GE-625/635

Programming

Reference Manual

GENERAL ELECTRIC

N O T I C E

For your convenience, the following Technical Information

Bulletins have been incorporated in this manual:

TIB No.   600-214
            600-228
            1004F-3

GE-625/635
Programming
Reference Manual

July 1964
Rev. April 1968

GENERAL ELECTRIC

PREFACE

The GE-625/635 Programming Reference Manual is the basic document for programming the GE-625/635. It essentially describes programming-related GE-625/635 machine features, the instruction repertoire, and the symbolic machine language oriented Macro Assembler. The Assembler chapter and the examples in Chapter IV describe how the programmer may write Processor instructions using a symbolic notation.

The Programming Reference Manual is one of a set of user publications for programming the GE-625/635 computer. The others of the set, together with pertinent and necessary programming information contained in each, are:

PUBLICATION	PROGRAMMING INFORMATION
GE-625/635 FORTRAN IV Reference Manual, CPB-1006	FORTRAN IV language specifications, coding rules and restrictions, and compiler information for the GE-625/635
GE-625/635 COBOL Reference Manual, CPB-1007	COBOL-61 Extended language specifications, coding rules and restrictions, and compiler information for the GE-625/635
GE-625/635 File and Record Control Reference Manual, CPB-1003 (GEFRC)	Standard input/output coding by use of calling sequences to software system input/output routines.
GE-625/635 Comprehensive Operating Supervisor Reference Manual, CPB-1195 (GECOS II)	1. Descriptions and functions of the Comprehensive Operating Supervisor modules and submodules
GE-625/635 Comprehensive Operating Supervisor (GECOS-III) Reference Manual, CPB-1518	2. Use of Operating Supervisor control cards
	3. Coding for information exchange between the programmer and the Operating Supervisor
	4. Alternative coding techniques for input/output operations
	5. Preparation of the user program fault transfer table
GE-625/635 General Loader Reference Manual, CPB-1008, (GELOAD)	1. Use of Loader control cards
	2. Use of the Loader debugging option and program segment overlays
	3. Descriptions of relocatable and absolute decks and their loading

PUBLICATION	PROGRAMMING INFORMATION
GE-625/635 Sort/Merge Program Reference Manual, CPB-1005	1. Descriptions of the sort and merge programs
	2. Use of the sort/merge and supplemental system MACROS
GE-625/635 Bulk Media Conversion Reference Manual, CPB-1096	Description of deck preparation for bulk media conversion run

This reference manual is addressed to programmers experienced with coding in the environment of a large-scale computer installation. It assumes some knowledge and experience in the use of address modification with indirection, hardware indicators, fault interrupts and recovery routines, macro operations, pseudo-operations, and other features normally encountered in a fast, large memory capacity computer with a very flexible instruction repertoire--under control of a master executive program. It is also assumed that the programmer is familiar with the 2's complement number system as used in a sign-number machine.

Suggestions and criticisms relative to form, content, purpose, or use of this manual are invited. Comments may be sent on the Document Review Sheet in the back of this manual or may be addressed directly to General Electric Company, Information Systems Equipment Division, C-83, 13430 North Black Canyon Highway, Phoenix, Arizona 85029.

> This manual includes features implemented in Systems Development Letter 2.

CONTENTS

APPENDIXES

ILLUSTRATIONS

I. SUMMARY OF SYSTEM FEATURES

The GE-600 Information Systems provide processing and input/output capabilities across a wide performance range. Systems are tailored to the specific workload and processing environment of an installation through the selection of the appropriate system model and by the configuration of central system modules and peripheral devices. The particular system model is determined by the speed of the central system components, and include the GE-615 and GE-635 systems. The Comprehensive Operating System, GECOS*, is the same for all models and configurations and provides 3-dimensional processing capabilities from the smallest to the largest system.

SYSTEM MODELS

System modules differ only in terms of their speed of operation, not in functional capability. The primary characteristics which identify each model are summarized in the following table:

System	Memory Cycle Time	No. of Words Per Memory Access	Instruction Overlap
615	2 microseconds	2	No
635	1 microsecond	2	Yes

Comparative instruction execution times for each system model are shown in Appendix A. Calculation of instruction execution time is shown on page 42.

COMPUTER COMPONENTS

Typical System and Functions

A typical GE-600 Line computer system offering the three dimensional processing capabilities - batch, remote batch, and time-sharing - consists of the following components:

1. Memory module 128K, 1 microsecond
2. Processor module (GE-635)
3. The Input/Output Controller module
4. Peripheral subsystems

---

*GECOS - Trademark of General Electric Company, U.S.A.

Each of the items perform specialized functions to be elaborated upon under separate headings that follow. For purposes of this discussion, we consider the typical computer system to be comprised of items 1 through 3 and the following complement of peripheral devices:

A Disc Storage Subsystem (90M characters)
A Dual Magnetic Tape Subsystem
Two Printers
Communications Processor (DATANET-30*)
Card Punch
Two Card Readers
Operator's Console with Typewriter

This system can be expanded in a variety of ways to develop multiprocessor and multi-computer systems that are restricted in size only by practical application considerations. (The computer system itself is theoretically capable of unlimited expansion, see the GE-615/635 Information Systems Manual, CPB-371).

Memory Module

The Memory module, unlike most computer systems which are processor-oriented, is the overall system control agency. It serves as a passive coordinating component that provides interim information storage and general system communication control. The module comprises two major functional units: the System Controller and the Magnetic Core Storage Unit. The principal featurs of the module and the performing units are:

FEATURE	FUNCTIONAL UNIT
1. Control of the selection and enabling of the eight or fewer channels between the Memory and Processor or Input/Output Controller modules	System Controller (eight priority-linked channel control cells plus an associated mask register)

---

*DATANET, Registered Trademark of General Electric Company, U. S. A

FEATURE	FUNCTIONAL UNIT
2. Recognition of program interrupts within the multiprogram environment	System Controller (32 priority-related program interrupt cells plus an associated mask register)
3. Selection of the type of Core Storage Unit memory cycle to be used--Read-Restore, Clear-Write, or Read-Alter-Rewrite	System Controller (control logic subunit)
4. Control of information transfers to and from the Core Storage Unit and on the selected system communication channel	System Controller (control logic subunit)
5. Storage of information	Magnetic Core Storage Unit

Processor Module

The Processor module is composed of two principal functional units: the Program Control Unit and the Operations Unit. The chief features of the module and the performing units are:

1. Decoding of instructions and indirect words with associated directions of the Operations Unit	Program Control Unit (operations decoder)
2. Development of effective addresses	Program Control Unit (address modification registers, adder, location counter, and control circuitry)
3. Memory protection of all executive routines and user programs not currently under execution	Program Control Unit (Base Address register and adder)
4. Dynamic relocation of user and other programs	Program Control Unit (Base Address register and adder)
5. Master and Slave Modes of operation whereby in the Master Mode all machine instructions can be executed, but in the Slave Mode the LBAR, LDT, SMIC, RMCM, SMCM, and CIOC instructions cannot be executed	Program Control Unit (Master Mode Indicator and mode control circuitry)
6. Performance of arithmetic, logical, shifting, and other operations involving fixed- and floating-point numbers in single or double precision	Operations Unit (control logic subunit, main and exponent adders, and associated registers)

Input/Output Controller Module

The Input/Output Controller module is the coordinator of all input/output data transfers between the complement of peripheral subsystems and the Memory module. It is in fact a separate processor which, when provided with certain required information from the Comprehensive Operating Supervisor and the user program, works independently of the Processor module under control of its own permanently-wired program.

The major functional units of the Input/Output Controller are (1) the Memory Interface, (2) the Buffer Storage, (3) the Micro-Program Generator, (4) the I/O Processor, and (5) the PUB* Interrupt Service. The main features of this module and the performing units are:

FEATURE	FUNCTIONAL UNIT
1. Transfer of characters and words to and from memory	Memory Interface (with the Buffer Storage as controlled by the Micro-Program Generator and I/O Processor)
2. Transfer of characters only to and from the programmer-designated peripheral type and Comprehensive Operating Supervisor selected physical device	PUB Interrupt Service (with the Buffer Storage as controlled by the Micro-Program Generator and the I/O Processor)
3. Memory protection of all executed routines and user programs, not currently involved in input/output operations, on all data transfers	I/O Processor (as controlled by the Micro-Program Generator)
4. Sensing and storing, in appropriate input/output queue lists of executive system (protected) memory, the status of every peripheral operation and/or device involved in input/output transfers	Micro-Program Generator and I/O Processor

Peripheral Subsystems

Peripheral subsystems used with the GE-625/635 are described in the following manuals:

1. GE-400/600 Series Punched Card Subsystems, CPB-1288
2. PRT201 Printer Reference Manual, CPB-1292
3. DSU200 Disc Storage Subsystem, CPB-4302
4. PTS200 Perforated Tape Subsystem, CPB-1100
5. MDS200/201 Magnetic Drum Subsystem, CPB-1123
6. Magnetic Tape Subsystems, CPB-1044
7. Seven/Nine Track Magnetic Tape Subsystems, CPB-1205

---

* Peripheral Unit Buffer; that is, peripheral device channel

CPB-1004F

SOFTWARE SYSTEM

Objectives

The primary objectives of the GE-635 software system are:

1. To reduce user-program "turn-around" time in large-scale installations (elapsed time from program submission to the machine room up to return of program solutions).

2. To assure that accounting information is based only on such time as the user program activity is worked upon by the Processor and peripheral devices

3. To increase the total "throughput" of the computer (the amount of work that may be performed in any given time)

4. To reduce computer operation "overhead" time in running the installation programs

5. To provide easy to-use programmer and operator interfaces with the executive software

The attainment of these objectives is achieved by the General Comprehensive Operating Supervisor (GECOS) (the overall manager of the software system) through efficient use of the hardware features and the supervision of a multiprogramming environment (which is the normal operating mode of the GE-625/635). The significant features provided by the Operating Supervisor as related to the several primary objectives above are summarized in the list following. These features are implemented by the modules and submodules within the Comprehensive Operating Supervisor.

1. Scheduling and coordination of jobs

2. Memory allocation for data and programs

3. Assignment of input/output peripherals

4. Input/Ouput supervision on an interrupt-oriented basis

5. File-oriented programming (instead of device-oriented)

6. Fault detection with standard Operating Supervisor or optional programmer-supplied corrective actions

7. Modular construction to simplify maintenance

8. Maximum system throughput via multiprogramming

9. Maximum efficiency of core memory by dynamic program relocation, and by system-controlled subprogram overlays

Multiprogramming

Although each user-programmer writes his job program as though he had exclusive use of the computer, he is in fact generating a program that will reside concurrently in memory with other user programs and will be executed in a time-shared manner; that is, any given program is processed until it is held up (usually because of the need for some input/output

CPB-1004F

4

to be completed) at which time the next most urgent program is processed. Transfer between programs under multiprogram execution is performed by means of the hardware interrupt facility (in the System Controller) working with the Dispatcher routines in the Input/Output Supervisor. The ways by which a user program can be temporarily delayed in execution are:

DELAY TYPE	REASON
Roadblock	Program cannot progress until all input/output requests have terminated
Relinquish	Program relinquishes control so that some other program may be executed
Forced Relinquish	Program was interrupted because a timer runout occurred.

Each time a program yields control to the Operating Supervisor by means of Roadblock, Relinquish or by Forced Relinquish listed above, the Supervisor has the opportunity to give control to another program in core which can make effective use of the Processor.

In giving such control, the Supervisor examines the following conditions:

1. Program urgency compared to other programs that reside in memory
2. Roadblock status involving completion of all input/output
3. Completion of input/output that was pending when the last Relinquish was given
4. Request present for use of the Processor

On-Line Media Conversion

Media conversions are of two basic types (1) bulk media conversion, whereby large volumes of data in a single format and for a single purpose are processed and, (2) system media conversion where low-volume sets of data--each with its own format and purpose--are processed.

Bulk media conversion is performed by a system routine which may be called into execution by use of a control card. Other control cards will direct the routine as to where to find the input and where to place the output.

On-line media conversions for both input and output are performed as a normal part of the multiprogramming environment of the GE-625/635. Normal job input is carried out by input media conversion, which reads card input from the card reader, scans the control cards for execution information, and records the job on the input queue located on the system drum.

System media conversions of program output data are automatically performed by the Output Media Conversion routine executed in protected memory. The programmer specifies that a particular output file be written on the permanently assigned system output (SYSOUT) file by use of the PRINT, PUNCH, or WTREC calling sequences described in the GE-625/635 File and Record Control Reference Manual. Once on the SYSOUT file, the output is converted to hard copy or punched cards by the Output Media Conversion routine, concurrently with other user programs under execution in the multiprogramming environment.

CPB-1004F

Centralized Input/Output

In the multiprogramming environment where several programs may concurrently request input/output, a facility must be provided (1) for processing such multiple requests in terms of the efficient use of the entire peripheral complement and, (2) for maintaining continuous processing of the multiple programs in core storage. The Comprehensive Operating Supervisor module that performs these general functions is the Input/Output Supervisor.

The main functions of the Input/Output Supervisor are to initiate an input/output activity and to respond to the termination of an input/output activity. In addition, the Input/Output Supervisor provides the following functions:

1. File code to physical unit translation

2. File protection of user files

3. Pseudo-tape processing on disc/drum

4. Supervision of all input/output interrupts

5. Queueing of input/output requests

6. Utilization of crossbarred magnetic tape channels

7. Maintenance of an awareness of the status of each peripheral

8. Accounting of time spent by the Processor and all peripherals for each program executed

When the Input/Output Supervisor receives a request to perform an input/output function, it looks at the communication cells and issues a connect instruction. If the particular channel is busy, the request is placed in a waiting queue. If the request queue is full or if the program indicated that it should be roadblocked until all input/output is complete, then control is given to another program residing in memory.

When the input/output operation terminates, control is given to the Input/Output Supervisor to perform all necessary termination functions. At this point, the request queue is examined and if any requests for the channel are in queue, they will be executed.

Master/Slave Relationship

Each Processor has the capability of operating in the Slave Mode or in the Master Mode. Master Mode is established for exclusive use by the Operating Supervisor. When executing a user program, a Processor is in Slave Mode. The prime reason for the Master Mode of operation is to protect the Operating Supervisor and user programs as well from modification by other user programs. This feature is vital in the multiprogramming environment and is closely tied in with memory protection, accounting determinations, multiprogram interrupt management, intermodule communications control, and input/output operations. Each of these functions is implemented by a Processor instruction that requires the Master Mode. These are listed below.

All instructions available to the Processor in Slave Mode are available in Master Mode. The following instructions can be executed only when the Processor is in Master Mode.

1. Load Base Address Register (LBAR)
2. Load Timer Register (LDT)
3. Set Memory Controller Interrupt Cells (SMIC)
4. Read Memory Controller Mask Registers (RMCM)
5. Set Memory Controller Mask Registers (SMCM)
6. Connect Input/Output Channel (CIOC)

The last of these instructions, Connect Input/Output Channel, is the beginning of every peripheral operation. Thus, all peripheral operations are reserved for execution in Master Mode, and in particular by the Input/Output Supervisor within the Comprehensive Operating Supervisor.

Master Mode Entry

Although Master Mode operation by the Procesor is a primary safeguard for executive routines and user programs in memory, the applications programmer can force the Processor into this mode but only for accessing routines that are part of the Operating Supervisor. This is done by use of the Master Mode Entry (MME) instruction and one of the system-symbol operands listed in Appendix E and described fully in the General Comprehensive Operating Supervisor Manual. Any other use of MME causes an abort of the user program. Thus, through the MME instruction, the programmer can communicate with modules of the Operating Supervisor to exchange any necessary information for the execution of his program.

Mass Storage Orientation

"Compute overhead" time is reduced and multiprogramming is enhanced through the use of an external disc (mass) storage unit. The disc (and optionally a drum storage device) enables optimized accessing of system routines and performs data transfers at higher rates than other external storage media.

The disc and/or drum is used primarily for the following purposes:

1. System storage area--Least used submodules of the Operating Supervisor and all system programs are stored on the disc. Included in this storage area are the Assembler, compilers (FORTRAN and COBOL), portions of the operating system, subroutine library, sort/merge, utility routines used by system routines, tables associated with storage allocation and file/record assignments, operational statistics, hardware diagnostics, and the General Loader with its debugging routines.

2. Temporary data storage--Temporary data files used during a single activity can be stored on the disc or drum for fast access.

3. Permanent user files--Permanent data files can be stored on the disc or drum and accessed through the software system.

Program File Orientation

The software system is further described as file oriented because (1) the Comprehensive Operating Supervisor assigns peripheral devices to an activity and (2) it manages all assigned peripherals during input or output operations so that the programmer never deals directly with input/output subsystems or devices. The programmer references all peripherals by use of file code designators, two alphanumeric characters, that are

7

referenced in two ways: (1) on file control cards used by the Allocator in the Operating Supervisor to specify those files needed to execute the activity and, (2) in communicating to the File and Record Control program or to the Input/Output Supervisor. The file code designators and their assigned peripheral devices are maintained in the Peripheral Assignment Table (PAT) used by the Input/Output Supervisor for peripheral identification.

Software Reference Documentation

The following manuals and documents contain detailed descriptions of items mentioned in this chapter.

1. GE-625/635 Comprehensive Operating Supervisor Reference Manual, CPB-1195
2. GE-625/635 File and Record Control Reference Manual, CPB-1003
3. GE-625/635 General Loader Reference Manual, CPB-1008
4. GE-625/635 FORTRAN IV Reference Manual, CPB-1006
5. GE-625/635 COBOL Reference Manual, CPB-1007
6. GE-625/635 Sort/Merge Program Reference Manual, CPB-1005
7. GE-625/635 FORTRAN IV Mathematical Routine Library, CPB-1083
8. GE-625/635 Operator's Reference Manual, CPB-1045

II. GE-635 PROCESSOR

GENERAL CHARACTERISTICS

Major Functional Units

The Processor consists of two relatively independent units: the Control Unit and the Operations Unit.

The Control Unit provides Processor control functions and also serves as an interface between the Operations Unit and memory. In addition, the Control Unit performs the following principal functions:

1. Address modification
2. Address relocation
3. Memory protection for user and executive programs
4. Fault recognition
5. Interrupt recognition
6. Operation decoding

Since the Control Unit runs independently of the Memory module, a single Processor can be connected to memories with different cycle times. The Processor is designed to eliminate adverse interaction when memories with different cycle times are employed.

The Operations Unit performs all arithmetic and logical operations as directed by the Control Unit. The Operations Unit contains most of the registers available to a user program. This unit performs such functions as:

1. Fractional and integer divisions and multiplications
2. Automatic alignment of fixed-point numbers for additions and subtractions
3. Inverted divisions on floating-point numbers
4. Automatic normalization of floating-point resultants
5. Separate operations on the exponents and mantissas of floating-point numbers
6. Shifts
7. Indicator Register loading and storing
8. Base Address Register loading and storing
9. Timer Register loading and decrementing

Master/Slave Mode of Operation

To permit separation of control and object programs with corresponding protection of control programs from undebugged object programs, two modes of operation, Master and Slave, are provided in the Processor. Control programs will run in the Master Mode, and object programs will run in the Slave Mode. Programs running in Master Mode have

access to the entire memory, can initiate peripheral and internal control functions, and do not have base address relocation applied. Programs running in Slave Mode have access to a limited portion of the memory, cannot generate peripheral control functions, and have the Base Address Register added to all relative memory addresses of the object program.

Master Mode operation is the state in which the Processor:

1. Presents an "unrelocated" address to the memory

2. Has an unbounded access to memory

3. Causes the memory to be in the unprotected state when accessed by the Processor

    a. This permits setting of execute interrupt cells.

    b. When this Processor is designated the "control" Processor by the memory, as set by Memory module switches, this also permits generation of execute interrupts.

4. Permits setting the timer and Base Address Register by the appropriate instructions (Load Timer Register or Load Base Register, LDT and LBAR)

The Processor is in the Master Mode when any of the following exists:

1. The Master Mode Indicator is in the master condition
2. An execute interrupt is recognized
3. A fault is recognized

Slave Mode operation is the state in which the Processor:

1. Presents a relocated address to Memory as specified by bit positions 0-8 of the Base Address Register.

2. Restricts the effective address formed to the bounds specified by bit positions 9-17 of the Base Address Register.

3. Causes the memory to be in the "protected" state when accessed by the Processor. This prohibits generation of peripheral commands, interrupt masks, or setting of execute interrupt cells, even if the Processor is designated the control Processor by the Memory module.

4. Prohibits setting of the timer and Base Address Register by the instructions LDT or LBAR.

The Processor is in the Slave Mode when the Master Mode Indicator is in the slave condition or when the Transfer and Set Slave (TSS) instruction is being executed. (See page 19.)

Operation Overlapping

Instruction words are fetched in pairs and sequentially transferred to the Control Unit of the Processor where the instructions are directed to the primary and secondary instruction registers of the instruction decoder. If required, address modification is then performed using the first of the two instructions.

CPB-1004F

10

As soon as this is accomplished, the operand specified by the first instruction is requested from memory while the Control Unit concurrently performs any address modification required by the second of the instruction pair.

When the operand called for by the first instruction is obtained, the Control Unit transfers the operand to the Operations Unit, thus initiating the specified operation to be carried out. While this operation is being carried out by the Operations Unit, the operand specified by the second instruction is requested by the Control Unit. As soon as the second operand is received and the Operations Unit has finished with the first operand, the Control Unit signals the Operations Unit to carry out the second operation. Finally, while the second operation is being carried out, the next instruction pair is requested from memory.

Address Range Protection

Any object program address to be used in a memory access request while the Processor is in the Slave Mode is checked, just prior to the fetch, for being within the address range allocated by the Comprehensive Operating Supervisor (GECOS) to the program for this execution. This address range protection is commonly referred to as memory protection.

For the purpose of memory protection, the 18-bit Processor Base Address Register is loaded by GECOS with an address range in bit positions 9-16. The check takes place only in the Slave Mode. It consists of subtracting bit positions 0-7 of the program address from this address range, using the boundary comparator, when the result is zero or negative then the program address is out of range; and a Memory Fault Trap occurs. (Refer to page 22.)

More specifically, the checking is actually based on nine bits, namely the Base Address Register positions 9-17 and the bit positions 0-8 of the program address. This permits address range allocation to job programs in multiples of 512 words. Because of a software requirement, bits 8 and 17 of the Base Address Register have been wired in such a way that they contain zeros permanently and cannot be altered by the LBAR instruction. Thus, memory allocation and protection is performed in multiples of 1024 words.

In the Master Mode no checking takes place; thus, any memory location (in those Memory modules that are connected to this Processor) can be accessed.

Execution of Interrupts

When an execute interrupt request present signal is received from a Memory module system controller for which the Processor is the control Processor, the Processor carries out the interrupt procedure as soon as an instruction from an odd memory location has been executed that:

1. Did not have its interrupt inhibit bit position 28 set to a 1

2. Did not cause an actual transfer of control (A transfer of control is effected if the instruction is an unconditional transfer, a conditional transfer with the condition satisfied, or a programmed fault such as ZOP, MME, DRL, Fault Tag, Connect).

3. Was not an Execute or Execute Double (XEC or XED) instruction (Note than an XEC or XED instruction and the one or two instructions carried out under its control are regarded as a single instruction execution.)

