewriter motor control switch (1) must manually be placed in the ON position to apply power to the typewriter. In the ON position the switch reveals a red area at its base.

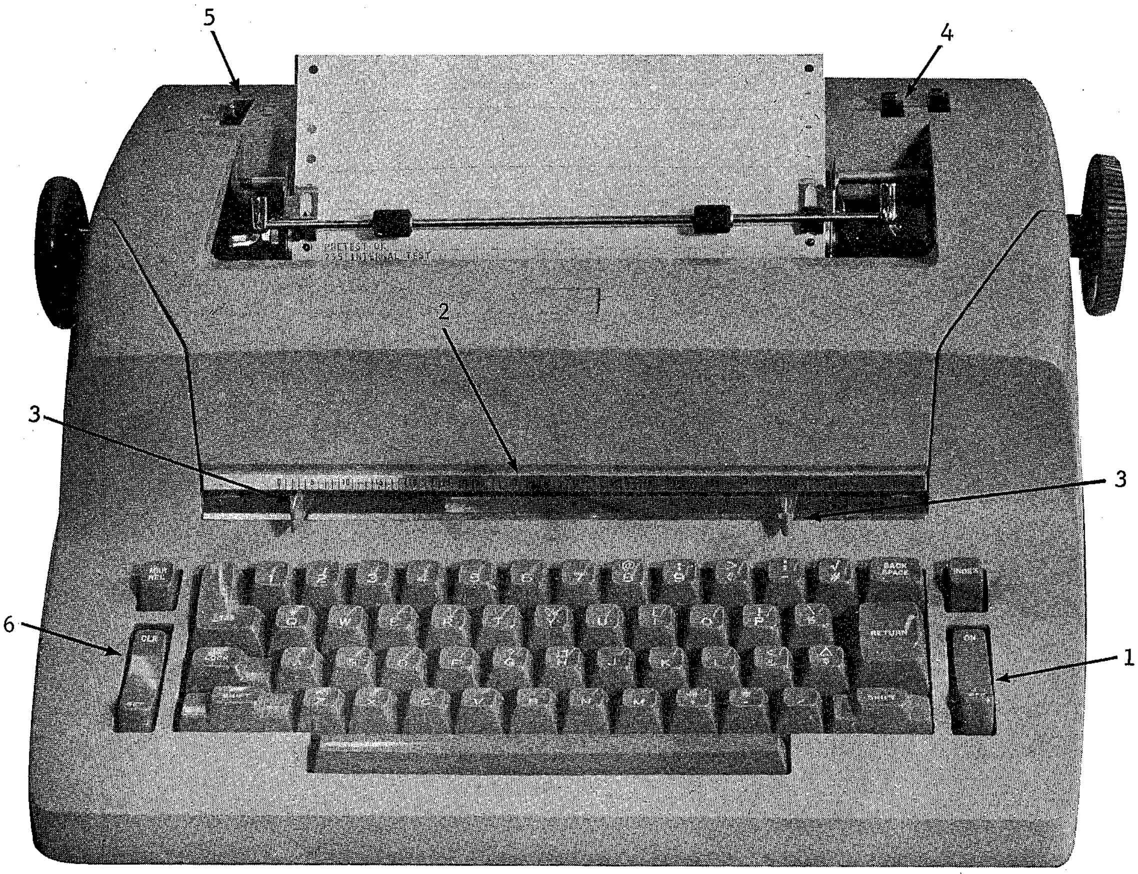


Figure XIII-2. Console Typewriter Controls

The margin scale (2) is directly above the typewriter keyboard. Margins are set by pushing in on the margin stops (3) and sliding them to the desired points on the scale. It is customary to set the left margin 10 spaces in from the edge of the paper and the right margin as near as possible to the edge of the paper.

The line space lever (4) may be set for either single or double spacing.

The multiple copy lever (5) must be moved from the forward position to one of the rear positions when carbon copies are to be made. It is recommended that the lever be moved no further toward the rear than is necessary for making clear copies.

To clear and set tabs, the typewriter motor control switch must be ON. Tab settings may be cleared by tabulating the carrier to each stop to be cleared and depressing the CLR end of the tab control key (6). New tab stops may be set by positioning the carrier at each desired stop point and depressing the SET end of the tab control key.

It is a COMPATIBLES/200 Series programming convention that the first two tabs be set at 10 and 20 spaces from the left margin, so that different types of messages may be programmed to originate at different positions on the typing line. Tabs must be set manually before a program is run, as clearing and setting operations cannot be programmed.

Manual Additions to the Programmed Log

When typewriter output is used as a program log, the operator may make additions to the log by manually operating the keyboard. This typing may be done at any time that there is no possibility the system will require the typewriter for output. The typewriter must not be in the KON (KEYBOARD ON) condition; otherwise it will transfer characters to the N-register.

Operator Duties when Typewriter Is To Be Used as Output Device

At the beginning of any shift during which the typewriter is to be used as an output device, the operator should make certain that:

1. The typewriter motor control switch is ON
2. Sufficient paper is loaded in the typewriter to last throughout the shift.

Before running each program he should make certain that any special instructions by the programmer for manual control settings have been complied with.

During processing the operator should immediately observe any typeouts, since they may contain information requiring operator action. Error messages are customarily programmed to be printed in red.

Typewriter output may be removed at any time. It is usually done at the end of each program or at the end of each shift.

Operator Duties when Typewriter Is To Be Used as Input Device

The operator should use the typewriter as an input device only when called for by a program. At such times he should operate the keyboard as on an ordinary typewriter, complying also with any special instructions supplied by the programmer.

Error Conditions and Corrective Action

OPERATOR ERRORS

If the typewriter fails to provide program output because the operator has not turned the motor control switch ON, the following corrective action should be taken:

1. Set the AUTO/MAN switch on the console to MAN
2. Turn the motor control switch ON
3. Restart program.

If the typewriter does not operate properly because the operator has not complied with the specified manual control settings, the procedures shown in steps 1 and 3, should be followed after the settings have been corrected.

PROGRAM ERRORS

Failure to program a TON instruction before a TYP instruction will result in the following:

1. Processing will halt at the TYP instruction
2. The typewriter will not produce output, even though its motor control switch is ON
3. The N REG READY indicator on the console will not be lighted.

Corrective action for this condition includes:

1. Manually entering a TON instruction (2500007_8) into the A-register through the console (but saving the present contents of the A-register)

2. Returning to the nearest restart point in the program

3. Making a record of the error condition for the programmer.

MANUAL OPERATING PROCEDURES

The option switches on the control console permit the operator to enter instructions and data manually; the register display lights permit the reading of the contents of memory and of certain registers. Thus, it is possible to feed in and execute a short program and read the results. Manual loading is used most, however, for getting a program started, correcting memory, branching, setting and clearing special modes, and reproducing damaged cards. Once an operator completes the necessary manual operations and gets a program started, control of operations is usually transferred to the central processor.

Loading an Instruction Manually

Any instruction that is intelligible to the GE-235 (see Alphabetical List of GE-235 Instructions at rear of manual) can be loaded manually into the A-register, as follows:

1. Set the AUTO/MANUAL switch to the MANUAL position
2. Set the INSTR/WORD switch to the INSTR position
3. Toggle the RESET A switch to clear the A-register
4. Load the octal equivalent of the instruction into the A-register (See instructions for toggling option switches in the previous section.)
5. Depress the A to I switch
6. Toggle the RESET A switch and load any necessary data into the A-register (Not necessary for some instructions.)
7. Depress the START switch.

The central processor will then execute the instruction placed in the I-register by the operator.

The following are the most-used instructions, and should be memorized as quickly as possible:

Mnemonic	Description	Octal Code
TON	Typewriter On	2500007
RON	Perforated Tape Reader On	2500014
PON	Perforated Tape Punch On	2500015
WCD	Write Card Decimal	250YY02
WCB	Write Card Binary	250YY03
WCF	Write Card Full	250YY17
RCD	Read Cards Decimal	250YY00
RCB	Read Cards Binary	250YY01
RCF	Read Cards Full	250YY10
BRU	Branch Unconditionally	2600000
SET DECMODE	Set Decimal Mode	2506011

SET BINMODE	Set Binary Mode	2506012
SXG Y	Select Index Group	2506YY3
STA	Store A	0300000
LDA	Load A	0000000
OFF	Input/Output Off	2500005

Loading Data Manually

When data is to be loaded manually into memory, follow this procedure:

1. Set the AUTO/MANUAL switch to the MANUAL position
2. Set the INSTR/WORD switch to the INSTR position
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 righthand 13 bits 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.

Manual Branching

Prior to executing a program, the operator may perform such manual operations as checking memory, feeding constants into the memory, or correcting memory. To then transfer to automatic operation, the operator must manually enter a branch instruction which contains the location of the first instruction to be executed. This is done as follows: (Power is on and the INSTR/WORD switch is in the INSTR position.)

1. Set the AUTO/MANUAL switch to the MANUAL position
2. Toggle the RESET A switch
3. Load a BRU instruction into the A-register (octal 26 in positions 0 through 4 and the memory location of first instruction to be executed in positions 7 through 19)
4. Depress the A to I switch

5. Set the AUTO/MANUAL switch to the AUTO position

6. Depress the START switch.

If the operator wishes to branch and remain in manual mode, he may use the above procedure omitting Step 5 (that is, leave AUTO/MANUAL switch in MANUAL mode). If the operator branches to the first instruction of a stored program, one instruction is executed each time the START switch is pressed.

Entering and Leaving Upper Memory

In order to get into upper memory (memory locations 8192 and above), it is necessary to use a basic branch instruction modified by index word 1, 2, or 3. The index word must contain a constant of 8192 (decimal). The octal equivalent of 8192 is 0020000, which means that bit position six is turned on and all others are turned off. Thus, any time bit position six is on in the P-counter indicator lights, the operator will know the central processor is in upper memory.

The following steps transfer operations to upper memory:

1. Set the AUTO/MANUAL switch to the MANUAL position (Assume that the INSTR switch is engaged.)

2. Set a Store A instruction into the A-register (STA is an octal 030000X, where X is 1, 2, or 3 for the index word.)

3. Depress the A to I switch

4. Toggle the RESET A switch

5. Set an octal 0020000 into the A-register

6. Depress the START switch (Stores constant into index word selected.)

7. Toggle the RESET A switch

8. Set into the A-register a BRU to the desired memory location modified by the selected index word used in step 2, above

9. Depress the START switch.

To transfer from upper memory to lower memory, follow the above steps, except step 5. The effect then is that step 6 (START) stores zeros into the index word being used.

Reproducing Individual Cards

In an emergency, it may be necessary to reproduce a single card. The following describes a quick method of doing this with manual instructions. It is assumed that the card punch is ready for operation and that its input hopper is loaded with blank cards:

1. Set the AUTO/MANUAL switch to the MANUAL position

2. Feed the card to be reproduced into the feed rollers of the card reader to position it on the sensing platform

3. By using option switches, set a read-card instruction into the A-register

   a. If a binary card, use RCB, octal 250YY01 (or Load Card)
   b. If a decimal card, use RCD, octal 250YY00

   YY is the starting address where the card is to be read. The handiest address is zero; if this cannot be used, remember that the address must be a multiple of 128 and less than 2048; that is, multiples of octal 200 and less than octal 4000.

4. Depress the A to I switch

5. Hold down the hopper-empty switch on the card reader and press the START switch on the console (reads cards)

6. By using option switches, set a write-card instruction into the A-register

   a. If a binary card, use WCB, octal 250YY03
   b. If a decimal card, use WCD, octal 250YY02

7. Depress the A to I switch

8. Depress the START switch (punches cards)

9. Depress the MANUAL CYCLE switch on the card punch twice to clear punched card into output hopper.

If more than one copy of the card is needed, repeat steps 7 and 8 as many times as there are cards needed (the write-card instruction remains in A and can be transferred to the I-register and executed as many times as necessary).

Saving Information in the A-Register

When an operator manually enters changes to a program, it is usually necessary to save information in the A-register before entering new information. It must be remembered, however, that changes should never be made by this method without instructions to do so by the programmer. The procedure for saving the information in the A-register and the location of the P-counter is as follows:

1. Place in a down position the option switches correspondiug to the ones (the lights) of the A-register. These switches are now a reminder to the operator of what was originally in A

2. Raise the RESET A switch to clear the A-register

3. Lower the SAVE P switch to save the information in the P-counter

4. Raise the option switches corresponding to the ones of the new information to be entered into A Return to the down position any of the switches which were in that position (as a result of step 1)

5. Move the new information now in the A-register to wherever it will be used in the program: for example, move it to the I-register by depressing A to I ("See Loading Data Manually").

6. Raise the RESET A switch to clear the A-register

7. Raise to the up position all of the option switches which are down (as a result of step 1.). Since the switches are spring loaded, they will automatically return to the normal position. This returns the original information to the A-register by entering ones into A to correspond with the down switches

8. Raise the SAVE P switch (lowered in step 3) to restore the original information to the P-counter.

Extracting Data from Memory

After a series of instructions or data has been loaded, the operator may want to check the contents of memory. The following steps can be used any time the operator wishes to know what is in any particular cell in memory. Once the information is displayed, it is a simple matter to correct it and return it to memory (see "Loading Data Manually"). It is assumed the AUTO/MANUAL switch is set to MANUAL and the INSTR/WORD switch is set to INSTR, and no alarms lights are on.

1. If it is desired to save P, set the SAVE P switch

2. Toggle the RESET A switch, thus leaving an LDA instruction in the A-register (Load A is an octal 00)

3. Load the memory location of the information desired into bit positions 7 through 19 of the A-register

4. Depress the A to I switch

5. Depress the START switch.

The contents of the memory location specified in step 3 now appears in the A-register.

Sequencing through Programs

It is possible to manually sequence through a program, step by step, and examine each instruction by reading the instruction register. This is accomplished with the following steps (Assume the INSTR switch is engaged):

1. Set the AUTO/MANUAL switch to MANUAL
2. Branch to the starting location of the program to be examined:
   a. Set a BRU instruction into A-register, including the address of the first instruction to be executed (BRU is octal 2600000)
   b. Depress A to I switch
3. Press START switch once for each instruction to be executed; read the I-register and P-counter after each step.

Special Modes

Normally the program will set and clear out special modes, such as the decimal mode, index group select, and the automatic program interrupt AIP. The operator will seldom need to set these conditions, but may occasionally have to clear them manually. For instance, a program being debugged may be aborted, and the special options are left on.

The procedure for clearing API, index group select, and decimal mode are described in the following paragraphs. If these procedures fail, more elaborate procedures are given in the sections immediately following.

CLEARING WITH PWR ON

A quick way to turn off the automatic program interrupt, clear out index group select, and change from the decimal mode to binary is to use the RESET MODES switch:

1. Set the AUTO/MANUAL switch to MANUAL
2. Depress RESET MODES switch
3. Make visual checks to see if goal is accomplished
4. Set the AUTO/MANUAL switch to AUTO when ready to resume automatic operation.

At step 3, note that the DEC MODE indicator will go out, the API indicator on the control console will go out (if on), and the index group indicators will go out. If these procedures fail, the operator can follow the procedures of the following section.

SETTING AND CLEARING DECIMAL MODE

Normally, the program will set and clear the decimal mode. If for any reason the operator finds it necessary to do this manually, the following procedures may be used. As previously mentioned, the DEC MODE indicator will be on when the central processor is in the decimal mode.

A. TO SET THE DECIMAL MODE:

1. Set the AUTO/MANUAL switch to MANUAL
2. Toggle the RESET A switch
3. Put the SET DECMODE instruction (octal 2506011) into the A- register
4. Depress the A to I switch
5. Depress the START switch
6. Check to be sure the DEC MODE indicator came on.

B. TO CLEAR DECIMAL MODE (SET BINARY MODE):

1. Set the AUTO/MANUAL switch to MANUAL
2. Toggle the RESET A switch
3. Put the Set Binmode instruction (octal 2506012) into the A-register
4. Depress the A to I switch
5. Depress the START switch
6. Be sure the DEC MODE indicator went off.

AUTOMATIC PROGRAM INTERRUPT MODE

The Automatic Program Interrupt Mode, which is present as an optional feature on some GE-235 Systems, is usually turned on and off by program control. Occasionally, however, it may be desired to turn this mode of operation on or off manually as follows:

A. MANUAL TURN-ON PROCEDURE (API):

1. Set the AUTO/MANUAL switch to the MANUAL position

2. If the program is to be resumed at the point of manual interruption, depress the SAVE P switch to preserve the contents of the P-register. Also, record the contents of the A-register on the log sheet

3. Introduce a SET PST instruction (octal 2506015) into the computer through the control console using the technique described under "Loading an Instruction Manually "

4. If the program is to be resumed at the point of manual interruption, return the SAVE P switch to its normal position and reinstate the contents of the A-register (as recorded in step 2) through the option switches.

B. MANUAL TURN-OFF PROCEDURE (API) :

1. Set the AUTO/MANUAL switch to MANUAL position

2. If the program is to be resumed at the point of manual interruption, depress the SAVE P switch to preserve the contents of the P-register. Also, record the contents of the A-register on the log sheet, or set the option switches

3. Introduce a SET PST instruction (octal 2506015) into the computer through the console switches using the technique described under "Loading an Instruction Manually "

4. After the SET PST instruction has been set and executed, follow that command with a SET PBK instruction (octal 2506016) through the console switches

5. Next, set a branch instruction with the console switches to return to the program

   a. If the program is to be resumed at the point of manual interruption, set a BRU 0 1 instruction (octal 2620000)

   b. If the program is to be resumed at the point of manual interruption, set a BRU Y instruction (where Y represents the memory location preserved in the P-register by the action described in step 2)

6. If the program is to be resumed at the point of manual interruption, return the SAVE P switch to its normal position and reinstate the contents of the A-register (as recorded in step 2) throuh the option switches.

RESETTING A PARITY ALARM

ıwo methods of resetting a MEM PARITY alarm will be described. The first method is to be used when the operator is starting a program and the parity error is in location zero. The second method is to be used when the program is beyond location zero.

A. RESETTING A PARITY ALARM IN LOCATION ZERO :

When the central processor's power is first turned on, a parity alert frequently occurs. The following procedure should be used to clear alerts:

1. Depress the RESET ALARM switch. It must be remembered that the RESET ALARM switch can damage a program, so this action must be taken only when directed by programmer instructions or when the operator is sure that the MEM PARITY alarm light goes off, the correction is made. If MEM PARITY alarm light does not go off, continue with steps 2 through 9

2. Set the STOP ON PARITY ALARM/NORM switch to NORM

3. Set the AUTO/MANUAL switch to MANUAL

4. Toggle the RESET A switch

5. Load the STA instruction (0300000) into the A-register

6. Depress the A to I switch

7. Toggle the RESET A switch (leaves all zeros in A)

8. Depress the START switch (loads zeros into memory location zero where the parity error supposedly occurred)

9. Depress the RESET ALARM switch.

B. RESETTING A PARITY ALARM NOT IN LOCATION ZERO :

It must be understood that any time the STOP ON PARITY ALARM switch is set and a parity alert is detected during a production run, the central processor halts and all peripherals halt after completing their latest instruction. At this time it is mandatory that the operator consult the operating instructions (run book) before doing anything to the equipment. It will usually be necessary to return the program to the nearest restart point. The occurrence of a parity alert in the central processor indicates that erroneous information is present. To depress the RESET ALARM and continue the program is apt to produce incorrect results. The RESET ALARM switch resets the overflow flip-flop and the carry flip-flop, and these could compound the problem rather than remedy it.

The procedure for resetting a MEM PARITY alarm in a location other than at the beginning of a program is as follows:

1. Set the AUTO/MANUAL switch to MANUAL

2. Set the STOP ON PARITY ALARM/NORM switch to NORM

3. Put a memory resetter, followed by 2 blank cards, into the input hopper of the card reader (8K or 16K depending on the size of the central processor, and either a zero or a minus resetter.)

4. Depress the LOAD CARD switch

5. Depress the RESET ALARM switch

6. Depress the LOAD CARD switch

7. Depress the RESET P switch

8. Set the AUTO/MANUAL switch to AUTO

9. Depress the START switch (Resetter will now clear memory.)

10. After memory has been cleared and the central processor stops, set the AUTO/MANUAL switch to MANUAL

11. Engage the STOP ON PARITY ALARM switch

12. Depress the RESET ALARM switch which should cause the MEM PARITY alarm light to go out

13. If the above steps do not clear the parity error condition, notify the service engineer.

Starting the Program

Most operators are principally concerned with only three types of program input: cards, magnetic tape, and perforated tape. The procedures for starting a program from each of these types of input are described in this section.

MEMORY RESETTERS

Before loading the program into memory, the operator should check the programmer's instructions to see what kind of memory resetter to use, if any.

Two types of memory resetters are often used. These are zero resetters and minus resetters. There are advantages and disadvantages to both types.

The zero memory resetter resets memory locations to zeros. The operator may use the zero resetter, then load a program and start running. If the machine should jump out of sequence for any reason, it may land in a location with all zeros, which is an LDA instruction. It will then proceed to continue loading the A-register until it comes back around to the program. It may enter the program at the wrong place and abort the program. Some service routines such as input/output routines require a special resetter.

The minus resetter sets all memory locations with ones. If the central processor accidentally jumps out of sequence during a run when memory has been reset with a minus resetter, the machine may jump into another part of the program or land in a location with all ones. Since all ones is a 37 which is an illegal instruction on most models, the central processor will halt.

A universal memory reset routine also is available which allows the user to reset memory to any desired constant.

CARD INPUT

Procedures for loading cards into memory depend on whether the cards contain a program or merely data for use after the program is already loaded into memory. Assume that power is on to the central processor, the INSTR/WORD switch is set to INSTR, and the card reader has been made ready. The following steps apply to the 400 card per minute reader.

The following procedure loads a program deck of cards into memory. The first card is usually a loader card punched in binary format and the last two cards must be blank:

1. Set the AUTO/MANUAL switch to MANUAL
2. Depress the RESET MODES switch
3. Depress the LOAD CARD switch to move the first card (assuming no alarms were on)
4. Depress the RESET ALARM switch (if indicator is on)
5. Depress the LOAD CARD switch to read the first card into memory
6. Depress the RESET P switch
7. Return the AUTO/MANUAL switch to AUTO
8. Depress the START switch to begin automatic feed of the cards under program control.

When loading a program deck of cards using the 1000-cpm card reader, use the following steps:

1. Set the AUTO/MANUAL switch to MANUAL

2. Depress the RESET MODES switch

3. Depress the LOAD CARD switch to move the first card (assuming no alarm indicators are on)

4. Depress the RESET ALARM switch (if an alarm indicator is on)

5. Depress the RESET P switch

6. Return the AUTO/MANUAL switch to AUTO

7. Depress the START switch to begin automatic feed of the cards under program control.

MAGNETIC TAPE INPUT

With system configurations having both a card reader and the magnetic tape system, it is a simple matter to read a call card, which calls a taped program into memory. The entire program and data input can be on tape, or the program can be on cards and the input data on tape. Without the card reader, instructions must be fed manually into the central processor to get the program started.

PERFORATED TAPE INPUT

When a card reader is available, perforated tape programs and data can easily be called into memory with a call card. Otherwise, a series of instructions must be fed manually into the central processor to get the program started.

ERRORS AND OPERATOR CORRECTIVE ACTION

The central processor may fail to operate correctly and cause program halts when the operator neglects to carry out any of the following operations.

Operator Checklist

1. See Figure XIII-3 for checklist
2. Reset alarms and modes before attempting to start
3. Put SAVE P switch in normal (up) position
4. Put INSTR/WORD switch in appropriate position.

Program Recovery or Restart

Watching and interpreting the indicator lights on the console will tell the operator much about the source of troubles when a program halts or refuses to start. The red alarm lights in the upper left corner of the console panel are danger signals which indicate that errors have been made, erroneous information has been received or transmitted, and the program may be aborted. The CARD PUNCH and CARD READER alarm lights mean the operator must restart or recover the program. In some cases, the operator can exercise care and save the program run. It is a good general rule to go into the manual mode of operation before attempting to correct error situations indicated on the console.

Whether a program can be saved or is aborted depends on the answer to the questions: has erroneous information been received or transmitted, or has information been missed in a read or write operation? False or missing information will usually abort a program. By studying the charts on error and corrective action in the manuals on individual peripherals the operator will often be able to determine whether a program can be saved.

When erroneous information has gotten into a run, the operator will correct any operator error, or have the service engineer correct any machine malfunction. The run book should indicate the nearest programmed recovery point. Successful recovery will save going back to the beginning. A good operator always looks in the run book or in other operator instructional material for programmer's instructions.

Alert Condition		Possible Cause	Corrective Action
PRIORITY indicator lights (yellow)		Central processor is in the manual mode	When ready to go into automatic mode, press RESET ALARM and the AUTO portion of the AUTO/MANUAL switch
		Alarm condition has occurred in card reader or card punch	See error and restart procedure on equipment concerned
PARITY indicators (red) light. If STOP ON PARITY ALARM/NORM switch is depressed, central processor halts	MEM	A parity error was detected in memory	See run book for recovery or restart procedures; press RESET ALARM and START switches to start processing if run book so specifies
		Random parity errors caused by system room overheating were detected on information coming from memory	Call service engineer to correct air conditioning
	N REG	A parity error was detected in perforated tape reader operation	Check perforated tape for damage; return to nearest rerun point; press RESET ALARM and START switches to start processing if run book so specifies
	I/O	A parity error was detected while receiving information via the priority control	Return to nearest rerun point; press RESET ALARM and START switches to start processing if run book so specifies
OVERFLOW alarm lights (red) comes on (no halt in program)		Capacity of the arithmetic unit has been exceeded	Normally this alarm will be reset by the program; if not, check run book for instructions
		The A-register overflowed on a shift-left instruction	
		An illegal division was attempted	
ECHO ALARMS light; program stops.	CONT SEL (red)	A peripheral controller that is operating through the controller selector was unable to respond when addressed (this pertains to every peripheral --except card readers and punches--which has a controller); off line, addressed channel not installed; power off; malfunction.	Check peripheral for incorrect setup

Figure XIII-3. Control Console Alert Conditions.

Alert Condition		Possible Cause	Corrective Action
ECHO ALARMS (cont'd.)	C PUNCH	Card punch not ready when a WCB, WCF, or WCD instruction was given	Make card punch ready and consult run book for restart procedures
	C READER	Card reader not ready when an RCB, RCF, or RCD instruction was given (busy, inoperable, card jam, or feed error)	Make card reader ready and consult run book for restart procedures
CARD READ ALERT (red) indicator lights		1. Card reader input hopper empty. 2. Card reader power off. 3. Card not positioned on sensing platform. 4. Card jam. 5. Card feed error (400 cpm). 6. Phantom feed (1000 cpm).	Inspect card reader and correct condition which caused alert
Central processor hangs in a loop and all peripherals halt		Program is branching on a peripheral ready or not ready. See console display lights.	Read console display lights to determine which peripheral is involved; refer to manual on that peripheral for corrective action
When loading a program deck of cards, the card reader reads one or two cards and halts; repeats		Processor may be in decimal mode	Check operator's panel; if DEC MODE light is on, reset to binary mode, rerun deck (RESET MODES switch)
		An index group (other than zero) has been selected	If any INDEX GROUP lights are on, reset group to zero (RESET MODES switch)
Central processor and peripherals halt		An illegal instruction is in I-register	Manually transfer contents of memory location indicated by P-counter to the A-register to determine if instruction is illegal. If programmer error, return program to originator. If machine error, try to rerun; if rerun fails, call service engineer
		Illegal instruction in I is all ones (a minus memory resetter was used), indicates machine has jumped out of the program	Restart and try to rerun program; if error recurs, call service engineer

Figure XIII-3. (Cont'd.)

XIV. GENERAL PROCEDURES

SETUP PROCEDURES

Setting up the central processor for operation involves turning on power and setting control switches. This procedure was covered in Chapter XII, both for the central processor and peripheral subsystems.

The following six steps are designed to save data in the core memory of the central processor. If the console was previously shut down by the normal procedure and switches have not been disturbed, steps 2 and 3 require only verification. Some operators check switches both visually and physically by actually pressing them even though they appear to be already in position. One exception: Do not press the POWER ON button when power is already on, except when performing special procedures to be described later:

1. Verify that blowers in central processor cabinets are operating
2. Set the AUTO/MANUAL switch to the MANUAL position
3. Set the INSTR/WORD switch to the INSTR position
4. Depress the POWER ON switch.
5. Depress the RESET ALARM and RESET MODES switches (this can be done either at this time or just prior to starting a program).

If the equipment is apparently not operating correctly when performing any of the above steps, the service engineer should be consulted.

OPERATING THE EQUIPMENT

During the running of programs and routines, operators perform a number of functions to assure accurate results, including:

1. Follow and execute the instructions as established by the programmers and generally contained in a program run book

2. Service the peripheral subsystems for individual runs

    a. Load punched cards, perforated tape, magnetic tape, paper forms and other input and output media

    b. Set the control switches on each peripheral to the correct positions for that run

3. Control the execution of the program through the console on the central processor

4. When operations stop before the program is completed, locate any errors and restart the program

5. Inspect the output to insure that it conforms to the examples in the run book

6. Obtain assistance from the programmer or service engineer as necessary to attain efficient operation of the programs on the system.