4. Was accessed while in the repeat mode or the instruction following the termination of the repeat mode.

The interrupt procedure consists of the following steps:

1. Enter the Master Mode (the Master Mode Indicator is not affected).

2. Return the Execute command code to the system controller that sent the interrupt request present signal.

3. Receive a five-bit interrupt code on the data lines from this Memory module (bit positions 12-16), specifying the number of the highest priority nonmasked interrupt cell that was set to ON when the transfer interrupt number command code was recognized at the system controller.

4. Carry out a "wired in" XED.

$$Y = \begin{array}{|c|c|c|c|} \hline 000 \quad 000 \quad 000 & \begin{array}{c} \text{Memory} \\ \text{No.} \end{array} & \begin{array}{c} \text{Interrupt} \\ \text{Cell No.} \end{array} & 0 \\ \hline 0 \qquad\qquad\qquad 8 & 9 \qquad 11 & 12 \qquad\qquad 16 & 17 \\ \end{array}$$
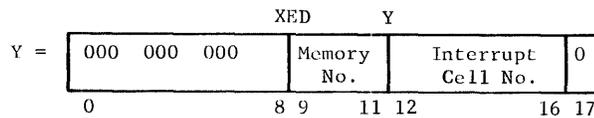
The cell number is determined by the highest priority unmasked interrupt cell (in the system controller) causing the execute interrupt.

5. Return to the mode specified by the Master Mode Indicator (see below) and continue with the instruction from the memory location specified by the Instruction Counter.

Each of the two instructions executed by the XED may affect the Master Mode Indicator as follows:

1. If this instruction results in an actual transfer of control and is not the Transfer and Set Slave instruction (TSS), then ON (that is, Master Mode).

2. If this instruction is either the Return instruction (RET) with bit 28 of the (RET) operand equal to 0 or the TSS instruction, then OFF (Slave Mode).

The first of the two instructions from the memory location Y must not alter the contents of the location of the second instruction, and must not be an XED instruction. If the first of the two instructions alters the contents of the Instruction Counter, then this transfer of control is effective immediately; and the second of the two instructions is not executed.

Interval Timer

The Processor contains a timer which provides a program interrupt at the end of a variable interval. The timer is loaded by GECOS and can be set to a maximum of approximately four minutes total elapsed time. (See pages 15 and 21.)

REGISTERS

The Processor block diagram (Figure 1) shows the program accessible registers as well as the major nonprogram accessible registers, adders, and switches. Only data and information paths are shown. The block diagram also shows the division between the Operations Unit and Control Unit.

The switches (rounded figures on the block diagram) control the flow of information between the registers, adders, and the memory interface.

Program Accessible Registers

The following table shows the registers accessible to the program.

Name	Mnemonic	Length
Accumulator Register	AQ	72 bits
Eight Index Registers (n=0,...,7)	Xn	18 bits each
Exponent Register	E	8 bits
Base Address Register	BAR	18 bits
Indicator Register	IR	18 bits
Timer Register	TR	24 bits
Instruction Counter	IC	18 bits

1. The AQ-register is used as follows:

   a. In floating-point operations as a mantissa register for single and double precision

   b. In fixed-point operations as an operand register for double precision

   c. In fixed-point operations as operands for single precision where each AQ half serves independently of the other. The halves then are called the A-register, (namely AQ_{0-35}) and the Q-register, (namely AQ_{36-71}).

   d. In address modification each half of A as well as of Q is an index register. These halves then are called AU (namely A_{0-17}), AL (namely A_{18-35}), QU (namely Q_{0-17}), and QL (namely Q_{18-35}).

2. The Xn-registers are used as follows:

   a. In fixed-point operations as operand registers for half precision

   b. In address modification as index registers

3. The E-register supplements the AQ-register in floating-point operations, serving as the exponent register.

CPB-1004F

13
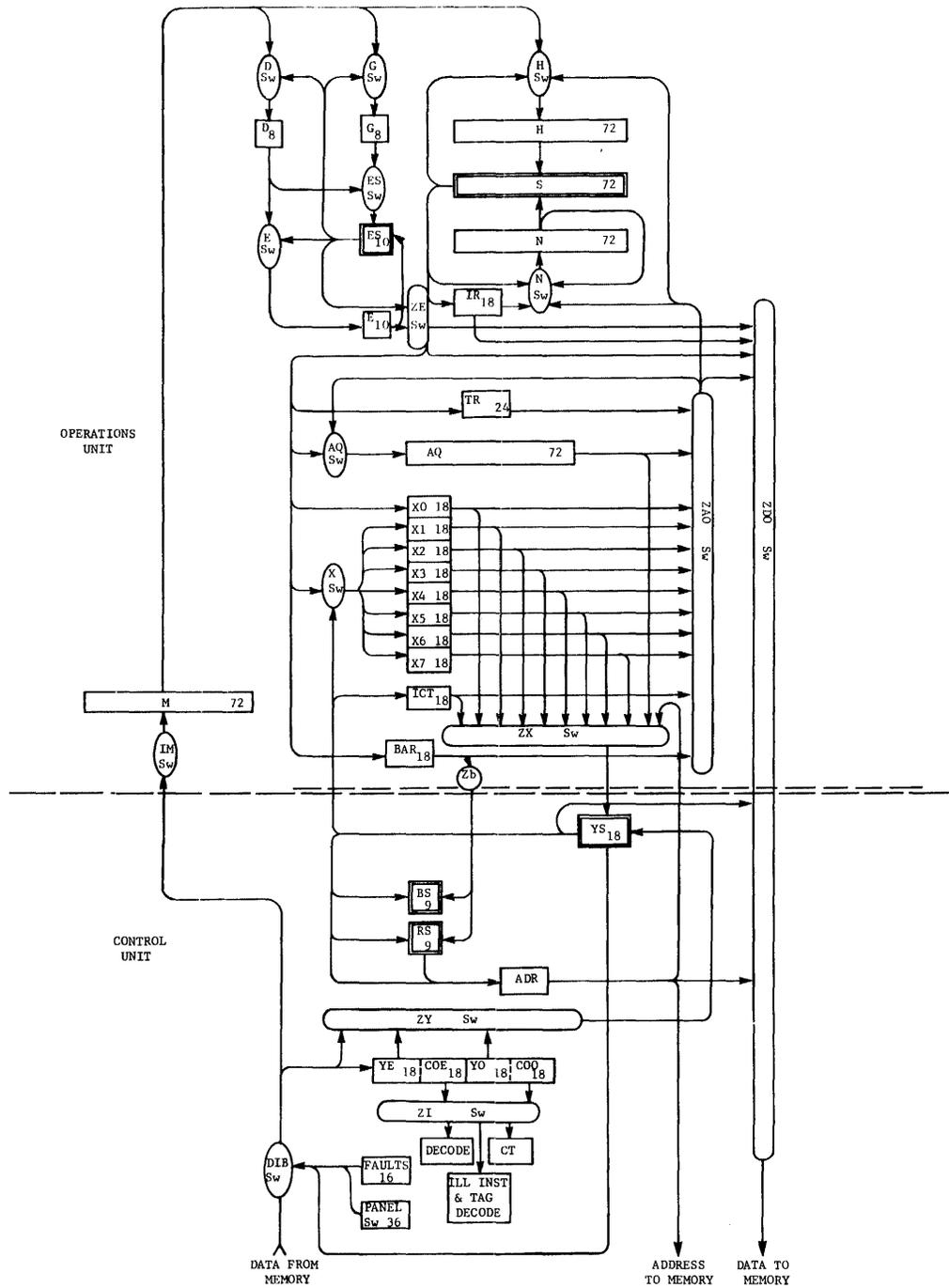
Figure 1. Block Diagram of Principal Processor Registers

14

4. The Base Address Register (BAR) is used in address translation and memory protection. It stores the base address and the number of 1024-word blocks assigned to the object program being executed.

5. The Indicator Register (IR) is a generic term for all the program-accessible indicators within the Processor. The name is used where the set of indicators appears as a register, that is, as source or destination of data.

6. The Timer Register (TR) is decremented by one each 15.625 microseconds, and a Timer Runout Fault Trap occurs whenever its contents reach zero. If Timer Runout occurs in Master Mode, the trap does not occur until the Processor returns to Slave Mode; but decrementation continues beyond zero.

7. The Instruction Counter holds the address of the next instruction to be executed.

Program Nonaccessible Registers

The following listed registers are used in Processor operations but are not referenced in machine instructions.

Mnemonic	Length
M	72 bits
H	72 bits
N	72 bits
D	8 bits
G	8 bits
ADR	18 bits
YE	18 bits
YO	18 bits
COE	18 bits
COO	18 bits

1. The M-register is an intermediate register used to buffer operands coming in from memory.

2. The H- and N-registers are intermediate registers used to hold the operands which are presented to the main, 72-bit (S) adder.

3. The D-register is used to hold the exponent of the operand from memory in floating-point operations.

4. The G-register contains the number of shifts necessary in shifting, floating-point, and fixed-point multiply and divide operations.

5. The ADR (Address)-register is used to hold the absolute address of memory cells when making memory accesses.

6. The YE- and YO-registers contain the address portions of the even and odd instruction respectively of an accessed instruction pair.

7. The COE- and COO-registers contain the lower half of each instruction word and include the operation code and the tag field portions of the even and odd instructions respectively of an instruction pair.

15

Adders

The following table lists the Processor adders.

Name	Length
S	72 bits
YS	18 bits
ES	10 bits
BC(Comparator)	9 bits
RS	9 bits

1. The S-adder is the main adder in the Processor. It is used for fixed- and floating-point additions, subtractions, multiplications, and divisions.

2. The YS-adder is used to compute the effective addresses of instructions and operands.

3. The ES-adder is the exponent adder; it is used for exponent operations in floating-point operations.

4. The RS-adder is used to compute the absolute addresses of instructions and operands.

5. The BC-adder, although not implemented as a complete adder, is used to determine if an effective address is out of the range allocated to the operating program (memory protection).

PROCESSOR INDICATORS

General

The indicators can be regarded as individual bit positions in an 18-bit half-word Indicator Register (IR).

An indicator is set to the ON or OFF state by certain events in the Processor, or by certain instructions. The ON state corresponds to a binary 1 in the respective bit position of the IR; the OFF state corresponds to a 0.

The description of each machine instruction on pages 39 through 148 includes a statement about (1) those indicators that may be affected by the instruction and (2) the condition under which a setting of the indicators to a specific state occurs. If the conditions stated are not satisfied, the status of this indicator remains unchanged.

The instruction set includes certain instructions which transfer data between the lower half of a storage location and the Indicator Register. The following table lists the indicators that have been implemented, their relation to the bit positions of the lower half of a memory location, and the instructions directly affecting indicators. (A detailed explanation of indicator settings is given with each machine instruction.)

Implementation	Bit Position	Indicator	Indicator Instructions
Assigned	18 19 20 21 22 23 24 25 26 27 28	Zero Negative Carry Overflow Exponent Overflow Exponent Underflow Overflow Mask Tally Runout Parity Error Parity Mask Master Mode	1. Load Indicators (LDI) 2. Store Indicators (STI) 3. Store Instruction Counter Plus 1 and Indicators (STC1) 4. Return (RET)
Unassigned	29 30 31 32 33 34 35	Must be Zero	

The following descriptions of the individual indicators are limited to general statements only.

Zero Indicator

The Zero Indicator is affected by instructions that change the contents of a Processor register (A, Q, AQ, Xn, BAR, IR, TR) or adder, and by comparison instructions.

The indicator is set ON when the new contents of the affected register or adder contains all binary 0's; otherwise the indicator is set OFF.

Negative Indicator

The Negative Indicator is affected by instructions that change the contents of a Processor register (A, Q, AQ, Xn, BAR, IR, TR) or adder, and by comparison instructions.

The indicator is set ON when the new contents of bit position 0 of this register or adder is a binary 1; otherwise it is set OFF.

Carry Indicator

The Carry Indicator is affected by left shifts, additions, subtractions, and comparisons.

The indicator is set ON when a carry is generated out of bit position 0; otherwise it is set OFF. If bit position 0 ever changes during the shift, set the indicator ON, otherwise OFF.

Overflow Indicator

The Overflow Indicator is affected by the arithmetic instructions, but not by compare instructions and Add Logical (ADL(R)) or Subtract Logical (SBL(R)) instructions.

Exponent Overflow Indicator

The Exponent Overflow Indicator is affected by arithmetic operations with floating-point numbers or with the exponent register (E).

The indicator is set ON when the exponent of the result is larger than +127 which is the upper limit of the exponent range.

Since it is not automatically set to OFF otherwise, the Exponent Overflow Indicator reports any exponent overflow that has happened since it was last set OFF by certain instructions (LDI, RET, and Transfer on Exponent Overflow (TEO)).

Exponent Underflow Indicator

The Exponent Underflow Indicator is affected by arithmetic operations with floating-point numbers, or with the exponent register (E).

The indicator is set ON when the exponent of the result is smaller than -128 which is the lower limit of the exponent range.

Since it is not automatically set to OFF otherwise, the Exponent Underflow Indicator reports any exponent underflow that has happened since it was last set OFF by certain instructions (LDI, RET, and Transfer on Exponent Underflow (TEU)).

Overflow Mask Indicator

The Overflow Mask Indicator can be set ON or OFF only by the instructions LDI and RET.

When the Overflow Mask Indicator is ON, then the setting ON of the Overflow Indicator, Exponent Overflow Indicator, or Exponent Underflow Indicator does not cause an Overflow Fault Trap to occur. When the Overflow Mask Indicator is OFF, such a trap will occur.

Clearing of the Overflow Mask Indicator to the unmask state does not generate a fault from a previously set Overflow Indicator, Exponent Overflow Indicator, or Exponent Underflow Indicator. The status of the Overflow Mask Indicator does not affect the setting, testing, or storing of the Overflow Indicator, Exponent Overflow Indicator, or Exponent Underflow Indicator.

CPB-1004F

Tally Runout Indicator

The Tally Runout Indicator is affected by the Indirect Then Tally (IT) address modification type (all designators except Indirect and Fault) and by the Repeat, Repeat Double, and Repeat Link instructions (RPT, RPD, and RPL).

The termination of a Repeat instruction because a specified termination condition is met sets the Tally Runout Indicator to OFF.

The termination of a Repeat instruction because the tally count reaches 0 (and for RPL because of a 0 link address) sets the Tally Runout Indicator to ON; the same is true for tally equal to 0 in some of the IT address modifications.

Parity Error Indicator

The Parity Error Indicator is set to ON when a parity error is detected during the access of one or both words of Y-pair from memory.

It may be set to OFF by the LDI or RET instruction.

Parity Mask Indicator

The Parity Mask Indicator can be set to ON or OFF only by the instructions LDI and RET.

When the Parity Mask Indicator is ON, the setting of the Parity Error Indicator does not cause a Parity Error Fault Trap to occur. When the Parity Mask Indicator is OFF, such a trap will occur.

Clearing of the Parity Mask Indicator to the unmasked state does not generate a fault from a previously set Parity Error Indicator. The status of the Parity Mask Indicator does not affect the setting, testing, or storing of the Parity Error Indicator.

Master Mode Indicator

The Master Mode Indicator can be changed only by an instruction. For a description of how the indicator can be changed, refer to the following instruction descriptions:

Instruction	Reference
Master Mode Entry (MME)	Page 132
Return (RET)	Page 125
Derail (DRL)	Page 133
Transfer and Set Slave (TSS)	Page 124

When the Master Mode Indicator is ON, the Processor is in the Master mode; however, the converse is not necessarily true. (See the MME and DRL descriptions.)

CPB-1004F

19

FAULT TRAPS

Trapping Procedure

Sixteen types of faults and other events each have a fault trap assigned. Some of these events have nothing to do with actual faults; they are included here because they are treated the same as faults.

The fault trap procedure is similar to the interrupt procedure (page 12) except that the effective address is defined differently. The fault trap procedure consists of the following steps:

1. Automatically enter the Master Mode (the Master Mode Indicator is not affected).

2. Carry out an Execute Double (XED) instruction (page 131) with an effective address (Y) as defined for bits 0-17 of a machine word as follows:

ZEROS	Constant	Code	0
0	5 6	12 13	16 17
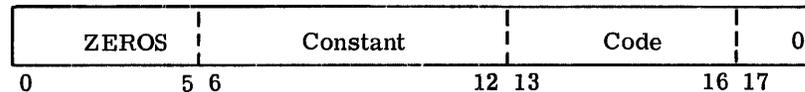
Constant: Set up by the fault switches in the Processor (also see the description of the instructions Master Mode Entry (MME) and Derail (DRL).

Code: The four-bit fault trap code which identifies the respective fault trap (see Figure 2).

3. Return to the mode specified by the Master Mode Indicator, and continue with the instruction from the memory location specified by the Instruction Counter.

Each of the two instructions from the memory location Y-pair may affect the Master Mode indicator as follows: If this instruction results in any actual transfer of control and is not the Transfer and Set Slave instruction (TSS), then ON; If this instruction is either a return instruction (RET) with bit 28 of the operand (RET) equal to 0 or the TSS instruction, then OFF (Slave Mode).

The first of the two instructions from the memory location Y must not alter the contents of the location of the second instruction, and must not be an Execute Double instruction (XED). If the first of the two instructions alters the contents of the Instruction Counter, then this transfer of control is effective immediately; and the second of the two instructions is not executed.

Fault Categories

There are four general categories of faults:

1. Instruction generated (by execution of instruction)
2. Program generated
3. Hardware generated
4. Manually generated

- **Instruction Generated Faults.** The Instruction generated faults are:

    1. Master Mode Entry (MME)

        The instruction Master Mode Entry has been executed (page 132).

    2. Derail (DRL)

        The instruction Derail has been executed (page 133).

    3. Fault Tag

        The address modifier IT where T=F has been recognized. The indirect cycle will not be made upon recognition of F, nor will the operation be completed.

    4. Connect (CON)

        The Processor has received a Connect from a Control Processor via a System Controller.

    5. Illegal OP Code (ZOP)

        An operation code of all zeros has been executed.

- **Program Generated Faults.** Program generated faults are defined as:

    1. The Arithmetic Faults

        a. Overflow (FOFL)--An arithmetic overflow, exponent overflow, or exponent underflow has been generated. The generation of this fault is inhibited when the Overflow Mask is in the mask state. Subsequent clearing of the Overflow Mask to the unmasked state will not generate this fault from previously set indicators. The Overflow Fault Mask state does not affect the setting, testing, or storing of indicators.

        b. Divide Check (FDIV)--A divide check fault occurs when the actual division cannot be carried out for one of the reasons specified with each divide instruction.

    2. The Elapsed Time Interval Faults

        a. Timer Runout (TROF)--This fault is generated when the timer count reaches zero. If the Processor is in Master Mode, recognition of this fault will be delayed until the Processor returns to the Slave Mode; this delay does not inhibit the counting in the Timer Register.

        b. Lockup (LUF)--The Processor is in a program lockup which inhibits recognizing an execute interrupt or interrupt type fault for greater than 16 milliseconds. Examples of this condition are the coding TRA*, the continuous use of inhibit bit, or Repeat Mode loops exceeding 16 milliseconds.

        c. Operation Not Completed (FONC)--This fault is generated due to one of the following:

            1) No System Controller attached to the Processor for the address.

            2) Operation Not Completed. (See Hardware Generator Faults, page 22.)

CPB-1004F

21

3. The Memory Faults

    a. Command (FCMD)--This fault is interpreted as an illegal request by the Processor for action of the System Controller. These illegal requests are:

        1) The Processor is in the Slave Mode, and issues a CIOC, RMCM, SMCM, or SMIC. The CIOC, SMCM, and SMIC commands will not be executed. (Refer to page 257 for descriptions and references concerning these instruction mnemonics.)

        2) The Processor has issued a connect to a channel that is masked off (by program or switch).

        3) The Processor is in the Slave Mode and encounters a Delay until Interrupt Signal (DIS) instruction.

    b. Memory (FMEM)--This fault is generated when:

        1) No physical memory existed for the address.

        2) An address (in Slave Mode) is outside the program boundary.

● Hardware-Generated Faults. The hardware-generated faults are defined as:

    1. Operation Not Completed (FONC)--This fault is generated due to one of the following:

        a. The Processor has not generated a memory operation within 1 to 2 milliseconds and is not executing the Delay Until Interrupt Signal (DIS) instruction.

        b. The System Controller closed out a double-precision or read-alter-rewrite cycle.

        c. See Operation Not Completed under Program Generated Faults (page 21).

    2. Parity (FPAR)--This fault is generated when a parity error exists in a word which is read from a core location:

        a. Single- or double-instruction word fetch--if the odd instruction contains a parity error, the instruction counter retains the location of the even instruction.

        b. Indirect word fetch--if a parity error exists in an indirect and tally word in which the word is normally altered and replaced, the contents of that memory location are destroyed.

        c. Operand fetch--when a single-precision operand, C(Y) is requested, the contents of the memory pair location at Y, Y+1 where Y is even, or Y-1, Y, where Y is odd are read from memory. The System Controller will not report a parity error if it occurs in C(Y+1) or C(Y-1), but will restore the C(Y+1) or C(y-1) with its parity bit unchanged.

            If a parity error occurs on any instruction for which the C(Y) are taken from a core location (this includes "to storage" instructions, ASA, ANSA, etc.,) the Processor operation is completed with the faulty operand before entering the fault routine.

The generation of this fault is inhibited when the Parity Mask Indicator is in the mask state. Subsequent clearing of the Parity Mask to the unmasked state will not generate this fault from a previously set Parity Error Indicator. The Parity Mask does not affect the setting, testing, or storing of the Parity Indicator.

● **Manually Generated Faults.** Manually generated faults are:

    1. Execute (EXF)

        a. The EXECUTE pushbutton on the Processor maintenance panel has been activated.

    2. The Power Turn On/Off Faults

        a. Startup (SUF)--A power turn-on has occurred.

        b. Shutdown (SDF)--Power will be turned off in approximately 1 millisecond.

Fault Priority

The 16 faults are organized into five groups to establish priority for the recognition of a specific fault when faults occur in more than one group. Group 1 has highest priority.

Only one fault within a priority group is allowed to be active at any one time. In the event that two or more faults occur concurrently, only the fault which occurs first through normal program sequence is permitted.

Fault Recognition

Faults in Groups I and II cause the operations in the Processor to abort unconditionally.

Faults in Groups III and IV cause the operations in the Processor to abort when the operation currently being executed is completed.

Faults in Group V are recognized under the same conditions that Program interrupts are recognized. (See page 12.) Faults in Group V have priority over Program Interrupts and are also subject to being inhibited from recognition by use of the inhibit bit in the instruction word.

CPB-1004F

Instruction Counter (IC)

Upon recognition of a fault, the contents of the Instruction Counter (IC) are as shown in the Table of Faults below.

Fault Code	Fault Name	Group (Priority)	IC Contents[1]
1100	Startup	I	N+0, +1, or +2
1111	Execute	I	N+0, +1, or +2
1011	Operation Not Completed	II	N+0, +1, or +2
0111	Lockup	II	N+0, +1, or +2
1110	Divide Check	III	N[3]
1101	Overflow	III	N
1001	Parity	IV	N[2]
0101	Command	IV	N
0001	Memory	IV	N[3]
0010	Master Mode Entry	IV	N[3]
0110	Derail	IV	N[3]
0011	Fault Tag	IV	N[3]
1010	Illegal Op Code	IV	N
1000	Connect	V	N
0100	Timer Runout	V	N
0000	Shut Down	V	N

Notes:

[1] N = Last operation

[2] If parity occurred on operand fetch, operation N+1 was completed with faulty data.
If parity occurred on instruction fetch, operation N+1 was not completed
If parity occurred on IT, IT was not completed.