The operator also may perform routine servicing at the start of daily operations. or as necessary, including:

1. Replacing worn ribbons on type-out equipment

2. Emptying chip boxes on punching equipment

3. Cleaning read and write heads on magnetic tape handlers with a special solvent as approved by the G-E service engineering representative

4. Turning on the system, following the steps given in Chapter XII

5. Shutting down the system at the end of operations.

OPERATING INSTRUCTIONS

The specific instructions from which the operating staff runs programs are usually compiled by the programming staff. Depending on the complexity of the program or routine, these instructions may range from simple verbal comments to a complete book on a single program. Standardized, often-used programs usually will have a complete book, while new programs still under test may have only temporary documentation which the programmer will finalize after testing has been completed.

Program Run Book

The Program Run Book is actually a procedures manual for a program, providing a written record of everything pertinent to a run. It contains step-by-step operating procedures from which to run the program. In addition, descriptions, flow charts, and other details of interest primarily to programmers usually are included.

Under the section heading of "Operator Instructions," the following types of information are found:

1. Error procedures, rerun and restart instructions, and average run time. A list of program halts, their meaning, and specified operator action.
2. Initial setup for tape input and output, including tape labels and controller channel numbers; disposition of tapes.
3. Printer controller channel number, paper form number, setup of print position, VFU tape name, line number, punch number, and length.
4. Card description for use with card punch, including form number of card and color code, if any.
5. Document handler controller channel number, description of or number of plugboard, if used.
6. Typewriter tab settings, if non-standard. Description of typeouts which could include a dictionary of informative typeouts and corresponding operator action.
7. Order of card placement in card reader, including program cards, data cards, loader cards, transfer cards, blank cards, and any others used.
8. All instructions for setup of the control console, such as option switch settings.
9. Memory dump requirements and instructions for the dumps.
10. Take-down procedure, which covers information such as disassembly of the program and data decks of cards, labeling of punched cards, identification of reports from the printer, rewinding and labeling of all tapes. It specifies disposition of all cards, tapes and reports.

Areas of information in a run book which are not written directly for the operator, but which should nevertheless be of interest and assistance to him are:

1. General run description. This is a narrative description of the scope and purpose of the program. It usually states the system components required for the program's operation.
2. A general (top level) flow chart. This usually shows the source and disposition of all input/output. It indicates the input/output tapes, paper tape, cards, printer etc., and specifies tape unit numbers, channels, and labels.
3. Input and output specifications. This covers card and tape formats and descriptions of each field. Printer formats and field descriptions are also given.
4. Detailed flow charts. These indicate all of the major programming steps.
5. Memory allocation chart. This lists the areas of the program in memory and shows locations of input, output, constants, working storage, and available memory. It also shows locations of major subroutines.

Other Program Run Records

INSTRUCTION CARDS AND SHEETS

Operating instructions for running simpler programs and routines may also be compiled on single cards or sheets. Examples of these types of instructions are shown in Chapter XVI.

OPERATOR MANUAL

Frequently a computer center will compile a manual containing excerpts of the operating portions of the larger program run books. It also may include pertinent operating information, such as utility routines, assembly, and compiler programs. It may often be more convenient for the operating staff to use a manual of this type rather than the more lengthy run books or manuals on specific programs.

Corrective Action for Errors

A number of corrective actions can be taken for errors that occur while running programs. In addition to the general actions to be discussed, more specific measures are given in Chapter XIII, also in the separate manuals covering each peripheral subsystem of the GE-235. These measures are intended to serve as a guide. The General Electric application and service engineers can recommend specific corrective actions for each specific system installation.

When a program being run unexpectedly halts, it is usually because of an operating or programming error, or equipment malfunction. Often these errors can be corrected immediately and the program run continued. A second run of the programs will often be successful. When the cause of the error is found, it should be noted for future use and correction of operating procedures or the program.

OPERATOR ERRORS

These are usually the result of not following the established order of procedure, or setting of switches, placement of tapes on tape handlers, methods of feeding input cards or perforated tape, etc. The operating instructions for the program or routine should be quickly reviewed. A recheck is then made of everything that has been done. Each peripheral in use should be checked to see if it is properly set up and in a ready condition.

Program decks of punched cards should be checked to see that all cards are in the proper order, and that none are missing. A mutilated card should be replaced as soon as possible.

A worn or torn spot on perforated tape can cause read errors, as can a bad spot on magnetic tape. These flaws can usually be detected by visual inspection. More specific recommendations for correcting or replacing faulty input/output media are given in the individual manuals which cover the GE-235's subsystems.

PROGRAM ERRORS

Once programs have been tested and run without difficulty, they should be virtually free of errors. If trouble is experienced in running a program, all possible operation checks should be made. Program errors usually fall into three general categories:

1. The program halts unexpectedly
2. The program continues running past the scheduled time without producing results
3. The program produces faulty or incorrect results.

If the program halts at a point where it was not programmed to stop, first check for operator errors or equipment malfunction. Then note the contents of the I, A, N and Q registers, and the P-counter. Also note the condition of the error indicators on the central processor and peripheral controllers in use and record this information. The I-register tells what instruction has not yet been executed. The reading of the P-counter tells the location of the I-register's instruction. If possible, run a complete memory dump. All of this information is helpful to the programmer in locating and correcting programming errors with a minimum of delay.

If the program overruns in time, or appears to be repeating, it may be "hung up" in a loop because of an error. The loop may keep repeating indefinitely unless stopped. Often when the central processor continues operating after all peripheral subsystems in use have ceased all visible operation, it is an indication that the program is hung up in a loop.

The program run book specifies program loops, and the operator can often identify the loop that is causing the error. A consultation with the programmer will usually locate the error and enable the program run to be continued. Faulty results, especially when the program output is on punched cards, perforated tape, or magnetic tape, are not easy to identify. If discovered before the program is removed from the system, the operator can run a memory dump to help the programmer analyze the program and find the error.

EQUIPMENT MALFUNCTIONS

If the operating and programming checks indicate that an error is being caused by equipment malfunction, the service engineer should be consulted. Each GE-235 installation has its own procedures for contacting the service engineer and enlisting his help. Usually he will need to know the readings on all console indicators, switch settings, and peripheral indicator readings, at the occurrence of the malfunction. This information is vital to his quickly diagnosing and locating the malfunction.

Assistance to the Programmer

In addition to the team aspect of normal operations, the operator and programmer cooperate when test runs of new programs are made. This process is called "debugging," the finding and correcting of errors.

When one small section of a new program is causing trouble during a run, the operator may occasionally be asked to step manually through that group of instructions to help locate an error. This is done by setting the AUTO/MANUAL switch on the console to MANUAL, and repetitively depressing the START button.

The combined experience of the operator and programmer usually is effective in producing smooth-running, error-free programs with a minimum of debugging time. The procedures covered under Program Errors are effective tools in the debugging process.

XV. PROGRAMMING AND REFERENCE LIBRARY

A systematic library storage and handling procedure should be established at the GE-235 system installation. Among the records and materials usually retained in a library are: punched cards, perforated tapes, magnetic tapes, program run books, plus manuals and other documents covering the system, and general programming and operating information (such as this manual); manuals covering specific programs and routines.

FACILITIES

The computer staff should have procedures for storage, checkout and use of this library material. Information is presented here to serve as a guide to operating the library.

The library should have facilities to store these materials properly to protect them from damage or deterioration. This can prevent much of the operating difficulty which can happen when attempting to run damaged or worn cards or tape on the equipment. Appropriate shelving for books and manuals, storage racks for tape reels, and suitable protective drawers for cards, should be available. The following information on storing and handling magnetic tapes is typical of library operations.

Magnetic Tapes

Establish procedures for storage, checkout and use of master tapes, save tapes, and scratch tapes:

1. Master tapes contain a master file of permanent or semi-permanent information. The data on the tapes is updated rather than replaced

2. Save tapes are those which contain information which must be saved for a short period of time

3. Scratch tapes are those which contain information which is of no further use and upon which new information may be recorded

Tapes may be identified by means of special labels, or by numbers painted on their reels. A complete description of the information on tapes may be marked on special forms, such as those shown in Chapter XVI.

MASTER TAPES

Because of the importance of master program tapes, special precautions should be taken. A duplicate copy of each master tape is usually made in case some of the information is destroyed and needs to be retrieved. As an added safety measure, these duplicate master tapes may be stored in a different location to prevent loss of information due to fire or other disaster. Duplicates may be easily made by following the tape copy routine described in the GE-215/225/235 Magnetic Tape Subsystem Reference Manual (CPB-339).

The following example of typical tape library procedure illustrates how records may be kept on special punched cards:

1. Two source logs of tapes are kept for each 24-hour period. One log is at the console where tape reels to be saved are listed. The other is in the tape library and lists save tapes released for use

2. The release of save tapes (to become scratch tapes) may be made by the programmer. Release may be made either by a retention schedule or by initialed entry of the library's tape log

3. At the beginning of each operating period the librarian performs the following functions:

   a. Collects the tape logs

   b. Updates the tape file (of tab cards on each save-tape reel) as to retention schedule or entries from the logs

   c. Updates cards by key punching information about scratch tapes that have changed to save tapes

   d. Removes cards that represent save tapes changed to scratch tapes, and places them in the history file

   e. Makes a new scratch-tape card for the tape released in step d

   f. Files newly made cards for save tapes by file number and date, with the latest date first in the files

   g. Files updated scratch cards behind remainder of scratch cards in the file to provide rotation of tapes

   h. Preselects all tapes for the console operator for all shifts and places these cards in an in-use section of the file. Scratch tapes to be used are taken to correspond to the first cards in the scratch file (this permits tape rotation)

i. Places cards corresponding to the scratch tapes used the previous day in the back of the scratch-tape file

4. The librarian selects scratch-tape reels corresponding to the cards in the front of the scratch-tape card file and places the cards in the in-use section of the file

5. After the card file is completely updated for the day, a complete listing is made of the cards including the in-use file.

The retention schedule mentioned in step 3b above is a schedule used with production-run tapes. The schedule lists data numbers which identify a reel as containing a certain kind of data. Depending on the kind of data, the reel must be kept a specified number of days after the save tape was originated.

Punched Cards

Punched cards are a basic input medium for the GE-235 system. They should not be bent, folded or otherwise damaged by improper handling, storage or use. Cards should be stored in properly designed file drawers or containers which have some method of keeping the card deck under slight compression. This will prevent curling, buckling, or warpage. Additional suggestions on card handling and storage can be found in the GE-215/225/235 Punched Card Subsystems Reference Manual (CPB-302).

The usual practice at computer installations is to maintain a master working deck, and a duplicate reserve deck, of all frequently used routines, such as sorts, loaders and dumps. The duplicate file is necessary to permit reproduction of the duplicate cards in case cards in the working deck are damaged or lost. Each time the cards are removed from the duplicate file, these cards should be reproduced and replaced so as to keep a complete duplicate file in reserve at all times.

Perforated Tape

Perforated tape that is needed for subsequent runs on the GE-235 system can be stored in rolls in the library. A label identifying its contents can be attached to the loose end. The same control and record system suggested for magnetic tape can be used for perforated tape in the library.

Make every effort to keep perforated tape clean. Never allow it to drag on the floor. Dirt on the tape can become caught in the reader mechanism and cause errors. Avoid getting the tape wet with water, oil or other liquids. Dampness not only weakens the tape and cause tears or breaks, but the moisture can damage the tape reader and punch.

XVI. OPERATIONS RECORD KEEPING

Efficient operation of a GE-235 system is enhanced by a coordinated program of record keeping. This may include operations schedules, logs, instructions and time records, malfunction reports, input/output media control records, and program control and change records. A number of typical examples of these records are shown here. Of course, GE-235 system users undoubtedly will wish to modify and otherwise change the basic formats of these records to suit their own individual needs and policies. Most of the sample records are designed for use by operating and library personnel.

SCHEDULE OF OPERATIONS

A sample schedule of operations is shown in Figure XVI-1. It can be used by the systems management for precise scheduling of computer time to sections or individuals in the organization using the system. The sample shows blocks of time assigned to sections in the first two columns. Time assigned to individuals is shown in the remaining columns on the form.

COMPUTER UTILIZATION RECORDS

Complete and accurate logs should be kept by the operating staff of all system operating time. These logs then provide:

1. Management with a comparison of actual operating time used for the various runs with the previously scheduled time
2. Programmers with a measurement of the time used for test runs and debugging new programs
3. Maintenance personnel with the details of difficulties experienced during operation and the efficiency of maintenance procedures
4. The finance group in an organization with information on charging the various sections for the computer time each has used

The sample logs shown in Figures XVI-2 and XVI-3 illustrate the types of information usually recorded.

LOG TERMINOLOGY

It is important that there be complete understanding of log terminology by both those who make the logs and those who use them. If there is any doubt about the meaning of any of the terms, the operator should insist that each term be defined in writing. The following partial list of terms may or may not provide the exact meaning necessary to understand the log terminology of a particular site, but it should provide some familiarity with the terms:

Machine Error. Lost time caused by machine malfunction.

Operator Error. Lost time caused from operator error.

Input Error. Time lost because of faulty input data.

Debugging. Applies to all work in program checkout stage.

Production. This code is used for all work that is checked out and known to be working properly.

Overhead. Time charged to computer operations, such as tape edit and tape cleaning.

Training. Time when a training group is using the computer, such as product service training, computer operations training, and other internal groups.

Utility Failure. Time lost through power failure.

Unused. Time when the machine is idle.

Facility Failure. Time when failure is caused by air conditioning, humidity control, etc.

Scheduled Maintenance. This is used when service engineering personnel are on the machine during the normal maintenance time.

Unscheduled Maintenance. This is used when it is necessary for service engineering personnel to work on the machine during normal center operation; but it does not necessarily halt operations on equipment other than that being repaired.

System Checkout. This category will be used if operator is having machine trouble and decides to run some diagnostics to determine trouble areas.

Power-on Time. Time starting when power is turned on at the beginning of each day and ending when power is turned off at the end of the day.

Total Available Time. Includes all time that the equipment has power on, less preventative maintenance.

TIME CARD LOG

Some systems have a time clock installed which will record the time on a punched card type of log form, shown in Figure XVI-4. The user fills out the card and gives it to the operator to stamp the starting and stopping time of a given program or routine. The time clock is usually installed under the console desk where it can be easily reached. Time is recorded in hours (24 hours to the day) and in hundredths of hours.

EQUIPMENT MALFUNCTION REPORT

A special form for reporting on equipment malfunctions by the operator is shown in Figure XVI-5. His comments is shown on the left-hand side of the form, while the notation by the service engineer are recorded on the right-hand side. This type of record helps document when malfunctions have been corrected.

OPERATOR INSTRUCTION FORMS

A special form with which the programmer can give special instructions to operators is shown in Figure XVI-6. It is useful in giving instructions for the running of certain utility routines, and during test runs of new programs before the run books have been written for them. More compact tabulating card size forms, shown in Figure XVI-7, provides the operator with the same type of information. There is space for the operator to record his initials, the date, the running time of the program, and any special remarks about the operation of the run.

MEDIA CONTROL FORMS

Operation of the library for a computer system involves records for controlling and identifying the cards and tapes filed in the library. A form for identifying magnetic tape reels is shown in Figure XVI-8. The left-hand portion of this form is made up in triplicate, and the first copy has an adhesive backing. The right-hand portion of the form is in duplicate. When a programmer checks out a tape from the library for use, he fills out the form and fastens the adhesive-backed stub to the reel. He keeps one copy of the form, and gives one copy to the librarian (or operator). Systems which have a sizeable and active tape library use the information from these forms to punch tabulating cards which can be sorted and used to give various printouts of tape library information. A log of tape usage may be recorded on a form like that illustrated in Figure XVI-9.

DEBUGGING RECORD FORMS

When a program on the computer does not run as intended, it usually either stops unintentionally, or hangs up in a loop from which it can't escape. It was mentioned earlier that a record should be made of the condition of all indicators and switches on the operator console to aid the programmer in finding the error. A special reporting form showing the GE-235 console, as illustrated in Figure XVI-10, is handy for quickly recording this information.

When a test run is being made of a new program--or one in which changes are being made--the programmer may wish to give special instructions to the operator. A Debug Instruction Sheet, such as that shown in Figure XVI-11, permits the programmer to designate specifically, without being present, what is to be done by the operator in case an error is encountered.

Schedule of Operation

225 Machine No. 1
Date 4-12 62

6 AM	AM	PM	PM	PM
6:05	9:05 Engr'g	12:05 Systems	3:05 Small	6:05 PR + D
6:10	9:10	12:10 Devel.	3:10	6:10
6:15	9:15	12:15	3:15	6:15
6:20	9:20	12:20	3:20 ▼	6:20
6:25	9:25	12:25	3:25 Product	6:25
6:30	9:30	12:30 ▼	3:30 Research	6:30
6:35 PR + D	9:35	12:35 Richards	3:35	6:35
6:40	9:40	12:40	3:40	6:40
6:45	9:45	12:45 ▼	3:45	6:45
6:50	9:50	12:50 Samuels	3:50	6:50
6:55	9:55	12:55	3:55	6:55
7 AM	10 AM ▼	1 PM	4 PM	7 PM
7:05	10:05 1st. Nat'l	1:05	4:05	7:05
7:10	10:10 Bank	1:10	4:10	7:10
7:15	10:15	1:15 ▼	4:15	7:15
7:20	10:20	1:20 White	4:20	7:20
7:25	10:25	1:25	4:25	7:25
7:30	10:30	1:30	4:30	7:30
7:35	10:35	1:35 ▼	4:35	7:35
7:40	10:40	1:40 Paulson	4:40	7:40
7:45	10:45	1:45	4:45	7:45
7:50	10:50	1:50	4:50	7:50
7:55	10:55	1:55	4:55	7:55
8 AM	11 AM	2 PM	5 PM	8 PM
8:05	11:05	2:05 ▼	5:05	8:05
8:10	11:10	2:10 Davidson	5:10	8:10
8:15	11:15 ▼	2:15	5:15	8:15
8:20	11:20 Natural	2:20	5:20	8:20
8:25	11:25 Gas Co.	2:25	5:25	8:25
8:30 ▼	11:30	2:30 ▼	5:30 ▼	8:30
8:35 Accounting	11:35	2:35 Smith	5:35 Special	8:35
8:40	11:40	2:40	5:40 Systems	8:40
8:45	11:45	2:45	5:45	8:45 ▼
8:50	11:50	2:50	5:50	8:50
8:55	11:55	2:55	5:55	8:55
9 AM ▼	12 NOON ▼	3 PM ▼	6 PM ▼	9 PM

Figure XVI-1. Sample Schedule of Operation

DAILY COMPUTER LOG

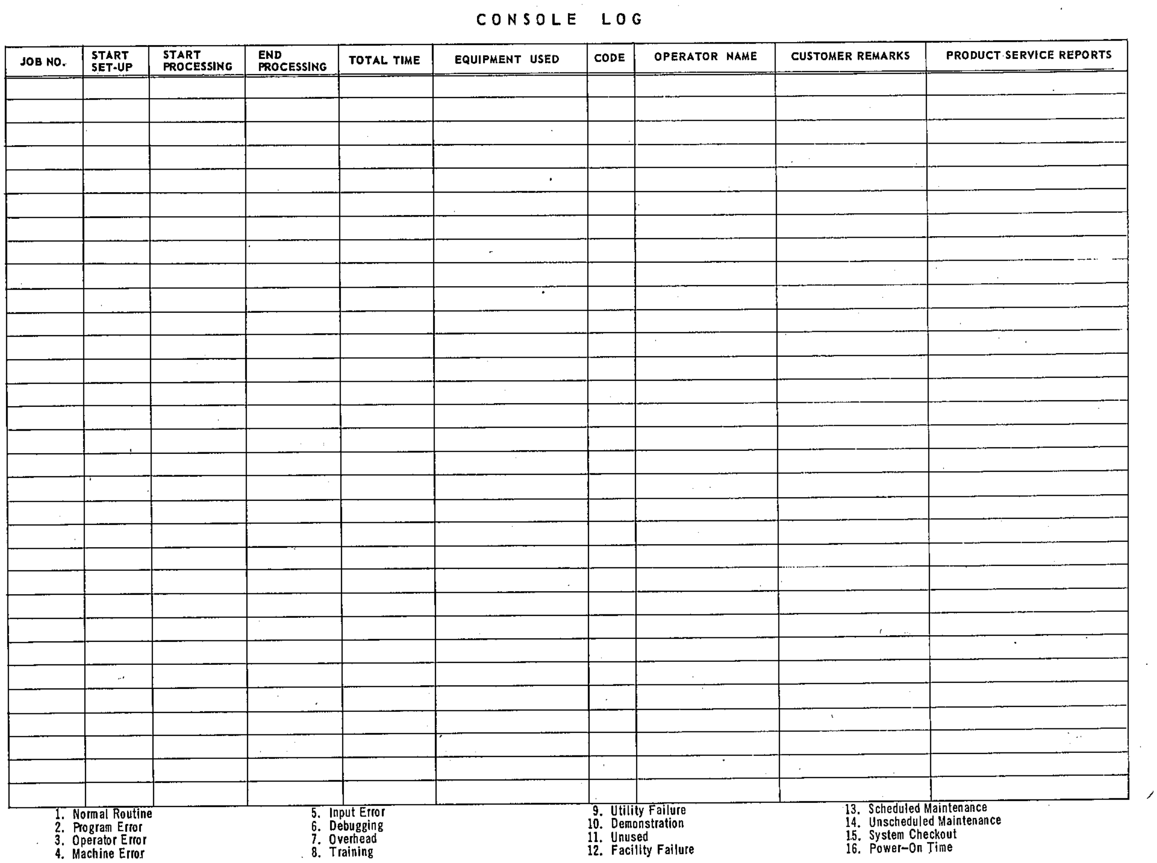
Shift First　　　　Date Aug. 20, 1962　　　　Operator G. Harris

1. Record all times
2. Fill out Hang-up sheet and Dump memory at all abnormal stops.
3. Explain all error time in remarks.

RUN DESCRIPTION	Programmer	Run Number	TIME RECORD			UTILIZATION			RERUN				MAINTENANCE			Remarks (Explain all Errors and Reruns)
			Start	Stop	Elapsed	Production	Debug	Idle	Program Error	Operator Error	Machine Error	Misc. Error	Scheduled	Unsched.	Facility Failure	
				8:03	3			3								
Posting	Roach	2010	8:03	8:33	30	30										
			8:33	9:03	30					30						Switch 3 was down
			TOTALS		63	30		3		30						

TOTAL RERUN 30　　　　TOTAL MAINT.

Figure XVI-2. Sample Daily Computer Log

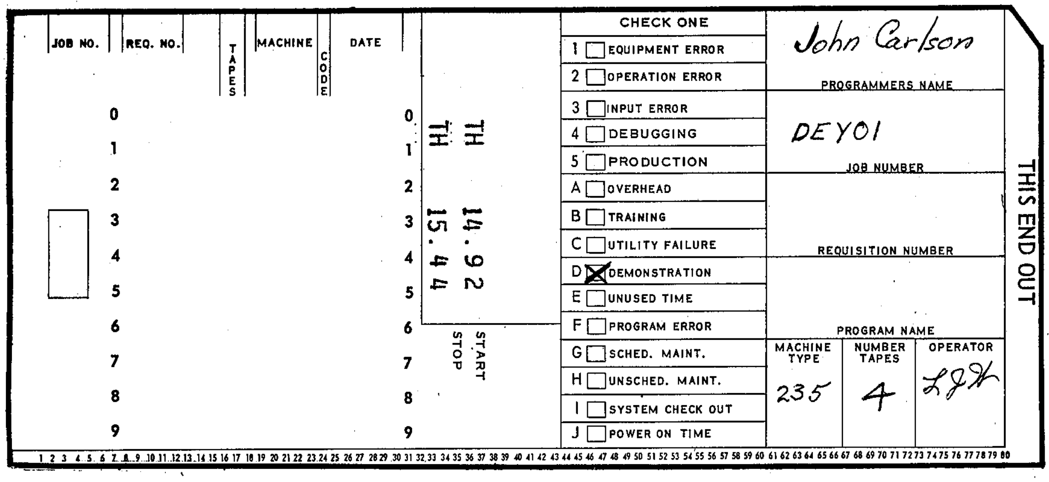


CONSOLE LOG

JOB NO.	START SET-UP	START PROCESSING	END PROCESSING	TOTAL TIME	EQUIPMENT USED	CODE	OPERATOR NAME	CUSTOMER REMARKS	PRODUCT SERVICE REPORTS

1. Normal Routine
2. Program Error
3. Operator Error
4. Machine Error
5. Input Error
6. Debugging
7. Overhead
8. Training
9. Utility Failure
10. Demonstration
11. Unused
12. Facility Failure
13. Scheduled Maintenance
14. Unscheduled Maintenance
15. System Checkout
16. Power-On Time

Figure XVI-3. Sample Operator and Maintenance Log



JOB NO. | REQ. NO. | TAPES | MACHINE | CODE | DATE

0 1 2 3 4 5 6 7 8 9

TH 14.92 START

TH 15.44 STOP

CHECK ONE

1 □ EQUIPMENT ERROR
2 □ OPERATION ERROR
3 □ INPUT ERROR
4 □ DEBUGGING
5 □ PRODUCTION
A □ OVERHEAD
B □ TRAINING
C □ UTILITY FAILURE
D ☒ DEMONSTRATION
E □ UNUSED TIME
F □ PROGRAM ERROR
G □ SCHED. MAINT.
H □ UNSCHED. MAINT.
I □ SYSTEM CHECK OUT
J □ POWER ON TIME

PROGRAMMERS NAME: John Carlson

JOB NUMBER: DEYOI

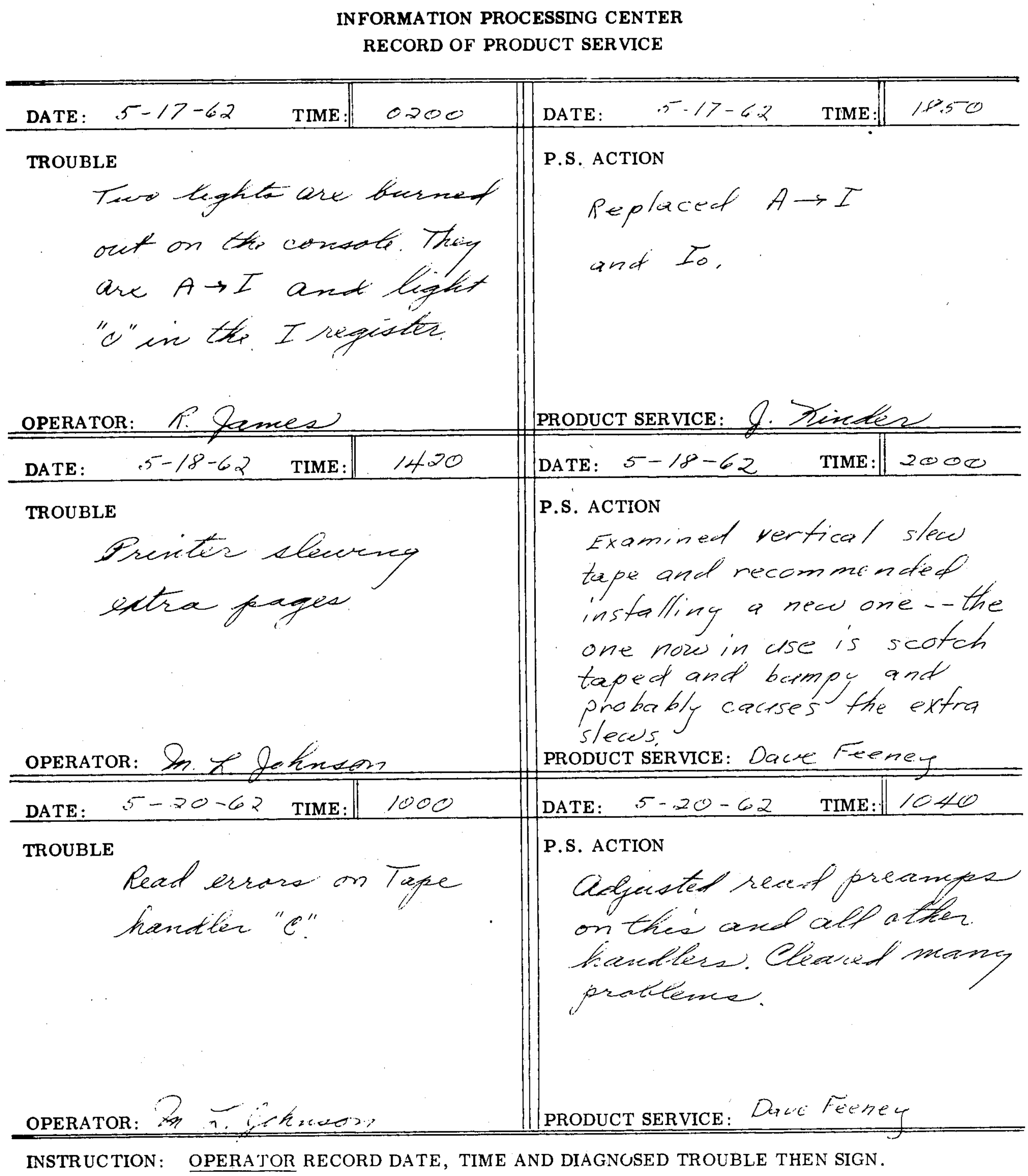
REQUISITION NUMBER:

PROGRAM NAME:

MACHINE TYPE	NUMBER TAPES	OPERATOR
235	4	

THIS END OUT

Figure XVI-4. Sample Time Card

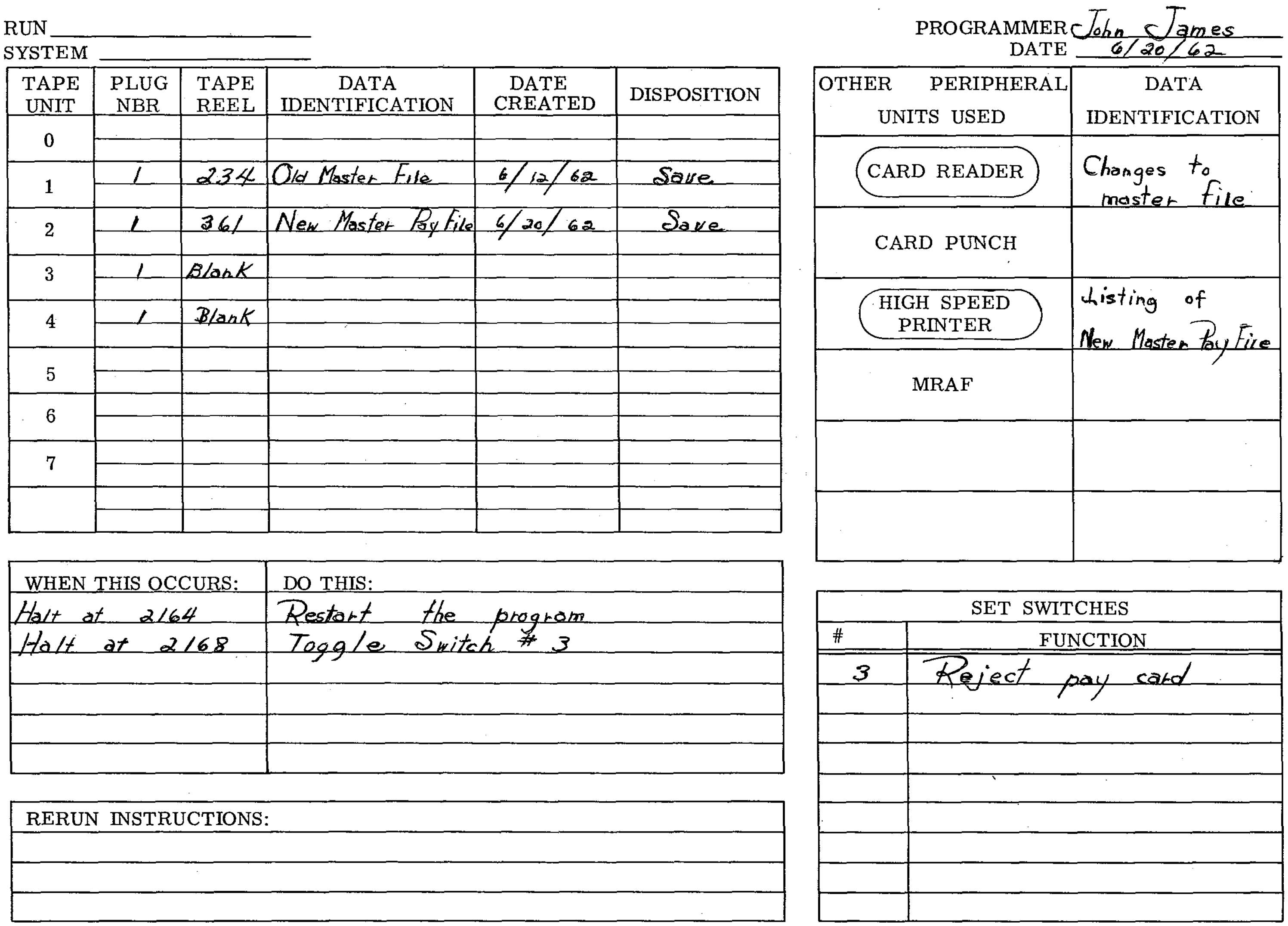


INFORMATION PROCESSING CENTER
RECORD OF PRODUCT SERVICE

DATE: 5-17-62 TIME: 0200	DATE: 5-17-62 TIME: 1850
TROUBLE Two lights are burned out on the console. They are A→I and light "0" in the I register.	P.S. ACTION Replaced A→I and I0,
OPERATOR: R. James	PRODUCT SERVICE: J. Kinder
DATE: 5-18-62 TIME: 1420	DATE: 5-18-62 TIME: 2000
TROUBLE Printer slewing extra pages	P.S. ACTION Examined vertical slew tape and recommended installing a new one--the one now in use is scotch taped and bumpy and probably causes the extra slews.
OPERATOR: M. L. Johnson	PRODUCT SERVICE: Dave Feeney
DATE: 5-20-62 TIME: 1000	DATE: 5-20-62 TIME: 1040
TROUBLE Read errors on Tape handler "C".	P.S. ACTION Adjusted read preamps on this and all other handlers. Cleared many problems.
OPERATOR: M. L. Johnson	PRODUCT SERVICE: Dave Feeney

INSTRUCTION: OPERATOR RECORD DATE, TIME AND DIAGNOSED TROUBLE THEN SIGN.
PRODUCT SERVICE RECORD DATE, TIME AND ACTION THEN SIGN.

Figure XVI-5. Sample Reporting Form ior Equipment Malfunction



RUN ____________________

SYSTEM ____________________

PROGRAMMER John James

DATE 6/20/62

TAPE UNIT	PLUG NBR	TAPE REEL	DATA IDENTIFICATION	DATE CREATED	DISPOSITION
0					
1	1	234	Old Master File	6/12/62	Save
2	1	361	New Master Pay File	6/20/62	Save
3	1	Blank			
4	1	Blank			
5					
6					
7					

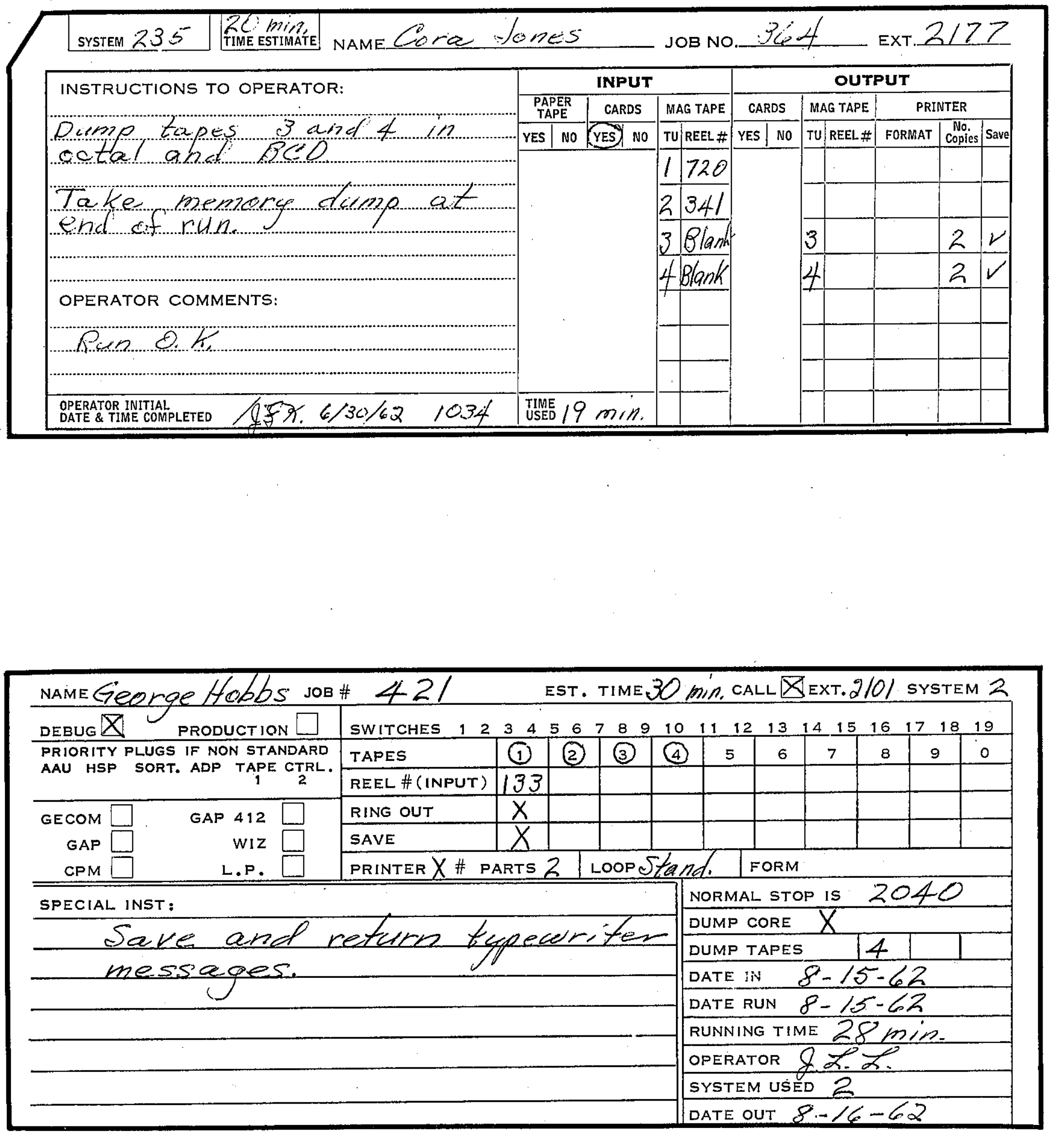
OTHER PERIPHERAL UNITS USED	DATA IDENTIFICATION
[✓ CARD READER]	Changes to master file
CARD PUNCH	
[✓ HIGH SPEED PRINTER]	Listing of New Master Pay File
MRAF	

WHEN THIS OCCURS:	DO THIS:
Halt at 2164	Restart the program
Halt at 2168	Toggle Switch # 3

SET SWITCHES	
#	FUNCTION
3	Reject pay card

RERUN INSTRUCTIONS:

Figure XVI-6. Operator Instruction Form



SYSTEM 235 | 20 min. TIME ESTIMATE | NAME Cora Jones | JOB NO. 364 | EXT. 2177

INSTRUCTIONS TO OPERATOR:

Dump tapes 3 and 4 in octal and BCD

Take memory dump at end of run.

OPERATOR COMMENTS:

Run O.K.

OPERATOR INITIAL DATE & TIME COMPLETED JFK. 6/30/62 1034

TIME USED 19 min.

INPUT						OUTPUT						
PAPER TAPE		CARDS		MAG TAPE		CARDS		MAG TAPE		PRINTER		
YES	NO	[✓ YES]	NO	TU	REEL #	YES	NO	TU	REEL #	FORMAT	No. Copies	Save
				1	720							
				2	341							
				3	Blank			3			2	✓
				4	Blank			4			2	✓

NAME George Hobbs JOB # 421 EST. TIME 30 min. CALL ☒ EXT. 2101 SYSTEM 2

DEBUG ☒ PRODUCTION □

PRIORITY PLUGS IF NON STANDARD
AAU HSP SORT. ADP TAPE CTRL. 1 2

GECOM □ GAP 412 □
GAP □ WIZ □
CPM □ L.P. □

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

TAPES	①	②	③	④	5	6	7	8	9	0
REEL # (INPUT)	133									
RING OUT	X									
SAVE	X									

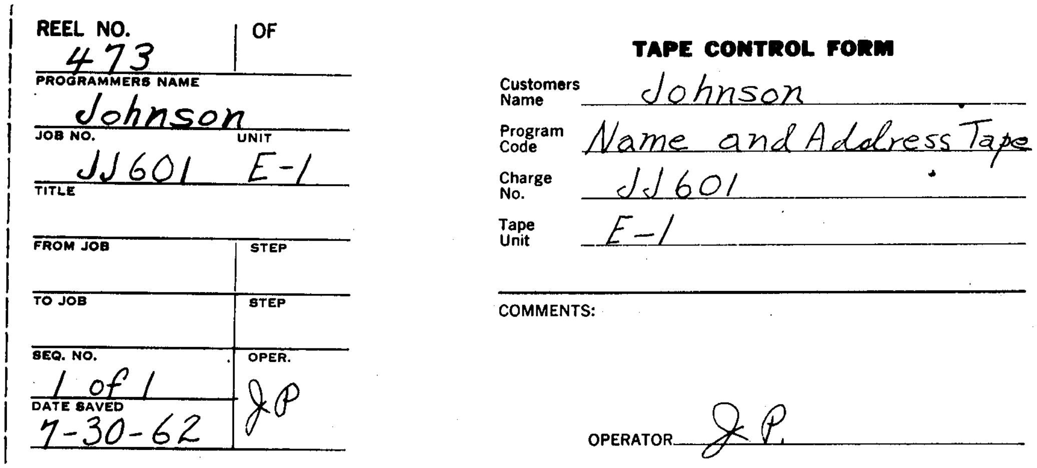
PRINTER X # PARTS 2 | LOOP Stand. | FORM

SPECIAL INST:

Save and return typewriter messages.

NORMAL STOP IS 2040
DUMP CORE X
DUMP TAPES 4
DATE IN 8-15-62
DATE RUN 8-15-62
RUNNING TIME 28 min.
OPERATOR J.L.L.
SYSTEM USED 2
DATE OUT 8-16-62

Figure XVI-7. Two Types of Operator Instruction Cards



REEL NO. 473 | OF

PROGRAMMERS NAME Johnson

JOB NO. JJ601 | UNIT E-1

TITLE

FROM JOB | STEP

TO JOB | STEP

SEQ. NO. 1 of 1 | OPER. JP

DATE SAVED 7-30-62

TAPE CONTROL FORM

Customers Name Johnson

Program Code Name and Address Tape

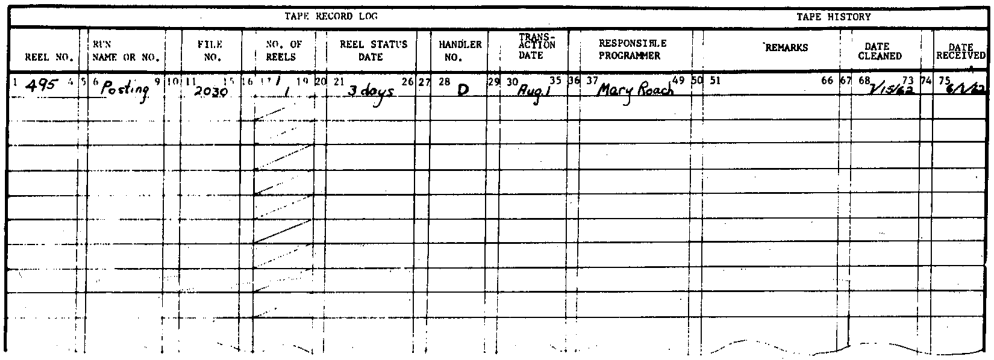
Charge No. JJ601

Tape Unit E-1

COMMENTS:

OPERATOR J.P.

Figure XVI-8. Sample Tape Control Form



TAPE RECORD LOG								TAPE HISTORY		
REEL NO.	RUN NAME OR NO.	FILE NO.	NO. OF REELS	REEL STATUS DATE	HANDLER NO.	TRANS-ACTION DATE	RESPONSIBLE PROGRAMMER	REMARKS	DATE CLEANED	DATE RECEIVED
1 495 4 5	6 Posting 9 10	11 2030 15 16	17 1 19 20	21 3 days 26 27	28 D 29	30 Aug. 1 35 36	37 Mary Roach 49 50	51 66 67	68 7/15/62 73 74	75 6/1/62

Figure XVI-9. Sample Log of Tape Use

HANG-UP SHEET

Posting Run NAME OR RUN # DATE 7/18/62

E. C. Jones PROGRAMMER TIME 1430

Caution: Be sure all information has been recorded before taking memory dump.

☒ ALERT HALT

□ BRANCH HALT

☒ LOOP (Try to give the Branches)

Remarks:

TAPE CONTROLLER

(⊗ LIGHT ON)

○ Ready	○ End of File	⊗ Alert
○ N Reg	⊗ Alert Last Record	○ Memory Alert
⊗ Lateral Parity	○ Horizontal Parity	MOD ○ Alert

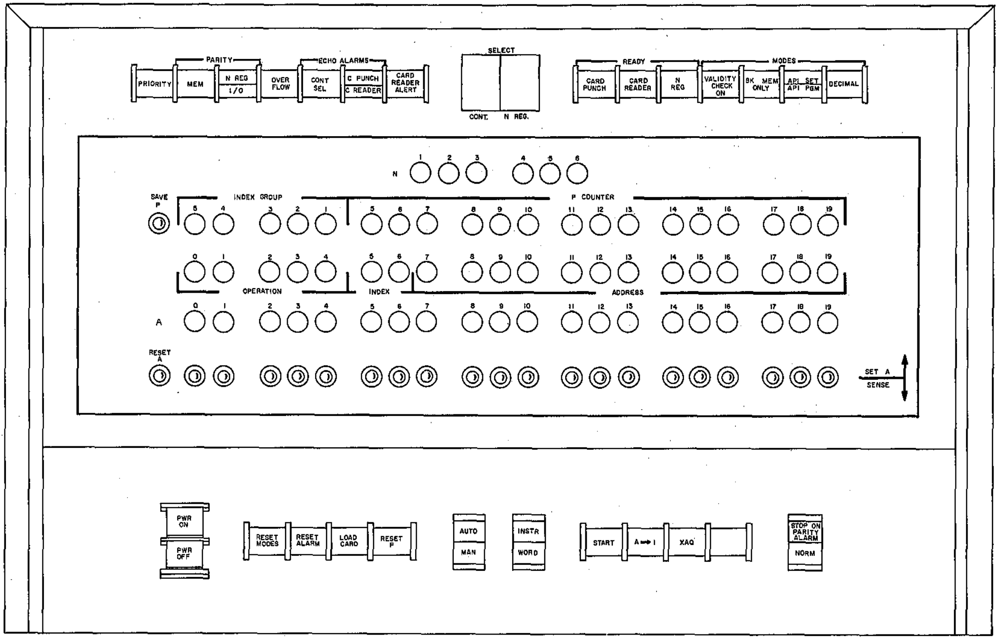
CONSOLE

BCD MODE

On □ Off ☒

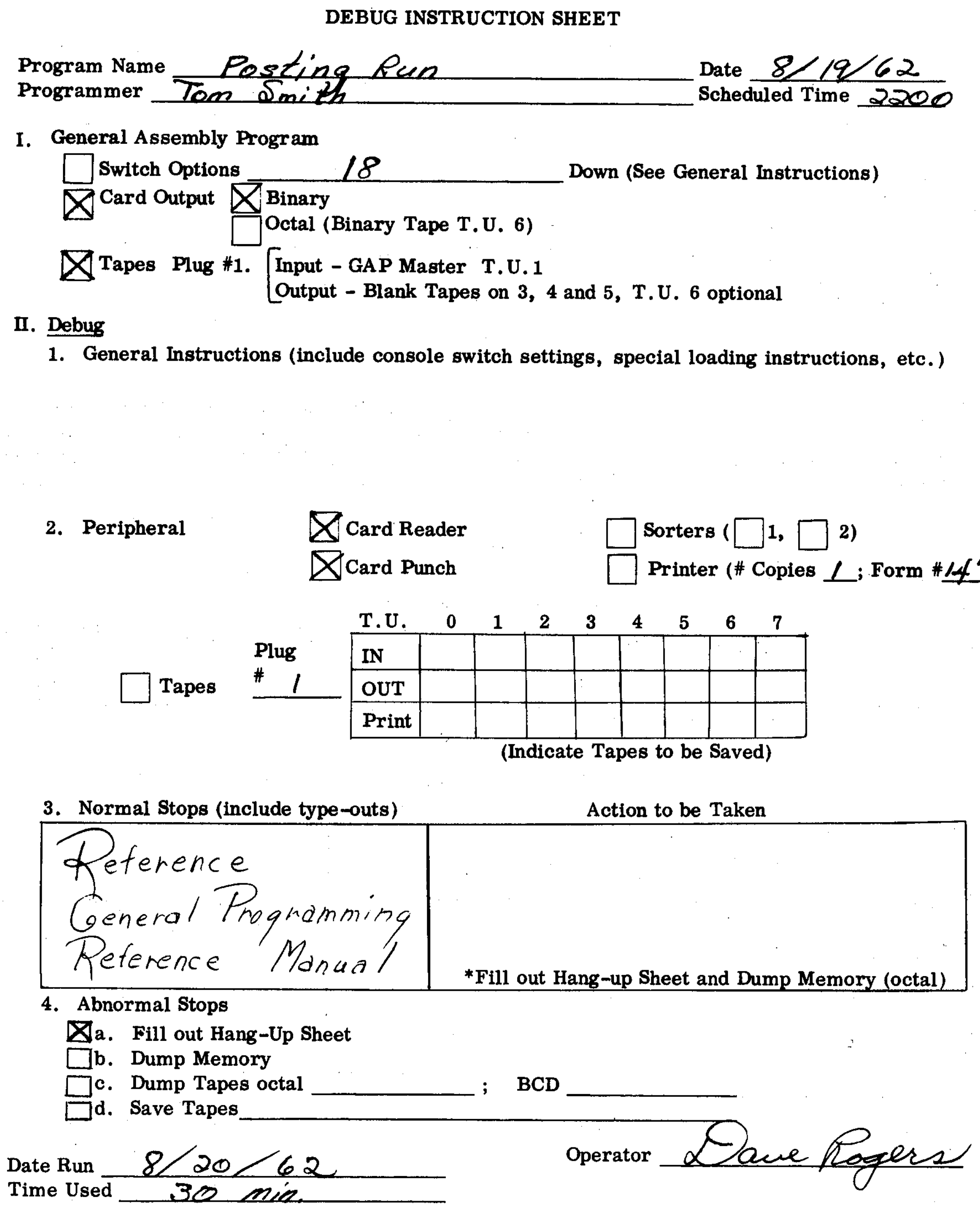
⊗ LIGHT ON

□ X For Switch Down



RUNNING TIME 30 min. OPERATOR Dave Rogers

Figure XVI-10. Sample Hang-up Sheet



DEBUG INSTRUCTION SHEET

Program Name Posting Run   Date 8/19/62
Programmer Tom Smith   Scheduled Time 2200

I. General Assembly Program

□ Switch Options 18 Down (See General Instructions)
☒ Card Output ☒ Binary
□ Octal (Binary Tape T.U. 6)
☒ Tapes Plug #1. Input - GAP Master T.U. 1
Output - Blank Tapes on 3, 4 and 5, T.U. 6 optional

II. Debug

1. General Instructions (include console switch settings, special loading instructions, etc.)

2. Peripheral ☒ Card Reader □ Sorters (□ 1, □ 2)
☒ Card Punch □ Printer (# Copies 1; Form # 14)

□ Tapes Plug # 1

T.U.	0	1	2	3	4	5	6	7
IN								
OUT								
Print								

(Indicate Tapes to be Saved)

3. Normal Stops (include type-outs)	Action to be Taken
Reference General Programming Reference Manual	*Fill out Hang-up Sheet and Dump Memory (octal)

4. Abnormal Stops
☒ a. Fill out Hang-Up Sheet
□ b. Dump Memory
□ c. Dump Tapes octal ________ ; BCD ________
□ d. Save Tapes ________

Date Run 8/20/62   Operator Dave Rogers
Time Used 30 min.

Figure XVI-11. Sample Debug Instruction Sheet

XVII. EMERGENCY PROCEDURES

The three general categories of emergency apt to be encountered by the staff of a computer center are: utilities failure, damage to the building or facilities, and injury to personnel. The staff should be thoroughly familiar with the procedures to be followed in all these situations. Telephone numbers of the various emergency services available should be posted in a conspicuous location in the computer area.

UTILITIES FAILURE

This situation falls into two general classes, power failure and air conditioning failure. If external power should fail, take the following steps:

1. Immediately turn off the master circuit breaker at the main power panel for the GE-235 system. This is necessary to avoid damage to the equipment.
2. Notify the appropriate service and maintenance personnel of the power failure.
3. When power is restored, follow the system startup procedure outlined in Chapter XII.

When air conditioning fails, the room temperature may rise to a point which endangers computer operation. Temperatures above 80 degrees Fahrenheit are considered too warm for continued operation. Take the following action:

1. Follow the normal shutdown procedure to remove power from the peripheral subsystems and the central processor.
2. Notify the appropriate service and maintenance personnel.
3. When the room temperature returns to normal, follow the normal startup procedure.

DAMAGE TO BUILDING

Damage to the building and facilities can result from fire, flood, earthquake, or a variety of other causes. Procedures for shutting down the system will depend on the urgency of the situation;

that is, whether the emergency is about to occur, occurring, or has already happened. In each situation, the operating staff should weigh the importance of saving data in the system against the amount of time remaining before the power must be shut off.

Since many paper forms and other flammable materials are in constant use in a computer area, the staff should exercise extreme caution in smoking, and discarding matches, ashes, cigarettes, etc.

If a fire starts in the computer room, take the following action:

1. Immediately shut off power with the master circuit breaker on the main power panel.
2. Notify the appropriate emergency services.
3. Use only carbon dioxide type fire extinguishers on fires in electrical equipment (never use any other type of extinguisher, such as water type, on an electrical fire).

If a flood occurs in the computer room or a wiring area (such as under a false floor), take the following steps:

1. If flooding is slight and gradual, follow the normal shutdown procedure.
2. If flooding is extensive and rapid, shut off power with the master circuit breaker on the main power panel.
3. Notify appropriate emergency personnel.
4. Evacuate personnel from the area as necessary.

In other emergency situations, the general procedure is to turn off power as soon as possible, notify emergency personnel, and evacuate the area as necessary.

INJURY TO PERSONNEL

In case of injury to personnel:

1. Do not move the injured person unless his position would cause further injury or otherwise endanger him
2. Follow the recommended first-aid practices (such as artificial respiration, etc.)
3. Notify the designated medical authorities
4. If conditions indicate a delay in continuing with computer operations, follow the normal shutdown procedures to remove power from the peripheral subsystems and the central processor.

SECTION D. REFERENCES

A. FLOW CHART CONVENTIONS

A standard set of symbols and notations has been developed by the General Electric Computer Department for the flow charts which appear in its publications. Illustrations of these symbols, and examples of how they are used, appear on the following pages. A template containing these symbols is available from GE Computer Department representatives.

The following is a list of recommended practices for drawing and documenting flow charts for GE-235 programming:

1. Flow charts should be prepared on 11- by 17-inch sheets and filed in program run books, or other documentation of each program.
2. Top level flow chart symbols should be used only in top level flow charts; detailed flow chart symbols should be used only in detailed flow charts.
3. The main path of flow on detailed flow charts should be from top to bottom of the page, starting at the top left-hand side. Horizontal branches from the main path should return to downward vertical flow.
4. Numbers, English words and abbreviations should be used to describe all operations, comparisons, decisions, etc. The only exception should be the use of the customary mathematical expressions shown on page A-3.
5. There should be only one entry to any detailed flow chart symbol except for the operation box; in general, however, the merge point is preferred.
6. Draw only one exit from any detailed flow chart symbol, except for the following operations:

   a. Decision
   b. Comparison
   c. Subroutine
   d. Variable connector

7. Detailed flow charts should be cross-referenced to the coding in a manner consistent with the programming language used (i.e. General Assembly Program symbolic names, TABSOL table names, etc.). This annotation should be placed at the upper left and outside of the symbol.