[3] These operations are considered complete when the fault is recognized.

Figure 2. Table of Faults

THE NUMBER SYSTEM

The binary system of notation is used throughout the GE-625/635 information processing system.

Many of the instructions, mainly additions, subtractions, and comparisons, can be used in two ways: either operands and results are regarded as signed binary numbers in the 2's complement form (the "arithmetic" case), or they are regarded as unsigned, positive binary numbers (the "logic" case). The Zero and the Negative Indicators facilitate the general interpretation of the results in the arithmetic case; the Zero and the Carry Indicators, in the logic case. The instruction set contains instruction types "Add Logic" and "Subtract Logic" which particularly facilitate arithmetic of the logic type with half-word, single-word, and double-word precision. See Appendix I for a description of the two's complement number system.

Subtractions are carried out internally by adding the 2's complement of the subtrahend.* It is a characteristic feature of the 2's complement representation that a "no borrow" condition in the case of true subtraction corresponds to a "carry" condition in the case of addition of the 2's complement, and vice versa.

A statement on the assumed location of the binary point has significance only for multiplications and divisions. These two operations are implemented for integer arithmetic as well as for fractional arithmetic with numbers in 2's complement form, "integer" meaning that the position of the binary point may be assumed to the right of the least-significant bit position (that is, to the right of bit position 35 or 71, depending on the precision of the respective number) and "fractional" meaning that the position of the binary point may be assumed to the left of the most-significant bit position (that is, between the bit positions 0 and 1).

REPRESENTATION OF INFORMATION

The Processor is fundamentally organized to deal with 36-bit groupings of information. Special features are also included for ease in manipulating 6-bit groups, 9-bit groups, 18-bit groups, and 72-bit, double-precision groups. These bit groupings are used by the hardware and software to represent a variety of forms of information.

Position Numbering

The numbering of bit positions, character positions, words, etc., increases in the direction of conventional reading and writing: from the most- to the least-significant digit of a number, and from left to right in conventional alphanumeric text.

Graphical presentations in this manual show registers and data with position numbers increasing from left to right.

The Machine Word

The machine word consists of 36 bits arranged as follows:

```
0                         17|18                    35
 ┌──────────────────────────┬──────────────────────┐
 │        One Machine       |Word                   │
 └──────────────────────────┴──────────────────────┘
       Upper Half word        |      Lower Half word
```
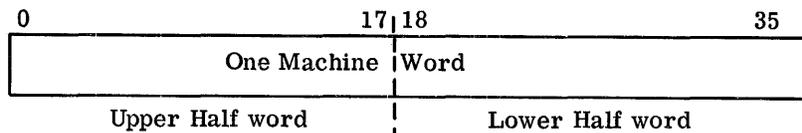
Data transfers between the Processor and memory are word oriented: 36 bits are transferred at a time for single-precision data and two successive 36-bit word transfers for double-precision data. When words are transferred to a Magnetic Core Storage Unit, this unit adds a parity bit to each 36-bit word before storing it. When words are requested

---

*When the subtrahend is zero, the algorithm for forming the 2's complement is still carried out. Thus, each bit of the subtrahend is complemented, and a 1 is added into the least-significant position of the parallel adder.

from a Magnetic Core Storage Unit, this unit verifies the parity bit read from the store and removes it from the word transferred prior to sending each word to the Processor.

The Processor has many built-in features for transferring and processing pairs of words. In transferring a pair of words to or from memory, a pair of memory locations is accessed; these addresses are an even and the next-higher odd number.
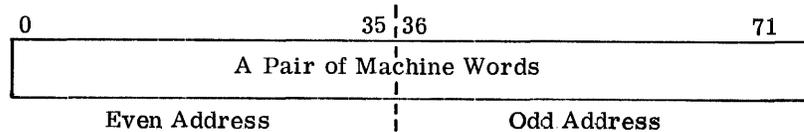
In addressing such a pair of memory locations in an instruction that is intended for handling pairs of machine words, either of the two addresses may be used as the effective address (Y). Thus,

If Y is even, the pair of locations (Y, Y+1) is accessed. If Y is odd, the pair of locations (Y-1, Y) is accessed. The term "Y-pair" is used for each such pair of addresses.

Alphanumeric Data

Alphanumeric data are represented by six-bit or nine-bit characters. A machine word contains either six or four characters:

Six-Bit Character   Nine-Bit Character

Operand from memory

Operand from
memory:

X	X	X	Char.	X	X
0	1	2	3	4	5

X	X	Char.	X
0	1	2	3

0	0	0	0	0	Char.
0	1	2	3	4	5

0	0	0	Char.
0	1	2	3

Operand for the operation

Result of the operation

Result to
memory:

X	X	X	X	X	Char.
0	1	2	3	4	5

X	X	X	Char.
0	1	2	3

X	X	X	Char.	X	X
0	1	2	3	4	5

X	X	Char.	X
0	1	2	3

Result in memory
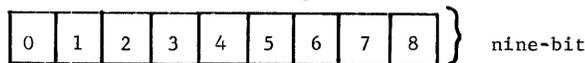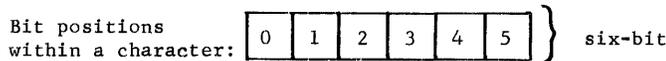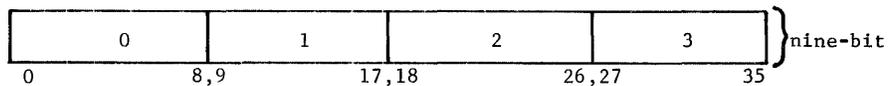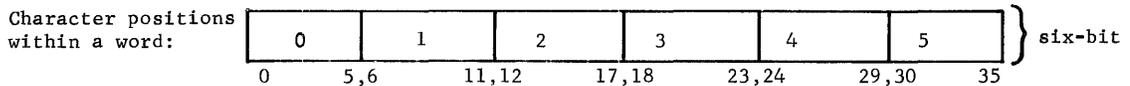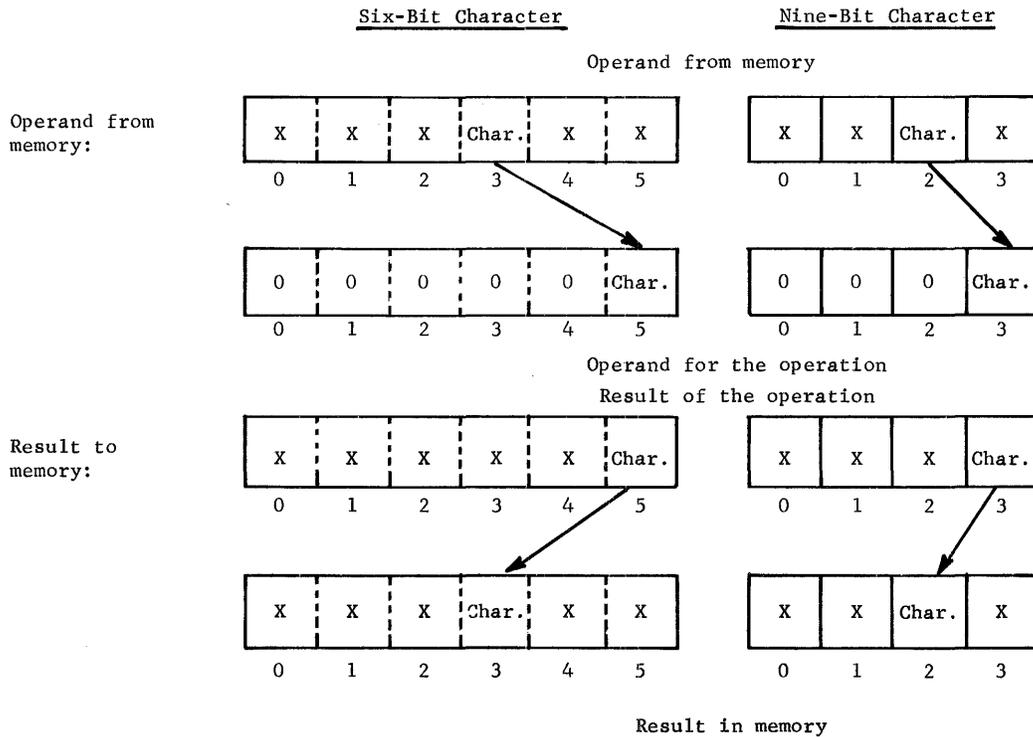
For six-bit character operations in which the operand is taken from memory, the effective operand from memory is presented as a single word with the specified character justified to character position 5; position 0-4 are presented as zero. For operations in which the resultant is placed in memory, character 5 of the resultant replaces the specified character in memory location Y; the remaining characters in memory location Y are not changed.

For nine-bit character operations in which the operand is taken from memory, the effective operand from memory is presented as a single word with the specified character justified to character position 3; positions 0-2 are presented as zero. For operations in which the resultant is placed in memory, character 3 of the resultant replaces the specified character in memory location Y; the remaining characters in memory location Y are not changed.

The character set used is the Computer Equipment Department Standard Character Set, which is readily convertible to and from the ASCII character set.

Binary Fixed-Point Numbers

The instruction set comprises instructions for binary fixed-point arithmetic with half-word, single-word, and double-word precision.

PRECISION   REPRESENTATION
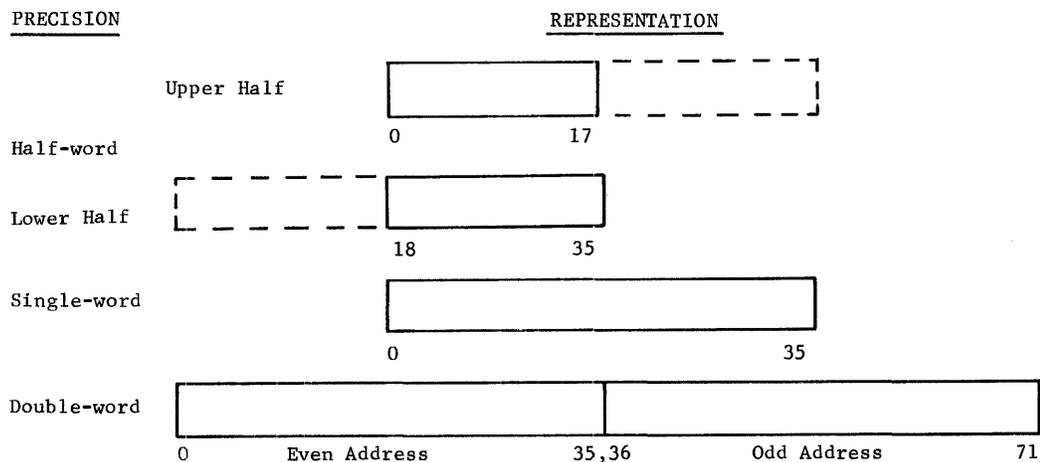
Instructions can be divided into two groups according to the way in which the operand is interpreted: the "logic" group and the "algebraic" group.

For the "logic" group, operands and results are regarded as unsigned, positive binary numbers. In the case of addition and subtraction, the occurrence of any overflow is reflected by the carry out of the most-significant (leftmost) bit position:

1. Addition   --   If the carry out of the leftmost bit position equals 1, then the result is above the range.

2. Subtraction --   If the carry out of the leftmost bit position equals 0, then the result is below the range.

In the case of comparisons, the Zero and Carry Indicators show the relation.

For the "algebraic" group, operands and results are regarded as signed, binary numbers, the leftmost bit being used as a sign bit, (a 0 being plus and 1 minus). When the sign is positive all the bits represent the absolute value of the number; and when the sign is negative, they represent the 2's complement of the absolute value of the number.

In the case of addition and subtraction the occurrence of an overflow is reflected by the carries into and out of the leftmost bit position (the sign position). If the carry into the leftmost bit position does not equal the carry out of that position then overflow has occurred. If overflow has been detected and if the sign bit equals 0, the resultant is below range; if with overflow, the sign bit equals 1, the resultant is above range.

An explicit statement about the assumed location of the binary point is necessary only for multiplication and division; for addition, subtraction, and comparison it is sufficient to assume that the binary points are "lined up."

CPB-1004F

28

In the GE-625/635 Processor, multiplication and division are implemented in two forms for 2's complement numbers: integer and fractional.

In integer arithmetic, the location of the binary point is assumed to the right of the least-significant bit position, that is, depending on the precision, to the right of bit position 35 or 71. The general representation of a fixed-point integer is then:

$$-a_n2^n + a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \ldots + a_12^1 + a_02^0$$

where a_n is the sign bit.

In fractional arithmetic, the location of the binary point is assumed to the left of the most-significant bit position, that is, to the left of bit position 1. The general representation of a fixed-point fraction

$$-a_02^0 + a_12^{-1} + a_22^{-2} + \ldots + a_{n-1}2^{-(n-1)} + a_n2^{-n}$$

The number ranges for the various cases of precision, interpretation, and arithmetic are listed in Figure 3.

Inter-pretation	Arithmetic	Precision		
		Half-Word (Xn, $Y_{0\ldots17}$)	Single-Word (A,Q,Y)	Double-Word (AQ, Y-pair)
Algebraic	Integral	$-2^{17} \leq N \leq (2^{17}-1)$	$-2^{35} \leq N \leq (2^{35}-1)$	$-2^{71} \leq N \leq (2^{71}-1)$
	Fractional	$-1 \leq N \leq (1-2^{-17})$	$-1 \leq N \leq (1-2^{-35})$	$-1 \leq N \leq (1-2^{-71})$
Logic	Integral	$0 \leq N \leq (2^{18}-1)$	$0 \leq N \leq (2^{36}-1)$	$0 \leq N \leq (2^{72}-1)$
	Fractional	$0 \leq N \leq (1-2^{-18})$	$0 \leq N \leq (1-2^{-36})$	$0 \leq N \leq (1-2^{-72})$

Figure 3. Ranges of Fixed-Point Numbers

CPB-1004F

29

Binary Floating-Point Numbers

The instruction set contains instructions for binary floating-point arithmetic with numbers of single-word and double-word precision. The upper 8 bits represent the integral exponent E in the 2's complement form, and the lower 28 or 64 bits represent the fractional mantissa M in 2's complement form. The notation for a floating-point number Z is:

$$Z_{(2)} = M_{(2)} \times 2^{E(2)}$$

Single-Word Precision:

```
0 1          7 8 9                              35
  ┌─┬────────┬─┬──────────────────────────────┐
  │S│        │S│                              │
  └─┴────────┴─┴──────────────────────────────┘
  ├──── E ───┤├────────── M ──────────────────┤
```

Double-Word Precision:

```
0 1          7 8 9                                              71
  ┌─┬────────┬─┬──────────────────────────────────────────────┐
  │S│        │S│                                              │
  └─┴────────┴─┴──────────────────────────────────────────────┘
  ├──── E ───┤├────────────── M ──────────────────────────────┤
```

where S = Sign bit
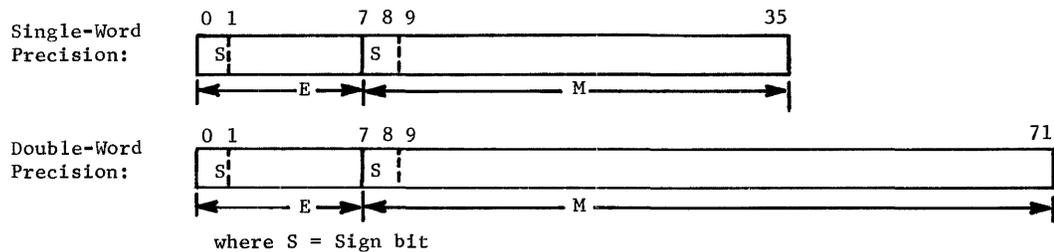
Before doing floating-point additions or subtractions, the Processor aligns the number which has the smaller positive exponent. To maintain accuracy, the lowest permissible exponent of -128 together with the mantissa equal to 0.00....0 has been defined as the machine representation of the number zero (which has no unique floating-point representation). Whenever a floating-point operation yields a resultant untruncated machine mantissa equal to zero (71 bits plus sign because of extended precision), the exponent is automatically set to -128.

The general representation of the exponent for single and double precision is:

$$-e_7 2^7 + e_6 2^6 + \ldots + e_1 2^1 + e_0 2^0$$

where e_7 is the sign.

The general representations of single- and double-precision mantissas are:

Single Precision: $\quad -m_0 2^0 + m_1 2^{-1} + m_2 2^{-2} + \ldots + m_{26} 2^{-26} + m_{27} 2^{-27}$

and

Double Precision: $\quad -m_0 2^0 + m_1 2^{-1} + m_2 2^{-2} + \ldots + m_{62} 2^{-62} + m_{63} 2^{-63}$

where m_0 is the sign in both cases.

Normalized Floating-Point Numbers

For normalized floating-point numbers, the binary point is placed at the left of the most-significant bit of the mantissa (to the right of the sign bit). Numbers are normalized by shifting the mantissa (and correspondingly adjusting the exponent) until no leading zeros are present in the mantissa for positive numbers, or until no leading ones are present in the mantissa for negative numbers. Zeros fill in the vacated bit positions. With the exception of the number zero (represented as 0×2^{-128}), all normalized floating-point numbers will contain a binary 1 in the most-significant bit position for positive numbers and a binary 0 in the most-significant bit position for negative numbers. Some examples are:

Unnormalized positive number	$(0\,\underset{S}{\mid}\,0001101) \times 2^{7}$
Same number normalized	$(0\,\underset{S}{\mid}\,1101000) \times 2^{4}$
Unnormalized negative number	$(1\,\underset{S}{\mid}\,11010111) \times 2^{-4}$
Same number normalized	$(1\,\underset{S}{\mid}\,01011100) \times 2^{-6}$

The number ranges resulting from the various cases of precision, normalization, and sign are listed in the table following:

	Sign	Single Precision	Double Precision
Normalized	Positive	$2^{-129} \le N \le (1-2^{-27})2^{127}$	$2^{-129} \le N \le (1-2^{-63})2^{127}$
	Negative	$-(1+2^{-26})2^{-129} \ge N \ge -2^{127}$	$-(1+2^{-62})2^{-129} \ge N \ge -2^{127}$
Unnormalized	Positive	$2^{-155} \le N \le (1-2^{-27})2^{127}$	$2^{-191} \le N \le (1-2^{-63})2^{127}$
	Negative	$-2^{-155} \ge N \ge -2^{127}$	$-2^{-191} \ge N \ge -2^{127}$

NOTE: The floating-point number zero is not included in the table.

Figure 4. Ranges of Floating-Point Numbers

Decimal Numbers

The instruction set does not comprise instructions for decimal arithmetic. The representation of decimal numbers in the machine therefore depends entirely on the programs used for performing the decimal arithmetic required.

CPB-1004F

The representation of the decimal digits as a subset of the character set is shown in Appendix F.

Instructions

Machine instructions have the following general format:

y	Op Code	0	i	0	Tag
0	17,18		26,27,28,29,30		35
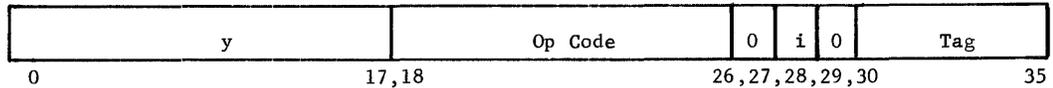
Where

y   = the address field; also used in some cases to augment the Op Code as in shift operations where it specifies the number of shifts

Op Code = the operation code, usually stated in the form of a 3-digit octal number

i   = interrupt inhibit bit

Tag   = the tag field, generally used to control the address modification

0   = the two bit positions 27 and 29 have no function at this time; however, they must be zero for compatibility with other 600-line Processors.

The three repeat instructions, Repeat, Repeat Double, and Repeat Link (RPT, RPD, and RPL), use a different instruction format. (See pages 134, 137, and 141.)

Indirect words have the same general format as the instruction words; however, the fields are used in a somewhat different way. (See page 35 and following.)

ADDRESS TRANSLATION AND MODIFICATION

Address Translation

Any program address to be used in a memory access request while the Processor is in the Slave Mode is first translated into an actual address and then submitted to the memory.

The term "program address" is used for the following addresses:

1. An instruction address which is the address used for fetching instructions

2. A tentative address which is the address used for fetching an indirect word

3. An effective address, which is the final address produced by the address modification process, is the address used for obtaining an operand, for storing a result, or for other special operations during which the memory is accessed using the effective address.

CPB-1004F

For the purpose of address translation, the Processor Base Address Register contains a base address in bit positions 0-7. The translation takes place only in the Slave Mode of operation. It consists of adding this base address to bit positions 0-7 of the program address, using the Relocation Adder (RS).

In the Master Mode no address translation takes place. Any program address to be used in a memory access request while the Processor is in the Master Mode is used directly as an actual address and submitted to the memory without any translation.

Address translation is actually based on nine bits, namely the Base Address Register positions 0-8 and the bit positions 0-8 of the program address; this permits address relocation by multiples of 512 words. Because of a software requirement, bit positions 8 and 17 of the Base Address Register have been wired in such a way that they contain 0's permanently and cannot be altered by the Load Base Address Register (LBAR) instruction. Thus, address relocation is performed in multiples of 1024.

Tag Field

Before the operation of an instruction is carried out, an address modification procedure generally takes place as directed by the tag field of the instruction and of indirect words. Only the repeat mode instructions (RPT, RPD, RPL) and character store instructions (STCA, STCQ, STBA, STBQ) do not provide for an address modification.   (See pages 134, 127, 129, and 56-59 respectively.)

The tag field consists of two parts, t_m and t_d, that are located within the instruction word as follows:
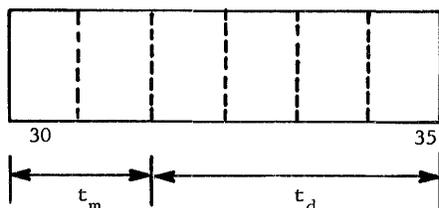
Where

t_m specifies one of the four possible modification types: Register (R), Register then Indirect (RI), Indirect then Register (IR), and Indirect then Tally (IT)

t_d specifies further the action for each modification type:

1. In the case of t_m R, RI, or IR, t_d is called the register designator and generally specifies the register to be used in indexing.

2. In the case of t_m = IT, t_d is called the tally designator and specifies the tallying in detail.

Modification Types

The following table gives a general characterization of each of the four modification types.

t_m	Binary	Modification Type
R	00	**Register** Indexing according to t_d as register designator and termination of the address modification procedure.
RI	01	**Register then Indirect** Indexing according to t_d as register designator, then substitution and continuation of the modification procedure as directed by the Tag field of this indirect word.
IR	11	**Indirect then Register** Saving of t_d as final register designator, then substitution and continuation of the modification procedure as directed by the Tag field of this indirect word.
IT	10	**Indirect then Tally** Substitution, then use of this indirect word according to t_d as tally designator.

Register Designator

Each of the three modification types R, RI, IR includes an indexing step which is further specified by the register designator t_d. In most cases, t_d really specifies the register from which the index is obtained. However, t_d may also specify a different action, namely that the effective address Y is to be used directly as operand and not as address of an operand (DU,DL), or that nothing takes place at all (N). Nevertheless, t_d is called "register designator" in these cases.