8. Detailed flow charts should be definitive enough to permit an inexperienced programmer to follow the complete logic.

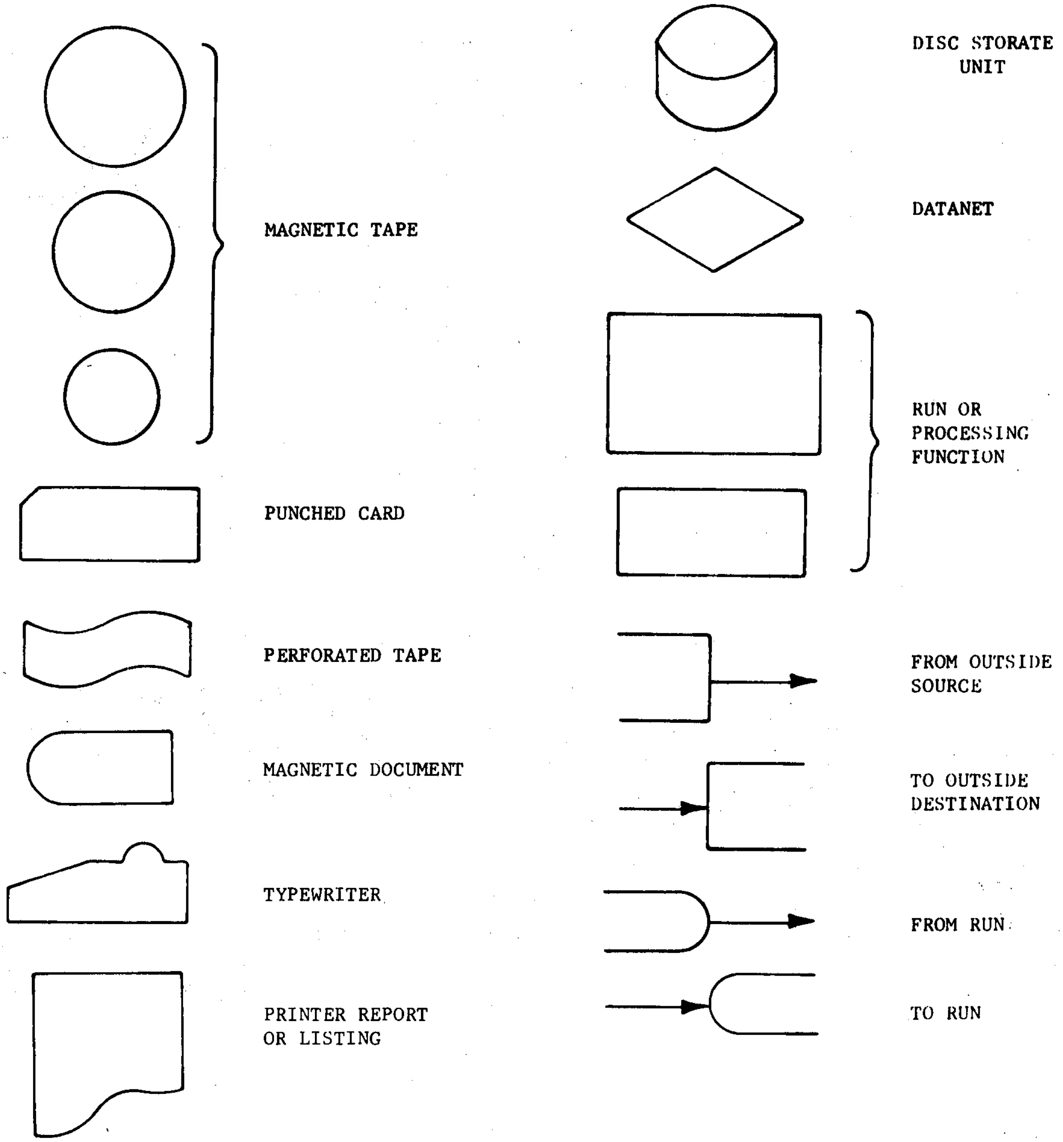


Figure A-1. Top Level Standard Flow Chart Symbols

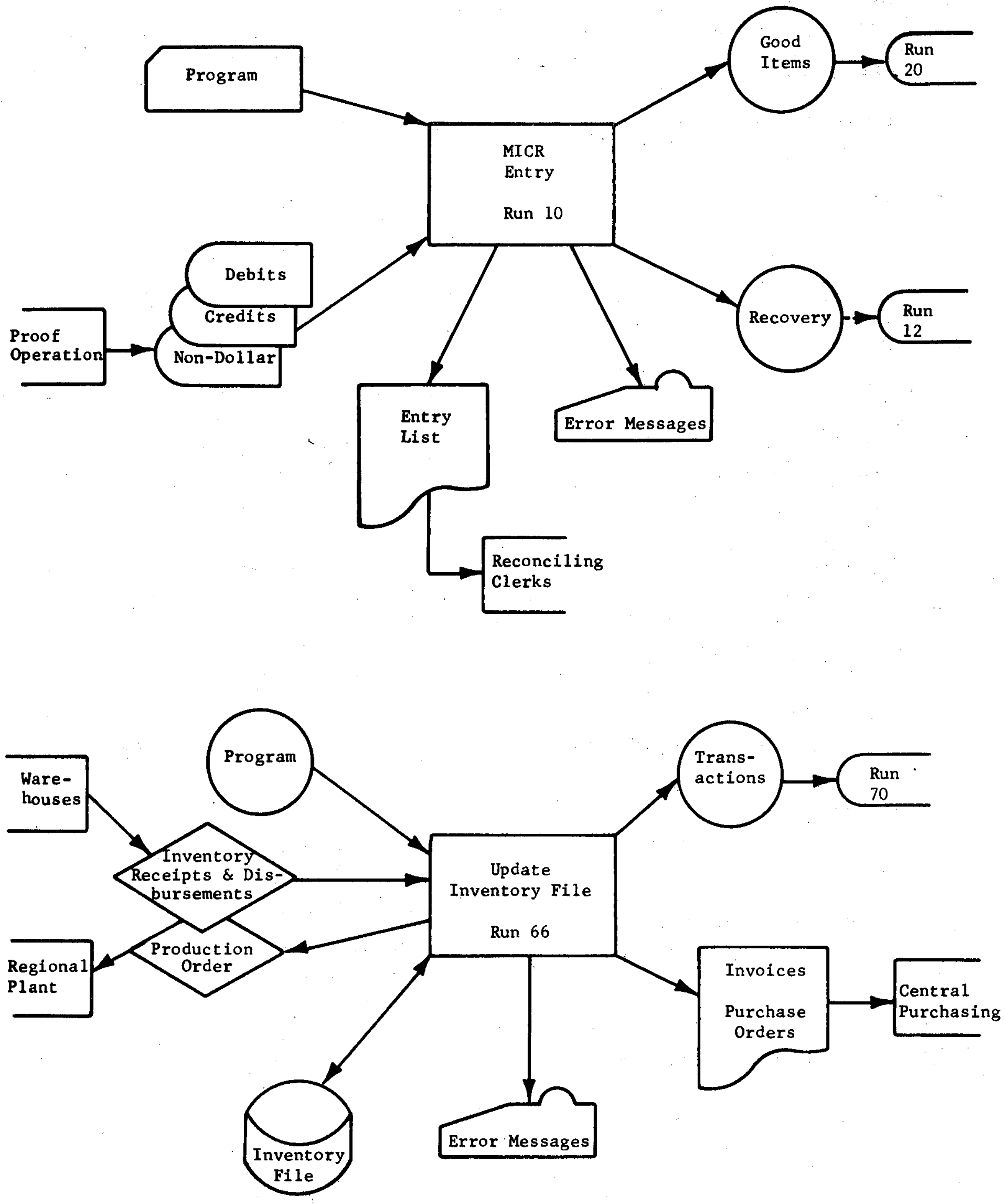
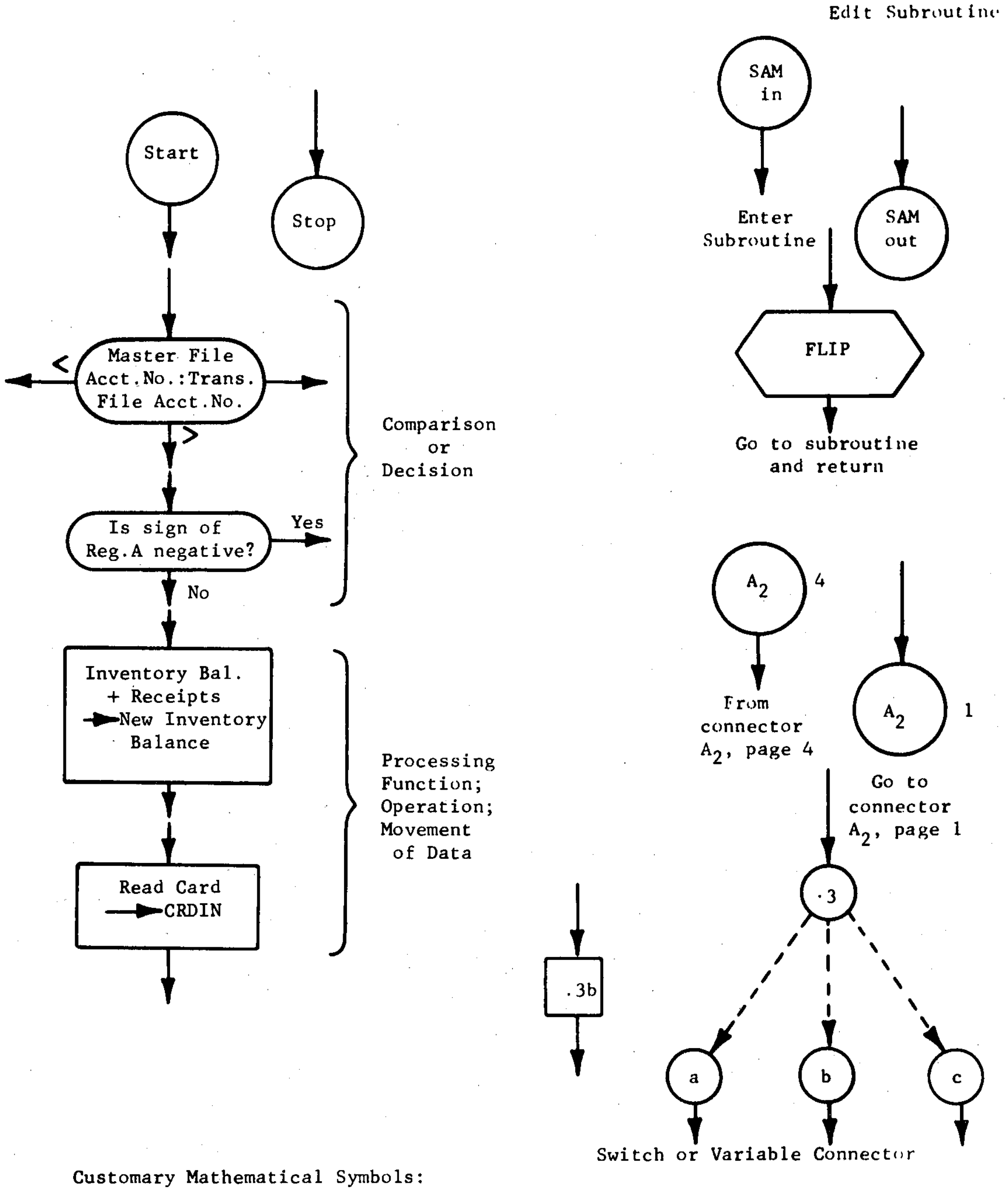


Figure A-2. Examples of Top Level Symbols in Flow Chart


Edit Subroutine
SAM in
Start
Stop
Enter Subroutine
SAM out
FLIP
Master File Acct.No.:Trans. File Acct.No.
<
>
Comparison or Decision
Go to subroutine and return
Is sign of Reg.A negative?
Yes
No
A2 4
Inventory Bal. + Receipts → New Inventory Balance
From connector A2, page 4
A2 1
Go to connector A2, page 1
Processing Function; Operation; Movement of Data
Read Card → CRDIN
.3
.3b
a
b
c
Switch or Variable Connector

Customary Mathematical Symbols:

:	Comparison	>	Greater Than	+	Plus	X	Multiplied By
=	Equal To	<	Less Than	-	Minus	÷	Divided By
≠	Not Equal To	≥	Greater Than or Equal	≤	Less Than or Equal	→	Replace

Figure A-3. Standard Detailed Logic Flow Chart Symbols

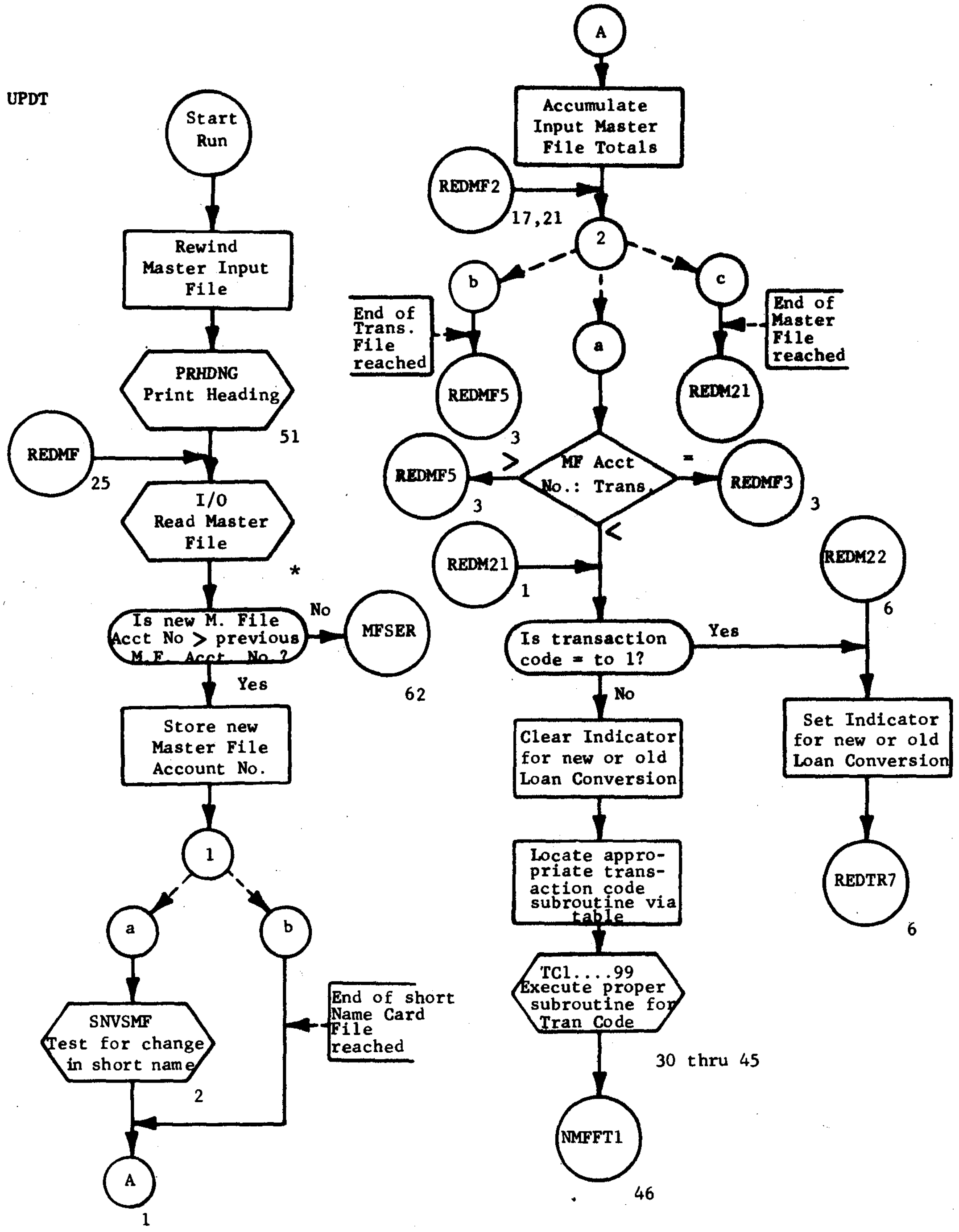

UPDT
Start Run
Rewind Master Input File
PRHDNG Print Heading
51
REDMF
25
I/O Read Master File
*
Is new M. File Acct No > previous M.F. Acct. No.?
No
MFSER
62
Yes
Store new Master File Account No.
1
a
b
SNVSMF Test for change in short name
2
End of short Name Card File reached
A
1
A
Accumulate Input Master File Totals
REDMF2
17,21
2
b
a
c
End of Trans. File reached
End of Master File reached
REDMF5
3
REDM21
REDMF5
3
>
MF Acct No.: Trans.
=
REDMF3
3
<
REDM21
1
REDM22
6
Is transaction code = to 1?
Yes
No
Clear Indicator for new or old Loan Conversion
Set Indicator for new or old Loan Conversion
REDTR7
6
Locate appropriate transaction code subroutine via table
TC1....99 Execute proper subroutine for Tran Code
30 thru 45
NMFFT1
46

Figure A-4. Examples of Detailed Logic Symbols in Flow Chart

B. NUMBER SYSTEMS

In Chapter 1 of this manual--covering the machine language of the GE-235 system--the importance of knowing and understanding the decimal, octal and binary number systems was stressed. Although the following section summarizes number systems and methods of conversion among them, there are several more extensive texts on the subject which may be reviewed.

Understanding Number Systems

To learn and understand new number systems, it is necessary to analyze principles which are true of all number systems:

1. A number is expressed as the sum of terms
2. Each term is the product of a digit times a base raised to a power
3. The base of a system is equal to the number of digits in the system
4. In any number system, the largest single digit is always one less than the base
5. The rightmost or least significant digit counts units. Each count in another column from the right contains a multiple of the base
6. Whenever any column holds the highest valued digit of a particular number system, and one is added to it, the column goes back to zero and develops a carry to the next most significant column.

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.

Any value less than infinity can be expressed in the decimal system by expanding the number of positional factors as far as necessary. The following table shows the relationship between positional factor and digit positions:

	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

As an 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	

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 B-1. Other arithmetic operations are similarly easy.

		Binary Digits	
	+	0	1
Binary Digits	0	0	1
	1	1	10

Figure B-1. Binary Addition Table

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

Example: Add the binary numbers 10110101 and 11010110.

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

Octal Number System

The octal number system uses eight digits, 0 through 7, and the base 8. Again, positional notation is similar to that of the decimal and binary systems. Successive positions in an octal number, from right to left, have values corresponding to increasing powers of 8. Thus the octal number 1376 is equal to 1 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 finite 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 B-2.

The table is useful in adding two octal numbers, which is the most common application the programmer will require. Note that if a sum of two octal numbers exceeds 7; the octal sum is greater than the decimal sum of the same two numbers by plus two (+2). If octal numbers are added or subtracted only two at a time, the above table will always apply. Also note that the above table serves for octal subtractions if one operand is chosen from the body of the table and the process is done in reverse. If one operand exceeds 7, the difference is less than the decimal difference of the same two numbers by minus two (-2).

		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 B-2. Octal Addition Table

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.

CONVERTING BETWEEN NUMBER SYSTEMS

Since the computer programmer is constantly converting between numbering systems, a discussion of the methods for doing conversions, and examples, is outlined in this text. Conversions may be done with arithmetic, or by using charts showing the conversion relationships between the systems. A useful table showing the powers of 8 and 2 appears in Figure B-3. In practice, most conversions use the octal number system as an intermediate step between the decimal and binary number systems.

The conversion examples and tables which follow use octal numbers below 3,777,777. This number represents the full 20-bit word length of the GE-235 system with all bits "on" (binary 11 111 111 111 111 111 111), and thus is sufficient for programming it.

Decimal-to-Octal Conversion

The usual procedure for converting a quantity expressed in one number system to another number system is to divide by the base number of the latter system. Thus, to convert a decimal number to the octal system, the decimal number is divided repeatedly by 8. The remainders from each division are combined in sequence from right to left to form an octal number which is equivalent to the decimal number.

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

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

Example: Convert 349, 798$_{10}$ to octal notation:

$8\overset{43724}{\overline{)349798}}$	Remainder is 6	→	6	1st remainder
$8\overset{5465}{\overline{)43724}}$	R 4	→	46	1st two remainders
$8\overset{683}{\overline{)5465}}$	R 1	→	146	1st three remainders
$8\overset{85}{\overline{)683}}$	R 3	→	3146	1st four remainders
$8\overset{10}{\overline{)85}}$	R 5	→	53146	1st five remainders
$8\overset{1}{\overline{)10}}$	R 2	→	253146	1st six remainders
$8\overset{0}{\overline{)1}}$	R 1	→	1253146_8	all remainders

Another method of conversion from decimal to octal notation involves the use of the conversion chart shown in Figure B-4. The octal equivalent of each decimal digit is located in the table. Addition is then performed, starting with the pair of equivalents having the lowest decimal position, as shown in the example below. The answer in octal notation is the same as obtained by the previous example, dividing by the octal base number 8.

Example: Convert 349,798$_{10}$ to octal notation using chart.

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	9	7	9	8	→						1	0
						→					1	3	2
											1	4	2
						→				1	2	7	4
										1	4	3	6
						→			2	1	4	5	0
									2	3	1	0	6
						→		1	1	6	1	0	0
								1	4	1	2	0	6
						→	1	1	1	1	7	4	0
							1	2	5	3	1	4	6_8

In the addition at the left, the octal equivalent of 8 in the 10^0 position is 10, and that of 9 in the 10^1 position is 132. Their sum is 142_8, and to this is added the 10^2 decimal position equivalent of 7 in octal notation, or 1274_8. Note in this second addition that the sum of 7 and 4 in the second column is octal 13, so the 3 is written down and the 1 is carried into the third column. **The octal equivalents of the remaining decimal numbers in their respective positions are obtained from the table and added in sequence. The answer following the fifth addition step is $1{,}253{,}146_8$.**

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 B-4. Decimal to Octal Conversion Charts

Octal-to-Decimal Conversion

Numbers expressed in octal notation can be converted to decimal notation by a mathematical process called expansion. This is performed by multiplying each octal digit by powers of 8 according to its position factor. The results of each such multiplication are then added together, the sum being the decimal equivalent of the octal number. Position factors were covered earlier in the information devoted to the octal number system.

To illustrate this process, a sample conversion of an octal number into its equivalent decimal number follows.

Example: Convert $1{,}253{,}146_8$ to decimal notation.

1 2 5 3 1 4 6

$6(8)^0$	=	6(1)	=	6
$4(8)^1$	=	4(8)	=	32
$1(8)^2$	=	1(64)	=	64
$3(8)^3$	=	3(512)	=	1,536
$5(8)^4$	=	5(4,096)	=	20,480
$2(8)^5$	=	2(32,768)	=	65,536
$1(8)^6$	=	1(262,144)	=	262,144
				$349{,}798_{10}$

Another method of octal-to-decimal conversion is to use a conversion table and merely look up the equivalent decimal number. The conversion table in Figure B-5 has been compiled for converting octal numbers up to 3,777,777 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.

Example: Convert $1{,}761{,}354_8$ to decimal notation.

Octal Positions

8^6	8^5	8^4	8^3	8^2	8^1	8^0
1	7	6	1	3	5	4

Decimal Positions

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

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 B-5. Octal-to-Decimal Conversion Chart

Octal-to-Binary Conversion

The converting of numbers from octal notation to their equivalent in binary notation is simply a process of substituting the binary equivalent of each octal number. Since octal digits range from 0 to 7, each binary equivalent will have three digits, or bits. Beginning with the right-hand digit of the octal number, each digit is converted to its binary equivalent. It is a simple matter to memorize the binary equivalent of each octal number, as shown below.

Example: Convert 1234567_8 into binary notation.

Octal Number

1	2	3	4	5	6	7
↓	↓	↓	↓	↓	↓	↓
001	010	011	100	101	110	111

binary equivalent

Binary-to-Octal Conversion

By reversing the above process, conversion from binary notation back to octal is simplified. Each group of three binary bits becomes one octal digit in the range of 0 to 7 in octal notation.

Or, a conversion weighting method can be used for converting binary numbers to octal notation. As above, each group of three binary bits is evaluated individually; the right-most bit has a weight of 1, the center bit is 2, and the left-most bit equals 4. These weights apply only when a 1 bit is shown in each position; 0 bits are not counted. For a 101 binary number, the weighting would add as follows.

4	2	1	Conversion weight
1	0	1	number in binary
4	+0	+1	substituting conversion weights
	5		equivalent in octal

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

Decimal-to-Binary Conversion

Undoubtedly the simplest method of converting a decimal number to binary is to perform the following two steps:

1. Convert the decimal number to octal by repeated division by 8, using the remainder from each division in sequence from right to left to form the octal equivalent of the decimal number.
2. Convert the octal number to binary by inspection of each digit and copying its binary equivalent. The result is a binary notation of the decimal number.

Example: Find the binary equivalent of the decimal 53.

1. $\begin{array}{r} 6 \\ 8\overline{)53} \end{array} \longrightarrow 5$ 1st remainder

   $\begin{array}{r} 0 \\ 8\overline{)6} \end{array} \longrightarrow 65$ all remainders

   Thus, 65_8 is the octal equivalent of 53_{10}

2. In 65_8, $6 = 110_2$, and $5 = 101_2$.

   So, . $53_{10} = 65_8 = 110101_2$.

Binary-to-Decimal Conversion

Binary numbers can be converted to decimal by the same two steps as for decimal-to-binary conversion simply by reversing the steps, as follows.

1. Convert the binary number to octal form by inspection.

2. Convert the octal number to decimal by expansion; i.e., multiplying the coefficients of the number expressed by the base, raised to its proper power. The result is the decimal equivalent of the binary number.

Example: Find the decimal equivalent of the binary number 101111011_2.

```
1.     101  111  011
        5    7    3   =  573₈
```

$$
\begin{aligned}
2.\quad 573_8 &= 5(8^2) + 7(8^1) + 3(8^0) \\
&= 5(64) + 7(8) + 3(1) \\
&= 320 + 56 + 3 \\
&= 379_{10}
\end{aligned}
$$

Thus, $101111011_2 = 379_{10}$.

Another method would be simply to look up the decimal equivalents of the corresponding powers of two in the table shown in Figure B-6 and add, as shown below. However, this method is mainly useful where such a conversion table is available.

Example: Convert 101111011_2 to decimal notation.

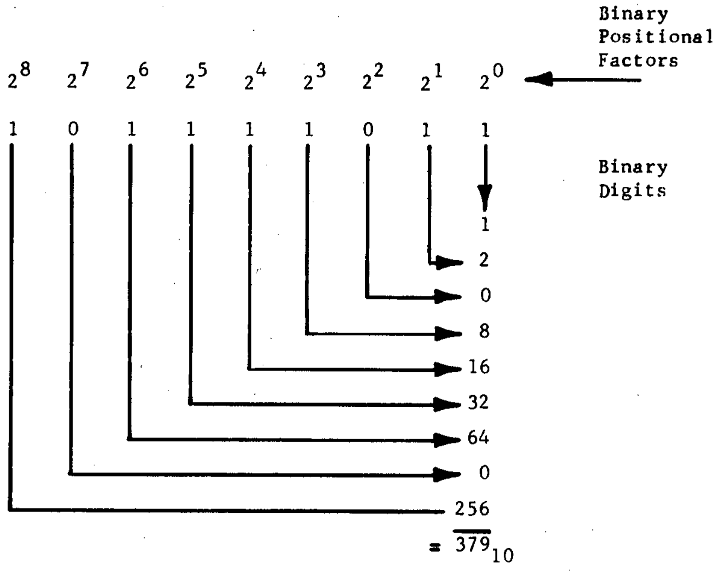


TABLE OF CONVERSION METHODS FOR MATHEMATICS

Decimal - Octal - Binary

FROM	TO	METHOD
Decimal Whole Numbers	Octal Whole Numbers	Repeated division by 8
Decimal Fractional Numbers	Octal Fractional Numbers	Repeated multiplication by 8
Octal Whole Numbers	Decimal Whole Numbers	Expansion
Octal Fractional Numbers	Decimal Fractional Numbers	Repeated multiplication by 12_8
Octal Numbers	Binary Numbers	Inspection
Binary Numbers	Octal Numbers	Inspection
Decimal Numbers	Binary Numbers	Convert to Octal numbers first; then follow Octal-to-Binary method above.
Binary Numbers	Decimal Numbers	Convert to Octal numbers first; then use Octal-to-Decimal method above.

Decimal-to-Binary Equivalents

$1 = 1_2$ (or 001_2)

$2 = 10_2$ (or 010_2)

$3 = 11_2$ (or 011_2)

$4 = 100_2$

$5 = 101_2$

$6 = 110_2$

$7 = 111_2$

$8 = 1{,}000_2$ (or $001{,}000_2$)

$9 = 1{,}001_2$ (or $001{,}001_2$)

$10 = 1{,}010_2$ (or $001{,}010_2$)

$11 = 1{,}011_2$ (or $001{,}011_2$)

$12 = 1{,}100_2$ (or $001{,}100_2$)

$13 = 1{,}101_2$ (or $001{,}101_2$)

$14 = 1{,}110_2$ (or $001{,}110_2$)

$15 = 1{,}111_2$ (or $001{,}111_2$)

$16 = 10{,}000_2$ (or $010{,}000_2$)

$17 = 10{,}001_2$ (or $010{,}001_2$)

$18 = 10{,}010_2$ (or $010{,}010_2$)

$19 = 10{,}011_2$ (or $010{,}011_2$)

$20 = 10{,}100_2$ (or $010{,}100_2$)

Figure B-6. Conversion Methods for Mathematics

Table of Powers of 2

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5

Octal-Decimal Integer Conversion Table

Octal	10000	20000	30000	40000	50000	60000	70000
Decimal	4096	8192	12288	16384	20480	24576	28672

Octal	0000 to 0377
Decimal	0000 to 0255

Octal	0	1	2	3	4	5	6	7
0000	0000	0001	0002	0003	0004	0005	0006	0007
0010	0008	0009	0010	0011	0012	0013	0014	0015
0020	0016	0017	0018	0019	0020	0021	0022	0023
0030	0024	0025	0026	0027	0028	0029	0030	0031
0040	0032	0033	0034	0035	0036	0037	0038	0039
0050	0040	0041	0042	0043	0044	0045	0046	0047
0060	0048	0049	0050	0051	0052	0053	0054	0055
0070	0056	0057	0058	0059	0060	0061	0062	0063
0100	0064	0065	0066	0067	0068	0069	0070	0071
0110	0072	0073	0074	0075	0076	0077	0078	0079
0120	0080	0081	0082	0083	0084	0085	0086	0087
0130	0088	0089	0090	0091	0092	0093	0094	0095
0140	0096	0097	0098	0099	0100	0101	0102	0103
0150	0104	0105	0106	0107	0108	0109	0110	0111
0160	0112	0113	0114	0115	0116	0117	0118	0119
0170	0120	0121	0122	0123	0124	0125	0126	0127
0200	0128	0129	0130	0131	0132	0133	0134	0135
0210	0136	0137	0138	0139	0140	0141	0142	0143
0220	0144	0145	0146	0147	0148	0149	0150	0151
0230	0152	0153	0154	0155	0156	0157	0158	0159
0240	0160	0161	0162	0163	0164	0165	0166	0167
0250	0168	0169	0170	0171	0172	0173	0174	0175
0260	0176	0177	0178	0179	0180	0181	0182	0183
0270	0184	0185	0186	0187	0188	0189	0190	0191
0300	0192	0193	0194	0195	0196	0197	0198	0199
0310	0200	0201	0202	0203	0204	0205	0206	0207
0320	0208	0209	0210	0211	0212	0213	0214	0215
0330	0216	0217	0218	0219	0220	0221	0222	0223
0340	0224	0225	0226	0227	0228	0229	0230	0231
0350	0232	0233	0234	0235	0236	0237	0238	0239
0360	0240	0241	0242	0243	0244	0245	0246	0247
0370	0248	0249	0250	0251	0252	0253	0254	0255

Octal	1000 to 1377
Decimal	0512 to 0767

Octal	0	1	2	3	4	5	6	7
1000	0512	0513	0514	0515	0516	0517	0518	0519
1010	0520	0521	0522	0523	0524	0525	0526	0527
1020	0528	0529	0530	0531	0532	0533	0534	0535
1030	0536	0537	0538	0539	0540	0541	0542	0543
1040	0544	0545	0546	0547	0548	0549	0550	0551
1050	0552	0553	0554	0555	0556	0557	0558	0559
1060	0560	0561	0562	0563	0564	0565	0566	0567
1070	0568	0569	0570	0571	0572	0573	0574	0575
1100	0576	0577	0578	0579	0580	0581	0582	0583
1110	0584	0585	0586	0587	0588	0589	0590	0591
1120	0592	0593	0594	0595	0596	0597	0598	0599
1130	0600	0601	0602	0603	0604	0605	0606	0607
1140	0608	0609	0610	0611	0612	0613	0614	0615
1150	0616	0617	0618	0619	0620	0621	0622	0623
1160	0624	0625	0626	0627	0628	0629	0630	0631
1170	0632	0633	0634	0635	0636	0637	0638	0639
1200	0640	0641	0642	0643	0644	0645	0646	0647
1210	0648	0149	0650	0651	0652	0653	0654	0655
1220	0656	0657	0658	0659	0660	0661	0662	0663
1230	0664	0665	0666	0667	0668	0669	0670	0671
1240	0672	0673	0674	0675	0676	0677	0678	0679
1250	0680	0681	0682	0683	0684	0685	0686	0687
1260	0688	0689	0690	0691	0692	0693	0694	0695
1270	0696	0697	0698	0699	0700	0701	0702	0703
1300	0704	0705	0706	0707	0708	0709	0710	0711
1310	0712	0713	0714	0715	0716	0717	0718	0719
1320	0720	0721	0722	0723	0724	0725	0726	0727
1330	0728	0729	0730	0731	0732	0733	0734	0735
1340	0736	0737	0738	0739	0740	0741	0742	0743
1350	0744	0745	0746	0747	0748	0749	0750	0751
1360	0752	0753	0754	0755	0756	0757	0758	0759
1370	0760	0761	0762	0763	0764	0765	0766	0767

Octal	0400 to 0777
Decimal	0256 to 0511

Octal	0	1	2	3	4	5	6	7
0400	0256	0257	0258	0259	0260	0261	0262	0263
0410	0264	0265	0266	0267	0268	0269	0270	0271
0420	0272	0273	0274	0275	0276	0277	0278	0279
0430	0280	0281	0282	0283	0284	0285	0286	0287
0440	0288	0289	0290	0291	0292	0293	0294	0295
0450	0296	0297	0298	0299	0300	0301	0302	0303
0460	0304	0305	0306	0307	0308	0309	0310	0311
0470	0312	0313	0314	0315	0316	0317	0318	0319
0500	0320	0321	0322	0323	0324	0325	0326	0327
0510	0328	0329	0330	0331	0332	0333	0334	0335
0520	0336	0337	0338	0339	0340	0341	0342	0343
0530	0344	0345	0346	0347	0348	0349	0350	0351
0540	0352	0353	0354	0355	0356	0357	0358	0359
0550	0360	0361	0362	0363	0364	0365	0366	0367
0560	0368	0369	0370	0371	0372	0373	0374	0375
0570	0376	0377	0378	0379	0380	0381	0382	0383
0600	0384	0385	0386	0387	0388	0389	0390	0391
0610	0392	0393	0394	0395	0396	0397	0398	0399
0620	0400	0401	0402	0403	0404	0405	0406	0407
0630	0408	0409	0410	0411	0412	0413	0414	0415
0640	0416	0417	0418	0419	0420	0421	0422	0423
0650	0424	0425	0426	0427	0428	0429	0430	0431
0660	0432	0433	0434	0435	0436	0437	0438	0439
0670	0440	0441	0442	0443	0444	0445	0446	0447
0700	0448	0449	0450	0451	0452	0453	0454	0455
0710	0456	0457	0458	0459	0460	0461	0462	0463
0720	0464	0465	0466	0467	0468	0469	0470	0471
0730	0472	0473	0474	0475	0476	0477	0478	0479
0740	0480	0481	0482	0483	0484	0485	0486	0487
0750	0488	0489	0490	0491	0492	0493	0494	0495
0760	0496	0497	0498	0499	0500	0501	0502	0503
0770	0504	0505	0506	0507	0508	0509	0510	0511

Octal	1400 to 1777
Decimal	0768 to 1023

Octal	0	1	2	3	4	5	6	7
1400	0768	0769	0770	0771	0772	0773	0774	0775
1410	0776	0777	0778	0779	0780	0781	0782	0783
1420	0784	0785	0786	0787	0788	0789	0790	0791
1430	0792	0793	0794	0795	0796	0797	0798	0799
1440	0800	0801	0802	0803	0804	0805	0806	0807
1450	0808	0809	0810	0811	0812	0813	0814	0815
1460	0816	0817	0818	0819	0820	0821	0822	0823
1470	0824	0825	0826	0827	0828	0829	0830	0831
1500	0832	0833	0834	0835	0836	0837	0838	0839
1510	0840	0841	0842	0843	0844	0845	0846	0847
1520	0848	0849	0850	0851	0852	0853	0854	0855
1530	0856	0857	0858	0859	0860	0861	0862	0863
1540	0864	0865	0866	0867	0868	0869	0870	0871
1550	0872	0873	0874	0875	0876	0877	0878	0879
1560	0880	0881	0882	0883	0884	0885	0886	0887
1570	0888	0889	0890	0891	0892	0893	0894	0895
1600	0896	0897	0898	0899	0900	0901	0902	0903
1610	0904	0905	0906	0907	0908	0909	0910	0911
1620	0912	0913	0914	0915	0916	0917	0918	0919
1630	0920	0921	0922	0923	0924	0925	0926	0927
1640	0928	0929	0930	0931	0932	0933	0934	0935
1650	0936	0937	0938	0939	0940	0941	0942	0943
1660	0944	0945	0946	0947	0948	0949	0950	0951
1670	0952	0953	0954	0955	0956	0957	0958	0959
1700	0960	0961	0962	0963	0964	0965	0966	0967
1710	0968	0969	0970	0971	0972	0973	0974	0975
1720	0976	0977	0978	0979	0980	0981	0982	0983
1730	0984	0985	0986	0987	0988	0989	0990	0991
1740	0992	0993	0994	0995	0996	0997	0998	0999
1750	1000	1001	1002	1003	1004	1005	1006	1007
1760	1008	1009	1010	1011	1012	1013	1014	1015
1770	1016	1017	1018	1019	1020	1021	1022	1023

Octal-Decimal Integer Conversion Table

Octal	10000	20000	30000	40000	50000	60000	70000
Decimal	4096	8192	12288	16384	20480	24576	28672

Octal	2000 to 2377
Decimal	1024 to 1279

Octal	0	1	2	3	4	5	6	7
2000	1024	1025	1026	1027	1028	1029	1030	1031
2010	1032	1033	1034	1035	1036	1037	1038	1039
2020	1040	1041	1042	1043	1044	1045	1046	1047
2030	1048	1049	1050	1051	1052	1053	1054	1055
2040	1056	1057	1058	1059	1060	1061	1062	1063
2050	1064	1065	1066	1067	1068	1069	1070	1071
2060	1072	1073	1074	1075	1076	1077	1078	1079
2070	1080	1081	1082	1083	1084	1085	1086	1087
2100	1088	1089	1090	1091	1092	1093	1094	1095
2110	1096	1097	1098	1099	1100	1101	1102	1103
2120	1104	1105	1106	1107	1108	1109	1110	1111
2130	1112	1113	1114	1115	1116	1117	1118	1119
2140	1120	1121	1122	1123	1124	1125	1126	1127
2150	1128	1129	1130	1131	1132	1133	1134	1135
2160	1136	1137	1138	1139	1140	1141	1142	1143
2170	1144	1145	1146	1147	1148	1149	1150	1151
2200	1152	1153	1154	1155	1156	1157	1158	1159
2210	1160	1161	1162	1163	1164	1165	1166	1167
2220	1168	1169	1170	1171	1172	1173	1174	1175
2230	1176	1177	1178	1179	1180	1181	1182	1183
2240	1184	1185	1186	1187	1188	1189	1190	1191
2250	1192	1193	1194	1195	1196	1197	1198	1199
2260	1200	1201	1202	1203	1204	1205	1206	1207
2270	1208	1209	1210	1211	1212	1213	1214	1215
2300	1216	1217	1218	1219	1220	1221	1222	1223
2310	1224	1225	1226	1227	1228	1229	1230	1231
2320	1232	1233	1234	1235	1236	1237	1238	1239
2330	1240	1241	1242	1243	1244	1245	1246	1247
2340	1248	1249	1250	1251	1252	1253	1254	1255
2350	1256	1257	1258	1259	1260	1261	1262	1263
2360	1264	1265	1266	1267	1268	1269	1270	1271
2370	1272	1273	1274	1275	1276	1277	1278	1279

Octal	2400 to 2777
Decimal	1280 to 1535

Octal	0	1	2	3	4	5	6	7
2400	1280	1281	1282	1283	1284	1285	1286	1287
2410	1288	1289	1290	1291	1292	1293	1294	1295
2420	1296	1297	1298	1299	1300	1301	1302	1303
2430	1304	1305	1306	1307	1308	1309	1310	1311
2440	1312	1313	1314	1315	1316	1317	1318	1319
2450	1320	1321	1322	1323	1324	1325	1326	1327
2460	1328	1329	1330	1331	1332	1333	1334	1335
2470	1336	1337	1338	1339	1340	1341	1342	1343
2500	1344	1345	1346	1347	1348	1349	1350	1351
2510	1352	1353	1354	1355	1356	1357	1358	1359
2520	1360	1361	1362	1363	1364	1365	1366	1367
2530	1368	1369	1370	1371	1372	1373	1374	1375
2540	1376	1377	1378	1379	1380	1381	1382	1383
2550	1384	1385	1386	1387	1388	1389	1390	1391
2560	1392	1393	1394	1395	1396	1397	1398	1399
2570	1400	1401	1402	1403	1404	1405	1406	1407
2600	1408	1409	1410	1411	1412	1413	1414	1415
2610	1416	1417	1418	1419	1420	1421	1422	1423
2620	1424	1425	1426	1427	1428	1429	1430	1431
2630	1432	1433	1434	1435	1436	1437	1438	1439
2640	1440	1441	1442	1443	1444	1445	1446	1447
2650	1448	1449	1450	1451	1452	1453	1454	1455
2660	1456	1457	1458	1459	1460	1461	1462	1463
2670	1464	1465	1466	1467	1468	1469	1470	1471
2700	1472	1473	1474	1475	1476	1477	1478	1479
2710	1480	1481	1482	1483	1484	1485	1486	1487
2720	1488	1489	1490	1491	1492	1493	1494	1495
2730	1496	1497	1498	1499	1500	1501	1502	1503
2740	1504	1505	1506	1507	1508	1509	1510	1511
2750	1512	1513	1514	1515	1516	1517	1518	1519
2760	1520	1521	1522	1523	1524	1525	1526	1527
2770	1528	1529	1530	1531	1532	1533	1534	1535

Octal	3000 to 3377
Decimal	1356 to 1791

Octal	0	1	2	3	4	5	6	7
3000	1536	1537	1538	1539	1540	1541	1542	1543
3010	1544	1545	1546	1547	1548	1549	1550	1551
3020	1552	1553	1554	1555	1556	1557	1558	1559
3030	1560	1561	1562	1563	1564	1565	1566	1567
3040	1568	1569	1570	1571	1572	1573	1574	1575
3050	1576	1577	1578	1579	1580	1581	1582	1583
3060	1584	1585	1586	1587	1588	1589	1590	1591
3070	1592	1593	1594	1595	1596	1597	1598	1599
3100	1600	1601	1602	1603	1604	1605	1606	1607
3110	1608	1609	1610	1611	1612	1613	1614	1615
3120	1616	1617	1618	1619	1620	1621	1622	1623
3130	1624	1625	1626	1627	1628	1629	1630	1631
3140	1632	1633	1634	1635	1636	1637	1638	1639
3150	1640	1641	1642	1643	1644	1645	1646	1647
3160	1648	1649	1650	1651	1652	1653	1654	1655
3170	1656	1657	1658	1659	1660	1661	1662	1663
3200	1664	1665	1666	1667	1668	1669	1670	1671
3210	1672	1673	1674	1675	1676	1677	1678	1679
3220	1680	1681	1682	1683	1684	1685	1686	1687
3230	1688	1689	1690	1691	1692	1693	1694	1695
3240	1696	1697	1698	1699	1700	1701	1702	1703
3250	1704	1705	1706	1707	1708	1709	1710	1711
3260	1712	1713	1714	1715	1716	1717	1718	1719
3270	1720	1721	1722	1723	1724	1725	1726	1727
3300	1728	1729	1730	1731	1732	1733	1734	1735
3310	1736	1737	1738	1739	1740	1741	1742	1743
3320	1744	1745	1746	1747	1748	1749	1750	1751
3330	1752	1753	1754	1755	1756	1757	1758	1759
3340	1760	1761	1762	1763	1764	1765	1766	1767
3350	1768	1769	1770	1771	1772	1773	1774	1775
3360	1776	1777	1778	1779	1780	1781	1782	1783
3370	1784	1785	1786	1787	1788	1789	1790	1791

Octal	3400 to 3777
Decimal	1792 to 2047

Octal	0	1	2	3	4	5	6	7
3400	1792	1793	1794	1795	1796	1797	1798	1799
3410	1800	1801	1802	1803	1804	1805	1806	1807
3420	1808	1809	1810	1811	1812	1813	1814	1815
3430	1816	1817	1818	1819	1820	1821	1822	1823
3440	1824	1825	1826	1827	1828	1829	1830	1831
3450	1832	1833	1834	1835	1836	1837	1838	1839
3460	1840	1841	1842	1843	1844	1845	1846	1847
3470	1848	1849	1850	1851	1852	1853	1854	1855
3500	1856	1857	1858	1859	1860	1861	1862	1863
3510	1864	1865	1866	1867	1868	1869	1870	1871
3520	1872	1873	1874	1875	1876	1877	1878	1879
3530	1880	1881	1882	1883	1884	1885	1886	1887
3540	1888	1889	1890	1891	1892	1893	1894	1895
3550	1896	1897	1898	1899	1900	1901	1902	1903
3560	1904	1905	1906	1907	1908	1909	1910	1911
3570	1912	1913	1914	1915	1916	1917	1918	1919
3600	1920	1921	1922	1923	1924	1925	1926	1927
3610	1928	1929	1930	1931	1932	1933	1934	1935
3620	1936	1937	1938	1939	1940	1941	1942	1943
3630	1944	1945	1946	1947	1948	1949	1950	1951
3640	1952	1953	1954	1955	1956	1957	1958	1959
3650	1960	1961	1962	1963	1964	1965	1966	1967
3660	1968	1969	1970	1971	1972	1973	1974	1975
3670	1976	1977	1978	1979	1980	1981	1982	1983
3700	1984	1985	1986	1987	1988	1989	1990	1991
3710	1992	1993	1994	1995	1996	1997	1998	1999
3720	2000	2001	2002	2003	2004	2005	2006	2007
3730	2008	2009	2010	2011	2012	2013	2014	2015
3740	2016	2017	2018	2019	2020	2021	2022	2023
3750	2024	2025	2026	2027	2028	2029	2030	2031
3760	2032	2033	2034	2035	2036	2037	2038	2039
3770	2040	2041	2042	2043	2044	2045	2046	2047

Octal-Decimal Integer ConversionTable

Octal	10000	20000	30000	40000	50000	60000	70000
Decimal	4096	8192	12288	16384	20480	24576	28672

Octal	4000 to 4377
Decimal	2048 to 2303

Octal	0	1	2	3	4	5	6	7
4000	2048	2049	2050	2051	2052	2053	2054	2055
4010	2056	2057	2058	2059	2060	2061	2062	2063
4020	2064	2065	2066	2067	2068	2069	2070	2071
4030	2072	2073	2074	2075	2076	2077	2078	2079
4040	2080	2081	2082	2083	2084	2085	2086	2087
4050	2088	2089	2090	2091	2092	2093	2094	2095
4060	2096	2097	2098	2099	2100	2101	2102	2103
4070	2104	2105	2106	2107	2108	2109	2110	2111
4100	2112	2113	2114	2115	2116	2117	2118	2119
4110	2120	2121	2122	2123	2124	2125	2126	2127
4120	2128	2129	2130	2131	2132	2133	2134	2135
4130	2136	2137	2138	2139	2140	2141	2142	2143
4140	2144	2145	2146	2147	2148	2149	2150	2151
4150	2152	2153	2154	2155	2156	2157	2158	2159
4160	2160	2161	2162	2163	2164	2165	2166	2167
4170	2168	2169	2170	2171	2172	2173	2174	2175
4200	2176	2177	2178	2179	2180	2181	2182	2183
4210	2184	2185	2186	2187	2188	2189	2190	2191
4220	2192	2193	2194	2195	2196	2197	2198	2199
4230	2200	2201	2202	2203	2204	2205	2206	2207
4240	2208	2209	2210	2211	2212	2213	2214	2215
4250	2216	2217	2218	2219	2220	2221	2222	2223
4260	2224	2225	2226	2227	2228	2229	2230	2231
4270	2232	2233	2234	2235	2236	2237	2238	2239
4300	2240	2241	2242	2243	2244	2245	2246	2247
4310	2248	2249	2250	2251	2252	2253	2254	2255
4320	2256	2257	2258	2259	2260	2261	2262	2263
4330	2264	2265	2266	2267	2268	2269	2270	2271
4340	2272	2273	2274	2275	2276	2277	2278	2279
4350	2280	2281	2282	2283	2284	2285	2286	2287
4360	2288	2289	2290	2291	2292	2293	2294	2295
4370	2296	2297	2298	2299	2300	2301	2302	2303

Octal	5000 to 5377
Decimal	2560 to 2815

Octal	0	1	2	3	4	5	6	7
5000	2560	2561	2562	2563	2564	2565	2566	2567
5010	2568	2569	2570	2571	2572	2573	2574	2575
5020	2576	2577	2578	2579	2580	2581	2582	2583
5030	2584	2585	2586	2587	2588	2589	2590	2591
5040	2592	2593	2594	2595	2596	2597	2598	2599
5050	2600	2601	2602	2603	2604	2605	2606	2607
5060	2608	2609	2610	2611	2612	2613	2614	2615
5070	2616	2617	2618	2619	2620	2621	2622	2623
5100	2624	2625	2626	2627	2628	2629	2630	2631
5110	2632	2633	2634	2635	2636	2637	2638	2639
5120	2640	2641	2642	2643	2644	2645	2646	2647
5130	2648	2649	2650	2651	2652	2653	2654	2655
5140	2656	2657	2658	2659	2660	2661	2662	2663
5150	2664	2665	2666	2667	2668	2669	2670	2671
5160	2672	2673	2674	2675	2676	2677	2678	2679
5170	2680	2681	2682	2683	2684	2685	2686	2687
5200	2688	2689	2690	2691	2692	2693	2694	2695
5210	2696	2697	2698	2699	2700	2701	2702	2703
5220	2704	2705	2706	2707	2708	2709	2710	2711
5230	2712	2713	2714	2715	2716	2717	2718	2719
5240	2720	2721	2722	2723	2724	2725	2726	2727
5250	2728	2729	2730	2731	2732	2733	2734	2735
5260	2736	2737	2738	2739	2740	2741	2742	2743
5270	2744	2745	2746	2747	2748	2749	2750	2751
5300	2752	2753	2754	2755	2756	2757	2758	2759
5310	2760	2761	2762	2763	2764	2765	2766	2767
5320	2768	2769	2770	2771	2772	2773	2774	2775
5330	2776	2777	2778	2779	2780	2781	2782	2783
5340	2784	2785	2786	2787	2788	2789	2790	2791
5350	2792	2793	2794	2795	2796	2797	2798	2799
5360	2800	2801	2802	2803	2804	2805	2806	2807
5370	2808	2809	2810	2811	2812	2813	2814	2815

Octal	4400 to 4777
Decimal	2304 to 2559

Octal	0	1	2	3	4	5	6	7
4400	2304	2305	2306	2307	2308	2309	2310	2311
4410	2312	2313	2314	2315	2316	2317	2318	2319
4420	2320	2321	2322	2323	2324	2325	2326	2327
4430	2328	2329	2330	2331	2332	2333	2334	2335
4440	2336	2337	2338	2339	2340	2341	2342	2343
4450	2344	2345	2346	2347	2348	2349	2350	2351
4460	2352	2353	2354	2355	2356	2357	2358	2359
4470	2360	2361	2362	2363	2364	2365	2366	2367
4500	2368	2369	2370	2371	2372	2373	2374	2375
4510	2376	2377	2378	2379	2380	2381	2382	2383
4520	2384	2385	2386	2387	2388	2389	2390	2391
4530	2392	2393	2394	2395	2396	2397	2398	2399
4540	2400	2401	2402	2403	2404	2405	2406	2407
4550	2408	2409	2410	2411	2412	2413	2414	2415
4560	2416	2417	2418	2419	2420	2421	2422	2423
4570	2424	2425	2426	2427	2428	2429	2430	2431
4600	2432	2433	2434	2435	2436	2437	2438	2439
4610	2440	2441	2442	2443	2444	2445	2446	2447
4620	2448	2449	2450	2451	2452	2453	2454	2455
4630	2456	2457	2458	2459	2460	2461	2462	2463
4640	2464	2465	2466	2467	2468	2469	2470	2471
4650	2472	2473	2474	2475	2476	2477	2478	2479
4660	2480	2481	2482	2483	2484	2485	2486	2487
4670	2488	2489	2490	2491	2492	2493	2494	2495
4700	2496	2497	2498	2499	2500	2501	2502	2503
4710	2504	2505	2506	2507	2508	2509	2510	2511
4720	2512	2513	2514	2515	2516	2517	2518	2519
4730	2520	2521	2522	2523	2524	2525	2526	2527
4740	2528	2529	2530	2531	2532	2533	2534	2535
4750	2536	3537	2538	2539	2540	2541	2542	2543
4760	2544	2545	2546	2547	2548	2549	2550	2551
4770	2552	2553	2554	2555	2556	2557	2558	2559

Octal	5400 to 5777
Decimal	2816 to 3071

Octal	0	1	2	3	4	5	6	7
5400	2816	2817	2818	2819	2820	2821	2822	2823
5410	2824	2825	2826	2827	2828	2829	2830	2831
5420	2832	2833	2834	2835	2836	2837	2838	2839
5430	2840	2841	2842	2843	2844	2845	2846	2847
5440	2848	2849	2850	2851	2852	2853	2854	2855
5450	2856	2857	2858	2859	2860	2861	2862	2863
5460	2864	2865	2866	2867	2868	2869	2870	2871
5470	2872	2873	2874	2875	2876	2877	2878	2879
5500	2880	2881	2882	2883	2884	2885	2886	2887
5510	2888	2889	2890	2891	2892	2893	2894	2895
5520	2896	2897	2898	2899	2900	2901	2902	2903
5530	2904	2905	2906	2907	2908	2909	2910	2911
5540	2912	2913	2914	2915	2916	2917	2918	2919
5550	2920	2921	2922	2923	2924	2925	2926	2927
5560	2928	2929	2930	2931	2932	2933	2934	2935
5570	2936	2937	2938	2939	2940	2941	2942	2943
5600	2944	2945	2946	2947	2948	2949	2950	2951
5610	2952	2953	2954	2955	2956	2957	2958	2959
5620	2960	2961	2962	2963	2964	2965	2966	2967
5630	2968	2969	2970	2971	2972	2973	2974	2975
5640	2976	2977	2978	2979	2980	2981	2982	2983
5650	2984	2985	2986	2987	2988	2989	2990	2991
5660	2992	2993	2994	2995	2996	2997	2998	2999
5670	3000	3001	3002	3003	3004	3005	3006	3007
5700	3008	3009	3010	3011	3012	3013	3014	3015
5710	3016	3017	3018	3019	3020	3021	3022	3023
5720	3024	3025	3026	3027	3028	3029	3030	3031
5730	3032	3033	3034	3035	3036	3037	3038	3039
5740	3040	3041	3042	3043	3044	3045	3046	3047
5750	3048	3049	3050	3051	3052	3053	3054	3055
5760	3056	3057	3058	3059	3060	3061	3062	3063
5770	3064	3065	3066	3067	3068	3069	3070	3071

Octal-Decimal Integer Conversion Table

Octal	10000	20000	30000	40000	50000	60000	70000
Decimal	4096	8192	12288	16384	20480	24576	28672

Octal	6000 to 6377
Decimal	3072 to 3327

Octal	0	1	2	3	4	5	6	7
6000	3072	3073	3074	3075	3076	3077	3078	3079
6010	3080	3081	3082	3083	3084	3085	3086	3087
6020	3088	3089	3090	3091	3092	3093	3094	3095
6030	3096	3097	3098	3099	3100	3101	3102	3103
6040	3104	3105	3106	3107	3108	3109	3110	3111
6050	3112	3113	3114	3115	3116	3117	3118	3119
6060	3120	3121	3122	3123	3124	3125	3126	3127
6070	3128	3129	3130	3131	3132	3133	3134	3135
6100	3136	3137	3138	3139	3140	3141	3142	3143
6110	3144	3145	3146	3147	3148	3149	3150	3151
6120	3152	3153	3154	3155	3156	3157	3158	3159
6130	3160	3161	3162	3163	3164	3165	3166	3167
6140	3168	3169	3170	3171	3172	3173	3174	3175
6150	3176	3177	3178	3179	3180	3181	3182	3183
6160	3184	3185	3186	3187	3188	3189	3190	3191
6170	3192	3193	3194	3195	3196	3197	3198	3199
6200	3200	3201	3202	3203	3204	3205	3206	3207
6210	3208	3209	3210	3211	3212	3213	3214	3215
6220	3216	3217	3218	3219	3220	3221	3222	3223
6230	3224	3225	3226	3227	3228	3229	3230	3231
6240	3232	3233	3234	3235	3236	3237	3238	3239
6250	3240	3241	3242	3243	3244	3245	3246	3247
6260	3248	3249	3250	3251	3252	3253	2354	3255
6270	3256	3257	3258	3259	3260	3261	3262	3263
6300	3264	3265	3266	3267	3268	3269	3270	3271
6310	3272	3273	3274	3275	3276	3277	3278	3279
6320	3280	3281	3282	3283	3284	3285	3286	3287
6330	3288	3289	3290	3291	3292	3293	3294	3295
6340	3296	3297	3298	3299	3300	3301	3302	3303
6350	3304	3305	3306	3307	3308	3309	3310	3311
6360	3312	3313	3314	3315	3316	3317	3318	3319
6370	3320	3321	3322	3323	3324	3325	3326	3327

Octal	7000 to 7377
Decimal	3584 to 3839

Octal	0	1	2	3	4	5	6	7
7000	3584	3585	3586	3587	3588	3589	3590	3591
7010	3592	3593	3594	3595	3596	3597	3598	3599
7020	3600	3601	3602	3603	3604	3605	3606	3607
7030	3608	3609	3610	3611	3612	3613	3614	3615
7040	3616	3617	3618	3619	3620	3621	3622	3623
7050	3624	3625	3626	3627	3628	3629	3630	3631
7060	3632	3633	3634	3635	3636	3637	3638	3639
7070	3640	3641	3642	3643	3644	3645	3646	3647
7100	3648	3649	3650	3651	3652	3653	3654	3655
7110	3656	3657	3658	3659	3660	3661	3662	3663
7120	3664	3665	3666	3667	3668	3669	3670	3671
7130	3672	3673	3674	3675	3676	3677	3678	3679
7140	3680	3681	3682	3683	3684	3685	3686	3687
7150	3688	3689	3690	3691	3692	3693	3694	3695
7160	3696	3697	3698	3699	3700	3701	3702	3703
7170	3704	3705	3706	3707	3708	3709	3710	3711
7200	3712	3713	3714	3715	3716	3717	3718	3719
7210	3720	3721	3722	3723	3724	3725	3726	3727
7220	3728	3729	3730	3731	3732	3733	3734	3735
7230	3736	3737	3738	3739	3740	3741	3742	3743
7240	3744	3745	3746	3747	3748	3749	3750	3751
7250	3752	3753	3754	3755	3756	3757	3758	3759
7260	3760	3761	3762	3763	3764	3765	3766	3767
7270	3768	3769	3770	3771	3772	3773	3774	3775
7300	3776	3777	3778	3779	3780	3781	3782	3783
7310	3784	3785	3786	3787	3788	3789	3790	3791
7320	3792	3793	3794	3795	3796	3797	3798	3799
7330	3800	3801	3802	3803	3804	3805	3806	3807
7340	3808	3809	3810	3811	3812	3813	3814	3815
7350	3816	3817	3818	3819	3820	3821	3822	3823
7360	3824	3825	3826	3827	3828	3829	3830	3831
7370	3832	3833	3834	3835	3836	3837	3838	3839