Register Designator		Action
Symbolic	Binary	
N	0000	$y = y$
X0	1000	
X1	1001	
.	.	$y = y + C(Xn)$
.	.	
X7	1111	
AU	0001	$y = y + C(A)_{0...17}$
AL	0101	$y = y + C(A)_{18..35}$
QU	0010	$y = y + C(Q)_{0...17}$
QL	0110	$y = y + C(Q)_{18..35}$
IC	0100	$y = y + C(IC)$
DU	0011	$y,00...0$ is the operand
DL	0111	$00...0,y$ is the operand

Tally Designator

The modification type IT causes the word at Y to be fetched and used as an indirect word as specified by the t_d of the word (instruction or previous indirect word) which contains Y.

The format of the indirect word is:

y	Tally	Tag
0 17,18	29,30	35
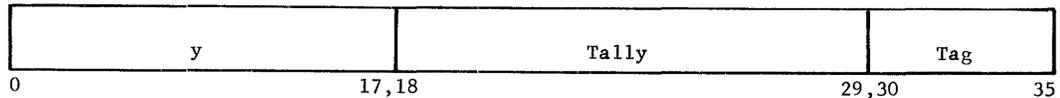
Where

       y = address field

     Tally = tally field

      Tag = tag field

CPB-1004F

35

Depending upon the prior tally designator, the tag field is used in one of three ways:

Tally Designator (Table Follows)   Tag Field

I, DI, ID, and F

Ignored					

DIC and IDC

t_m		t_d			

CI, SC

t_b	0	0	C_f		

AD, SD

Delta					
30	31	32	33	34	35
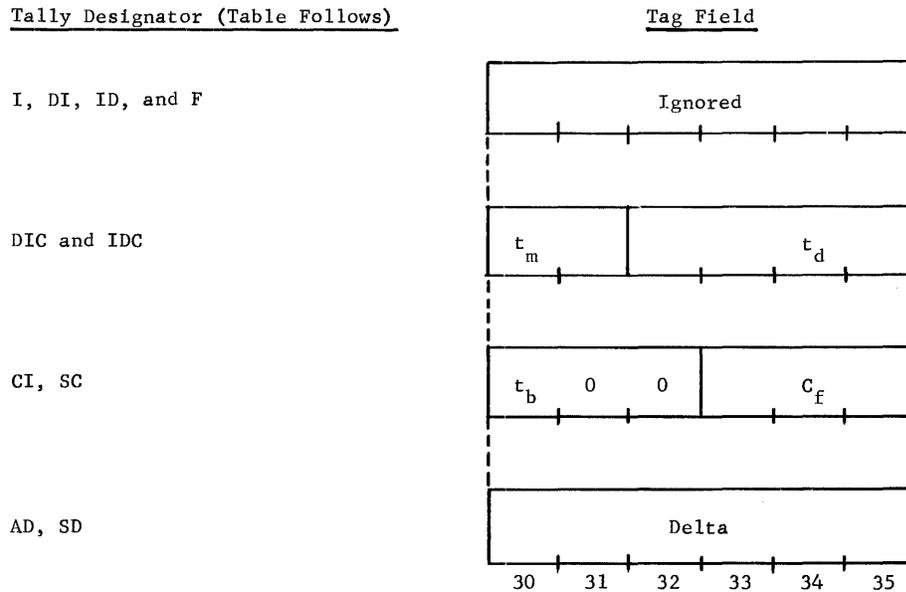
Where

t_m = modifier

t_d = designator

t_b = character size (0 = 6-bit, 1 = 9-bit)

C_f = character field

Delta = delta field

The following table gives the possible tally designators under IT type modification.

Tally Designator		Name
Symbolic	Binary	
I	1001	Indirect only
DI	1100	Decrement Address, Increment Tally
AD	1011	Add Delta (to address field)
SD	0100	Subtract Delta (from address field)
ID	1110	Increment Address, Decrement Tally
DIC	1101	Decrement Address, Increment Tally, and Continue
IDC	1111	Increment Address, Decrement Tally, and Continue
CI	1000	Character from Indirect
SC	1010	Sequence Character
F	0000	Fault

CPB-1004F

36

Address Modification Flowcharts

All possible types and sequences of address modification are shown on the following two flowcharts.

Modification Type	Flowchart
R, IR, and RI address modification	Figure 5A
IT address modification	Figure 5B

See explanation of symbols and descriptions of modifications immediately following these figures.
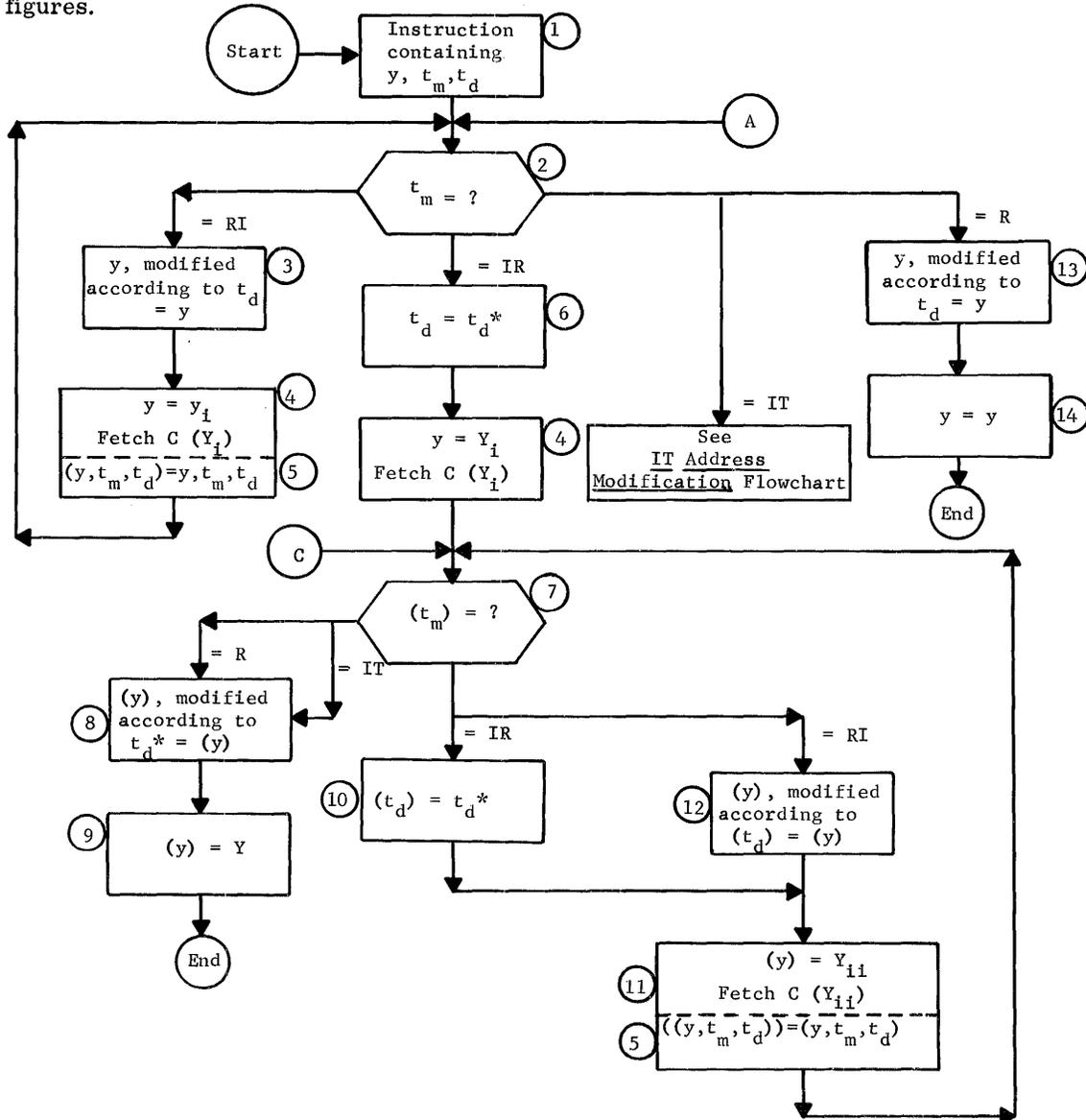
Figure 5A. Address Modification Flowchart

CPB-1004F
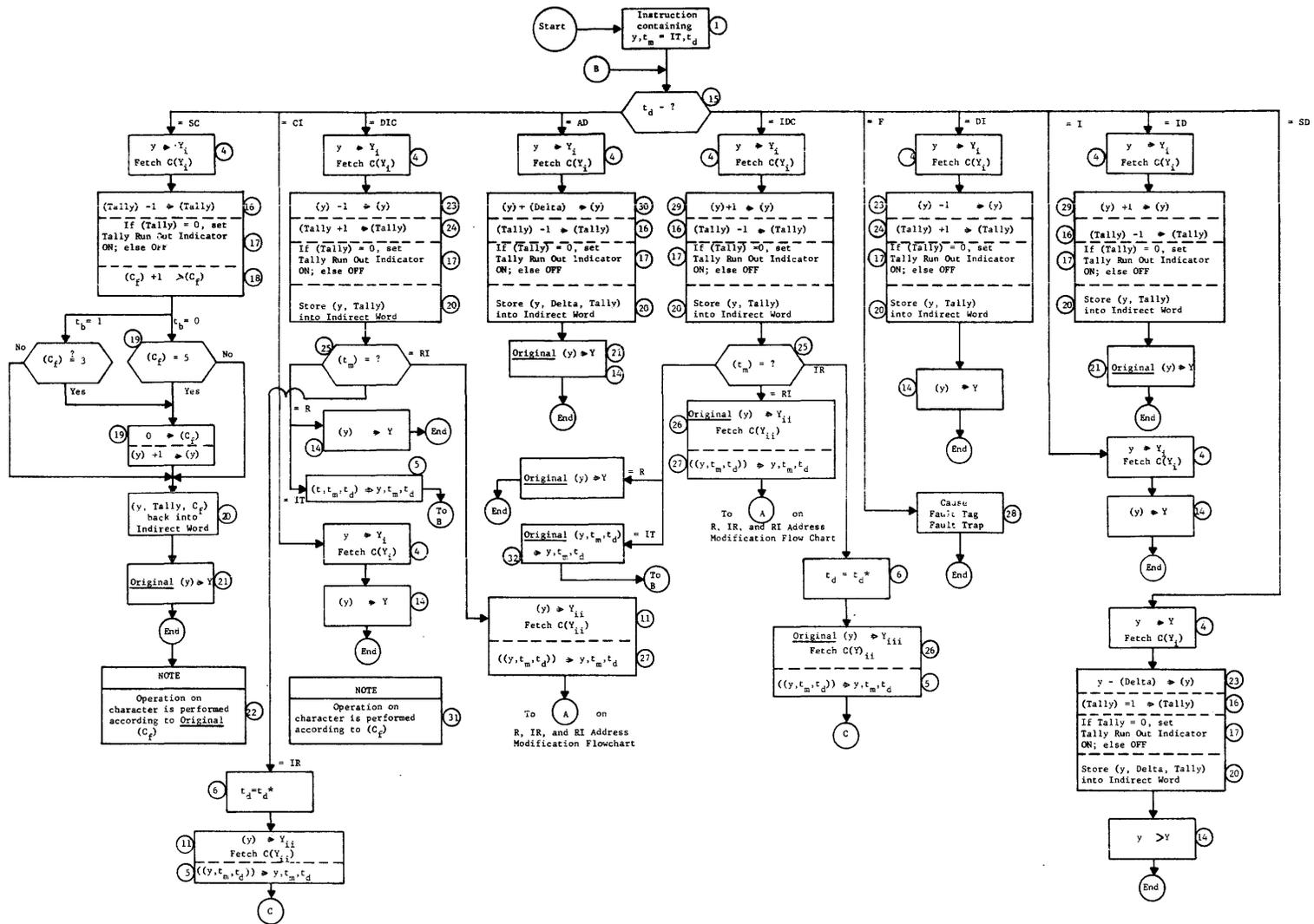
Figure 5B. Address Modification Flowchart

● <u>Explanation of Symbols Used on Flowcharts</u>

y, t_m, t_d	is the original address, tag modifier, and tag designator, respectively.
C_f, Tally, Delta	is the value of the character field, tally field, and delta field of an indirect word.
$\Rightarrow$	should be read "replaces."
C(---)	should be read "the contents of---."
Y	is the final effective address to be used in carrying out an instruction operation.
Y_i	is the address of an indirect word which will be used for further modification.
Y_{ii}	is the address, obtained from another indirect word, of an indirect word which will be used for further modification.
(---)	represents quantities obtained from the contents of an indirect word.
((---))	represents quantities obtained from the contents of an indirect word which was obtained through another indirect word.
t_d^*	is the register designator to be used as a final register modifier under IR modification.
Original	Most indirect words which are used under IT modification utilize the read-alter-rewrite (RAR) memory cycle. This RAR cycle must be completed before another indirect cycle can occur. The word original refers to the quantity contained in an indirect word <u>before</u> that quantity is incremented (during the alter part of the RAR cycle). Omission of the word original refers to the quantity <u>after</u> it is incremented or decremented during the alter portion of the RAR cycle.
End	indicates that the modification procedure for that instruction has terminated and the effective address Y, developed up to that point, is used to carry out the instruction operation.

● <u>Detailed Description of Flowcharts</u>

(1) The instruction word address field serves as the initial value of the tentative address y, and its tag field supplies the initial modifier t_m as well as initial designator t_d.

(2) t_m is one of the four modification types: R, RI, IR, or IT.

(3) y modified by t_d replaces the former tentative address y. If t_d = DU or DL, DU or DL is ignored and the modification proceeds as if t_d = N.

(4) The tentative address y, developed up to that point, becomes the address Y_i to be used in accessing an indirect word which will be used for further modification. Using Y_i, the indirect word is fetched.

CPB-1004F

⑤ The address and tag fields of the last indirect word replace the tentative address and the tag of the instruction.

⑥ The last designator t_d, becomes the final designator t_d^*, to be used as a final register modifier under IR modification.

⑦ The t_m, of the indirect word, designates one of the four modification types: R, RI, IR, or IT.

⑧ The address of the indirect word (y), modified by the final register modifier t_d^*, replaces the former tentative address.

⑨ The tentative indirect address (y), developed up to that point, is used as the effective address Y for carrying out the instruction operation.

⑩ The designator of the indirect word (t_d) replaces the final register designator t_d^*.

⑪ The tentative indirect address (y), developed up to that point becomes the address Y_{ii}, to be used in accessing another indirect word which will be used for further modification. Using Y_{ii}, the indirect word is fetched.

⑫ The address (y), contained in the indirect word and modified by the designator of the indirect word (t_d), replaces the tentative indirect address (y).

⑬ The y modified by t_d replaces the former tentative address y.

⑭ The tentative address y, developed up to that point, is used as the effective address Y for carrying out the instruction's operation.

⑮ The t_d is one of the 10 tally designators: SC, CI, DIC, AD, IDC, F, DI, I, ID, or SD.

⑯ A value one less than or one greater than the value of the tally field loaded from the indirect word becomes the new value of the tally field, depending on the use of the AD or SD designator.

⑰ The Tally Runout Indicator is set to ON if the tally field equals zero after incrementation or decrementation; the Indicator is set of OFF if the tally field does not equal zero after incrementation or decrementation.

⑱ A value one greater than the value of the character field loaded from the indirect word becomes the new value of the character field.

⑲ If the value of the character field C_f equals six, the character field is set to zero; and a value one greater than the value of the address field loaded from the indirect word becomes the new value of the address field.

⑳ During the rewrite portion of the read-alter-rewrite cycle used for updating an indirect word, the updated fields--(y), (C_f), (Tally), (Delta), (t_m), (t_d), where applicable--are returned to storage in memory.

㉑ The original value of the address field (y), as loaded from the indirect word before any incrementation or decrementation, becomes the effective address Y which is used to carry out the instruction operation.

(22) The original value of the character field C_f, as loaded from the indirect word <u>before</u> any incrementation (or setting to zero), is the value used in carrying out the instruction operation. (See note at end of this listing.)

(23) A value one less than the value of the address field loaded from the indirect word becomes the new value of the address field.

(24) A value one greater than the value of the tally field loaded from the indirect word becomes the new value of the tally field.

(25) Under IDC or DIC types of modification, the modifiers permitted within the indirect are:

$$t_m = R \qquad\qquad t_d = N$$

$$t_m = IR \qquad\qquad t_d = any$$

$$t_m = RI \qquad\qquad t_d = N$$

$$t_m = IT \qquad\qquad t_d = any$$

$t_m = R$ effectively terminates the modification procedure while

$t_m = RI$, IR, or IT seeks at least an additional level of modification.

(26) The original value of the address field (y), as loaded from the indirect word <u>before</u> incrementation, becomes the address Y_{ii} to be used in accessing the next indirect word which will be used for further modification.

(27) The address and tag fields of Y_{ii} replace the address and tag fields of the original instruction, and modification proceeds.

(28) Occurs when $t_m = IT$ and $t_d = F$, or when Fault tag fault is initiated and no further indirect addressing occurs.

(29) A value one greater than the value of the address field loaded from the indirect word becomes the new value of the address field.

(30) A value equal to the value of the address field (loaded from the indirect word) plus or minus Delta (a constant also loaded from the indirect word) replaces the value of the address field.   The constant is positive for the AD designator and negative for the SD designator.

(31) The value of the character field C_f, <u>after</u> incrementation (or setting to zero), is used in carrying out the instruction operation.   (See the note at the end of this listing.)

(32) The original value of the address field and the tag field of the last indirect word replace the tentative address and tag of the instruction.

NOTE: When the tally designator is CI or SC, the character field of the last indirect word is an octal number which specifies the character position of the memory location Y to be used in carrying out the instruction operation (the example uses a value of 3 in the character field).

CALCULATION OF INSTRUCTION EXECUTION TIMES

The instruction execution times are based on fetching of instructions in pairs from storage, and in the case of overlap type instructions* on overlap between the operation execution of the overlap type instruction and the fetching and address modification of the next instruction.

Certain operations prevent the fetching of instructions in pairs and certain instructions prevent overlapping. Under these conditions the following time adjustments should be made.

		GE-615	GE-635
1.	If an instruction from an even storage location alters a register, and the next instruction (from the successive odd location) begins its address modification procedure with an R or RI type of modification which uses this same register, then	Add 0 microseconds	Add 1.8 microseconds
2.	If an instruction from an even storage location alters the next instruction, then	Add 2.2 microseconds	Add 2.7 microseconds
3.	If a transfer of control instruction is located at an odd storage location, then	Add 0.5 microsecond	Add 0.5 microsecond
4.	If a transfer of control transfers to an instruction located at an odd storage location, then	Add 1.3 microseconds	Add 0.8 microsecond
5.	If a store type** instruction located in an odd storage location is preceded by a nonstore type instruction, then		

	GE-615	GE-635
Single Precision Store	Add 0.6 microsecond	Add 1.1 microseconds
Read-Alter-Rewrite	Add 0.6 microsecond	Add 1.2 microseconds
Double Precision Store	Add 0.9 microsecond	Add 1.2 microseconds

*Overlap type instructions include multiplications, divisions, shifts, and floating-point operations except loads and stores.

**Store type instructions = store, floating store, add and subtract stored, AND, OR, and EXCLUSIVE OR to storage, etc.

	GE-615	GE-635

6. If an overlap type in-
struction is followed either
by a store type instruc-
tion from an odd storage
location or by a trans-
fer of control instruction,
then (depending on the
particular instruction
sequence)   Add 2-3 microseconds   Add 1-2 microseconds

7. The instruction execution
times of shift and floating-
point operations are listed
as "average" times based
on a number of 4 shift steps.
Note that a single shift
step may effect a shift by
one, four, or sixteen po-
sitions. Actual times for
these instructions may
vary by up to ± 0.8 micro-
second. Where unnormal-
ized operands are used
in normalizing floating-
point operations, worst
case conditions can add
as much as 1.5 micro-
seconds.

8. Address modifications do not
require any time adjustments
except of the following cases:

RI - type, for the indirect
cycle   Add 1.7 microseconds   Add 1.7 microseconds

IR - type, for the indirect
cycle   Add 1.7 microseconds   Add 1.7 microseconds

IT - type, for the indirect
cycle with restoring
of the indirect word
         IT Even   Add 3.3 microseconds   Add 3.2 microseconds
         IT Odd   Add 3.3 microseconds   Add 4.9 microseconds

IT - type, for the indirect
cycle with nonrestoring
of the indirect word
(CI and I)   Add 1.7 microseconds   Add 1.7 microseconds

Index designator DU or DL   Subt 1.4 microseconds   Subt 0.5 microsecond
except when used with
a first modification of
the R or RI type and the
preceding instruction
being an overlap type
instruction.

THE INSTRUCTION REPERTOIRE

The GE-625/635 instruction set described under this heading is arranged by functional class, as listed in Appendix A. Appendix A together with Appendix B, which lists the instructions in alphabetical order by mnemonic, afford convenient page references to the instructions in this section. Appendix C presents the instruction mnemonics grouped by operation code.

For the description of the machine instructions that follow it is assumed that the reader is familiar with the general structure of the Processor, the representation of information, the data formats, and the method of address modifications, as presented in the preceding paragraphs of this chapter.

FORMAT OF INSTRUCTION DESCRIPTION

Each instruction in the repertoire is described in the following pages of this chapter. The descriptions are presented in the standardized format shown below.

Mnemonic:	Name of the Instruction:	Op Code (Octal)

SUMMARY:

MODIFICATIONS:

INDICATORS:

NOTES:

Line 1: Mnemonic, Name of the Instruction, Op Code (octal)

This line has three headings that appear over boxes containing the following:

1. Mnemonic--The mnemonic code for the Operation field of the programming form.

2. Name of the Instruction--The name of the machine instruction from which the Mnemonic was derived.

3. Op Code (octal)--The octal operation code for the instruction.

Line 2: SUMMARY

The change in the status of the information processing system effected by the execution of the instructions operation is described in a short and generally symbolic form. If reference is made here to the status of an indicator, then it is the status of this indicator before the operation is executed.

Line 3: MODIFICATIONS

Those designators are listed explicitly that shall not be used with this instruction either because they are not permitted with this instruction or because their effect cannot be predicted from the general address modification procedure.

CPB-1004F

44

Line 4: INDICATORS

Only those indicators are listed whose status can be changed by the execution of this instruction. In most cases, a condition for setting ON as well as one for setting OFF is stated. If only one of the two is stated, then this indicator remains unchanged. Unless explicitly stated otherwise, the conditions refer to the contents of registers, etc., as existing after the execution of the instruction's operation.

Line 5: NOTES

This part of the description exists only in those cases where the SUMMARY is not sufficient for an understanding of the operation.

Abbreviations and Symbols

The following abbreviations and symbols will be used for the description of the machine operations.

Registers:

A =	Accumulator Register (36 bits)
Q =	Quotient Register (36 bits)
AQ =	Combined Accumulator-Quotient Register (72 bits)
Xn =	Index Register n (n = 0, 1,..., 7) (18 bits)
E =	Exponent Register (8 bits)
EA =	Combined Exponent-Accumulator Register (8 + 36 bits)
EAQ =	Combined Exponent-Accumulator-Quotient Register (8 + 72 bits)
BAR =	Base Address Register (18 bits)
IC =	Instruction Counter (18 bits)
IR =	Indicator Register (18 bits, 11 of which are used at this time)
TR =	Timer Register (24 bits)
Z =	Temporary Pseudo-result of a non-store comparative Operation.

Effective Address and Memory Locations

Y =	The effective address (18 bits) of the respective instruction.
Y-pair =	A symbol denoting that the effective address Y designates a pair of memory locations (72 bits) with successive addresses, the lower one being even. When the effective address is even, then it designates the pair (Y, Y+1), and when it is odd, then the pair (Y-1, Y). In any case the memory location with the lower (even) address contains the more significant part of a double-precision number or the first of a pair of instructions.

Register Positions and Contents:

("R" standing for any of the registers listed above as well as for a memory location or a pair of memory locations.)

R_i	=	the ith position of R
$R_{i...j}$	=	the positions i through j of R
$C(R)$	=	the contents of the full register R
$C(R)_i$	=	the contents of the ith position of R
$C(R)_{i...j}$	=	the contents of the positions i through j of R

When the description of an instruction states a change only for a part of a register or memory location, then it is always understood that the part of the register or memory location which is not mentioned remains unchanged.

Other Symbols:

⇒	=	replaces
: :	=	compare with
AND	=	the Boolean connective AND (symbol Λ)
OR	=	the Boolean connective OR (symbol V)
≢	=	the Boolean connective NON-EQUIVALENCE (or EXCLUSIVE OR)

Memory Accessing

It is a characteristic feature of the GE-625/635 computer that an address translation takes place with each memory access when the Processor operates in the Slave Mode.

During the execution of a program a base address is contained in the bit positions 0-7 of the Processor Base Address Register. With each memory access, this base address is added to bit positions 0-7 of the program address supplied by this program in order to generate the actual address used in accessing the memory. In this way, the address translation provides complete independence of the program address range that is used with a specific execution of this program.

Only when the Processor is in the Master Mode is the program address used directly as an actual address; in this case, program addresses generally refer to the Comprehensive Operating System which has allocated to it the actual address range beginning at zero.

The descriptions of the individual machine instructions in this chapter do not mention the address translation. It is understood here that an address translation has to be performed immediately prior to each memory access request (in the Slave Mode) regardless of whether:

1. The program address is an instruction address, and the memory is accessed for fetching an instruction.

2. The program address is a tentative address, and the memory is accessed for fetching an indirect word

3. The program address is an effective address, and the memory is accessed for obtaining an operand or for storing a result.

CPB-1004F

No address translations take place for effective addresses which are used either as operands directly or in other ways (for example, shifts).

Floating-Point Arithmetic

Numbers in floating-point representation are stored in memory as follows:

	Integer Exponent	Fractional Mantissa
Single-word precision	$C(Y)_{0-7}$	$C(Y)_{8-35}$
Double-word precision	$C(Y\text{-pair})_{0-7}$	$C(Y\text{-pair})_{8-71}$

When a floating-point number is held in the register EAQ, its mantissa length is allowed to increase to the full length of the register AQ.
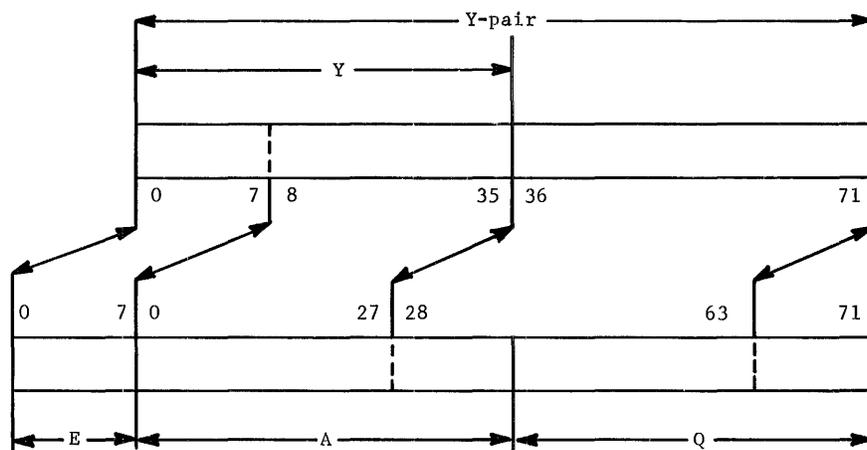
In storing a floating-point number, a truncation of the mantissa takes place. With single-word precision store instructions, only $C(AQ)_{0-27}$ will be stored as mantissa, and with double-word precision store instructions, only $C(AQ)_{0-63}$.

DESCRIPTIONS OF THE MACHINE INSTRUCTIONS

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LDA	Load A	235

SUMMARY: $C(Y) \Rightarrow C(A)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(A)$ = 0, then ON; otherwise OFF
Negative	If $C(A)_0$ = 1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LDQ	Load Q	236

SUMMARY: $C(Y) \Rightarrow C(Q)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Q)$ = 0, then ON; otherwise OFF
Negative	If $C(Q)_0$ = 1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LDAQ	Load AQ	237

SUMMARY: $C(Y\text{-pair}) \Rightarrow C(AQ)$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(AQ)$ = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0$ = 1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:		Op Code (Octal)
LDXn	Load Xn from Upper	(n=0, 1, . . . ,7)	22n

SUMMARY: $C(Y)_{0...17} \Rightarrow C(Xn)$

MODIFICATIONS: All except CI, SC, DL

INDICATORS: (Indicators not listed are not affected)

Zero	If C(Xn) = 0, then ON; otherwise OFF
Negative	If $C(Xn)_0$ = 1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:		Op Code (Octal)
LXLn	Load Xn from Lower	(n=0, 1, . . . ,7)	72n

SUMMARY: $C(Y)_{18...35} \Rightarrow C(Xn)$

MODIFICATIONS: All except DU, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(Xn) = 0, then ON; otherwise OFF
Negative	If $C(Xn)_0$ = 1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LREG	Load Registers	073

SUMMARY: $C(Y,Y+1,....Y+6) \Rightarrow C(X0,X1,...X7,A,Q,E)$
where Y_{15-17} = 000 for the first location only and increases by one for
each succeeding location.

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

NOTE:
$C(Y)_{0-17} \Rightarrow C(X0)$ $C(Y+3)_{0-17} \Rightarrow C(X6)$

$C(Y)_{18-35} \Rightarrow C(X1)$ $C(Y+3)_{18-35} \Rightarrow C(X7)$

$C(Y+1)_{0-17} \Rightarrow C(X2)$ $C(Y+4)_{0-35} \Rightarrow C(A)$

$C(Y+1)_{18-35} \Rightarrow C(X3)$ $C(Y+5)_{0-35} \Rightarrow C(Q)$

$C(Y+2)_{0-17} \Rightarrow C(X4)$ $C(Y+6)_{0-7} \Rightarrow C(E)$

$C(Y+2)_{18-35} \Rightarrow C(X5)$

CPB-1004F

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LCA	Load Complement A	335

SUMMARY: $- C(Y) \Rightarrow C(A)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(A) = 0$, then ON; otherwise OFF
Negative	If $C(A)_0 = 1$, then ON; otherwise OFF
Overflow	If range of A is exceeded, then ON; otherwise OFF

NOTE: This instruction changes the number to its negative
(if $\neq 0$) while moving it from the memory to A.
The operation is executed by forming the two's com-
plement of the string of 36 bits.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LCQ	Load Complement Q	336

SUMMARY: $- C(Y) \Rightarrow C(Q)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Q) = 0$, then ON; otherwise OFF
Negative	If $C(Q)_0 = 1$, then ON; otherwise OFF
Overflow	If range of Q is exceeded, then ON

NOTE: This instruction changes the number to its negative (if $\neq 0$) while moving it from Y to Q. The operation is executed by forming the two's complement of the string of 36 bits.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LCAQ	Load Complement AQ	337

SUMMARY: $- C(Y\text{-pair}) \Rightarrow C(AQ)$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(AQ) = 0$, then ON; otherwise OFF
Negative	If $C(AQ)_0 = 1$, then ON; otherwise OFF
Overflow	If range of AQ is exceeded, then ON

NOTE: This instruction changes the number to its negative (if $\neq 0$) while moving it from Y-pair to AQ. The operation is executed by forming the two's complement of the string of 72 bits.

CPB-1004F

51

Mnemonic:	Name of the Instruction:		Op Code (Octal)
LCXn	Load Complement Xn	(n = 0,1,...,7)	32n

SUMMARY: $- C(Y)_{0...17} \Rightarrow C(Xn)$

MODIFICATIONS: All except CI, SC, DL

INDICATORS: (Indicators not listed are not affected)

Zero	If C(Xn) = 0, then ON; otherwise OFF
Negative	If $C(Xn)_0 = 1$, then ON; otherwise OFF
Overflow	If range of Xn is exceeded, then ON

NOTE: This instruction changes the number to its negative (if $\neq 0$) while moving it from $Y_{0...17}$ to Xn. The operation is executed by forming the two's complement of the string of 18 bits.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
EAA	Effective Address to A	635

SUMMARY: $Y \Rightarrow C(A)_{0...17}; \ 00...0 \Rightarrow C(A)_{18...35}$

MODIFICATIONS: All except DU, DL

INDICATORS: (Indicators not listed are not affected)

Zero	If C(A) = 0, then ON; otherwise OFF
Negative	If $C(A)_0 = 1$, then ON; otherwise OFF

NOTE: This instruction, and the instructions EAQ and EAXn, facilitate interregister data movements; the data source is specified by the address modification, and the data destination by the operation code of the instruction.

CPB-1004F

52

Mnemonic:	Name of the Instruction:	Op Code (Octal)
EAQ	Effective Address to Q	636

SUMMARY: $Y \Rightarrow C(Q)_{0...17}$; $00...0 \Rightarrow C(Q)_{18...35}$

MODIFICATIONS: All except DU, DL

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Q) = 0$, then ON; otherwise OFF
Negative	If $C(Q)_0 = 1$, then ON; otherwise OFF

NOTE: This instruction, and the instructions EAA and EAXn, facilitate interregister data movements; the data source is specified by the address modification, and the data destination by the operation code of the instruction.

Mnemonic:	Name of the Instruction:		Op Code (Octal)
EAXn	Effective Address to Xn	(n=0,1,...,7)	62n

SUMMARY: $Y \Rightarrow C(Xn)$

MODIFICATIONS: All except DU, DL

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Xn) = 0$, then ON; otherwise OFF
Negative	If $C(Xn)_0 = 1$, then ON; otherwise OFF

NOTE: This instruction, and the instructions EAA and EAQ facilitate interregister data movements; the data source is specified by the address modification, and the data destination by the operation code of the instruction.

CPB-1004F

53

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LDI	Load Indicator Register	634

SUMMARY: $C(Y)_{18...35} \Rightarrow C(IR)$

MODIFICATIONS: All except CI, SC

INDICATORS: (Indicators not listed are not affected)

Master Mode	Not Affected!
All other indicators	If corresponding bit in C(Y) is ONE, then ON; otherwise OFF

NOTE: 1. The relation between bit positions of C(Y) and the indicators is as follows:

Bit Position	Indicators
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent Overflow
23	Exponent Underflow
24	Overflow Mask
25	Tally Runout
26	Parity Error
27	Parity Mask
28	Master Mode
29	
30	
31	
32	00...0
33	
34	
35	

2. The Tally Runout Indicator will reflect $C(Y)_{25}$ regardless of what address modification is performed on the LDI instruction (for Tally Operations).

CPB-1004F

54

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STA	Store A	755

SUMMARY: $C(A) \Rightarrow C(Y)$

MODIFICATIONS: All except DU, DL

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STQ	Store Q	756

SUMMARY: $C(Q) \Rightarrow C(Y)$

MODIFICATIONS: All except DU, DL

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STAQ	Store AQ	757

SUMMARY: $C(AQ) \Rightarrow C(Y\text{-pair})$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

Mnemonic:	Name of the Instruction:		Op Code (Octal)
STXn	Store Xn into Upper	$(n = 0,1\ldots,7)$	74n

SUMMARY: $C(Xn) \Rightarrow C(Y)_{0\ldots17}$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

Mnemonic:	Name of the Instruction:		Op Code (Octal)
SXLn	Store Xn into Lower	(n = 0, 1,...,7)	44n

SUMMARY: $C(Xn) \Rightarrow C(Y)_{18...35}$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SREG	Store Registers	753

SUMMARY: $C(X0,X1,X2,....X7,A,Q,E,TR) \Rightarrow C(Y,Y+1,....Y+7)$
where $Y_{15-17} = 000$ for the first location only and increases by one for each succeeding location

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

NOTE:

$C(X0) \Rightarrow C(Y)_{0-17}$		$C(X6) \Rightarrow C(Y+3)_{0-17}$	
$C(X1) \Rightarrow C(Y)_{18-35}$		$C(X7) \Rightarrow C(Y+3)_{18-35}$	
$C(X2) \Rightarrow C(Y+1)_{0-17}$		$C(A) \Rightarrow C(Y+4)_{0-35}$	
$C(X3) \Rightarrow C(Y+1)_{18-35}$		$C(Q) \Rightarrow C(Y+5)_{0-35}$	
$C(X4) \Rightarrow C(Y+2)_{0-17}$		$C(E) \Rightarrow C(Y+6)_{0-7}; 00..0 \Rightarrow C(Y+6)_{8-35}$	
$C(X5) \Rightarrow C(Y+2)_{18-35}$		$C(TR) \Rightarrow C(Y+7)_{0-23}; 00...0 \Rightarrow C(Y+7)_{24-35}$	

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STCA	Store Character of A (Six Bit)	751

SUMMARY: Characters of C(A) ⇒ corresponding characters of C(Y), the character positions affected being specified in the Tag field.

MODIFICATIONS: No modification can take place

INDICATORS: None affected

NOTE: Binary ones in the tag field of this instruction specify the character positions of A and Y that are affected by this instruction. The control relation is shown in the diagram below.
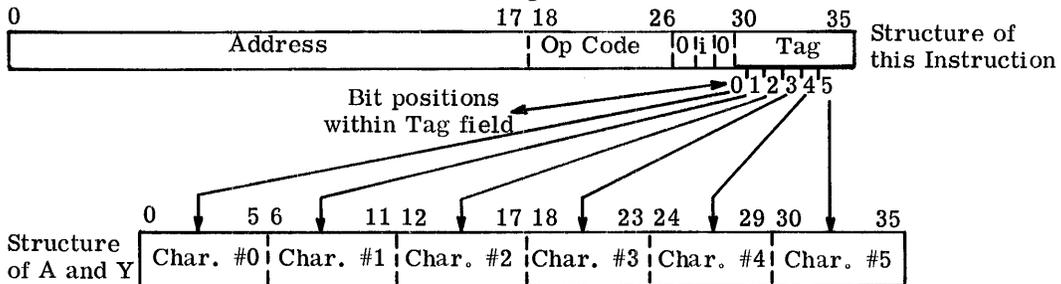
EXAMPLE:

1	8	16	32
	STCA	LOC, 07	

The instruction in this example moves the 6-bit characters #3, #4, and #5 from C(A) to the corresponding character positions of memory location LOC. Character positions #0, #1, and #2 of LOC are unaffected.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STCQ	Store Character of Q (Six Bit)	752

SUMMARY: Characters of C(Q) ⇒ corresponding characters of C(Y), the character positions affected being specified by the Tag field.

MODIFICATIONS: No modification can take place

INDICATORS: None affected

NOTE: Binary ones in the Tag field of this instruction specify the character positions of Q and Y that are affected by this instruction. See the example for STCA. The control relation is shown in the diagram below.
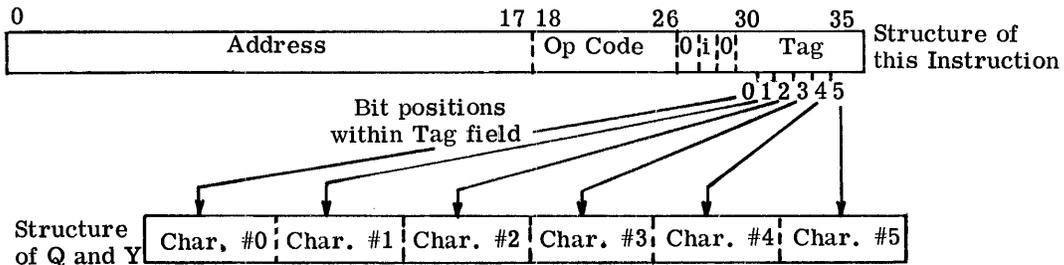
Mnemonic:	Name of the Instruction:	Op Code (Octal)
STBA	Store Character of A (Nine Bit)	551

SUMMARY: Characters of C(A) ⇒ corresponding characters of C(Y),
the character positions affected being specified in the Tag field.

MODIFICATIONS: No modification can take place

INDICATORS: None affected

NOTE: Binary ones in the Tag field of this instruction specify the character positions of A and Y that are affected by this instruction. The control relation is shown in the diagram below:
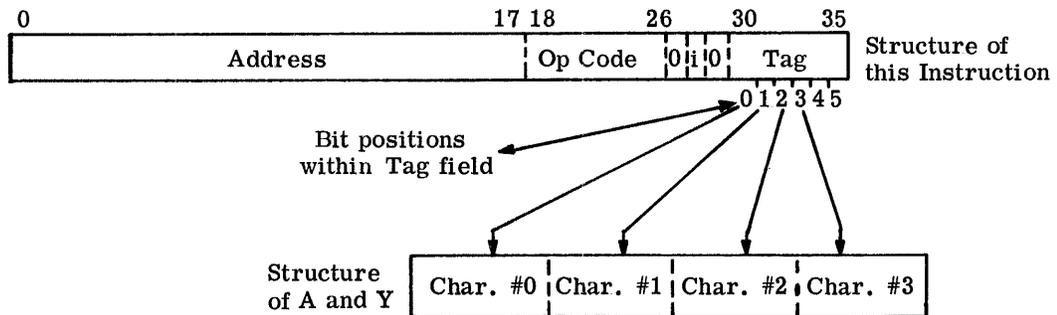
EXAMPLE:

```
 1        8       16                32
 |        |STBA   |LOC,04           |
```

The instruction in this example moves the low order 9-bit character #3 from C(A) to the corresponding character position of memory location LOC. Character positions #0, #1, and #2 of LOC are unaffected.

Bit positions 4 and 5 of the Tag field are ignored.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STBQ	Store Character of Q (Nine Bit)	552

SUMMARY: Characters of C(Q) ⇒ corresponding characters of C(Y), the character positions affected being specified in the Tag field.

MODIFICATIONS: No modification can take place

INDICATORS: None affected

NOTE: Binary ones in the Tag field of this instruction specify the character positions of A and Y that are affected by this instruction. See the example for STBA. The control relation is shown in the diagram below:
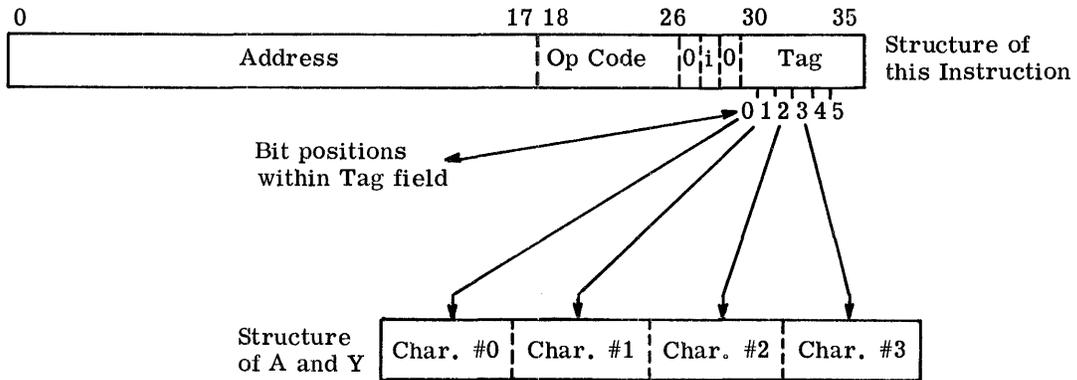
Bit positions 4 and 5 of the Tag field are ignored.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STI	Store Indicator Register	754

SUMMARY: $C(IR) \Rightarrow C(Y)_{18...35}$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

NOTE: 1. The relation between bit positions of C(Y) and the indicators
is as follows:

Bit Position	Indicators
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent Overflow
23	Exponent Underflow
24	Overflow Mask
25	Tally Runout
26	Parity Error
27	Parity Mask
28	Master Mode
29	
30	
31	
32	00...0
33	
34	
35	

2. The ON state corresponds to a ONE bit, the OFF state to a ZERO
bit.

3. The $C(Y)_{25}$ will contain the state of the Tally Runout Indicator
prior to address modification of the STI instruction (for Tally
operations).

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STT	Store Timer Register	454

SUMMARY: $C(TR) \Rightarrow C(Y)_{0\ldots23}$
$00\ldots0 \Rightarrow C(Y)_{24\ldots35}$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SBAR	Store Base Address Register	550

SUMMARY: $C(BAR) \Rightarrow C(Y)_{0\ldots17}$ $C(Y)_{18-35}$ Unchanged

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STZ	Store Zero	450

SUMMARY: $00\ldots0 \Rightarrow C(Y)$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STC1	Store Instruction Counter plus 1	554

SUMMARY: $C(IC) + 0...01 \rightarrow C(Y)_{0...17}$ (Note the difference between STC1 and STC2)

$C(IR) \rightarrow C(Y)_{18...35}$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

NOTES: 1. The relation between bit positions of C(Y) and the indicators is as follows:

Bit Position	Indicators
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent Overflow
23	Exponent Underflow
24	Overflow Mask
25	Tally Runout
26	Parity Error
27	Parity Mask
28	Master Mode
29	
30	
31	
32	00...0
33	
34	
35	

2. The ON state corresponds to a ONE bit, the OFF state to a ZERO bit.

3. The $C(Y)_{25}$ will contain the state of the Tally Runout Indicator prior to address modification of the STC1 instruction (for Tally operations).