Octal	6400 to 6777
Decimal	3328 to 3583

Octal	0	1	2	3	4	5	6	7
6400	3328	3329	3330	3331	3332	3333	3334	3335
6410	3336	3337	3338	3339	3340	3341	3342	3343
6420	3344	3345	3346	3347	3348	3349	3350	3351
6430	3352	3353	3354	3355	3356	3357	3358	3359
6440	3360	3361	3362	3363	3364	3365	3366	3367
6450	3368	3369	3370	3371	3372	3373	3374	3375
6460	3376	3377	3378	3379	3380	3381	3382	3383
6470	3384	3385	3386	3387	3388	3389	3390	3391
6500	3392	3393	3394	3395	3396	3397	3398	3399
6510	3400	3401	3402	3403	3404	3405	3406	3407
6520	3408	3409	3410	3411	3412	3413	3414	3415
6530	3416	3417	3418	3419	3420	3421	3422	3423
6540	3424	3425	3426	3427	3428	3429	3430	3431
6550	3432	3433	3434	3435	3436	3437	3438	3439
6560	3440	3441	3442	3443	3444	3445	3446	3447
6570	3448	3449	3450	3451	3452	3453	3454	3455
6600	3456	3457	3458	3459	3460	3461	3462	3463
6610	3464	3465	3466	3467	3468	3469	3470	3471
6620	3472	3473	3474	3475	3476	3477	3478	3479
6630	3480	3481	3482	3483	3484	3485	3486	3487
6640	3488	3489	3490	3491	3492	3493	3494	3495
6650	3496	3497	3498	3499	3500	3501	3502	3503
6660	3504	3505	3506	3507	3508	3509	3510	3511
6670	3512	3513	3514	3515	3516	3517	3518	3519
6700	3520	3521	3522	3523	3524	3525	3526	3527
6710	3528	3529	3530	3531	3532	3533	3534	3535
6720	3536	3537	3538	3539	3540	3541	3542	3543
6730	3544	3545	3546	3547	3548	3549	3550	3551
6740	3552	3553	3554	3555	3556	3557	3558	3559
6750	3560	3561	3562	3563	3564	3565	3566	3567
6760	3568	3569	3570	3571	3572	3573	3574	3575
6770	3576	3577	3578	3579	3580	3581	3582	3583

Octal	7400 to 7777
Decimal	3840 to 4095

Octal	0	1	2	3	4	5	6	7
7400	3840	3841	3842	3843	3844	3845	3846	3847
7410	3848	3849	3850	3851	3852	3853	3854	3855
7420	3856	3857	3858	3859	3860	3861	3862	3863
7430	3864	3865	3866	3867	3868	3869	3870	3871
7440	3872	3873	3874	3875	3876	3877	3878	3879
7450	3880	3881	3882	3883	3884	3885	3886	3887
7460	3888	3889	3890	3891	3892	3893	3894	3895
7470	3896	3897	3898	3899	3900	3901	3902	3903
7500	3904	3905	3906	3907	3908	3909	3910	3911
7510	3912	3913	3914	3915	3916	3917	3918	3919
7520	3920	3921	3922	3923	3924	3925	3926	3927
7530	3928	3929	3930	3931	3932	3933	3934	3935
7540	3936	3937	3938	3939	3940	3941	3942	3943
7550	3944	3945	3946	3947	3948	3949	3950	3951
7560	3952	3953	3954	3955	3956	3957	3958	3959
7570	3960	3961	3962	3963	3964	3965	3966	3967
7600	3968	3969	3970	3971	3972	3973	3974	3975
7610	3976	3977	3978	3979	3980	3981	3982	3983
7620	3984	3985	3986	3987	3988	3989	3990	3991
7630	3992	3993	3994	3995	3996	3997	3998	3999
7640	4000	4001	4002	4003	4004	4005	4006	4007
7650	4008	4009	4010	4011	4012	4013	4014	4015
7660	4016	4017	4018	4019	4020	4021	4022	4023
7670	4024	4025	4026	4027	4028	4029	4030	4031
7700	4032	4033	4034	4035	4036	4037	4038	4039
7710	4040	4041	4042	4043	4044	4045	4046	4047
7720	4048	4049	4050	4051	4052	4053	4054	4055
7730	4056	4057	4058	4059	4060	4061	4062	4063
7740	4064	4065	4066	4067	4068	4069	4070	4071
7750	4072	4073	4074	4075	4076	4077	4078	4079
7760	4080	4081	4082	4083	4084	4085	4086	4087
7770	4088	4089	4090	4091	4092	4093	4094	4095

Octal--Decimal Integer Conversion Table

Addition to octal-to-decimal conversion table to cover numbers up to 3777777_8, the maximum total value possible in a GE-215/225/235 word.

Octal	1,000,000	2,000,000	3,000,000
Decimal	262,144	524,288	786,432

Octal	100,000	200,000	300,000	400,000	500,000	600,000	700,000
Decimal	32,768	65,536	98,304	131,072	163,840	196,608	229,376

Example: Convert 1777000_8 to decimal

1,000,000 = 262,144

700,000 = 229,376

70,000 = 28,672

7,000 = 3,584

$523{,}776_{10}$

Octal-Decimal Fraction Conversion Table

OCTAL	DECIMAL	OCTAL	DECIMAL	OCTAL	DECIMAL	OCTAL	DECIMAL
.000	.000000	.100	.125000	.200	.250000	.300	.375000
.001	.001953	.101	.126953	.201	.251953	.301	.376953
.002	.003906	.102	.128906	.202	.253906	.302	.378906
.003	.005859	.103	.130859	.203	.255859	.303	.380859
.004	.007812	.104	.132812	.204	.257812	.304	.382812
.005	.009765	.105	.134765	.205	.259765	.305	.384765
.006	.011718	.106	.136718	.206	.261718	.306	.386718
.007	.013671	.107	.138671	.207	.263671	.307	.388671
.010	.015625	.110	.140625	.210	.265625	.310	.390625
.011	.017578	.111	.142578	.211	.267578	.311	.392578
.012	.019531	.112	.144531	.212	.269531	.312	.394531
.013	.021484	.113	.146484	.213	.271484	.313	.396484
.014	.023437	114	.148437	.214	.273437	.314	.398437
.015	.025390	.115	.150390	.215	.275390	.315	.400390
.016	.027343	.116	.152343	.216	.277343	.316	.402343
.017	.029296	.117	.154296	.217	.279296	.317	.404296
.020	.031250	.120	.156250	.220	.281250	.320	.406250
.021	.033203	.121	.158203	.221	.283203	.321	.408203
.022	.035156	.122	.160156	.222	.285156	.322	.410156
.023	.037109	.123	.162109	.223	.287109	.323	.412109
.024	.039062	.124	.164062	.224	.289062	.324	.414062
.025	.041015	.125	.166015	.225	.291015	.325	.416015
.026	.042968	.126	.167968	.226	.292968	.326	.417968
.027	.044921	.127	.169921	.227	.294921	.327	.419921
.030	.046875	.130	.171875	.230	.296875	.330	.421875
.031	.048828	.131	.173828	.231	.298828	.331	.423828
.032	.050781	.132	.175781	.232	.300781	.332	.425781
.033	.052734	.133	.177734	.233	.302734	.333	.427734
.034	.054687	.134	.179687	.234	.304687	.334	.429687
.035	.056640	.135	.181640	.235	.306640	.335	.431640
.036	.058593	.136	.183593	.236	.308593	.336	.433593
.037	.060546	.137	.185546	.237	.310546	.337	.435546
.040	.062500	.140	.187500	.240	.312500	.340	.437500
.041	.064453	.141	.189453	.241	.314453	.341	.439453
.042	.066406	.142	.191406	.242	.316406	.342	.441406
.043	.068359	.143	.193359	.243	.318359	.343	.443359
.044	.070312	.144	.195312	.244	.320312	.344	.445312
.045	.072265	.145	.197265	.245	.322265	.345	.447265
.046	.074218	.146	.199218	.246	.324218	.346	.449218
.047	.076171	.147	.201171	.247	.326171	.347	.451171
.050	.078125	.150	.203125	.250	.328125	.350	.453125
.051	.080078	.151	.205078	.251	.330078	.351	.455078
.052	.082031	.152	.207031	.252	.332031	.352	.457031
.053	.083984	.153	.208984	.253	.333984	.353	.458984
.054	.085937	.154	.210937	.254	.335937	.354	.460937
.055	.087890	.155	.212890	.255	.337890	.355	.462890
.056	.089843	.156	.214843	.256	.339843	.356	.464843
.057	.091796	.157	.216796	.257	.341796	.357	.466796
.060	.093750	.160	.218750	.260	.343750	.360	.468750
.061	.095703	.161	.220703	.261	.345703	.361	.470703
.062	.097656	.162	.222656	.262	.347656	.362	.472656
.063	.099609	.163	.224609	.263	.349609	.363	.474609
.064	.101562	.164	.226562	.264	.351562	.364	.476562
.065	.103515	.165	.228515	.265	.353515	.365	.478515
.066	.105468	.166	.230468	.266	.355468	.366	.480468
.067	.107421	.167	.232421	.267	.357421	.367	.482421
.070	.109375	.170	.234375	.270	.359375	.370	.484375
.071	.111328	.171	.236328	.271	.361328	.371	.486328
.072	.113281	.172	.238281	.272	.363281	.372	.488281
.073	.115234	.173	.240234	.273	.365234	.373	.490234
.074	.117187	.174	.242187	.274	.367187	.374	.492187
.075	.119140	.175	.244140	.275	.369140	.375	.494140
.076	.121093	.176	.246093	.276	.371093	.376	.496093
.077	.123046	.177	.248046	.277	.373046	.377	.498046

Octal-Decimal Fractioh Conversion Table

OCTAL	DECIMAL	OCTAL	DECIMAL	OCTAL	DECIMAL	OCTAL	DECIMAL
.000000	.000000	.000100	.000244	.000200	.000488	.000300	.000732
.000001	.000003	.000101	.000247	.000201	.000492	.000301	.000736
.000002	.000007	.000102	.000251	.000202	.000495	.000302	.000740
.000003	.000011	.000103	.000255	.000203	.000499	.000303	.000743
.000004	.000015	.000104	.000259	.000204	.000503	.000304	.000747
.000005	.000019	.000105	.000263	.000205	.000507	.000305	.000751
.000006	.000022	.000106	.000267	.000206	.000511	.000306	.000755
.000007	.000026	.000107	.000270	.000207	.000514	.000307	.000759
.000010	.000030	.000110	.000274	.000210	.000518	.000310	.000762
.000011	.000034	.000111	.000278	.000211	.000522	.000311	.000766
.000012	.000038	.000112	.000282	.000212	.000526	.000312	.000770
.000013	.000041	.000113	.000286	.000213	.000530	.000313	.000774
.000014	.000045	.000114	.000289	.000214	.000534	.000314	.000778
.000015	.000049	.000115	.000293	.000215	.000537	.000315	.000782
.000016	.000053	.000116	.000297	.000216	.000541	.000316	.000785
.000017	.000057	.000117	.000301	.000217	.000545	.000317	.000789
.000020	.000061	.000120	.000305	.000220	.000549	.000320	.000793
.000021	.000064	.000121	.000308	.000221	.000553	.000321	.000797
.000022	.000068	.000122	.000312	.000222	.000556	.000322	.000801
.000023	.000072	.000123	.000316	.000223	.000560	.000323	.000805
.000024	.000076	.000124	.000320	.000224	.000564	.000324	.000808
.000025	.000080	.000125	.000324	.000225	.000568	.000325	.000812
.000026	.000083	.000126	.000328	.000226	.000572	.000326	.000816
.000027	.000087	.000127	.000331	.000227	.000576	.000327	.000820
.000030	.000091	.000130	.000335	.000230	.000579	.000330	.000823
.000031	.000095	.000131	.000339	.000231	.000583	.000331	.000827
.000032	.000099	.000132	.000343	.000232	.000587	.000332	.000831
.000033	.000102	.000133	.000347	.000233	.000591	.000333	.000835
.000034	.000106	.000134	.000350	.000234	.000595	.000334	.000839
.000035	.000110	.000135	.000354	.000235	.000598	.000335	.000843
.000036	.000114	.000136	.000358	.000236	.000602	.000336	.000846
.000037	.000118	.000137	.000362	.000237	.000606	.000337	.000850
.000040	.000122	.000140	.000366	.000240	.000610	.000340	.000854
.000041	.000125	.000141	.000370	.000241	.000614	.000341	.000858
.000042	.000129	.000142	.000373	.000242	.000617	.000342	.000862
.000043	.000133	.000143	.000377	.000243	.000621	.000343	.000865
.000044	.000137	.000144	.000381	.000244	.000625	.000344	.000869
.000045	.000141	.000145	.000385	.000245	.000629	.000345	.000873
.000046	.000144	.000146	.000389	.000246	.000633	.000346	.000877
.000047	.000148	.000147	.000392	.000247	.000637	.000347	.000881
.000050	.000152	.000150	.000396	.000250	.000640	.000350	.000885
.000051	.000156	.000151	.000400	.000251	.000644	.000351	.000888
.000052	.000160	.000152	.000404	.000252	.000648	.000352	.000892
.000053	.000164	.000153	.000408	.000253	.000652	.000353	.000896
.000054	.000167	.000154	.000411	.000254	.000656	.000354	.000900
.000055	.000171	.000155	.000415	.000255	.000659	.000355	.000904
.000056	.000175	.000156	.000419	.000256	.000663	.000356	.000907
.000057	.000179	.000157	.000423	.000257	.000667	.000357	.000911
.000060	.000183	.000160	.000427	.000260	.000671	.000360	.000915
.000061	.000186	.000161	.000431	.000261	.000675	.000361	.000919
.000062	.000190	.000162	.000434	.000262	.000679	.000362	.000923
.000063	.000194	.000163	.000438	.000263	.000682	.000363	.000926
.000064	.000198	.000164	.000442	.000264	.000686	.000364	.000930
.000065	.000202	.000165	.000446	.000265	.000690	.000365	.000934
.000066	.000205	.000166	.000450	.000266	.000694	.000366	.000938
.000067	.000209	.000167	.000453	.000267	.000698	.000367	.000942
.000070	.000213	.000170	.000457	.000270	.000701	.000370	.000946
.000071	.000217	.000171	.000461	.000271	.000705	.000371	.000949
.000072	.000221	.000172	.000465	.000272	.000709	.000372	.000953
.000073	.000225	.000173	.000469	.000273	.000713	.000373	.000957
.000074	.000228	.000174	.000473	.000274	.000717	.000374	.000961
.000075	.000232	.000175	.000476	.000275	.000720	.000375	.000965
.000076	.000236	.000176	.000480	.000276	.000724	.000376	.000968
.000077	.000240	.000177	.000484	.000277	.000728	.000377	.000972

Octal-Decimal Fraction Conversion Table

OCTAL	DECIMAL	OCTAL	DECIMAL	OCTAL	DECIMAL	OCTAL	DECIMAL
.000400	.000976	.000500	.001220	.000600	.001464	.000700	.001708
.000401	.000980	.000501	.001224	.000601	.001468	.000701	.001712
.000402	.000984	.000502	.001228	.000602	.001472	.000702	.001716
.000403	.000988	.000503	.001232	.000603	.001476	.000703	.001720
.000404	.000991	.000504	.001235	.000604	.001480	.000704	.001724
.000405	.000995	.000505	.001239	.000605	.001483	.000705	.001728
.000406	.000999	.000506	.001243	.000606	.001487	.000706	.001731
.000407	.001003	.000507	.001247	.000607	.001491	.000707	.001735
.000410	.001007	.000510	.001251	.000610	.001495	.000710	.001739
.000411	.001010	.000511	.001255	.000611	.001499	.000711	.001743
.000412	.001014	.000512	.001258	.000612	.001502	.000712	.001747
.000413	.001018	.000513	.001262	.000613	.001506	.000713	.001750
.000414	.001022	.000514	.001266	.000614	.001510	.000714	.001754
.000415	.001026	.000515	.001270	.000615	.001514	.000715	.001758
.000416	.001029	.000516	.001274	.000616	.001518	.000716	.001762
.000417	.001033	.000517	.001277	.000617	.001522	.000717	.001766
.000420	.001037	.000520	.001281	.000620	.001525	.000720	.001770
.000421	.001041	.000521	.001285	.000621	.001529	.000721	.001773
.000422	.001045	.000522	.001289	.000622	.001533	.000722	.001777
.000423	.001049	.000523	.001293	.000623	.001537	.000723	.001781
.000424	.001052	.000524	.001296	.000624	.001541	.000724	.001785
.000425	.001056	.000525	.001300	.000625	.001544	.000725	.001789
.000426	.001060	.000526	.001304	.000626	.001548	.000726	.001792
.000427	.001064	.000527	.001308	.000627	.001552	.000727	.001796
.000430	.001068	.000530	.001312	.000630	.001556	.000730	.001800
.000431	.001071	.000531	.001316	.000631	.001560	.000731	.001804
.000432	.001075	.000532	.001319	.000632	.001564	.000732	.001808
.000433	.001079	.000533	.001323	.000633	.001567	.000733	.001811
.000434	.001083	.000534	.001327	.000634	.001571	.000734	.001815
.000435	.001087	.000535	.001331	.000635	.001575	.000735	.001819
.000436	.001091	.000536	.001335	.000636	.001579	.000736	.001823
.000437	.001094	.000537	.001338	.000637	.001583	.000737	.001827
.000440	.001098	.000540	.001342	.000640	.001586	.000740	.001831
.000441	.001102	.000541	.001346	.000641	.001590	.000741	.001834
.000442	.001106	.000542	.001350	.000642	.001594	.000742	.001838
.000443	.001110	.000543	.001354	.000643	.001598	.000743	.001842
.000444	.001113	.000544	.001358	.000644	.001602	.000744	.001846
.000445	.001117	.000545	.001361	.000645	.001605	.000745	.001850
.000446	.001121	.000546	.001365	.000646	.001609	.000746	.001853
.000447	.001125	.000547	.001369	.000647	.001613	.000747	.001857
.000450	.001129	.000550	.001373	.000650	.001617	.000750	.001861
.000451	.001132	.000551	.001377	.000651	.001621	.000751	.001865
.000452	.001136	.000552	.001380	.000652	.001625	.000752	.001869
.000453	.001140	.000553	.001384	.000653	.001628	.000753	.001873
.000454	.001144	.000554	.001388	.000654	.001632	.000754	.001876
.000455	.001148	.000555	.001392	.000655	.001636	.000755	.001880
.000456	.001152	.000556	.001396	.000656	.001640	.000756	.001884
.000457	.001155	.000557	.001399	.000657	.001644	.000757	.001888
.000460	.001159	.000560	.001403	.000660	.001647	.000760	.001892
.000461	.001163	.000561	.001407	.000661	.001651	.000761	.001895
.000462	.001167	.000562	.001411	.000662	.001655	.000762	.001899
.000463	.001171	.000563	.001415	.000663	.001659	.000763	.001903
.000464	.001174	.000564	.001419	.000664	.001663	.000764	.001907
.000465	.001178	.000565	.001422	.000665	.001667	.000765	.001911
.000466	.001182	.000566	.001426	.000666	.001670	.000766	.001914
.000467	.001186	.000567	.001430	.000667	.001674	.000767	.001918
.000470	.001190	.000570	.001434	.000670	.001678	.000770	.001922
.000471	.001194	.000571	.001438	.000671	.001682	.000771	.001926
.000472	.001197	.000572	.001441	.000672	.001686	.000772	.001930
.000473	.001201	.000573	.001445	.000673	.001689	.000773	.001934
.000474	.001205	.000574	.001449	.000674	.001693	.000774	.001937
.000475	.001209	.000575	.001453	.000675	.001697	.000775	.001941
.000476	.001213	.000576	.001457	.000676	.001701	.000776	.001945
.000477	.001216	.000577	.001461	.000677	.001705	.000777	.001949

E. REPRESENTATION OF GE-235 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)
		INPUT	OUTPUT				
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
		ƀ	(Index)		2-8	12	12
#	#	ø	#	(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	
		[			12-5-8	35	75
		<		(Tab)	12-6-8	36	76
		ǂ	(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
		Δ			11-7-8	57	57
‡		ǂ	(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
		⁕	(Ignore)	(Delete)	0-7-8	77	37

*The 400-card per minute card reader reads 11-0 and 11-2-8 as octal 52, and 12-2-8 as octal 32. If the BCD validity check switch is On, the 11-2-8 and 12-2-8 punches are treated as illegal characters.

** 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.

F. GE-235 ALPHABETICAL LIST OF GENERAL ASSEMBLY PROGRAM INSTRUCTIONS

This listing of General Assembly Program instructions is given in the alphabetical order of their mnemonic abbreviations. Also listed in columns are:

The Operand Symbol Code

Address Modification Symbol (blank if instruction cannot be modified)

Octal code representation of the instruction

Type of instruction, or GE-235 system unit affected

Description of the operation performed

Word time required to execute the instruction (NOTE: One GE-235 Word Time is 6 microseconds).

Page number on which the detailed description of the instructions--and examples of useage --may be found in this manual; or, the manual in which the instruction appears if it is not part of the central processor instruction repertoire.

The pseudo-instructions used for assembly operations in the General Assembly Program are indicated by a PSUDO in the TYPE column. No execution time is shown for them. Types of instructions found in the central processor manual are:

TYPE	REFERENCE	TYPE	REFERENCE
ADMOD	Address Modification	OPTNL	Optional Instructions
API	Automatic Program Interrupt	PSUDO	Pseudo-Instructions
ARITH	Arithmetic Instructions	SEL	Priority Control Instructions
CLOCK	Real Time Clock	SHIFT	Shift Instructions
CONSL	Central Processor Console	TRANS	Data Transfer Instructions
ITAB	Internal Test and Branch	TYPE	Console Typewriter Instructions

TYPES of instructions found in other manuals for the GE-235 system are:

TYPE	REFERENCE	TYPE	REFERENCE
AAU	Auxiliary Arithmetic Unit	DOC	Document Handler (Sorter)
CDPUN	Card Punch Equipment Manual	DSU	Disc Storage Unit
CDRD	Card Reader Equipment Manual	MAG	Magnetic Tape Unit
		PAPTP	Perforated Tape Reader and Punch
DN15	DATANET-15	PRINT	High-Speed Printer

MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	OCTAL CODE	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
* ADD	Y	X	0100000	ARITH	Decimal Add to A	2	VIII-16
ADD	Y	X	0100000	ARITH	Add to A	2	VIII-4
* ADO			2504032	ARITH	Decimal Add One to A	2	VIII-19
ADO			2504032	ARITH	Add One to A	2	VIII-7
ALF			*******	PSUDO	Alphanumeric		VII-14
ANQ	K		2511400	SHIFT	Shift A into N and Q	2 - 5	VIII-51
BAR	BAN	7	2516720	AAU	Branch on AAU NOT Ready	2	AAU
BAR	BAR	7	2514720	AAU	Branch on AAU Ready	2	AAU
BAR	BDC	7	2514730	AAU	Branch on Divide Check	2	AAU
BAR	BDN	7	2516730	AAU	Branch on No Divide Check	2	AAU
BAR	BER	7	2514727	AAU	Branch on AAU Error	2	AAU
BAR	BMI	7	2514721	AAU	Branch on AAU Minus	2	AAU
BAR	BNE	7	2516727	AAU	Branch on AAU No Error	2	AAU
BAR	BNO	7	2516723	AAU	Branch on AAU No Overflow	2	AAU
BAR	BNU	7	2516724	AAU	Branch on AAU No Underflow	2	AAU
BAR	BNZ	7	2516722	AAU	Branch on AAU Not Zero	2	AAU
BAR	BON	7	2516725	AAU	Branch on AAU Overflow Hold Not On	2	AAU
BAR	BOO	7	2514725	AAU	Branch on AAU Overflow Hold On	2	AAU
BAR	BOV	7	2514723	AAU	Branch on AAU Overflow	2	AAU
BAR	BPL	7	2516721	AAU	Branch on AAU Plus	2	AAU
BAR	BUF	7	2514724	AAU	Branch on AAU Underflow	2	AAU
BAR	BUN	7	2516726	AAU	Branch on AAU Underflow Hold Not On	2	AAU
BAR	BUO	7	2514726	AAU	Branch on AAU Underflow Hold On	2	AAU
BAR	BZE	7	2514722	AAU	Branch on AAU Zero	2	AAU
BCN			2516006	CDRD	Branch on Card Reader Not Ready	2	CDRD
BCR			2514006	CDRD	Branch on Card Reader Ready	2	CDRD

* This Instruction is an optional feature.

MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	OCTAL CODE	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
BCS	BAA	P	2514P21	PRINT	Branch on Printer Any Alert	4 - 6	PRINT
BCS	BEF	P	2514P21	MAG	Branch on Mag Tape End-Of-File	4 - 6	MAG
BCS	BER	P	2514P27	MAG	Branch on Mag Tape Error	4 - 6	MAG
				DSU	Branch on Disc Storage Unit Error	4 - 6	DSU
				PRINT	Branch on High Speed Printer Error	4 - 6	PRINT
BCS	BET	P	2514P22	MAG	Branch on Mag Tape End-of-Tape	4 - 6	MAG
BCS	BIC	P	2516P25	MAG	Branch on Mag Tape Input/Output Buffer Correct	4 - 6	MAG
				DSU	Branch On Disc Storage Unit Buffer Correct	4 - 6	DSU
BCS	BIO	P	2514P25	MAG	Branch on Mag Tape Input/Output Buffer Error	4 - 6	MAG
				DSU	Branch on Disc Storage Unit Buffer Error	4 - 6	DSU
BCS	BME	P	2514P26	MAG	Branch on Mag Tape Mod 3 or 4 Error	4 - 6	MAG
BCS	BNA	P	2516P21	PRINT	Branch on Printer No Alert	4 - 6	PRINT
BCS	BNE	P	2516P27	MAG	Branch on Mag Tape No Error	4 - 6	MAG
				DSU	Branch on Disc Storage Unit No Error	4 - 6	DSU
				PRINT	Branch on Printer No Error	4 - 6	PRINT
BCS	BNF	P	2516P21	MAG	Branch on Mag Tape No End-of-File	4 - 6	MAG
BCS	BNM	P	2516P26	MAG	Branch on Mag Tape No Mod 3 or 4 Error	4 - 6	MAG
BCS	BNO	P	2516P23	PRINT	Branch on Printer No Buffer Overflow	4 - 6	PRINT
BCS	BNP	P	2516P22	PRINT	Branch on Printer Not Out of Paper	4 - 6	PRINT
BCS	BNR	P	2516P23	MAG	Branch on Mag Tape Not Rewinding	4 - 6	MAG
BCS	BNS	P	2516P24	PRINT	Branch on Printer No Slew Alert	4 - 6	PRINT

MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	OCTAL CODE	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
BCS	BNT	P	2516P22	MAG	Branch on Mag Tape No End-of-Tape	4 - 6	MAG
BCS	BOP	P	2514P22	PRINT	Branch on Printer Out Of Paper	4 - 6	PRINT
BCS	BOV	P	2514P23	PRINT	Branch on Printer Buffer Overflow	4 - 6	PRINT
BCS	BPC	P	2516P24	MAG	Branch on Mag Tape Parity Correct	4 - 6	MAG
BCS	BPE	P	2514P24	MAG	Branch on Mag Tape Parity Error	4 - 6	MAG
BCS	BPN	P	2516P20	PRINT	Branch on Printer Not Ready	4 - 6	PRINT
BCS	BPR	P	2514P20	PRINT	Branch on Printer Ready	4 - 6	PRINT
BCS	BRN	P	2516P20	DSU	Branch on Disc Storage Unit Not Ready	4 - 6	DSU
BCS	BRR	P	2514P20	DSU	Branch on Disc Storage Unit Ready	4 - 6	DSU
BCS	BRW	P	2514P23	MAG	Branch on Mag Tape Rewinding	4 - 6	MAG
BCS	BSA	P	2514P24	PRINT	Branch on Printer Slew Alert	4 - 6	PRINT
BCS	BTN	P	2516P20	MAG	Branch on Mag Tape Controller Not Ready	4 - 6	MAG
BCS	BTR	P	2514P20	MAG	Branch on Mag Tape Controller Ready	4 - 6	MAG
* BCS	DQK (DQ1)	P	2514P32	DOC	Branch on Sorter 1 Document TCD Correct	4 - 6	DOC
* BCS	DQK (DQ2)	P	2514P33	DOC	Branch on Sorter 2 Document TCD Correct	4 - 6	DOC
BCS	FAC	P	2516P31	DSU	Branch on Any File Correct	4 - 6	DSU
BCS	FAE	P	2514P31	DSU	Branch on Any File Error	4 - 6	DSU
BCS	FKC (F0C)	P	2516P32	DSU	Branch on File 0 Correct	4 - 6	DSU
	(F1C)	P	2516P33	DSU	Branch on File 1 Correct	4 - 6	DSU
	(F2C)	P	2516P34	DSU	Branch on File 2 Correct	4 - 6	DSU
	(F3C)	P	2516P35	DSU	Branch on File 3 Correct	4 - 6	DSU
BCS	FKE (F0E)	P	2514P32	DSU	Branch on File 0 Error	4 - 6	DSU

* This Instruction is an optional Feature

MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	OCTAL CODE	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
	(F1E)	P	2514P33	DSU	Branch on File 1 Error	4 - 6	DSU
	(F2E)	P	2514P34	DSU	Branch on File 2 Error	4 - 6	DSU
	(F3E)	P	2514P35	DSU	Branch on File 3 Error	4 - 6	DSU
BCS	FKN (F0N)	P	2516P21	DSU	Branch on File 0 Not Ready	4 - 6	DSU
	(F1N)	P	2516P22	DSU	Branch on File 1 Not Ready	4 - 6	DSU
	(F2N)	P	2516P23	DSU	Branch on File 2 Not Ready	4 - 6	DSU
	(F3N)	P	2516P24	DSU	Branch on File 3 Not Ready	4 - 6	DSU
BCS	FKR (F0R)	P	2514P21	DSU	Branch on File 0 Ready	4 - 6	DSU
	(F1R)	P	2514P22	DSU	Branch on File 1 Ready	4 - 6	DSU
	(F2R)	P	2514P23	DSU	Branch on File 2 Ready	4 - 6	DSU
	(F3R)	P	2514P24	DSU	Branch on File 3 Ready	4 - 6	DSU
BCS	FSK (FS1)	P	2514P24	DOC	Branch on Sorter 1 Ready and Feed Command Given	4 - 6	DOC
	FSK (FS2)	P	2514P25	DOC	Branch on Sorter 2 Ready and Feed Command Given	4 - 6	DOC
BCS	ICK (IC1)	P	2514P26	DOC	Branch on Sorter 1 Invalid Character	4 - 6	DOC
	ICK (IC2)	P	2514P27	DOC	Branch on Sorter 2 Invalid Character	4 - 6	DOC
BCS	NFK (NF1)	P	2516P24	DOC	Branch on Sorter 1 Not Ready or Feed Command Not Given	4 - 6	DOC
	NFK (NF2)	P	2516P25	DOC	Branch on Sorter 2 Not Ready or Feed Command Not Given	4 - 6	DOC
BCS	NPK (NP1)	P	2514P22	DOC	Branch too Late for Pocket Decision, Sorter 1	4 - 6	DOC
	(NP2)	P	2514P23	DOC	Branch too Late for Pocket Decision, Sorter 2	4 - 6	DOC
BCS	NQK (NQ1)	P	2516P32	DOC	Branch on Sorter 1 Document TCD Not Correct	4 - 6	DOC
	(NQ2)	P	2516P33	DOC	Branch on Sorter 2 Document TCD Not Correct	4 - 6	DOC
BCS	PDK (PD1)	P	2516P22	DOC	Branch in Time For Sorter 1 Pocket Decision	4 - 6	DOC

MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	OCTAL CODE	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
BCS	PD2K(PD2)	P	2516P23	DOC	Branch in Time For Sorter 2 Pocket Decision	4 - 6	DOC
BCS	RAE	P	2514P27	DN15	Branch on DATANET-15 Any Error	4 - 6	DN15
BCS	RAH	P	2514P22	DN15	Branch on DATANET-15 Alert Halt	4 - 6	DN15
BCS	RAI	P	2514P34	DN15	Branch on DATANET-15 API Attempted	4 - 6	DN15
BCS	RCN	P	2516P20	DN15	Branch on DATANET-15 Not Ready	4 - 6	DN15
BCS	RCP	P	2514P25	DN15	Branch on DATANET-15 Command Word Parity Error	4 - 6	DN15
BCS	RCR	P	2514P20	DN15	Branch on DATANET-15 Ready	4 - 6	DN15
BCS	RDP	P	2514P24	DN15	Branch on DATANET-15 Parity Error	4 - 6	DN15
BCS	REC	P	2514P23	DN15	Branch on DATANET-15 Error Code Detected	4 - 6	DN15
BCS	REM	P	2514P30	DN15	Branch on DATANET-15 End-of-Message Code Detected	4 - 6	DN15
BCS	REX	P	2514P31	DN15	Branch on DATANET-15 End of Transmission	4 - 6	DN15
BCS	RNA	P	2516P22	DN15	Branch on DATANET-15 No Alert Halt	4 - 6	DN15
BCS	RNC	P	2516P23	DN15	Branch on DATANET-15 No Error Code Detected	4 - 6	DN15
BCS	RND	P	2516P24	DN15	Branch on DATANET-15 No Data Parity Error	4 - 6	DN15
BCS	RNE	P	2516P27	DN15	Branch on DATANET-15 No Error	4 - 6	DN15
BCS	RNI	P	2516P34	DN15	Branch on DATANET-15 Did Not Attempt API	4 - 6	DN15
BCS	RNM	P	2516P30	DN15	Branch on DATANET-15 No End-of-Message Code Detected	4 - 6	DN15
BCS	RNO	P	2516P33	DN15	Branch on DATANET-15 No Character Counter Overflow	4 - 6	DN15
BCS	RNP	P	2516P25	DN15	Branch on DATANET-15 No Command Word Parity Error	4 - 6	DN15

MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	OCTAL CODE	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
BCS	RNT	P	2516P21	DN15	Branch on DATANET-15 No 15-Second Delay Occurred	4 - 6	DN15
BCS	RNX	P	2516P31	DN15	Branch on DATANET-15 No End Of Transmission	4 - 6	DN15
BCS	ROV	P	2514P33	DN15	Branch on DATANET-15 Character Counter Overflow	4 - 6	DN15
BCS	RPC	P	2516P26	DSU	Branch on DSU Parity Correct	4 - 6	DSU
BCS	RPE	P	2514P26	DSU	Branch on DSU Parity Error	4 - 6	DSU
BCS	RPH	P	2514P32	DN15	Branch on DATANET-15 Perforated Tape Halt	4 - 6	DN15
BCS	RPT	P	2516P32	DN15	Branch on DATANET-15 No Perforated Tape Halt	4 - 6	DN15
BCS	RSN	P	2516P26	DN15	Branch on DATANET-15 Scanner Not Positioned On Station Requesting Access	4 - 6	DN15
BCS	RSP	P	2514P26	DN15	Branch on DATANET-15 Scanner Positioned On Station Requesting Access	4 - 6	DN15
BCS	RTD	P	2514P21	DN15	Branch on DATANET-15 If 15-Second Delay Occurred	4 - 6	DN15
BCS	SKC (S1C)	P	2516P30	DOC	Branch on Sorter 1 Correct	4 - 6	DOC
	(S2C)	P	2516P31	DOC	Branch on Sorter 2 Correct	4 - 6	DOC
BCS	SKE (S1E)	P	2514P30	DOC	Branch on Sorter 1 Error	4 - 6	DOC
	(S2E)	P	2514P31	DOC	Branch on Sorter 2 Error	4 - 6	DOC
BCS	SKN (S1N)	P	2516P20	DOC	Branch on Sorter 1 Not Ready	4 - 6	DOC
	(S2N)	P	2516P21	DOC	Branch on Sorter 2 Not Ready	4 - 6	DOC
BCS	SKR (S1R)	P	2514P20	DOC	Branch on Sorter 1 Ready	4 - 6	DOC
	(S2R)	P	2514P21	DOC	Branch on Sorter 2 Ready	4 - 6	DOC
BCS	VCK (VC1)	P	2516P26	DOC	Branch on Sorter 1 Valid Character	4 - 6	DOC
	(VC2)	P	2516P27	DOC	Branch on Sorter 2 Valid Character	4 - 6	DOC
BCS	0+F	P	2516P20	SEL	Controller Condition 0 False	4 - 6	IX-1
BCS	0+T	P	2514P20	SEL	Controller Condition 0 True	4 - 6	IX-1

MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	OCTAL CODE	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
BCS	1+F	P	2516P21	SEL	Controller Condition 1 False	4 - 6	IX-1
BCS	1+T	P	2514P21	SEL	Controller Condition 1 True	4 - 6	IX-1
BCS	2+F	P	2516P22	SEL	Controller Condition 2 False	4 - 6	IX-1
BCS	2+T	P	2514P22	SEL	Controller Condition 2 True	4 - 6	IX-1
BCS	3+F	P	2516P23	SEL	Controller Condition 3 False	4 - 6	IX-1
BCS	3+T	P	2514P23	SEL	Controller Condition 3 True	4 - 6	IX-1
BCS	4+F	P	2516P24	SEL	Controller Condition 4 False	4 - 6	IX-1
BCS	4+T	P	2514P24	SEL	Controller Condition 4 True	4 - 6	IX-1
BCS	5+F	P	2516P25	SEL	Controller Condition 5 False	4 - 6	IX-1
BCS	5+T	P	2514P25	SEL	Controller Condition 5 True	4 - 6	IX-1
BCS	6+F	P	2516P26	SEL	Controller Condition 6 False	4 - 6	IX-1
BCS	6+T	P	2514P26	SEL	Controller Condition 6 True	4 - 6	IX-1
BCS	7+F	P	2516P27	SEL	Controller Condition 7 False	4 - 6	IX-1
BCS	7+T	P	2514P27	SEL	Controller Condition 7 True	4 - 6	IX-1
BCS	8+F	P	2516P30	SEL	Controller Condition 8 False	4 - 6	IX-1
BCS	8+T	P	2514P30	SEL	Controller Condition 8 True	4 - 6	IX-1
BCS	9+F	P	2516P31	SEL	Controller Condition 9 False	4 - 6	IX-1
BCS	9+T	P	2514P31	SEL	Controller Condition 9 True	4 - 6	IX-1
BEV			2516000	ITAB	Branch on Even	2	VIII-57
BKW		T	1600000 TT00000	MAG	Backspace and Position Write Head	6	MAG
BMI			2514001	ITAB	Branch on Minus	2	VIII-57
BNN			2516005	CONSL	Branch on N-Register Not Ready	2	IX-5
BNO			2516003	ITAB	Branch on No Overflow	2	VIII-57
BNR			2514005	CONSL	Branch on N-Register Ready	2	IX-5
BNZ			2516002	ITAB	Branch on Non-Zero	2	VIII-58
BOD			2514000	ITAB	Branch on Odd	2	VIII-57
BOV			2514003	ITAB	Branch on Overflow	2	VIII-57
BPC			2516004	ITAB	Branch on Parity Correct	2	VIII-58

MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	OCTAL CODE	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
BPE			2514004	ITAB	Branch on Parity Error	2	VIII-58
BPL			2516001	ITAB	Branch on Plus	2	VIII-57
BPN			2516007	CDPUN	Branch on Card Punch Not Ready	2	CDPUN
BPR			2514007	CDPUN	Branch on Card Punch Ready	2	CDPUN
BRU	Y	X	2600000	ITAB	Branch Unconditionally	1	VIII-55
BSS			*******	PSUDO	Block Started By Symbol		VII-27
BXH	K	X	0500000	ADMOD	Branch If Index Word Is High or Equal	2	VIII-64
BXL	K	X	0400000	ADMOD	Branch If Index Word Is Low	2	VIII-65
BZE			2514002	ITAB	Branch On Zero	2	VIII-58
* CAB	Y		2100000	OPTNL	Compare And Branch	2 - 3	VIII-60
CAX			3200005	AAU	Clear AX-Register	1	AAU
CHS			2504040	ARITH	Change Sign of A-Register	2	VIII-44
CPL			2504502	ARITH	Complement A-Register	2	VIII-43
CQX			3500005	AAU	Clear QX-Register	1	AAU
* DAD	Y	X	1100000	ARITH	Decimal Double Length Add	3	VIII-17
DAD	Y	X	1100000	ARITH	Double Length Add	3	VIII-5
* DCB	Y		2200000	OPTNL	Double Length Compare And Branch	2 - 5	VIII-60
DDC			*******	PSUDO	Double Length Decimal		VII-20
DEC			*******	PSUDO	Decimal		VII-17
DLD	Y	X	1000000	TRANS	Double Length Load	3	VIII-29
DNO	K		2513200	SHIFT	Double Length Normalize	3 - 6	VIII-54
DST	Y	X	1300000	TRANS	Double Length Store	3	VIII-31
* DSU	Y	X	1200000	ARITH	Decimal Double Length Subtract	3	VIII-18
DSU	Y	X	1200000	ARITH	Double Length Subtract	3	VIII-6
DVD	Y	X	1600000	ARITH	Divide	7 - 11	VIII-9
EJT			*******	PSUDO	Eject Printer Paper		VII-31

* This Instruction is an optional feature

MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	OCTAL CODE	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
END			*******	PSUDO	End of Program		VII-29
EQO			*******	PSUDO	Equals Octal		VII-28
EQU			*******	PSUDO	Equals		VII-28
ERB	N (N=1)		0520000	DOC	Document Handler 1, End Read Busy	6	DOC
	(N=2)		1120000	DOC	Document Handler 2, End Read Busy	6	DOC
ERT		T	1300000	MAG	Erase Magnetic Tape	6	MAG
	N		HHOKKKK				
EXT	Y	X	2000000	TRANS	Extract Into A	2	VIII-34
FAD	Y		3100000	AAU	Floating Point Add - Normalized	4 - 6	AAU
	Y		3100000	AAU	Floating Point Add - Unnormalized	4 - 5	AAU
	Y		3100000	AAU	Double Precision Fixed Point Add	3	AAU
FDC			*******	PSUDO	Floating Point Decimal		VII-20
FDV	Y		3600000	AAU	Floating Point Divide - Normalized	12 - 13	AAU
	Y		3600000	AAU	Floating Point Divide - Unnormalized	12 - 13	AAU
	Y		3600000	AAU	Double Precision Fixed Point Divide	16 - 17	AAU
FLD	Y		3000000	AAU	Floating Point Load	3	AAU
FMP	Y		3500000	AAU	Floating Point Multiply - Normalized	5 - 9	AAU
	Y		3500000	AAU	Floating Point Multiply - Unnormalized	5 - 9	AAU
	Y		3500000	AAU	Double Precision Fixed Point Multiply	5 - 10	AAU
FST	Y		3300000	AAU	Floating Point Store	3	AAU
FSU	Y		3200000	AAU	Floating Point Subtract - Normalized	4 - 6	AAU
	Y		3200000	AAU	Floating Point Subtract - Unnormalized	4 - 5	AAU
	Y		3200000	AAU	Double Precision Fixed Point Subtract	3	AAU

MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	OCTAL CODE	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
HCR			2500004	CDRD	Halt Card Reader	2	CDRD
HLT	M	K =1	0500000	DOC	Halt Continuous Feeding, Sorter 1	6	DOC
		K =2	1100000	DOC	Halt Continuous Feeding, Sorter 2	6	DOC
HPT			2500016	PAPTP	Halt Perforated Tape Reader	2	PAPTP
INX	K	X	1400000	ADMOD	Increment Index Word X	3	VIII-63
KON			2500013	TYPE	Turn Typewriter Keyboard On	2	IX-4
* LAC			2504202	CLOCK	Load A-Register From C Register	2	VIII-38
LAQ			2504001	TRANS	Load A-Register From Q Register	2	VIII-36
LAQ	A		3600002	AAU	Load AX From QX	1	AAU
* LCA			2504210	CLOCK	Load C-Register From A Register	2	VIII-39
LDA	Y	X	0000000	TRANS	Load A-Register	2	VIII-29
LDO			2504022	TRANS	Load One Into A-Register	2	VIII-42
LDX	Y	X	0600000	ADMOD	Load Index Word	3	VIII-66
LDZ			2504002	TRANS	Load Zero Into A-Register	2	VIII-42
LMO			2504102	TRANS	Load Minus One Into A-Register	2	VIII-43
LOC			*******	PSUDO	Location In Octal		VII-26
LQA			2504004	TRANS	Load Q Register From A-Register	2	VIII-37
LQA	A		3200002	AAU	Load QX From AX	1	AAU
LST			*******	PSUDO	List		VII-33
MAL			*******	PSUDO	Multiple Alphanumeric		VII-16
MAQ			2504006	TRANS	Move A-Register To Q-Register	2	VIII-37
MAQA	A		3100002	AAU	Move AX to QX	1	AAU
* MOV	Y		2400000	OPTNL	Move a Block of Data	4+2N	VIII-35
MPY	Y	X	1500000	ARITH	Multiply	5 - 27	VIII-8

* This Instruction is an optional feature

MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	OCTAL CODE	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
NAL			*******	PSUDO	Negative Alphanumeric		VII-16
NAM			*******	PSUDO	Print Name on General Assembly Program Listing		VII-32
NAQ	K		2511100	SHIFT	Shift N, A and Q Right	2 - 5	VIII-52
NEG			2504522	ARITH	Negate A-Register	2	VIII-44
NLS			*******	PSUDO	No List		VII-32
NOP			2504012	ITAB	No Operation	2	VIII-45
NOR	K		2513000	SHIFT	Normalize A-Register	3 - 6	VIII-52
NOX			3100005	AAU	Normalize AX and QX	2	AAU
OCT			*******	PSUDO	Octal		VII-21
OFF			2500005	CONSL	Turn Off Typewriter, PT Punch, PT Reader	2	IX-5
OGA			2500111	TRANS	OR G Into A-Register	2	VIII-33
ORG			*******	PSUDO	Origin		VII-25
ORY	Y	X	2300000	TRANS	OR A Into Y	2	VIII-32
PAL			*******	PSUDO	Multiple Alphanumeric For Printer With Print Line Indicator		VII-17
PKT	X	K=1	0460000	DOC	Pocket Select, Sorter 1	6	DOC
		K=2	1060000	DOC	Pocket Select, Sorter 2	6	DOC
PLD			*******	PSUDO	Punch Loader Cards		VII-31
PON			2500015	PAPTP	Turn On Perforated Tape Punch	2	PAPTP
PRF OCT	 (DSU ADDRESS) K		2500000 MMMMMMM	DSU	Position Arm of Disc Storage Unit	6	DSU
RAW	N (Zero)	K	1202000 0000000	DSU	Read Disc Storage Unit After Write Check	6	DSU
RBB	M N	T	1500000 TTNNNNN	MAG	Read Backward Binary	6	MAG
RBD	M N	T	1400000 TTNNNNN	MAG	Read Backward Decimal	6	MAG
RBS	M N	T	3500000 TTNNNNN	MAG	Read Backward Special Binary	6	MAG
RCB			250YY01	CDRD	Read Punched Cards Binary	2	CDRD

MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	OCTAL CODE	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
RCD	Y		250YY00	CDRD	Read Punched Cards Decimal	2	CDRD
RCF	Y		2500010	CDRD	Read Punched Cards Full	2	CDRD
RCM	Y		2500012	CDRD	Read Punched Cards Mixed Modes, Alphanumeric and Binary	2	CDRD
RCS			2500011	CONSL	Read Console Switches	2	IX-8
RDC	M	K=1	0440000	DOC	Read Document and Feed Next Document, Sorter 1	6	DOC
		K=2	1040000	DOC	Read Document and Feed Next Document, Sorter 2	6	DOC
REM			*******	PSUDO	Remarks		VII-30
RIN			3500004	AAU	Reset AAU Indicators	1	AAU
RON			2500014	PAPTP	Turn On Perforated Tape Reader	2	PAPTP
ROV			3100004	AAU	Reset AAU Overflow Hold	1	AAU
RPT			2500006	PAPTP	Read Perforated Tape	2	PAPTP
RRD	N M	K	1201000 00MMMMM	DSU	Read Disc Storage Unit K and Release Arm	6	DSU
RRF	N M	K	1200000 00MMMMM	DSU	Read Disc Storage Unit K	6	DSU
RRM	C Y		1000000	DN15	Read DATANET-15 Remote Message	6	DN15
RRT	C Y		1200000	DN15	Read DATANET-15 Remote Perforated Tape	6	DN15
RSD	M	K=1	0420000	DOC	Read Single Document, Sorter 1	6	DOC
		K=2	1020000	DOC	Read Single Document, Sorter 2	6	DOC
RTB	M N	T	0500000 TTNNNNN	MAG	Read Mag Tape Binary	6	MAG
RTD	M N	T	0400000 TTNNNNN	MAG	Read Mag Tape Decimal	6	MAG
RTS	M N	T	2500000 TTNNNNN	MAG	Read Mag Tape Special Binary	6	MAG
RUN			3200004	AAU	Reset AAU Underflow Hold	1	AAU
RWD		T	2000000 TT00000	MAG	Rewind Mag Tape	6	MAG
SAN	K	X	2510400	SHIFT	Shift A and N Right	2 - 5	VIII-50

MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	OCTAL CODE	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
* SBO			2504112	ARITH	Decimal Subtract One From A	2	VIII-20
SBO			2504112	ARITH	Subtract One From A	2	VIII-8
SBR			*******	PSUDO	Call Subroutine		VII-23
SCA	K		2510040	SHIFT	Shift Circular A	2 - 5	VIII-49
SCD	K		2511200	SHIFT	Shift Circular Double A	2 - 5	VIII-50
SCN	O		1400000	DN15	Start Scanning Channels	2	DN15
SEL	P		2500P20	SEL	Select Controller on Channel P	4 - 6	IX-1
SEQ			*******	PSUDO	Check Source Program Card Sequence Numbers		VII-32
* SET	BINMODE		2506012	OPTNL	Set Binary Mode	2	VIII-16
* SET	DECMODE		2506011	OPTNL	Set Decimal Mode	2	VIII-15
SET	FIXPOINT		3500010	AAU	Set AAU Fixed Point Mode	1	AAU
SET	NFLPOINT		3100010	AAU	Set AAU Normalized Floating Point Mode	1	AAU
SET	NTPMODE		3200001	AAU	Set Floating Point Trap Mode Off	1	AAU
SET	PBK		2506016	API	Set Priority Break	2	IX-10
SET	PST		2506015	API	Set Priority Start	2	IX-10
SET	TRPMODE		3100001	AAU	Reset Floating Point Trap Mode On	1	AAU
SET	UFLPOINT		3200010	AAU	Set AAU Unnormalized Floating Point Mode	1	AAU
SLA	K		2512000	SHIFT	Shift Left A-Register	2 - 5	VIII-47
SLD	K		2512200	SHIFT	Shift Left Double A-Register	2 - 5	VIII-49
SLT	K		0X00000 XX00000	PRINT	Slew Printer Paper to Format Tape	6	PRINT
SLW	N		0600000 NN00000	PRINT	Slew Printer Paper Specified (N) Number of Lines	6	PRINT
SNA	K		2510100	SHIFT	Shift N and A Right	2 - 5	VIII-51
* SNR			2500114	OPTNL	Set N-Register To Receive	2	PRINT

*This Instruction is an optional feature.

MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	OCTAL CODE	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
* SNT			2500115	OPTNL	Set N Register To Transmit	2	PRINT
SPB	Y	X	0700000	ITAB	Store P Counter and Branch	2	VIII-55
SRA	K	X	2510000	SHIFT	Shift Right A-Register	2 - 5	VIII-46
SRD	K	X	2511000	SHIFT	Shift Right Double A-Register	2 - 5	VIII-48
STA	Y	X	0300000	TRANS	Store Contents of A	2	VIII-30
STO	Y	X	2700000	TRANS	Store Operand Address	2	VIII-32
STX	Y	X	1700000	ADMOD	Store Index Word	3	VIII-66
* SUB	Y	X	0200000	ARITH	Decimal Subtract From A	2	VIII-17
SUB	Y	X	0200000	ARITH	Subtract From A	2	VIII-4
* SXG	Y		2506YY3	ADMOD	Select Index Register Group	2	VIII-67
TCD			*******	PSUDO	Punch Transfer Card		VII-27
TON			2500007	TYPE	Turn Typewriter On	2	IX-4
TOP			2500116	PRINT	Turn On N-Register Printer Power	2	PRINT
TYP			2500006	TYPE	Type	2	IX-5
WAI			2504010	ITAB	Who Am I?	2	VIII-61
WCB			250YY03	CDPUN	Punch Card Binary	2	CDPUN
WCD	Y		250YY02	CDPUN	Punch Card Decimal	2	CDPUN
WCF	Y		250YY17	CDPUN	Punch Card Full	2	CDPUN
WEF		T	0200000 TT00000	MAG	Write Mag Tape End-Of-File	6	MAG
WFL (WPL)	Y X		30YYYYY 01XXXXX	PRINT PRINT	Print Format Line	6	PRINT
WPL	Y	N	2000000 01YYYYY	PRINT	Write Print Line	6	PRINT
WPT			2500006	PAPTP	Punch Perforated Tape	2	PAPTP
WRD	N M	K	3701000 00MMMMM	DSU	Write Disc Storage File K and Release Arm	6	DSU
WRF	N M	K	3700000 00MMMMM	DSU	Write Disc Storage File K	6	DSU

* This Instruction is an optional feature.

MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	OCTAL CODE	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
WRM	C+S Y		2000000	DN15	Write DATANET-15 Remote Message	6	DN15
WRT	C Y		2100000	DN15	Write DATANET-15 Remote Tape	6	DN15
WTB	M N	T	0300000 TTNNNNN	MAG	Write Mag Tape Binary	6	MAG
WTD	M N	T	0200000 TTNNNNN	MAG	Write Mag Tape Decimal	6	MAG
WTS	M N	T	2300000 TTNNNNN	MAG	Write Mag Tape Special Binary	6	MAG
XAQ			2504005	TRANS	Exchange A and Q	2	VIII-38
XAQ	A		3500002	AAU	Exchange AAU AX and QX	1	AAU
ZXX			*******	PSUDO	XX Denotes Octal Configuration Desired		VII-22

G. GE-235 OCTAL LIST OF GENERAL ASSEMBLY PROGRAM INSTRUCTIONS

This listing of General Assembly Program instructions is given in the numerical order of their octal numbers. The pseudo-instructions--those which have no octal numbers--are not included in this listing.

Otherwise, it contains all instructions shown in the previous reference section containing the Alphabetical Listing of General Assembly Program instructions.

TYPE	REFERENCE	TYPE	REFERENCE
ADMOD	Address Modification	OPTNL	Optional Instructions
API	Automatic Program Interrupt	PSUDO	Pseudo Instructions
ARITH	Arithmetic Instructions	SEL	Controller Selector Instructions
CLOCK	Real Time Clock	SHIFT	Shift Instructions
CONSL	Central Processor Console	TRANS	Data Transfer Instructions
ITAB	Internal Test and Branch	TYPE	Console Typewriter Instructions

Types of Instructions found in other manuals for the GE-235 System are:

TYPE	REFERENCE	TYPE	REFERENCE
AAU	Auxiliary Arithmetic Unit	DOC	Document Handler (Sorter)
CDPUN	Card Punches	DSU	Disc Storage Unit
CDRD	Card Readers	MAG	Magnetic Tape Unit
		PAPTP	Perforated Tape Reader and Punch
DN15	DATANET-15	PRINT	High-Speed Printer

OCTAL CODE	MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
0000000	LDA	Y	X	TRANS	Load A-Register	2	VIII-29
0100000	* ADD	Y	X	ARITH	Decimal Add to A	2	VIII-16
0100000	ADD	Y	X	ARITH	Add to A	2	VIII-4
0200000	* SUB	Y	X	ARITH	Decimal Subtract From A	2	VIII-17
0200000	SUB	Y	X	ARITH	Subtract From A	2	VIII-4
0200000 TT00000	WEF		T	MAG	Write Mag Tape End Of File	6	MAG
0200000 TTNNNNN	WTD	M N	T	MAG	Write Mag Tape Decimal	6	MAG
0300000	STA	Y	X	TRANS	Store Contents of A	2	VIII-30
0300000 TTNNNNN	WTB	M N	T	MAG	Write Mag Tape Binary	6	MAG
0400000	BXL	K	X	ADMOD	Branch If Index Word Is Low	2	VIII-65
0400000 TTNNNNN	RTD	M N	T	MAG	Read Mag Tape Decimal	6	MAG
0420000	RSD	M	K=1	DOC	Read Single Document, Sorter 1	6	DOC
0440000	RDC	M	K=1	DOC	Read Document and Feed Next Document, Sorter 1	6	DOC
0460000	PKT	X	K=1	DOC	Pocket Select, Sorter 1	6	DOC
0500000	BXH	K	X	ADMOD	Branch If Index Word Is High Or Equal	2	VIII-64
0500000	HLT	M	K=1	DOC	Halt Continuous Feeding, Sorter 1	6	DOC
0500000 TTNNNNN	RTB	M N	T	MAG	Read Mag Tape Binary	6	MAG
0520000 0000000	ERB	K (K=1)		DOC	Document Handler 1, End Read Busy	6	DOC
0600000	LDX	Y	X	ADMOD	Load Index Word	3	VIII-66
0600000	SLW	N		PRINT	Slew Printer Paper Specified (N) Number of Lines	6	PRINT
0700000	SPB	Y	X	ITAB	Store P Register and Branch	2	VIII-55
0X00000 XX00000	SLT	K		PRINT	Slew Printer Paper To Format Tape	6	PRINT

* This Instruction is an optional feature.