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STC2	Store Instruction Counter plus 2	750

SUMMARY: $C(IC) + 0...010_2 \rightarrow C(Y)_{0...17}$ (Note the difference between STC1 and STC2)

$C(Y)_{18-35}$ remain unchanged

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ARS	A Right Shift	731

SUMMARY: Shift right C(A) by $Y_{11\ldots17}$ positions; fill vacated positions with $C(A)_0$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(A) = 0, then ON; otherwise OFF
Negative	If $C(A)_0$ = 1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
QRS	Q Right Shift	732

SUMMARY: Shift right C(Q) by $Y_{11\ldots17}$ positions; fill vacated positions with $C(Q)_0$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(Q) = 0, then ON; otherwise OFF
Negative	If $C(Q)_0$ = 1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LRS	Long Right Shift	733

SUMMARY: Shift right C(AQ) by $Y_{11\ldots17}$ positions; fill vacated positions with $C(AQ)_0$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0$ = 1, then ON; otherwise OFF

CPB-1004F

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ALS	A Left Shift	735

SUMMARY: Shift left C(A) by $Y_{11...17}$ positions; fill vacated positions with zeros

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(A) = 0, then ON; otherwise OFF
Negative	If $C(A)_0$ =1, then ON; otherwise OFF
Carry	If $C(A)_0$ ever changes during the shift, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
QLS	Q Left Shift	736

SUMMARY: Shift left C(Q) by $Y_{11...17}$ positions; fill vacated positions with zeros

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(Q) = 0, then ON; otherwise OFF
Negative	If $C(Q)_0$ = 1, then ON; otherwise OFF
Carry	If $C(Q)_0$ ever changes during the shift, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LLS	Long Left Shift	737

SUMMARY: Shift left C(AQ) by $Y_{11...17}$ positions; fill vacated positions with zeros

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0$ = 1, then ON; otherwise OFF
Carry	If $C(AQ)_0$ ever changes during the shift, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ARL	A Right Logic	771

SUMMARY: Shift right C(A) by $Y_{11...17}$ positions; fill vacated positions with zeros

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(A) = 0, then ON; otherwise OFF
Negative	If $C(A)_0$ =1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
QRL	Q Right Logic	772

SUMMARY: Shift right C(Q) by $Y_{11...17}$ positions; fill vacated positions with zeros

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(Q) = 0, then ON; otherwise OFF
Negative	If $C(Q)_0$ =1, then ON; otherwise OFF

CPB-1004F

65

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LRL	Long Right Logic	773

SUMMARY: Shift right C(AQ) by $Y_{11\ldots17}$ positions; fill vacated positions with zeros

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0$ =1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ALR	A Left Rotate	775

SUMMARY: Rotate C(A) by $Y_{11\ldots17}$ positions; enter each bit leaving position 0 into position 35

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(A) = 0, then ON; otherwise OFF
Negative	If $C(A)_0$ =1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
QLR	Q Left Rotate	776

SUMMARY: Rotate C(Q) by $Y_{11\ldots17}$ positions; enter each bit leaving position 0 into position 35

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(Q) = 0, then ON; otherwise OFF
Negative	If $C(Q)_0$ = 1, then ON; otherwise OFF

CPB-1004F

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LLR	Long Left Rotate	777

SUMMARY: Rotate C(AQ) by $Y_{11...17}$ positions; enter each bit leaving position 0 into position 71

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0$ =1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ADA	Add to A	075

SUMMARY: $C(A) + C(Y) \Rightarrow C(A)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(A) = 0$, then ON; otherwise OFF
Negative	If $C(A)_0 = 1$, then ON; otherwise OFF
Overflow	If range of A is exceeded, then ON
Carry	If a carry out of A_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ADQ	Add to Q	076

SUMMARY: $C(Q) + C(Y) \Rightarrow C(Q)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Q) = 0$, then ON; otherwise OFF
Negative	If $C(Q)_0 = 1$, then ON; otherwise OFF
Overflow	If range of Q is exceeded, then ON
Carry	If a carry out of Q_0 is generated, then ON; otherwise OFF

CPB-1004 F

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ADAQ	Add to AQ	077

SUMMARY: $C(AQ) + C(Y\text{-pair}) \Rightarrow C(AQ)$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(AQ) = 0$, then ON; otherwise OFF
Negative	If $C(AQ)_0 = 1$, then ON; otherwise OFF
Overflow	If range of AQ exceeded, then ON
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:		Op Code (Octal)
ADXn	Add to Xn	$(n = 0, 1, \ldots, 7)$	06n

SUMMARY: $C(Xn) + C(Y)_{0\ldots17} \Rightarrow C(Xn)$

MODIFICATIONS: All except CI, SC, DL

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Xn) = 0$, then ON; otherwise OFF
Negative	If $C(Xn)_0 = 1$, then ON; otherwise OFF
Overflow	If range of Xn is exceeded; then ON
Carry	If a carry out of Xn_0 is generated, then ON; otherwise OFF

CPB–1004F

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ASA	Add to Storage from A	055

SUMMARY: $C(A) + C(Y) \Rightarrow C(Y)$ $C(A)$ unchanged

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Y) = 0$, then ON; otherwise OFF
Negative	If $C(Y)_0 = 1$, then ON; otherwise OFF
Overflow	If range of Y is exceeded, then ON
Carry	If a carry out of Y_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ASQ	Add to Storage from Q	056

SUMMARY: $C(Q) + C(Y) \Rightarrow C(Y)$ $C(Q)$ unchanged

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Y) = 0$, then ON; otherwise OFF
Negative	If $C(Y)_0 = 1$, then ON; otherwise OFF
Overflow	If range of Y is exceeded, then ON
Carry	If a carry out of Y_0 is generated, then ON; otherwise OFF

CPB-1004F

70

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ASXn	Add to Storage from Xn	04n

SUMMARY: $C(Xn) + C(Y)_{0...17} \Rightarrow C(Y)_{0...17}$ $C(Xn)$ unchanged

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Y)_{0...17} = 0$, then ON; otherwise OFF
Negative	If $C(Y)_0 = 1$, then ON; otherwise OFF
Overflow	If range of $Y_{0...17}$ exceeded, then ON
Carry	If a carry out of Y_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ADLA	Add Logic to A	035

SUMMARY: $C(A) + C(Y) \Rightarrow C(A)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(A) = 0$, then ON; otherwise OFF
Negative	If $C(A)_0 = 1$, then ON; otherwise OFF
Overflow	Not Affected!
Carry	If a carry out of A_0 is generated then ON, otherwise OFF

NOTE: This instruction is identical to the ADA instruction with the exception that the Overflow Indicator is not affected by this instruction. Operands and results are regarded as unsigned, positive binary integers. (See page 28, 72, and 73.)

CPB-1004F

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ADLQ	Add Logic to Q	036

SUMMARY: $C(Q) + C(Y) \Rightarrow C(Q)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Q) = 0$, then ON; otherwise OFF
Negative	If $C(Q)_0 = 1$, then ON; otherwise OFF
Overflow	Not Affected!
Carry	If a carry out of Q_0 is generated then ON; otherwise OFF

NOTE: This instruction is identical to the ADQ instruction with the exception that the Overflow Indicator is not affected by this instruction. Operands and results are regarded as unsigned, positive binary integers. (See page 28.)

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ADLAQ	Add Logic to AQ	037

SUMMARY: $C(AQ) + C(Y\text{-pair}) \Rightarrow C(AQ)$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(AQ) = 0$, then ON; otherwise OFF
Negative	If $C(AQ)_0 = 1$, then ON; otherwise OFF
Overflow	Not Affected!
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

NOTE: This instruction is identical to the ADAQ instruction with the exception that the Overflow Indicator is not affected by this instruction. Operands and results are regarded as unsigned, positive binary integers. (See page 28.)

CPB-1004F

Mnemonic:	Name of the Instruction:		Op Code (Octal)
ADLXn	Add Logic to Xn	$(n = 0, 1,\ldots,7)$	02n

SUMMARY: $C(Xn) + C(Y)_{0\ldots17} \Rightarrow C(Xn)$

MODIFICATIONS: All except CI, SC, DL

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Xn) = 0$, then ON; otherwise OFF
Negative	If $C(Xn)_0 = 1$, then ON; otherwise OFF
Overflow	Not Affected!
Carry	If a carry out of Xn_0 is generated, then ON; otherwise OFF

NOTE: This instruction is identical to the ADXn instruction with the exception that the Overflow Indicator is not affected by this instruction. Operands and results are regarded as unsigned, positive binary integers. (See page 28.)

Mnemonic:	Name of the Instruction:	Op Code (Octal)
AWCA	Add with Carry to A	071

SUMMARY: Carry Indicator OFF: $C(A) + C(Y) \Rightarrow C(A)$
Carry Indicator ON: $C(A) + C(Y) + 0\ldots01 \Rightarrow C(A)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(A) = 0$, then ON; otherwise OFF
Negative	If $C(A)_0 = 1$, then ON; otherwise OFF
Overflow	If range of A is exceeded, then ON
Carry	If a carry out of A_0 is generated, then ON; otherwise OFF

NOTE: This instruction is identical to the ADA instruction with the exception that, when the Carry Indicator is ON at the beginning of the instruction, then a + 1 is added to the least-significant position.

CPB-1004F

73

Mnemonic:	Name of the Instruction:	Op Code (Octal)
AWCQ	Add with Carry to Q	072

SUMMARY:　　Carry Indicator OFF:　$C(Q) + C(Y) \Rightarrow C(Q)$
　　　　　　　Carry Indicator ON:　　$C(Q) + C(Y) + 0\ldots01 \Rightarrow C(Q)$

MODIFICATIONS:　　All

INDICATORS:　　(Indicators not listed are not affected)

Zero	If $C(Q) = 0$, then ON; otherwise OFF
Negative	If $C(Q)_0 = 1$, then ON; otherwise OFF
Overflow	If range of Q is exceeded, then ON
Carry	If carry out of Q_0 is generated, then ON; otherwise OFF

NOTE:　　　　This instruction is identical to the ADQ instruction with the exception that, in case the Carry Indicator is ON at the beginning of the instruction, then a + 1 is added to the least-significant position.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ADL	Add Low to AQ	033

SUMMARY: $C(AQ) + C(Y)$, right adjusted, $\to C(AQ)$
(See the description below.)

MODIFICATIONS: All except CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(AQ) = 0$, then ON; otherwise OFF
Negative	If $C(AQ)_0 = 1$, then ON; otherwise OFF
Overflow	If range of AQ is exceeded, then ON
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

DESCRIPTION: A 72-bit number is formed:

$$\underbrace{C(Y_0), \ C(Y_0), \ \ldots\ldots, \ C(Y_0),}_{36 \text{ bits}} \ C(Y).$$

Its lower half (bits 36-71) is identical to $C(Y)$, and each of the bits of its upper half (bits 0-35) is identical to the sign bit of $C(Y)$, i.e., to $C(Y_0)$.

This number is added to the contents of the combined AQ-register, effecting the addition of $C(Y)$ to the lower half of the combined AQ-register, with a possible carry out of the Q-part being passed on to the A-part.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
AOS	Add One to Storage	054

SUMMARY: $C(Y) + 0_0\ldots01 \to C(Y)$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Y) = 0$, then ON; otherwise OFF
Negative	If $C(Y)_0 = 1$, then ON; otherwise OFF
Overflow	If range of Y is exceeded, then ON
Carry	If a carry out of Y_0 is generated, then ON; otherwise OFF

CPB-1004F

75

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SBA	Subtract from A	175

SUMMARY: $C(A) - C(Y) \Rightarrow C(A)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(A) = 0$, then ON; otherwise OFF
Negative	If $C(A)_0 = 1$, then ON; otherwise OFF
Overflow	If range of A is exceeded, then ON
Carry	If a carry out of A_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SBQ	Subtract from Q	176

SUMMARY: $C(Q) - C(Y) \Rightarrow C(Q)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(A) = 0$, then ON; otherwise OFF
Negative	If $C(Q)_0 = 1$, then ON; otherwise OFF
Overflow	If range of Q is exceeded, then ON
Carry	If a carry out of Q_0 is generated, then ON; otherwise OFF

CPB-1004F

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SBAQ	Subtract from AQ	177

SUMMARY: $C(AQ) - C(\text{Y-pair}) \Rightarrow C(AQ)$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(AQ) = 0$, then ON; otherwise OFF
Negative	If $C(AQ)_0 = 1$, then ON; otherwise OFF
Overflow	If range of AQ is exceeded, then ON; otherwise OFF
Carry	If carry out of AQ_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:		Op Code (Octal)
SBXn	Subtract from Xn	$(n = 0,1,\ldots,7)$	16n

SUMMARY: $C(Xn) - C(Y)_{0\ldots17} \Rightarrow C(Xn)$

MODIFICATIONS: All except CI, SC, DL

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Xn) = 0$, then ON; otherwise OFF
Negative	If $C(Xn)_0 = 1$, then ON; otherwise OFF
Overflow	If range of Xn is exceeded, then ON
Carry	If a carry out of Xn_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SSA	Subtract Stored from A	155

SUMMARY: $C(A) - C(Y) \Rightarrow C(Y)$ C(A) unchanged

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Y) = 0$, then ON; otherwise OFF
Negative	If $C(Y)_0 = 1$, then ON; otherwise OFF
Overflow	If range of Y is exceeded, then ON
Carry	If a carry out of Y_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SSQ	Subtract Stored from Q	156

SUMMARY: $C(Q) - C(Y) \Rightarrow C(Y)$ C(Q) unchanged

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Y) = 0$, then ON; otherwise OFF
Negative	If $C(Y)_0 = 1$, then ON; otherwise OFF
Overflow	If range of Y is exceeded, then ON
Carry	If a carry out of Y_0 is generated, then ON; otherwise OFF

CPB-1004F

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SSXn	Subtract Stored from Xn	14n

SUMMARY: $C(Xn) - C(Y)_{0...17} \Rightarrow C(Y)_{0...17}$ $C(Xn)$ unchanged

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Y)_{0...17} = 0$, then ON, otherwise OFF
Negative	If $C(Y)_0 = 1$, then ON, otherwise OFF
Overflow	If range of $Y_{0...17}$ exceeded, then ON
Carry	If a carry out of Y_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SBLA	Subtract Logic from A	135

SUMMARY: $C(A) - C(Y) \Rightarrow C(A)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(A) = 0$, then ON; otherwise OFF
Negative	If $C(A)_0 = 1$, then ON; otherwise OFF
Overflow	Not Affected!
Carry	If a carry out of A_0 is generated, then ON; otherwise OFF

NOTE: This instruction is identical to the SBA instruction with the exception that the Overflow Indicator is not affected by this instruction. Operands and results are regarded as unsigned, positive binary integers. (See page 28.)

CPB-1004F

79

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SBLQ	Subtract Logic from Q	136

SUMMARY: $C(Q) - C(Y) \Rightarrow C(Q)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Q) = 0$, then ON; otherwise OFF
Negative	If $C(Q)_0 = 1$, then ON; otherwise OFF
Overflow	Not Affected!
Carry	If a carry out of Q_0 is generated, then ON; otherwise OFF

NOTE: This instruction is identical to the SBQ instruction with the exception that the Overflow Indicator is not affected by this instruction. Operands and results are regarded as unsigned, positive binary integers. (See page 28.)

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SBLAQ	Subtract Logic from AQ	137

SUMMARY: $C(AQ) - C(Y\text{-pair}) \Rightarrow C(AQ)$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(AQ) = 0$, then ON; otherwise OFF
Negative	If $C(AQ)_0 = 1$, then ON; otherwise OFF
Overflow	Not Affected!
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

NOTE: This instruction is identical to the SBAQ instruction with the exception that the Overflow Indicator is not affected by this instruction. Operands and results are regarded as unsigned, positive binary integers. (See page 28.)

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SBLXn	Subtract Logic from Xn $\quad$ (n = 0, 1,...,7)	12n

SUMMARY: $\quad C(Xn) - C(Y)_{0...17} \Rightarrow C(Xn)$

MODIFICATIONS: $\quad$ All except CI, SC, DL

INDICATORS: $\quad$ (Indicators not listed are not affected)

Zero	If $C(Xn) = 0$, then ON; otherwise OFF
Negative	If $C(Xn)_0 = 1$, then ON; otherwise OFF
Overflow	Not Affected!
Carry	If a carry out of Xn_0 is generated, then ON; otherwise OFF

NOTE: $\quad$ This instruction is identical to the SBXn instruction with the exception that the Overflow Indicator is not affected by this instruction. Operands and results are regarded as unsigned, positive binary integers. (See page 28.)

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SWCA	Subtract with Carry from A	171

SUMMARY: $\quad$ Carry Indicator ON: $\quad C(A) - C(Y) \Rightarrow C(A)$
$\qquad\qquad\quad$ Carry Indicator OFF: $\quad C(A) - C(Y) - 0...01 \Rightarrow C(A)$

MODIFICATIONS: $\quad$ All

INDICATORS: $\quad$ (Indicators not listed are not affected)

Zero	If $C(A) = 0$, then ON; otherwise OFF
Negative	If $C(A)_0 = 1$, then ON; otherwise OFF
Overflow	If range of A is exceeded, then ON
Carry	If a carry out of A_0 is generated, then ON; otherwise OFF

NOTE: $\quad$ 1. This instruction is identical to the SBA instruction with the exception that, when the Carry Indicator is OFF at the beginning of the instruction, then a + 1 is subtracted from the least-significant position.

CPB-1004F

81

2. This instruction is used for multiple-word precision arithmetic. The SUMMARY can also be worded as follows in order to show the intended use:

Carry Indicator ON: $C(A)$ + 1's complement of $C(Y)$
+ 0...01 $\Rightarrow C(A)$
Carry Indicator OFF: $C(A)$ + 1's complement of $C(Y)$
$\Rightarrow C(A)$

(The +1 which is added in the first case represents the carry from the next lower part of the multiple-length subtraction.)

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SWCQ	Subtract with Carry from Q	172

SUMMARY: Carry Indicator ON: $C(Q) - C(Y)$ $\Rightarrow C(Q)$
                 Carry Indicator OFF: $C(Q) - C(Y) - 0...01 \Rightarrow C(Q)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Q) = 0$, then ON; otherwise OFF
Negative	If $C(Q)_0 = 1$, then ON; otherwise OFF
Overflow	If range of Q is exceeded, then ON
Carry	If carry out of Q_0 is generated, then ON; otherwise OFF

NOTES:

1. This instruction is identical to the SBQ instruction with the exception that, in case the Carry Indicator is OFF at the beginning of the instruction, then a +1 is subtracted from the least-significant position.

2. This instruction is used for multiple-word precision arithmetic. The SUMMARY can also be worded as follows in order to show the intended use:

Carry Indicator ON: $C(Q)$ + 1's complement of $C(Y)$
+ 0...01 $\Rightarrow C(Q)$

Carry Indicator OFF: $C(Q)$ + 1's complement of $C(Y)$
$\Rightarrow C(Q)$

(The +1 which is added in the first case represents the carry from the next lower part of the multiple-length subtraction).

CPB-1004F

Mnemonic:	Name of the Instruction:	Op Code (Octal)
MPY	Multiply Integer	402

SUMMARY: $C(Q) \times C(Y) \Rightarrow C(AQ)$, right-adjusted

MODIFICATIONS: All except CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(AQ) = 0$, then ON; otherwise OFF
Negative	If $C(AQ)_0 = 1$, then ON; otherwise OFF

NOTES: 1. Two 36-bit integer factors (including sign) are multiplied to form a 71-bit integer product (including sign), which is stored in AQ, right-adjusted. Bit position AQ_0 is filled with an "extended sign bit".

Combined AQ-register
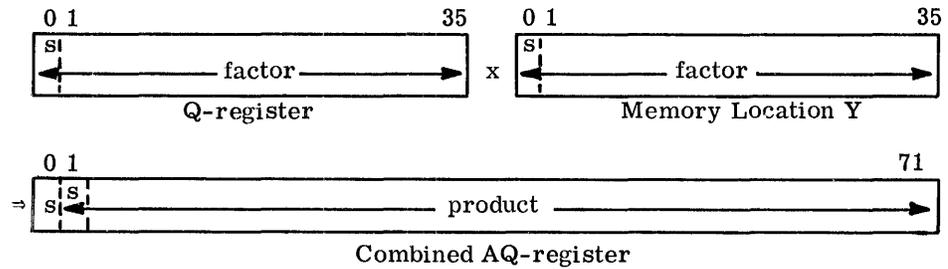
2. In the case of $(-2^{35}) \times (-2^{35}) = +2^{70}$, the position AQ_1 is used to represent this product without causing an overflow.

CPB-1004F

83

Mnemonic:	Name of the Instruction:	Op Code (Octal)
MPF	Multiply Fraction	401

SUMMARY: C(A) x C(Y) → C(AQ), left-adjusted

MODIFICATIONS: All except CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0$ = 1, then ON; otherwise OFF
Overflow	If range of AQ is exceeded, then ON

NOTES: 1. Two 36-bit fractional factors (including sign) are multiplied to form
a 71-bit fractional product (including sign), which is stored in AQ,
left-adjusted. Bit position AQ_{71} is filled with a zero bit.

Combined AQ-register
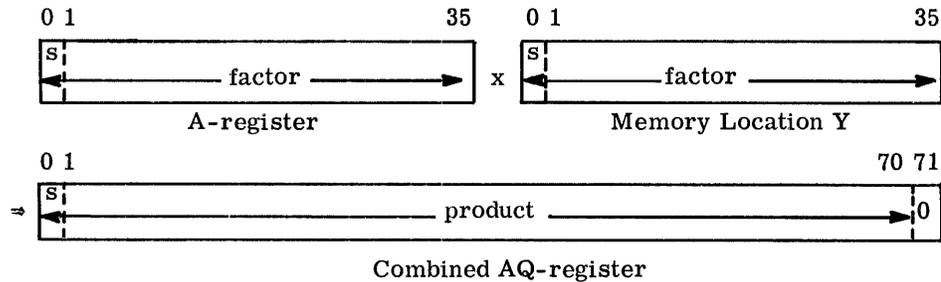
2. An overflow can occur only in the case (-1) x (-1).

CPB-1004F

84

Mnemonic:	Name of the Instruction:	Op Code (Octal)
DIV	Divide Integer	506

SUMMARY: C(Q) ÷ C(Y); integer quotient   → C(Q)
                                 integer remainder → C(A)

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

	If division takes place:	If no division takes place:
Zero	If C(Q) = 0, then ON; otherwise OFF	If divisor = 0, then ON; otherwise OFF
Negative	If $C(Q)_0$ = 1, then ON; otherwise OFF	If dividend < 0, then ON; otherwise OFF

NOTES: 1. A 36-bit integer dividend (including sign) is divided by a 36-bit
             integer divisor (including sign) to form a 36-bit integer quotient
             (including sign) and a 36-bit fractional remainder (including sign).
             The remainder sign is equal to the dividend sign unless the
             remainder is zero.

2. If dividend = -2^{35} and divisor = -1 or if divisor = 0, then the division
    itself does not take place.

    Instead, a Divide-Check Fault Trap occurs; the divisor C(Y) remains
    unchanged, C(Q) contains the dividend magnitude in absolute, and the
    Negative Indicator reflects the dividend sign.

CPB-1004F

85

Mnemonic:	Name of the Instruction:	Op Code (Octal)
DVF	Divide Fraction	507

SUMMARY: $C(AQ) \div C(Y)$; fractional quotient $\rightarrow C(A)$
remainder $\rightarrow C(Q)$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

	If division takes place:	If no division takes place:
Zero	If $C(A) = 0$, then ON; otherwise OFF	If divisor = 0, then ON; otherwise OFF
Negative	If $C(A)_0 = 1$, then ON; otherwise OFF	If dividend <0, then ON; otherwise OFF

NOTES:

1. A 71-bit fractional dividend (including sign) is divided by a 36-bit fractional divisor (including sign) to form a 36-bit fractional quotient (including sign) and a 36-bit remainder (including sign), bit position 35 of the remainder corresponding to bit position 70 of the dividend. The remainder sign is equal to the dividend sign unless the remainder is zero.
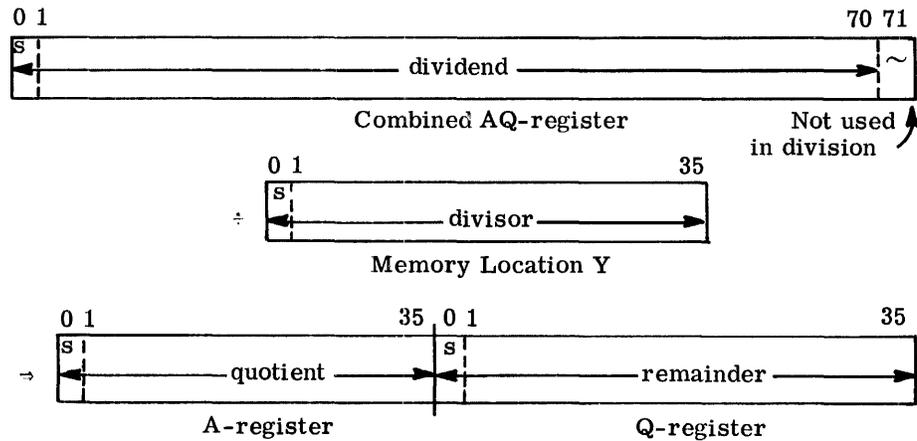
2. If $|\text{dividend}| \geq |\text{divisor}|$ or if divisor = 0, then the division itself does not take place.

Instead, a Divide-Check Fault Trap occurs; the divisor C(Y) remains unchanged, C(AQ) contains the dividend magnitude in absolute, and the Negative Indicator reflects the dividend sign.

CPB-1004F

86

Mnemonic:	Name of the Instruction:	Op Code (Octal)
NEG	Negate A	531

SUMMARY: - C(A) → C(A)

MODIFICATIONS: Are without any effect on the operation

INDICATORS: (Indicators not listed are not affected)

Zero	If C(A) = 0, then ON; otherwise OFF
Negative	If $C(A)_0$ = 1, then ON; otherwise OFF
Overflow	If range of A is exceeded, then ON

NOTE: This instruction changes the number in A to its negative (if ≠ 0). The operation is executed by forming the two's complement of the string of 36 bits.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
NEGL	Negate Long	533

SUMMARY: - C(AQ) → C(AQ)

MODIFICATIONS: Are without any effect on the operation

INDICATORS: (Indicators not listed are not affected)

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0$ = 1, then ON; otherwise OFF
Overflow	If range of AQ is exceeded, then ON

NOTE: This instruction changes the number in AQ to its negative (if ≠ 0). The operation is executed by forming the two's complement of the string of 72 bits.

CPB-1004F

87

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ANA	AND to A	375

SUMMARY: $C(A)_i$ AND $C(Y)_i \Rightarrow C(A)_i$ for all $i = 0, 1, \ldots, 35$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(A) = 0$, then ON; otherwise OFF
Negative	If $C(A)_0 = 1$, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ANQ	AND to Q	376

SUMMARY: $C(Q)_i$ AND $C(Y)_i \Rightarrow C(Q)_i$ for all $i = 0, 1, \ldots, 35$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Q) = 0$, then ON; otherwise OFF
Negative	If $C(Q)_0 = 1$, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ANAQ	AND to AQ	377

SUMMARY: $C(AQ)_i$ AND $C(Y\text{-pair})_i \Rightarrow C(AQ)_i$ for all $i = 0, 1, \ldots, 71$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(AQ) = 0$, then ON; otherwise OFF
Negative	If $C(AQ)_0 = 1$, then ON; otherwise OFF

CPB-1004F

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ANXn	AND to Xn (n = 0, 1,...,7)	36n

SUMMARY: $C(Xn)_i$ AND $C(Y)_i \Rightarrow C(Xn)_i$ for all i = 0, 1,...,17

MODIFICATIONS: All except CI, SC, DL

INDICATORS: (Indicators not listed are not affected)

Zero	If C(Xn) = 0, then ON; otherwise OFF
Negative	If $C(Xn)_0$ = 1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ANSA	AND to Storage A	355

SUMMARY: $C(A)_i$ AND $C(Y)_i \Rightarrow C(Y)_i$ for all i = 0, 1,...,35

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(Y) = 0, then ON; otherwise OFF
Negative	If $C(Y)_0$ = 1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ANSQ	AND to Storage Q	356

SUMMARY: $C(Q)_i$ AND $C(Y)_i \Rightarrow C(Y)_i$ for all i = 0, 1,...,35

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(Y) = 0, then ON; otherwise OFF
Negative	If $C(Y)_0$ = 1, then ON; otherwise OFF

CPB-1004F

89

Mnemonic:	Name of the Instruction:		Op Code (Octal)
ANSXn	AND to Storage Xn	(n = 0, 1,...,7)	34n

SUMMARY: $C(Xn)_i$ AND $C(Y)_i \Rightarrow C(Y)_i$ for all i = 0, 1,...,17

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Y)_{0...17}$ = 0, then ON; otherwise OFF
Negative	If $C(Y)_0$ = 1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ORA	OR to A	275

SUMMARY: $C(A)_i$ OR $C(Y)_i \Rightarrow C(A)_i$ for all i = 0, 1,...,35

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If C(A) = 0, then ON; otherwise OFF
Negative	If $C(A)_0$ = 1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ORQ	OR to Q	276

SUMMARY: $C(Q)_i$ OR $C(Y)_i \Rightarrow C(Q)_i$ for all i = 0, 1,...,35

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If C(Q) = 0, then ON; otherwise OFF
Negative	If $C(Q)_0$ = 1, then ON; otherwise OFF

CPB-1004F

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ORAQ	OR to AQ	277

SUMMARY: $C(AQ)_i$ OR $C(Y\text{-pair})_i \Rightarrow C(AQ)_i$ for all i = 0, 1,...,71

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0$ = 1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:		Op Code (Octal)
ORXn	OR to Xn	(n = 0, 1,...,7)	26n

SUMMARY: $C(Xn)_i$ OR $C(Y)_i \Rightarrow C(Xn)_i$ for all i = 0, 1,...,17

MODIFICATIONS: All except CI, SC, DL

INDICATORS: (Indicators not listed are not affected)

Zero	If C(Xn) = 0, then ON; otherwise OFF
Negative	If $C(Xn)_0$ = 1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ORSA	OR to Storage A	255

SUMMARY: $C(A)_i$ OR $C(Y)_i \Rightarrow C(Y)_i$ for all i = 0, 1,...,35

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(Y) = 0, then ON; otherwise OFF
Negative	If $C(Y)_0$ = 1, then ON; otherwise OFF

CPB-1004F

91

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ORSQ	OR to Storage Q	256

SUMMARY: $C(Q)_i$ OR $C(Y)_i \Rightarrow C(Y)_i$ for all i = 0, 1,..., 35

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Y)$ = 0, then ON; otherwise OFF
Negative	If $C(Y)_0$ = 1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:		Op Code (Octal)
ORSXn	OR to Storage Xn	(n = 0, 1,...,7)	24n

SUMMARY: $C(Xn)_i$ OR $C(Y)_i \Rightarrow C(Y)_i$ for all i = 0, 1,...,17

MODIFICATIONS: For all except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Y)_{0...17}$ = 0, then ON; otherwise OFF
Negative	If $C(Y)_0$ = 1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ERA	EXCLUSIVE OR to A	675

SUMMARY: $C(A)_i \neq C(Y)_i \Rightarrow C(A)_i$ for i = 0, 1,...,35

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(A)$ = 0, then ON; otherwise OFF
Negative	If $C(A)_0$ = 1, then ON; otherwise OFF

CPB-1004F

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ERQ	EXCLUSIVE OR to Q	676

SUMMARY: $C(Q)_i \neq C(Y)_i \to C(Q)_i$ for $i = 0, 1, \ldots, 35$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Q) = 0$, then ON; otherwise OFF
Negative	If $C(Q)_0 = 1$, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ERAQ	EXCLUSIVE OR to AQ	677

SUMMARY: $C(AQ)_i \neq C(Y\text{-pair})_i \to C(AQ)_i$ for all $i = 0, 1, \ldots, 71$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(AQ) = 0$, then ON; otherwise OFF
Negative	If $C(AQ)_0 = 1$, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:		Op Code (Octal)
ERXn	EXCLUSIVE OR to Xn	(n = 0, 1,...,17	66n

SUMMARY: $C(Xn)_i \neq C(Y)_i \to C(Xn)_i$ for $i = 0, 1, \ldots 17$

MODIFICATIONS: All except CI, SC, DL

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Xn) = 0$, then ON; otherwise OFF
Negative	If $C(Xn)_0 = 1$, then ON; otherwise OFF

CPB-1004F

93

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ERSA	EXCLUSIVE OR to Storage A	655

SUMMARY: $C(A)_i \neq C(Y)_i \Rightarrow C(Y)_i$ for $i = 0, 1, \ldots, 35$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Y) = 0$, then ON; otherwise OFF
Negative	If $C(Y)_0 = 1$, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ERSQ	EXCLUSIVE OR to Storage Q	656

SUMMARY: $C(Q)_i \neq C(Y)_i \Rightarrow C(Y)_i$ for $i = 0, 1, \ldots, 35$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Y) = 0$, then ON; otherwise OFF
Negative	If $C(Y)_0 = 1$, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
ERSXn	EXCLUSIVE OR to Storage Xn (n = 0,1,...,7)	64n

SUMMARY: $C(Xn)_i \neq C(Y)_i \Rightarrow C(Y)_i$ for $i = 0, 1, \ldots, 17$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(Y)_{0\ldots 17} = 0$, then ON; otherwise OFF
Negative	If $C(Y)_0 = 1$, then ON; otherwise OFF

CPB-1004F

94

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CMPA	Compare with A	115

SUMMARY: Comparison $C(A) \; :: \; C(Y)$

MODIFICATION: All

INDICATORS: (Indicators not listed are not affected)

Zero	Negative	Carry	Algebraic (Signed Fixed-Point) Comparison	
			Relation	Sign
0	0	0	$C(A) > C(Y)$	$C(A)_0 = 0, \; C(Y)_0 = 1$
0	0	1	$C(A) > C(Y)$	$C(A)_0 = C(Y)_0$
1	0	1	$C(A) = C(Y)$	
0	1	0	$C(A) < C(Y)$	
0	1	1	$C(A) < C(Y)$	$C(A)_0 = 1, \; C(Y)_0 = 0$

Zero	Carry	Logic (Unsigned Fixed-Point) Comparison
		Relation
0	0	$C(A) < C(Y)$
1	1	$C(A) = C(Y)$
0	1	$C(A) > C(Y)$

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CMPQ	Compare with Q	116

SUMMARY: Comparison $C(Q) :: C(Y)$

MODIFICATION: All

INDICATORS: (Indicators not listed are not affected)

Zero	Negative	Carry	Algebraic (Signed Fixed-Point) Comparison	
			Relation	Sign
0	0	0	$C(Q) > C(Y)$	$C(Q)_0 = 0,\ C(Y)_0 = 1$
0	0	1	$C(Q) > C(Y)$	$\left.\begin{array}{c} \\ \\ \\ \end{array}\right\}\ C(Q)_0 = C(Y)_0$
1	0	1	$C(Q) = C(Y)$	
0	1	0	$C(Q) < C(Y)$	
0	1	1	$C(Q) < C(Y)$	$C(Q)_0 = 1,\ C(Y)_0 = 0$

Zero	Carry	Logic (Unsigned Fixed-Point) Comparison
		Relation
0	0	$C(Q) < C(Y)$
1	1	$C(Q) = C(Y)$
0	1	$C(Q) > C(Y)$

CPB-1004F

96

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CMPAQ	Compare with AQ	117

SUMMARY: Comparison C(AQ) : : C(Y-pair)

MODIFICATION: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	Negative	Carry	Algebraic (Signed Fixed-Point) Comparison	
			Relation	Sign
0	0	0	$C(AQ) > C(Y\text{-pair})$	$C(AQ)_0 = 0,\ C(Y\text{-pair})_0 = 1$
0	0	1	$C(AQ) > C(Y\text{-pair})$	$C(AQ)_0 = C(Y\text{-pair})_0$
1	0	1	$C(AQ) = C(Y\text{-pair})$	
0	1	0	$C(AQ) < C(Y\text{-pair})$	
0	1	1	$C(AQ) < C(Y\text{-pair})$	$C(AQ)_0 = 1,\ C(Y\text{-pair})_0 = 0$

Zero	Carry	Logic (Unsigned Fixed-Point) Comparison
		Relation
0	0	$C(AQ) < C(Y\text{-pair})$
1	1	$C(AQ) = C(Y\text{-pair})$
0	1	$C(AQ) > C(Y\text{-pair})$

CPB-1004F

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CMPXn	Compare with Xn (n = 0, 1,..., 7)	10n

SUMMARY: Comparison $C(Xn)$ $::$ $C(Y)_{0...17}$

MODIFICATION: All except CI, SC, DL

INDICATORS: (Indicators not listed are not affected)

Zero	Negative	Carry	Algebraic (Signed Fixed-Point) Comparison	
			Relation	Sign
0	0	0	$C(Xn) > C(Y)_{0...17}$	$C(Xn)_0 = 0$, $C(Y)_0 = 1$
0	0	1	$C(Xn) > C(Y)_{0...17}$	
1	0	1	$C(Xn) = C(Y)_{0...17}$	$C(Xn)_0 = C(Y)_0$
0	1	0	$C(Xn) < C(Y)_{0...17}$	
0	1	1	$C(Xn) < C(Y)_{0...17}$	$C(Xn)_0 = 1$, $C(Y)_0 = 0$

Zero	Carry	Logic (Unsigned Fixed-Point) Comparison
		Relation
0	0	$C(Xn) < C(Y)_{0...17}$
1	1	$C(Xn) = C(Y)_{0...17}$
0	1	$C(Xn) > C(Y)_{0...17}$

CPB-1004F

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CWL	Compare with Limits	111

SUMMARY: Algebraic comparison of C(Y) with the closed interval
$[\ C(A);\ C(Q)]$ and also with the number C(Q)

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If C(Y) is contained in the closed interval $[\ C(A)\ ;\ C(Q)]$, i.e., either $C(A) \leqq C(Y) \leqq C(Q)$ or $C(A) \geqq C(Y) \geqq C(Q)$, then ON; otherwise OFF

Negative	Carry	Relation between C(Q) and C(Y)	Signs of C(Q) and C(Y)
0	0	$C(Q) > C(Y)$	$C(Q)_0 = 0,\ C(Y)_0 = 1$
0	1	$C(Q) \geqq C(Y)$	$C(Q)_0 = C(Y)_0$
1	0	$C(Q) < C(Y)$	
1	1	$C(Q) < C(Y)$	$C(Q)_0 = 1,\ C(Y)_0 = 0$

CPB-1004F

99

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CMG	Compare Magnitude	405

SUMMARY: Comparison $\left|C(A)\right|$ :: $\left|C(Y)\right|$

MODIFICATION: All

INDICATORS: (Indicators not listed are not affected)

Zero	Negative	Relation
0	0	$\left|C(A)\right|$ > $\left|C(Y)\right|$
1	0	$\left|C(A)\right|$ = $\left|C(Y)\right|$
0	1	$\left|C(A)\right|$ < $\left|C(Y)\right|$

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SZN	Set Zero and Negative Indicators from Memory	234

SUMMARY: Test the number C(Y)

MODIFICATION: All

INDICATORS: (Indicators not listed are not affected)

Zero	Negative	Relation
0	0	Number C(Y) > 0
1	0	Number C(Y) = 0
0	1	Number C(Y) < 0

CPB-1004F

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CMK	Compare Masked	211

SUMMARY: $Z_i = \overline{C(Q)}_i$ AND $[C(A)_i \neq C(Y)_i]$ for all $i = 0, 1, \ldots, 35$

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $Z = 0$, then ON; otherwise OFF
Negative	If $Z_0 = 1$, then ON; otherwise OFF

NOTE: This instruction compares those corresponding bit positions of A and Y for identity that are not masked by a 1 in the corresponding bit position of Q

The Zero Indicator is set ON, if the comparison is successful for bit positions; i.e. if for all $i = 0, 1, \ldots, 35$ there is

$$\text{either} \quad C(A)_i \equiv C(Y)_i \quad \text{or} \quad C(Q)_i = 1$$
$$(\text{identical}) \quad\quad\quad (\text{masked})$$

Otherwise it is set OFF

The Negative Indicator is set ON, if the comparison is unsuccessful for bit position 0, i.e. if

$$C(A)_0 \neq C(Y)_0 \quad \text{as well as} \quad C(Q)_0 = 0$$
$$(\text{nonidentical}) \quad\quad\quad (\text{nonmasked})$$

Otherwise it is set OFF.

CODING EXAMPLE:

In the following example, the comparison is equal after execution of CMK, and the TZE exit is taken. Only the 1's in NUMBER AND DATA are compared.

```
 0        8         16
          | LDQ     | MASK
          | LDA     | NUMBER
          | CMK     | DATA
          | TZE     | OUT
          | Continue|
          |         |
          |         |
  MASK    | OCT     | 777777777707
  NUMBER  | OCT     | 300333333316
  DATA    | OCT     | 666666666615
```

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CANA	Comparative AND with A	315

SUMMARY: $Z_i = C(A)_i \text{ AND } C(Y)_i$ for all i = 0, 1,...,35

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $Z = 0$, then ON; otherwise OFF
Negative	If $Z_0 = 1$, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CANQ	Comparative AND with Q	316

SUMMARY: $Z_i = C(Q)_i \text{ AND } C(Y)_i$ for all i = 0, 1,...,35

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If $Z = 0$, then ON; otherwise OFF
Negative	If $Z_0 = 1$, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CANAQ	Comparative AND with AQ	317

SUMMARY: $Z_i = C(AQ)_i \text{ AND } C(Y\text{-pair})_i$ for all i = 0, 1,...,71

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $Z = 0$, then ON; otherwise OFF
Negative	If $Z_0 = 1$, then ON; otherwise OFF

CPB-1004F

Mnemonic:	Name of the Instruction:		Op Code (Octal)
CANXn	Comparative AND with Xn	(n = 0,1,...,7)	30n

SUMMARY: $Z_i = C(Xn)_i$ AND $C(Y)_i$ for all i = 0, 1,...,17

MODIFICATIONS: All except CI, SC, DL

INDICATORS: (Indicators not listed are not affected)

Zero	If Z = 0, then ON; otherwise OFF
Negative	If Z_0 = 1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CNAA	Comparative NOT with A	215

SUMMARY: $Z_i = C(A)_i$ AND $\overline{C(Y)_i}$ for all i = 0, 1,...,35

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If Z = 0, then ON; otherwise OFF
Negative	If Z_0 = 1, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CNAQ	Comparative NOT with Q	216

SUMMARY: $Z_i = C(Q)_i$ AND $\overline{C(Y)_i}$ for all i = 0, 1,...,35

MODIFICATIONS: All

INDICATORS: (Indicators not listed are not affected)

Zero	If Z = 0, then ON; otherwise OFF
Negative	If Z_0 = 1, then ON; otherwise OFF

CPB-1004F

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CNAAQ	Comparative NOT with AQ	217

SUMMARY: $Z_i = C(AQ)_i$ AND $\overline{C(Y\text{-pair})}_i$ for all $i = 0, 1, \ldots, 71$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $Z = 0$, then ON; otherwise OFF
Negative	If $Z_0 = 1$, then ON; otherwise OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CNAXn	Comparative NOT with Xn	20n

SUMMARY: $Z_i = C(Xn)_i$ AND $\overline{C(Y)}_i$ for all $i = 0, 1, \ldots, 17$

MODIFICATIONS: All except CI, SC, DL

INDICATORS: (Indicators not listed are not affected)

Zero	If $Z = 0$, then ON; otherwise OFF
Negative	If $Z_0 = 1$, then ON; otherwise OFF

CPB-1004F

104

Mnemonic: Name of the Instruction: Op Code (Octal)

FLD	Floating Load	431

SUMMARY: $C(Y), 00...0 \Rightarrow C(EAQ)$

MODIFICATIONS: All except CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(AQ) = 0$, then ON; otherwise OFF
Negative	If $C(AQ)_0 = 1$, then ON; otherwise OFF

NOTE: $C(Y)_{0...7} \Rightarrow C(E)$
$C(Y)_{8...35} \Rightarrow C(AQ)_{0...27}$
$00...0 \Rightarrow C(AQ)_{28...71}$

Mnemonic: Name of the Instruction: Op Code (Octal)

DFLD	Double-Precision Floating Load	433

SUMMARY: $C(Y\text{-pair}), 00...0 \Rightarrow C(EAQ)$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If $C(AQ) = 0$, then ON; otherwise OFF
Negative	If $C(AQ)_0 - 1$, then ON; otherwise OFF

NOTE: $C(Y\text{-pair})_{0...7} \Rightarrow C(E)$
$C(Y\text{-pair})_{8...71} \Rightarrow C(AQ)_{0...63}$
$00...0 \Rightarrow C(AQ)_{64...71}$

Mnemonic: Name of the Instruction: Op Code (Octal)

LDE	Load Exponent Register	411

SUMMARY: $C(Y)_{0...7} \Rightarrow C(E)$

MODIFICATIONS: All except CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	Set OFF
Negative	Set OFF

CPB-1004F

105

Mnemonic:	Name of the Instruction:	Op Code (Octal)
FST	Floating Store	455

SUMMARY: $C(EAQ) \Rightarrow C(Y)$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

NOTE: This instruction is executed as follows:

$$C(E) \Rightarrow C(Y)_{0\ldots7}$$
$$C(A)_{0\ldots27} \Rightarrow C(Y)_{8\ldots35}$$

Mnemonic:	Name of the Instruction:	Op Code (Octal)
DFST	Double-Precision Floating Store	457

SUMMARY: $C(EAQ) \Rightarrow C(Y\text{-pair})$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

NOTE: This instruction is executed as follows:

$$C(E) \Rightarrow C(Y\text{-pair})_{0\ldots7}$$
$$C(AQ)_{0\ldots63} \Rightarrow C(Y\text{-pair})_{8\ldots71}$$

Mnemonic:	Name of the Instruction:	Op Code (Octal)
STE	Store Exponent Register	456

SUMMARY: $C(E) \Rightarrow C(Y)_{0\ldots7} ; 00\ldots0 \Rightarrow C(Y)_{8\ldots17}$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
FSTR	Floating Store Rounded	470

SUMMARY: $C(EAQ)$ rounded $\Rightarrow C(Y)$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

EXP. Overflow	If exponent above +127, then ON

NOTE: During single-precision floating point stores, this instruction rounds the number (positive or negative) as it is stored.

The instruction is executed by adding a binary one to bit position 28 of AQ, truncating, then storing the contents of AQ. Steps in the execution are as follows:

$$C(AQ)_{0...71} + 2^{-28} \Rightarrow C(AQ)_{0...27}$$

$$0...0 \Rightarrow C(AQ)_{28...71}$$

$$C(E) \Rightarrow C(Y)_{0...7}$$

$$C(A)_{0...27} \Rightarrow C(Y)_{8...35}$$

Restore $C(EAQ)$ to original values

All registers remain unchanged.

An exponent overflow occurs only if $C(E) = +127$ and $C(AQ)_{0...28} = 0.111...111$ before rounding.

If the original operand is a negative number $\left[C(AQ)_{0...28} = 1.0111...111 \text{ and } C(AQ)_{29...71} = 0 \right]$, the number is rounded towards zero, not towards a more negative value, and the result becomes unnormalized.

Normalization occurs only if the mantissa overflows when it is rounded.