OCTAL CODE	MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
1000000	DLD	Y	X	TRANS	Double Length Load	3	VIII-29
1000000	RRM	C Y		DN15	Read DATANET-15 Remote Message	6	DN15
1020000	RSD	M	K=2	DOC	Read Single Document, Sorter 2	6	DOC
1040000	RDC	M	K=2	DOC	Read Document and Feed Next Document, Sorter 2	6	DOC
1060000	PKT	X	K=2	DOC	Pocket Select, Sorter 2	6	DOC
1100000	* DAD	Y	X	ARITH	Decimal Double Length Add	3	VIII-17
1100000	DAD	Y	X	ARITH	Double Length Add	3	VIII-5
1100000	HLT	M	K=2	DOC	Halt Continuous Feeding, Sorter 2	6	DOC
1120000 0000000	ERB	K (K=2)		DOC	Document Handler 2, End Read Busy	6	DOC
1200000	* DSU	Y	X	ARITH	Decimal Double Length Subtract	3	VIII-18
1200000	DSU	Y	X	ARITH	Double Length Subtract	3	VIII-6
1200000 00MMMMM	RRF	N M	K	DSU	Read Disc Storage Unit File K	6	DSU
1200000	RRT	C Y		DN15	Read DATANET-15 Remote Perforated Tape	6	DN15
1201000 00MMMMM	RRD	N M	K	DSU	Read Disc Storage Unit K and Release Arm	6	DSU
1202000 0000000	RAW	N (zero)	K	DSU	Read Disc Storage Unit After Write Check	6	DSU
1300000	DST	Y	X	TRANS	Double Length Store	3	VIII-31
1300000 HHOKKKK	ERT	 K	T	MAG	Erase Mag Tape, K Units	6	MAG
1400000	INX	K	X	ADMOD	Increment Index Word	3	VIII-63
1400000 TTNNNNN	RBD	M N	T	MAG	Read Backward Decimal	6	MAG
1400000	SCN	O		DN15	Start Scanning Channels	2	DN15
1500000	MPY	Y	X	ARITH	Multiply	5 - 7	VIII-8
1500000 TTNNNNN	RBB	M N	T	MAG	Read Backward Binary	6	MAG

* This Instruction is an optional feature.

OCTAL CODE	MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
1600000	DVD	Y	X	ARITH	Divide	7 - 11	VIII-9
1600000 TT00000	BKW		T	MAG	Backspace and Position Write Head	6	MAG
1700000	STX	Y	X	ADMOD	Store Index Word	3	VIII-66
2000000	EXT	Y	X	TRANS	Extract Into A	2	VIII-34
2000000 TT00000	RWD		T	MAG	Rewind Mag Tape	6	MAG
2000000 0100000	WPL	Y	N	PRINT	Write Print Line	6	PRINT
2000000	WRM	C+S Y		DN15	Write DATANET-15 Remote Message	6	DN15
2100000	* CAB	Y		OPTNL	Compare and Branch	2 - 3	VIII-60
2100000	WRT	C Y		DN15	Write DATANET-15 Remote Tape	6	DN15
2200000	* DCB	Y		OPTNL	Double Length Compare and Branch	2 - 5	VIII-60
2300000	ORY	Y	X	TRANS	OR A Into Y	2	VIII-32
2300000 TTNNNNN	WTS	M N	T	MAG	Write Mag Tape Special Binary	6	MAG
2400000	* MOV	Y		OPTNL	Move A Block of Data	4+2N	VIII-35
2500000 MMMMMMM	PRF OCT	 (DSU Address)	K	DSU	Position Arm of Disc Storage Unit	6	DSU
2500000 TTNNNNN	RTS	M N	T	MAG	Read Mag Tape Special Binary	6	MAG
2500004	HCR			CDRD	Halt Card Reader	2	CDRD
2500005	OFF			CONSL	Turn Off Typewriter, Perforated Tape Punch, Perforated Tape Reader	2	IX-5
2500006	RPT			PAPTP	Read Perforated Tape	2	PAPTP
2500006	TYP			TYPE	Type	2	IX-5
2500006	WPT			PAPTP	Punch Perforated Tape	2	PAPTP
2500007	TON			TYPE	Turn On Typewriter	2	IX-4
2500011	RCS			CONSL	Read Console Switches	2	IX-8

* This Instruction is an optional feature.

OCTAL CODE	MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
2500013	KON			TYPE	Turn Typewriter Keyboard On	2	IX-4
2500014	RON			PAPTP	Turn On Perforated Tape Reader	2	PAPTP
2500015	PON			PAPTP	Turn On Perforated Tape Punch	2	PAPTP
2500016	HPT			PAPTP	Halt Perforated Tape Reader	2	PAPTP
2500111	OGA			TRANS	OR G Into A-Register	2	VIII-33
2500114	* SNR			OPTNL	Set N-Register To Receive	2	PRINT
2500115	* SNT			OPTNL	Set N-Register To Transmit	2	PRINT
2500116	TOP			PRINT	Turn On N-Register Printer Power	2	PRINT
2500P20	SEL	P		SEL	Select Priority Control Address	4 - 6	IX-1
2504001	LAQ			TRANS	Load A-Register From Q-Register	2	VIII-36
2504002	LDZ			TRANS	Load Zero Into A-Register	2	VIII-42
2504004	LQA			TRANS	Load Q-Register From A-Register	2	VIII-37
2504005	XAQ			TRANS	Exchange A and Q-Registers	2	VIII-38
2504006	MAQ			TRANS	Move A-Register To Q-Register	2	VIII-37
2504010	WAI			ITAB	Who Am I?	2	VIII-61
2504012	NOP			ITAB	No Operation	2	VIII-45
2504022	LDO			TRANS	Load One Into A-Register	2	VIII-42
2504032	* ADO			ARITH	Decimal Add One to A-Register	2	VIII-19
2504032	ADO			ARITH	Add One to A-Register	2	VIII-7
2504040	CHS			ARITH	Change Sign of A-Register	2	VIII-44
2504102	LMO			TRANS	Load Minus One Into A-Register	2	VIII-43
2504112	* SBO			ARITH	Decimal Subtract One From A	2	VIII-20
2504112	SBO			ARITH	Subtract One From A	2	VIII-8
2504202	* LAC			CLOCK	Load A-Register From C-Register	2	VIII-38
2504210	* LCA			CLOCK	Load C-Register From A-Register	2	VIII-39
2504502	CPL			ARITH	Complement A-Register	2	VIII-43
2504522	NEG			ARITH	Negate A-Register	2	VIII-44

* This Instruction is an optional feature.

OCTAL CODE	MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
2506003	* SXG	Y		ADMOD	Select Index Register Group	2	VIII-67
2506011	* SET	DECMODE		OPTNL	Set Decimal Mode	2	VIII-15
2506012	* SET	BIMODE		OPTNL	Set Binary Mode	2	VIII-16
2506015	SET	PST		API	Set Priority Start	2	IX-10
2506016	SET	PBK		API	Set Priority Break	2	IX-10
250YY00	RCD	Y		CDRD	Read Punched Card Decimal	2	CDRD
250YY01	RCB			CDRD	Read Punched Card Binary	2	CDRD
250YY02	WCD	Y		CDPUN	Punch Card Decimal	2	CDPUN
250YY03	WCB	Y		CDPUN	Punch Card Binary	2	CDPUN
250YY10	RCF	Y		CDRD	Read Punched Card Full	2	CDRD
250YY12	RCM	Y		CDRD	Read Punched Card Mixed Modes, Alphanumeric and Binary	2	CDRD
250YY17	WCF	Y		CDPUN	Punch Card Full	2	CDPUN
2510000	SRA	K		SHIFT	Shift Right A-Register	2 - 5	VIII-46
2510040	SCA	K		SHIFT	Shift Circular A	2 - 5	VIII-49
2510100	SNA	K		SHIFT	Shift N and A Right	2 - 5	VIII-51
2510400	SAN	K		SHIFT	Shift A and N Right	2 - 5	VIII-50
2511000	SRD	K		SHIFT	Shift Right Double A-Register	2 - 5	VIII-48
2511100	NAQ	K		SHIFT	Shift N, A,and Q Right	2 - 5	VIII-52
2511200	SCD	K		SHIFT	Shift Circular Double A	2 - 5	VIII-50
2511400	ANQ	K		SHIFT	Shift A Into N and Q	2 - 5	VIII-51
2512000	SLA	K		SHIFT	Shift Left A-Register	2 - 5	VIII-47
2512200	SLD	K		SHIFT	Shift Left Double A-Register	2 - 5	VIII-49
2513000	NOR	K		SHIFT	Normalize A-Register	3 - 6	VIII-52
2513200	DNO	K		SHIFT	Double Length Normalize	3 - 6	VIII-54
2514000	BOD			ITAB	Branch On Odd	2	VIII-57
2514001	BMI			ITAB	Branch on Minus	2	VIII-57
2514002	BZE			ITAB	Branch on Zero	2	VIII-58

* This Instruction is an optional feature.

OCTAL CODE	MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
2514003	BOV			ITAB	Branch on Overflow	2	VIII-57
2514004	BPE			ITAB	Branch on Parity Error	2	VIII-58
2514005	BNR			CONSL	Branch on N-Register Ready	2	IX-5
2514006	BCR			CDRD	Branch on Card Reader Ready	2	CDRD
2514007	BPR			CDPUN	Branch on Card Punch Ready	2	CDPUN
2514720	BAR	BAR	7	AAU	Branch on AAU Ready	2	AAU
2514721	BAR	BMI	7	AAU	Branch on AAU Minus	2	AAU
2514722	BAR	BZE	7	AAU	Branch on AAU Zero	2	AAU
2514723	BAR	BOV	7	AAU	Branch on AAU Overflow	2	AAU
2514724	BAR	BUF	7	AAU	Branch on AAU Underflow	2	AAU
2514725	BAR	BOO	7	AAU	Branch on AAU Overflow Hold On	2	AAU
2514726	BAR	BUO	7	AAU	Branch on AAU Underflow Hold On	2	AAU
2514727	BAR	BER	7	AAU	Branch on AAU Error	2	AAU
2514730	BAR	BDC	7	AAU	Branch on Divide Check	2	AAU
2514P20	BCS	BPR	P	PRINT	Branch on Printer Ready	4 - 6	PRINT
2514P20	BCS	BRR	P	DSU	Branch on Disc Storage Unit Ready	4 - 6	DSU
2514P20	BCS	BTR	P	MAG	Branch on Mag Tape Controller Ready	4 - 6	MAG
2514P20	BCS	RCR	P	DN15	Branch on DATANET-15 Ready	4 - 6	DN15
2514P20	BCS	SKR (S1R)	P	DOC	Branch on Sorter 1 Ready	4 - 6	DOC
2514P20	BCS	XXX (0+T)	P	SEL	Controller Condition 0 True	4 - 6	IX-1
2514P21	BCS	BAA	P	PRINT	Branch on Printer Any Alert	4 - 6	PRINT
2514P21	BCS	BEF	P	MAG	Branch on Mag Tape End-Of-File	4 - 6	MAG
2514P21	BCS	FKR (FOR)	P	DSU	Branch on File 0 Ready	4 - 6	DSU
2514P21	BCS	RTD	P	DN15	Branch on DATANET-15 If 15-Second Delay Occurred	4 - 6	DN15
2514P21	BCS	SKR (S2R)	P	DOC	Branch on Sorter 2 Ready	4 - 6	DOC
2514P21	BCS	XXX (1+T)	P	SEL	Controller Condition 1 True	4 - 6	IX-1

OCTAL CODE	MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
2514P22	BCS	BET	P	MAG	Branch on Mag Tape End-of-Tape	4 - 6	MAG
2514P22	BCS	BOP	P	PRINT	Branch on Printer Out Of Paper	4 - 6	PRINT
2514P22	BCS	FKR (F1R)	P	DSU	Branch on File 1 Ready	4 - 6	DSU
2514P22	BCS	NPK (NP1)	P	DOC	Branch Too Late For Pocket Decision, Sorter 1	4 - 6	DOC
2514P22	BCS	RAH	P	DN15	Branch on DATANET-15 Alert Halt	4 - 6	DN15
2514P22	BCS	XXX (2+T)	P	SEL	Controller Condition 2 True	4 - 6	IX-1
2514P23	BCS	BOV	P	PRINT	Branch on Printer Buffer Overflow	4 - 6	PRINT
2514P23	BCS	BRW	P	MAG	Branch on Mag Tape Rewinding	4 - 6	MAG
2514P23	BCS	FKR (F2R)	P	DSU	Branch on File 2 Ready	4 - 6	DSU
2514P23	BCS	NPK (NP2)	P	DOC	Branch Too Late for Pocket Decision, Sorter 2	4 - 6	DSU
2514P23	BCS	REC	P	DN15	Branch on DATANET-15 Error Code Detected	4 - 6	DN15
2514P23	BCS	XXX (3+T)	P	SEL	Controller Condition 3 True	4 - 6	IX-1
2514P24	BCS	BPE	P	MAG	Branch on Mag Tape Parity Error	4 - 6	MAG
2514P24	BCS	BSA	P	PRINT	Branch on Printer Slew Alert	4 - 6	PRINT
2514P24	BCS	FKR (F3R)	P	DSU	Branch on File 3 Ready	4 - 6	DSU
2514P24	BCS	FSK (FS1)	P	DOC	Branch on Sorter 1 Ready and Feed Command Given	4 - 6	DOC
2514P24	BCS	RDP	P	DN15	Branch on DATANET-15 Parity Error	4 - 6	DN15
2514P24	BCS	XXX (4+T)	P	SEL	Controller Condition 4 True	4 - 6	IX-1
2514P25	BCS	BIO	P	MAG	Branch on Mag Tape Input/Output Buffer Error	4 - 6	MAG
				DSU	Branch on Disc Storage Unit Buffer Error	4 - 6	DSU
2514P25	BCS	FSK (FS2)	P	DOC	Branch on Sorter 2 Ready and Feed Command Given	4 - 6	DOC
2514P25	BCS	RCP	P	DN15	Branch on DATANET-15 Command Word Parity Error	4 - 6	DN15

OCTAL CODE	MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
2514P25	BCS	XXX (5+T)	P	SEL	Controller Condition 5 True	4 - 6	IX-1
2514P26	BCS	BME	P	MAG	Branch on Mag Tape Mod 3 or 4 Error	4 - 6	MAG
2514P26	BCS	ICK (IC1)	P	DOC	Branch on Sorter 1 Invalid Character	4 - 6	DOC
2514P26	BCS	RPE	P	DSU	Branch on DSU Parity Error	4 - 6	DSU
2514P26	BCS	RSP	P	DN15	Branch on DATANET-15 Scanner Positioned On Station Requesting Access	4 - 6	DN15
2514P26	BCS	XXX (6+T)	P	SEL	Controller Condition 6 True	4 - 6	IX-1
2514P27	BCS	BER	P	MAG	Branch on Mag Tape Error	4 - 6	MAG
				DSU	Branch on Disc Storage Unit Error	4 - 6	DSU
				PRINT	Branch on High-Speed Printer Error	4 - 6	PRINT
2514P27	BCS	ICK (IC2)	P	DOC	Branch on Sorter 2 Invalid Character	4 - 6	DOC
2514P27	BCS	RAE	P	DN15	Branch on DATANET-15 Any Error	4 - 6	DN15
2514P27	BCS	XXX (7+T)	P	SEL	Controller Condition 7 True	4 - 6	IX-1
2514P30	BCS	REM	P	DN15	Branch on DATANET-15 End-of-Message Code Detected	4 - 6	DN15
2514P30	BCS	SKE (S1E)	P	DOC	Branch on Sorter 1 Error	4 - 6	DOC
2514P30	BCS	XXX (8+T)	P	SEL	Controller Condition 8 True	4 - 6	IX-1
2514P31	BCS	FAE	P	DSU	Branch on Any File Error	4 - 6	DSU
2514P31	BCS	REX	P	DN15	Branch on DATANET-15 End of Transmission	4 - 6	DN15
2514P31	BCS	SKE (S2E)	P	DOC	Branch on Sorter 2 Error	4 - 6	DOC
2514P31	BCS	XXX (9+T)	P	SEL	Controller Condition 9 True	4 - 6	IX-1
2514P32	* BCS	DQK (DQ1)	P	DOC	Branch on Sorter 1 Document TCD Correct	4 - 6	DOC
2514P32	BCS	FKE (F0E)	P	DSU	Branch on File 0 Error	4 - 6	DSU
2514P32	BCS	RPH	P	DN15	Branch on DATANET-15 Perforated Tape Halt	4 - 6	DN15

* This Instruction is an optional feature.

OCTAL CODE	MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
2514P33	BCS	DQK (DQ2)	P	DOC	Branch on Sorter 2 Document TCD Correct	4 - 6	DOC
2514P33	BCS	FKE (F1E)	P	DSU	Branch on File 1 Error	4 - 6	DSU
2514P33	BCS	ROV	P	DN15	Branch on DATANET-15 Character Counter Overflow	4 - 6	DN15
2514P34	BCS	FKE (F2E)	P	DSU	Branch on File 2 Error	4 - 6	DSU
2514P34	BCS	RAI	P	DN15	Branch on DATANET-15 API Attempted	4 - 6	DN15
2514P35	BCS	FKE (F3E)	P	DSU	Branch on File 3 Error	4 - 6	DSU
2516000	BEV			ITAB	Branch on Even	2	VIII-57
2516001	BPL			ITAB	Branch on Plus	2	VIII-57
2516002	BNZ			ITAB	Branch on Non-Zero	2	VIII-58
2516003	BNO			ITAB	Branch on No Overflow	2	VIII-57
2516004	BPC			ITAB	Branch on Parity Correct	2	VIII-58
2516005	BNN			CONSL	Branch on N-Register Not Ready	2	IX-5
2516006	BCN			CDRD	Branch on Card Reader Not Ready	2	CDRD
2516007	BPN			CDPUN	Branch on Card Punch Not Ready	2	CDPUN
2516720	BAR	BAN	7	AAU	Branch on AAU Not Ready	2	AAU
2516721	BAR	BPL	7	AAU	Branch on AAU Plus	2	AAU
2516722	BAR	BNZ	7	AAU	Branch on AAU Not Zero	2	AAU
2516723	BAR	BNO	7	AAU	Branch on AAU No Overflow	2	AAU
2516724	BAR	BNU	7	AAU	Branch on AAU No Underflow	2	AAU
2516725	BAR	BON	7	AAU	Branch on AAU Overflow Hold Not On	2	AAU
2516726	BAR	BUN	7	AAU	Branch on AAU Underflow Hold Not On	2	AAU
2516727	BAR	BNE	7	AAU	Branch on AAU No Error	2	AAU
2516730	BAR	BDN	7	AAU	Branch on AAU No Divide Check	2	AAU
2516P20	BCS	BPN	P	PRINT	Branch on Printer Not Ready	4 - 6	PRINT
2516P20	BCS	BRN	P	DSU	Branch on Disc Storage Unit Not Ready	4 - 6	DSU

OCTAL CODE	MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
2516P20	BCS	BTN	P	MAG	Branch on Mag Tape Controller Not Ready	4 - 6	MAG
2516P20	BCS	RCN	P	DN15	Branch on DATANET-15 Not Ready	4 - 6	DN15
2516P20	BCS	SKN (S1N)	P	DOC	Branch on Sorter 1 Not Ready	4 - 6	DOC
2516P20	BCS	XXX (0+F)	P	SEL	Controller Condition 0 False	4 - 6	IX-1
2516P21	BCS	BNA	P	PRINT	Branch on Printer No Alert	4 - 6	PRINT
2516P21	BCS	BNF	P	MAG	Branch on Mag Tape No End-of-File	4 - 6	MAG
2516P21	BCS	FKN (F0N)	P	DSU	Branch on File 0 Not Ready	4 - 6	DSU
2516P21	BCS	RNT	P	DN15	Branch on DATANET-15 No 15-Second Delay Occurred	4 - 6	DN15
2516P21	BCS	SKN (S2N)	P	DOC	Branch on Sorter 2 Not Ready	4 - 6	DOC
2516P21	BCS	XXX (1+F)	P	SEL	Controller Condition 1 False	4 - 6	IX-1
2516P22	BCS	BNP	P	PRINT	Branch on Printer Not Out Of Paper	4 - 6	PRINT
2516P22	BCS	BNT	P	MAG	Branch on Mag Tape No End-of-Tape	4 - 6	MAG
2516P22	BCS	FKN (F1N)	P	DSU	Branch on File 1 Not Ready	4 - 6	DSU
2516P22	BCS	PDK (PD1)	P	DOC	Branch in Time For Pocket Decision, Sorter 1	4 - 6	DOC
2516P22	BCS	RNA	P	DN15	Branch on DATANET-15 No Alert Halt	4 - 6	DN15
2516P22	BCS	XXX (2+F)	P	SEL	Controller Condition 2 False	4 - 6	IX-1
2516P23	BCS	BNO	P	PRINT	Branch on Printer No Buffer Overflow	4 - 6	PRINT
2516P23	BCS	BNR	P	MAG	Branch on Mag Tape Not Rewinding	4 - 6	MAG
2516P23	BCS	FKN (F2N)	P	DSU	Branch on File 2 Not Ready	4 - 6	DSU
2516P23	BCS	PDK (PD2)	P	DOC	Branch in Time For Pocket Decision, Sorter 2	4 - 6	DOC
2516P23	BCS	RNC	P	DN15	Branch on DATANET-15 No Error Code Detected	4 - 6	DN15
2516P23	BCS	XXX (3+F)	P	SEL	Controller Condition 3 False	4 - 6	IX-1

OCTAL CODE	MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
2516P24	BCS	BNS	P	PRINT	Branch on Printer No Slew Alert	4 - 6	PRINT
2516P24	BCS	BPC	P	MAG	Branch on Mag Tape Parity Correct	4 - 6	MAG
2516P24	BCS	FKN (F3N)	P	DSU	Branch on File 3 Not Ready	4 - 6	DSU
2516P24	BCS	NFK (NF1)	P	DOC	Branch on Sorter 1 Not Ready Or Feed Command Not Given	4 - 6	DOC
2516P24	BCS	RND	P	DN15	Branch on DATANET-15 No Data Parity Error	4 - 6	DN15
2516P24	BCS	XXX (4+F)	P	SEL	Controller Condition 4 False	4 - 6	IX-1
2516P25	BCS	BIC	P	MAG	Branch on Mag Tape Input/ Output Buffer Correct	4 - 6	MAG
				DSU	Branch on Disc Storage Unit Buffer Correct	4 - 6	DSU
2516P25	BCS	NFK (NF2)	P	DOC	Branch on Sorter 2 Not Ready Or Feed Command Not Given	4 - 6	DOC
2516P25	BCS	RNP	P	DN15	Branch on DATANET-15 No Command Word Parity Error	4 - 6	DN15
2516P25	BCS	XXX (5+F)	P	SEL	Controller Condition 5 False	4 - 6	IX-1
2516P26	BCS	BNM	P	MAG	Branch on Mag Tape No Mod 3 or 4 Error	4 - 6	MAG
2516P26	BCS	RPC	P	DSU	Branch on DSU Parity Correct	4 - 6	DSU
2516P26	BCS	RSN	P	DN15	Branch on DATANET-15 Scanner Not Positioned On Station Requesting Access	4 - 6	DN15
2516P26	BCS	VCK (VC1)	P	DOC	Branch on Sorter 1 Valid Character	4 - 6	DOC
2516P26	BCS	XXX (6+F)	P	SEL	Controller Condition 6 False	4 - 6	IX-1
2516P27	BCS	BNE	P	MAG	Branch on Mag Tape No Error	4 - 6	MAG
				DSU	Branch on Disc Storage Unit No Error	4 - 6	DSU
				PRINT	Branch on Printer No Error	4 - 6	PRINT
2516P27	BCS	RNE	P	DN15	Branch on DATANET-15 No Error	4 - 6	DN15
2516P27	BCS	VCK (VC2)	P	DOC	Branch on Sorter 2 Valid Character	4 - 6	DOC

OCTAL CODE	MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
2516P27	BCS	XXX (7+F)	P	SEL	Controller Condition 7 False	4 - 6	IX-1
2516P30	BCS	RNM	P	DN15	Branch on DATANET-15 No End of Message Code Detected	4 - 6	DN15
2516P30	BCS	SKC (S1C)	P	DOC	Branch on Sorter 1 Correct	4 - 6	DOC
2516P30	BCS	XXX (8+F)	P	SEL	Controller Condition 8 False	4 - 6	IX-1
2516P31	BCS	FAC	P	DSU	Branch on Any File Correct	4 - 6	DSU
2516P31	BCS	RNX	P	DN15	Branch on DATANET-15 No End of Transmission	4 - 6	DN15
2516P31	BCS	SKC (S2C)	P	DOC	Branch on Sorter 2 Correct	4 - 6	DOC
2516P31	BCS	XXX (9+F)	P	SEL	Controller Condition 9 False	4 - 6	IX-1
2516P32	BCS	FKC (F0C)	P	DSU	Branch on File 0 Correct.	4 - 6	DSU
2516P32	BCS	NQK (NQ1)	P	DOC	Branch on Sorter 1 Document TCD Not Correct	4 - 6	DOC
2516P32	BCS	RPT	P	DN15	Branch on DATANET-15 No Perforated Tape Halt	4 - 6	DN15
2516P33	BCS	FKC (F1C)	P	DSU	Branch on File 1 Correct	4 - 6	DSU
2516P33	BCS	NQK (NQ2)	P	DOC	Branch on Sorter 2 Document TCD Not Correct	4 - 6	DOC
2516P33	BCS	RNO	P	DN15	Branch on DATANET-15 No Character Counter Overflow	4 - 6	DN15
2516P34	BCS	FKC (F2C)	P	DSU	Branch on File 2 Correct	4 - 6	DSU
2516P34	BCS	RNI	P	DN15	Branch on DATANET-15 Did Not Attempt API	4 - 6	DN15
2516P35	BCS	FKC (F3C)	P	DSU	Branch on File 3 Correct	4 - 6	DSU
2600000	BRU	Y	X	ITAB	Branch Unconditionally	1	VIII-55
2700000	STO	Y	X	TRANS	Store Operand Address	2	VIII-32
3000000	FLD	Y		AAU	Floating Point Load	3	AAU
30YYYYY	WFL	Y		PRINT	Print Format Line	6	PRINT
01XXXXX	(WPL)	X	N				
3100000	FAD	Y		AAU	Floating Point Add - Normalized	4 - 6	AAU
		Y		AAU	Floating Point Add - Unnormalized	4 - 5	AAU
		Y		AAU	Double Precision Fixed Point Add	3	AAU

OCTAL CODE	MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
3100001	SET	TRPMODE		AAU	Set AAU Trapping Mode On	1	AAU
3100002	MAQ	A		AAU	Move AX to QX	1	AAU
3100004	ROV			AAU	Reset AAU Overflow Hold	1	AAU
3100005	NOX			AAU	Normalize AX and QX	2	AAU
3100010	SET	NFLPOINT		AAU	Set AAU Normalized Floating Point Mode	1	AAU
3200000	FSU	Y		AAU	Floating Point Subtract - Normalized	4 - 6	AAU
		Y		AAU	Floating Point Subtract - Unnormalized	4 - 5	AAU
		Y		AAU	Double Precision Fixed Point Subtract	3	AAU
3200001	SET	NTPMODE		AAU	Set AAU Trapping Mode Off	1	AAU
3200002	LQA	A		AAU	Load QX From AX	1	AAU
3200004	RUN			AAU	Reset AAU Underflow Hold	1	AAU
3200005	CAX			AAU	Clear AX-Register	1	AAU
3200010	SET	UFLPOINT		AAU	Set AAU Unnormalized Floating Point Mode	1	AAU
3300000	FST	Y		AAU	Floating Point Store	3	AAU
3500000	FMP	Y		AAU	Floating Point Multiply - Normalized	5 - 9	AAU
		Y		AAU	Floating Point Multiply - Unnormalized	5 - 9	AAU
		Y		AAU	Double Precision Fixed Point Multiply	5 - 10	AAU
3500000 TTNNNNN	RBS	M N	T	MAG	Read Backward Special Binary	6	MAG
3500002	XAQA	A		AAU	Exchange AX and QX	1	AAU
3500004	RIN			AAU	Reset AAU Indicators	1	AAU
3500005	CQX			AAU	Clear QX-Register	1	AAU
3500010	SET	FIXPOINT		AAU	Set AAU Fixed Point Mode	1	AAU
3600000	FDV	Y		AAU	Floating Point Divide - Normalized	12 - 13	AAU

OCTAL CODE	MNEMONIC	OPERAND SYMBOL	ADDRESS MODIF.	TYPE	DESCRIPTION	WORD TIME	PAGE OR MANUAL
		Y		AAU	Floating Point Divide - Unnormalized	12 - 13	AAU
		Y		AAU	Double Precision Fixed Point Divide	16 - 17	AAU
3600002	LAQ	A		AAU	Load AX From QX	1	AAU
3700000 00MMMMM	WRF	N M	K	DSU	Write Disc Storage File K	6	DSU
3701000 00MMMMM	WRD	N M	K	DSU	Write Disc Storage File K and Release Arm	6	DSU