Mnemonic:	Name of the Instruction	Op Code (Octal)
FAD	Floating Add	475

SUMMARY: $\left[C(EAQ) + C(Y)\right]$ normalized $\Rightarrow$ C(EAQ)

MODIFICATIONS: All except CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0$ = 1, then ON; otherwise OFF
Exp. Overflow	If Exponent above +127, then ON
Exp. Underflow	If Exponent below -128, then ON
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction	Op Code (Octal)
UFA	Unnormalized Floating Add	435

SUMMARY: $\left[C(EAQ) + C(Y)\right]$ not normalized $\Rightarrow$ C(EAQ)

MODIFICATIONS: All except CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0$ = 1, then ON; otherwise OFF
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

CPB-1004F

Mnemonic:	Name of the Instruction	Op Code (Octal)
DFAD	Double-Precision Floating Add	477

SUMMARY: $\left[C(EAQ) + C(Y\text{-pair})\right]$ normalized $\Rightarrow$ C(EAQ)

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0$ = 1, then ON; otherwise OFF
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction	Op Code (Octal)
DUFA	Double-Precision Unnormalized Floating Add	437

SUMMARY: $\left[C(EAQ) + C(Y\text{-pair})\right]$ not normalized $\Rightarrow$ C(EAQ)

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0$ = 1, then ON; otherwise OFF
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

CPB-1004F

108

Mnemonic:	Name of the Instruction	Op Code (Octal)
ADE	Add to Exponent Register	415

SUMMARY: $C(E) + C(Y)_{0...7} \Rightarrow C(E)$

MODIFICATIONS: All except CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	Set OFF
Negative	Set OFF
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON

```
FLOATING POINT
SUBTRACTION
```

Mnemonic:	Name of the Instruction	Op Code (Octal)
FSB	Floating Subtract	575

SUMMARY: $\left[C(EAQ) - C(Y) \right]$ normalized $\Rightarrow$ C(EAQ)

MODIFICATIONS: All except CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0$ = 1, then ON; otherwise OFF
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction	Op Code (Octal)
UFS	Unnormalized Floating Subtract	535

SUMMARY: $\left[C(EAQ) - C(Y) \right]$ not normalized $\Rightarrow$ C(EAQ)

MODIFICATIONS: All except CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0 = 1$, then ON; otherwise OFF
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction	Op Code (Octal)
DFSB	Double-Precision Floating Subtract	577

SUMMARY: $\left[C(EAQ) - C(Y\text{-pair}) \right]$ normalized $\Rightarrow$ C(EAQ)

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0 = 1$, then ON; otherwise OFF
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

CPB-1004F

Mnemonic:	Name of the Instruction	Op Code (Octal)
DUFS	Double-Precision Unnormalized Floating Subtract	537

SUMMARY: $\left[C(EAQ) - C(Y\text{-pair})\right]$ not normalized $\Rightarrow$ C(EAQ)

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(AQ) $= 0$, then ON; otherwise OFF
Negative	If $C(AQ)_0 = 1$, then ON; otherwise OFF
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON
Carry	If a carry out of AQ_0 is generated, then ON; otherwise OFF

Mnemonic:	Name of the Instruction	Op Code (Octal)
FMP	Floating Multiply	461

SUMMARY: $\left[C(EAQ) \times C(Y)\right]$ normalized $\Rightarrow$ C(EAQ)

MODIFICATIONS: All except CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(AQ) $= 0$, then ON; otherwise OFF
Negative	If $C(AQ)_0 = 1$, then ON; otherwise OFF
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON

NOTES: This multiplication is executed as follows:

1. $C(E) + C(Y)_{0...7} \Rightarrow C(E)$

2. $C(AQ) \times C(Y)_{8...35}$ results in a 98-bit product plus sign, the leading 71 bits plus sign of which $\Rightarrow$ C(AQ)

3. C(EAQ) normalized $\Rightarrow$ C(EAQ) .

CPB-1004F

111

Mnemonic:	Name of the Instruction	Op Code (Octal)
UFM	Unnormalized Floating Multiply	421

SUMMARY:　　$\left[C(EAQ) \times C(Y)\right]$ not normalized $\Rightarrow$ C(EAQ)

MODIFICATIONS:　　All except CI, SC

INDICATORS:　　(Indicators not listed are not affected)

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0 = 1$, then ON, otherwise OFF
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON

NOTE:　　This multiplication is executed like the instruction FMP with the exception that the final normalization is performed only in the case of both factor mantissas being $= -1.00\cdots 0$.

Mnemonic:	Name of the Instruction	Op Code (Octal)
DFMP	Double-Precision Floating Multiply	463

SUMMARY:　　$\left[C(EAQ) \times C(Y\text{-pair})\right]$ normalized $\Rightarrow$ C(EAQ)

MODIFICATIONS:　　All except DU, DL, CI, SC

INDICATORS:　　(Indicators not listed are not affected)

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0 = 1$, then ON; otherwise OFF
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON

NOTE:　　This multiplication is executed as follows:

1. $C(E) + C(Y\text{-pair})_{0\ldots 7} \Rightarrow C(E)$
2. $C(AQ) \times C(Y\text{-pair})_{8\ldots 71}$ results in a 134-bit product plus sign, the leading 71 bits plus sign of which $\Rightarrow$ C(AQ)
3. $C(EAQ)$ normalized $\Rightarrow$ C(EAQ).

Mnemonic:	Name of the Instruction	Op Code (Octal)
DUFM	Double-Precision Unnormalized Floating Multiply	423

SUMMARY: $\left[C(EAQ) \times X(Y\text{-pair}) \right]$ not normalized $\Rightarrow$ C(EAQ)

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0$ =1, then ON; otherwise OFF
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON

NOTE: This multiplication is executed like the instruction DFMP, with the exception that the final normalization is performed only in the case of both factor mantissas being = - 1.00...0 .

```
┌─────────────────────┐
│ FLOATING POINT      │
│    DIVISION         │
└─────────────────────┘
```

Mnemonic:	Name of the Instruction	Op Code (Octal)
FDV	Floating Divide	565

SUMMARY: $C(EAQ) \div C(Y) \Rightarrow C(EA)$; $00...0 \Rightarrow C(Q)$

MODIFICATIONS: All except CI, SC

INDICATORS: (Indicators not listed are not affected)

	If division takes place:	If no division takes place:
Zero	If C(A) = 0, then ON; otherwise OFF	If divisor mantissa = 0, then ON; otherwise OFF
Negative	If $C(A)_0$ = 1, then ON; otherwise OFF	If dividend < 0, tnen ON; otherwise OFF
Exp. Overflow	If exponent above +127, then ON	
Exp. Underflow	If exponent below -128, then ON	

NOTES: 1. This division is executed as follows:

The dividend mantissa C(AQ) is shifted right and the dividend exponent C(E) increased accordingly until

$$\left| C(AQ)_{0...27} \right| < \left| C(Y)_{8...35} \right| ;$$

$$C(E) - C(Y)_{0...7} \Rightarrow C(E) ;$$

$$C(AQ) \div C(Y)_{8...35} \Rightarrow C(A) ;$$

$$00...0 \Rightarrow C(Q) .$$

2. If mantissa of divisor = 0, then the division itself does not take place. Instead, a Divide-Check Fault Trap occurs. The divisor C(Y) remains unchanged, C(AQ) contains the dividend magnitude in absolute, and the Negative indicator reflects the dividend sign.

CPB-1004F

Mnemonic:	Name of the Instruction	Op Code (Octal)
FDI	Floating Divide Inverted	525

SUMMARY: $C(Y) \div C(EAQ) \Rightarrow C(EA)$; $00...0 \Rightarrow C(Q)$

MODIFICATIONS: All except CI, SC

INDICATORS: (Indicators not listed are not affected)

	If division takes place:	If no division takes place:
Zero	If $C(A) = 0$, then ON; otherwise OFF	If divisor mantissa = 0, then ON; otherwise OFF
Negative	If $C(A)_0 = 1$, then ON; otherwise OFF	If dividend < 0, then ON; otherwise OFF
Exp. Overflow	If exponent above +127, then ON	
Exp. Underflow	If exponent below -128, then ON	

NOTES:

1. This division is executed as follows:

   The dividend mantissa $C(Y)_{8...35}$ is shifted right and the dividend exponent $C(Y)_{0...7}$ increased accordingly until $\left| C(Y)_{8...35} \right|$ < $\left| C(AQ)_{0...27} \right|$;

   $C(Y)_{0...7} - C(E) \Rightarrow C(E)$;

   $C(Y)_{8...35} \div C(AQ) \Rightarrow C(A)$;

   $00...0 \qquad \Rightarrow C(Q)$.

2. If mantissa of divisor = 0, the division itself does not take place. Instead, a Divide-Check Fault Trap occurs; and all the registers remain unchanged.

Mnemonic:	Name of the Instruction	Op Code (Octal)
DFDV	Double-Precision Floating Divide	567

SUMMARY: $C(EAQ) \div C(Y\text{-pair}) \Rightarrow C(EAQ)$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

	If division takes place:	If no division takes place:
Zero	If $C(AQ) = 0$, then ON; otherwise OFF	If divisor mantissa $= 0$, then ON; otherwise OFF
Negative	If $C(AQ)_0 = 1$, then ON; otherwise OFF	If dividend < 0, then ON; otherwise OFF
Exp. Overflow	If exponent above +127, then ON	
Exp. Underflow	If exponent below -128, then ON	

NOTES: 1. This division is executed as follows:

The dividend mantissa $C(AQ)$ is shifted right and the dividend exponent $C(E)$ increased accordingly until $\left| C(AQ)_{0...63} \right| < \left| C(Y\text{-pair})_{8...71} \right|$;

$C(E) - C(Y\text{-pair})_{0...7} \Rightarrow C(E)$;

$C(AQ) \div C(Y\text{-pair})_{8...71} \Rightarrow C(AQ)_{0...63}$;

$00...0 \Rightarrow C(AQ)_{64...71}$.

2. If mantissa of divisor $= 0$, then the division itself does not take place. Instead, a Divide-Check Fault Trap occurs. The divisor $C(Y)$ remains unchanged, $C(AQ)$ contains the dividend magnitude in absolute, and the Negative indicator reflects the dividend sign.

CPB-1004F

Mnemonic:	Name of the Instruction	Op Code (Octal)
DFDI	Double-Precision Floating Divide Inverted	527

SUMMARY: $C(\text{Y-pair}) \div C(EAQ) \Rightarrow C(EAQ)$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

	If division takes place:	If no division takes place:
Zero	If $C(AQ) = 0$, then ON; otherwise OFF	If divisor mantissa = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0 = 1$, then ON; otherwise OFF	If dividend < 0, then ON; otherwise OFF
Exp. Overflow	If exponent above +127, then ON	
Exp. Underflow	If exponent below -128, then ON	

NOTES: 1. This division is executed as follows:

The dividend mantissa $C(\text{Y-pair})_{8\ldots71}$ is shifted right and the dividend exponent $C(\text{Y-pair})_{0\ldots7}$ increased accordingly until $\left| C(\text{Y-pair})_{8\ldots71} \right| < \left| C(AQ)_{0\ldots63} \right|$

$C(\text{Y-pair})_{0\ldots7} - C(E) \Rightarrow C(E)$;

$C(\text{Y-pair})_{8\ldots71} \div C(AQ) \Rightarrow C(AQ)_{0\ldots63}$;

$00\ldots0 \Rightarrow C(AQ)_{64\ldots71}$.

2. If mantissa of divisor = 0, then the division itself does not take place. Instead, a Divide-Check Fault Trap occurs; and all the registers remain unchanged.

Mnemonic:	Name of the Instruction	Op Code (Octal)
FNEG	Floating Negate	513

SUMMARY: $-$ C(AQ) normalized $\Rightarrow$ C(AQ)

MODIFICATIONS: Are without any effect on the operation

INDICATORS: (Indicators not listed are not affected)

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0$ = 1, then ON; otherwise OFF
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON

NOTES: 1. This instruction changes the number in EAQ to its normalized negative (if C(AQ) $\neq$ 0). The operation is executed by first forming the two's complement of C(AQ), and then normalizing C(EAQ).

2. Even if originally C(EAQ) were normalized, an exponent overflow can still occur, namely when originally C(AQ) = -1.00...0 and C(E) = +127.

Mnemonic:	Name of the Instruction	Op Code (Octal)
FNO	Floating Normalize	573

SUMMARY: C(EAQ) normalized $\Rightarrow$ C(EAQ)

MODIFICATIONS: Are without any effect on the operation

INDICATORS: (Indicators not listed are not affected)

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0$ = 1, then ON; otherwise OFF
Exp. Overflow	If exponent above +127, then ON
Exp. Underflow	If exponent below -128, then ON
Overflow	Set OFF

See NOTE on following page.

CPB-1004F

NOTE: The instruction normalizes the number in EAQ.
If the Overflow Indicator is ON, then the number in EAQ is
normalized one place to the right; and then the sign bit $C(AQ)_0$
is inverted in order to reconstitute the actual sign. Furthermore,
the Overflow Indicator is set OFF.

This instruction can be used to correct overflows that occurred
with fixed-point numbers.

FLOATING POINT
COMPARE

Mnemonic:	Name of the Instruction:	Op Code (Octal)
FCMP	Floating Compare	515

SUMMARY: Algebraic comparison $C\left[(E)(AQ_{0...27})\right]$ :: $C(Y)$

MODIFICATION: All except CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	Negative	Relation
0	0	$C\left[(E)(AQ_{0...27})\right] > C(Y)$
1	0	$C\left[(E)(AQ_{0...27})\right] = C(Y)$
0	1	$C\left[(E)(AQ_{0...27})\right] < C(Y)$

NOTE: This comparison is executed as follows:

1.   Compare $(C(E) :: C(Y)_{0...7}$, select the number with the lower
exponent, and shift its mantissa right as many places as the
difference of the exponents. If the number of shifts equals or
exceeds 72, the number with the lower exponent is defined as
zero.

2.   Then compare the mantissas and set the indicators accordingly.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
FCMG	Floating Compare Magnitude	425

SUMMARY: Comparison $\left| C \left[(E)(AQ_{0\ldots27}) \right] \right|$ :: $\left| C(Y) \right|$

MODIFICATION: All except CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	Negative	Relation
0	0	$\left| C \left[(E)(AQ_{0\ldots27}) \right] \right| > \left| C(Y) \right|$
1	0	$\left| C \left[(E)(AQ_{0\ldots27}) \right] \right| = \left| C(Y) \right|$
0	1	$\left| C \left[(E)(AQ_{0\ldots27}) \right] \right| < \left| C(Y) \right|$

NOTE: This comparison is executed as follows:

1.   Compare $C(E)$ :: $C(Y)_{0\ldots7}$, select the number with the lower exponent, and shift its mantissa right as many places as the difference of the exponents. If the number of shifts equals or exceeds 72, the number with the lower exponent is defined as zero.

2.   Then compare the absolute value of the mantissas and set the indicators accordingly.

CPB-1004F

120

Mnemonic:	Name of the Instruction:	Op Code (Octal)
DFCMP	Double-Precision Floating Compare	517

SUMMARY: Algebraic comparison $C\left[(E)(AQ_{0...63})\right]$ :: C(Y-pair)

MODIFICATION: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	Negative	Relation
0	0	$C\left[(E)(AQ_{0...63})\right] >$ C(Y-pair)
1	0	$C\left[(E)(AQ_{0...63})\right] =$ C(Y-pair)
0	1	$C\left[(E)(AQ_{0...63})\right] <$ C(Y-pair)

NOTE: This comparison is executed as follows:

1.   Compare $C(E)$ :: $C(Y)_{0...7}$, select the number with the lower exponent, and shift its mantissa right as many places as the difference of the exponents. If the number of shifts equals or exceeds 72, the number with the lower exponent is defined as zero.

2.   Then compare the mantissas and set the indicators accordingly.

CPB-1004 F

121

Mnemonic:	Name of the Instruction:	Op Code (Octal)
DFCMG	Double-Precision Floating Compare Magnitude	427

SUMMARY: Comparison $\left| C\left[(E)(AQ_{0...63})\right]\right| :: \left| C(Y\text{-pair})\right|$

MODIFICATION: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	Negative	Relation
0	0	$\left| C\left[(E)(AQ_{0...63})\right]\right| > \left| C(Y\text{-pair})\right|$
1	0	$\left| C\left[(E)(AQ_{0...63})\right]\right| = \left| C(Y\text{-pair})\right|$
0	1	$\left| C\left[(E)(AQ_{0...63})\right]\right| < \left| C(Y\text{-pair})\right|$

NOTE: This comparison is executed as follows:

1. Compare $C(E) :: C(Y)_{0...7}$, select the number with the lower exponent, and shift its mantissa right as many places as the difference of the exponents. If the number of shifts equals or exceeds 72, the number with the lower exponent is defined as zero.

2. Then compare the absolute value of the mantissas and set the indicators accordingly.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
FSZN	Floating Set Zero and Negative Indicators from Memory	430

SUMMARY: Test the Number C(Y)

MODIFICATION: All except CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	Negative	Relation
0	0	Mantissa $C(Y)_{8...35} > 0$
1	0	Mantissa $C(Y)_{8...35} = 0$
0	1	Mantissa $C(Y)_{8...35} < 0$

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TRA	Transfer Unconditionally	710

SUMMARY: $Y \Rightarrow C(IC)$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

Mnemonic:	Name of the Instruction:		Op Code (Octal)
TSXn	Transfer and Set Xn	(n = 0,1,...,7)	70n

SUMMARY: $C(IC) + 0_{\circ}..01 \Rightarrow C(Xn); \quad Y \Rightarrow C(IC)$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TSS	Transfer and Set Slave	715

SUMMARY: $Y \Rightarrow C(IC)$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Master Mode	Set OFF	

CPB-1004F

124

Mnemonic:	Name of the Instruction:	Op Code (Octal)
RET	Return	630

SUMMARY: $C(Y)_{0...17} \to C(IC); \quad C(Y)_{18...35} \to C(IR)$

MODIFICATIONS: All except CI, SC, DU, DL

INDICATORS: (Indicators not listed are not affected)

Master Mode	If $C(Y)_{28}$ is 1, then no change; otherwise OFF
All other indicators	If Corresponding bit in C(Y) is 1, then ON; otherwise OFF

NOTES: 1. The relation between bit position of C(Y) and the indicators is as follows:

Bit Position	Indicator
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent Overflow
23	Exponent Underflow
24	Overflow Mask
25	Tally Runout
26	Parity Error
27	Parity Mask
28	Master Mode
29	
30	
31	
32	00...0
33	
34	
35	

2. A possible change of the status of the Master Mode Indicator takes place as the last part of the instruction execution.

3. The Tally Runout Indicator will reflect $C(Y)_{25}$ regardless of what address modification is performed on the RET instruction (for tally operations).

CPB-1004F

125

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TZE	Transfer on Zero	600

SUMMARY: If Zero Indicator ON, then Y $\Rightarrow$ C(IC)

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TNZ	Transfer on Not Zero	601

SUMMARY: If Zero Indicator OFF, then Y $\Rightarrow$ C(IC)

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TMI	Transfer on Minus	604

SUMMARY: If Negative Indicator ON, then Y $\Rightarrow$ C(IC)

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TPL	Transfer on Plus	605

SUMMARY: If Negative Indicator OFF, then Y $\Rightarrow$ C(IC)

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

CPB-1004F

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TRC	Transfer on Carry	603

SUMMARY: If Carry Indicator ON, then Y $\Rightarrow$ C(IC)

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TNC	Transfer on No Carry	602

SUMMARY: If Carry Indicator OFF, then Y $\Rightarrow$ C(IC)

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TOV	Transfer on Overflow	617

SUMMARY: If Overflow Indicator ON, then Y $\Rightarrow$ C(IC)

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Overflow	Set OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TEO	Transfer on Exponent Overflow	614

SUMMARY: If Exponent Overflow Indicator ON, then Y $\Rightarrow$ C(IC)

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Exp. Overflow	Set OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TEU	Transfer on Exponent Underflow	615

SUMMARY: If Exponent Underflow Indicator ON, then Y $\Rightarrow$ C(IC)

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Exp. Underflow	Set OFF

Mnemonic:	Name of the Instruction:	Op Code (Octal)
TTF	Transfer on Tally Runout Indicator OFF	607

SUMMARY: If Tally Runout Indicator OFF, then Y $\Rightarrow$ C(IC)

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

CPB-1004F

Mnemonic:	Name of the Instruction:	Op Code (Octal)
NOP	No Operation	011

SUMMARY: No operation takes place

MODIFICATIONS: Generally the only modification that should be used is DU or DL (see NOTES)

INDICATORS: None affected

NOTES: 1. The use of a modification ID, DI, IDC, DIC, SC causes the respective changes in the address and the tally.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
BCD	Binary to Binary-Coded-Decimal	505

SUMMARY: Shift C(A) left 3 positions

$|C(A)| \div C(Y) \Rightarrow$ 4-bit quotient; $C(A) - C(Y) \times$ quotient $\Rightarrow$ remainder

Shift C(Q) left 6 positions; 4-bit quotient $\Rightarrow C(Q)_{32...35}$ and remainder $\Rightarrow C(A)$.

MODIFICATIONS: All except CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(A) = 0, then ON; otherwise OFF
Negative	If before execution $C(A)_0 = 1$, then ON; otherwise OFF

Restrictions:

1. The largest number which can be converted with the BCD instruction is that which is represented by 33 bits.

2. One 6-bit character is produced each time the BCD instruction is executed.

3. The character produced represents a decimal digit from 0 to 9.

4. One full 36-bit word cannot be directly converted by the BCD instructions.

NOTE: This instruction carries out one step of an algorithm for the conversion
of a binary number to the equivalent binary-coded decimal, which requires
the repeated short division of the binary number or last remainder by a
36-bit constant from store.

$$c_i = 8^i \times 10^{n-i} \text{ (for } i=1, 2,\ldots),$$

with n being defined by

$$10^{n-1} \leq \left| \text{number} \right| \leq 10^n - 1.$$

Mnemonic:	Name of the Instruction:	Op Code (Octal)
GTB	Gray to Binary	774

SUMMARY: C(A) converted from Gray Code to binary representation $\Rightarrow$ C(A)

MODIFICATIONS: Are without any effect on the operation

INDICATORS: (Indicators not listed are not affected)

Zero	If C(A) = 0, then ON; otherwise OFF
Negative	If $C(A)_0$ = 1, then ON; otherwise OFF

NOTE: This conversion is defined by the following algorithm, when R_i and S_i
denote the contents of bit positions i of the A-register before and
after the conversion:

$$S_0 = R_0$$
$$S_i = (R_i \text{ AND } \overline{S_{i-1}}) \text{ OR } (\overline{R_i} \text{ AND } S_{i-1})$$
$$\text{for } i = 1, 2,\ldots,35.$$

Mnemonic:	Name of the Instruction:	Op Code (Octal)
XEC	Execute	716

SUMMARY: Obtain and execute the instruction stored at the memory location Y

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

	The XEC instruction itself does not affect any indicator. However, the execution of the instruction from Y may affect indicators.

NOTES:

1. After the execution of the instruction obtained from location Y, the next instruction to be executed is obtained from C(IC) + 1. This is the one stored in memory right after this XEC instruction, unless the contents of the Instruction Counter have been changed by the execution of the instruction obtained from memory location Y.

2. To Execute (XEC) a Repeat Double (RPD) instruction, the XEC instruction must be in an odd location. Note that the instructions that are repeated are those which immediately follow the XEC instruction.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
XED	Execute Double	717

SUMMARY: Obtain and execute the two instructions stored at the memory Y-pair locations

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

	The XED instruction itself does not affect any indicator. However, the execution of the two instructions from Y-pair may affect indicators.

NOTES:

1. The first instruction obtained from Y-pair MUST NOT alter the memory location from which the second instruction is obtained, and MUST NOT be another XED instruction.

2. If the first instruction obtained from Y-pair alters the contents of the Instruction Counter, then this transfer of control is effective immediately; and the second instruction of the pair is not executed.

3. After the execution of the two instructions obtained from Y-pair, the next instruction to be executed is obtained from C(IC) +1. This is the instruction stored in memory right after this XED instruction unless the contents of the Instruction Counter have been changed by the execution of the two instructions obtained from the memory locations Y-pair.

4. To Execute Double (XED) a pair which has Repeat Double (RPD) as the odd instruction of the pair, XED must be located at the odd address. Note that the instructions that are repeated are those which immediately follow the XEC instruction.

5. If RPD is specified anywhere within a sequence of XED's, the original and all subsequent XED's in the sequence must be in odd locations.

CPB-1004F

131

Mnemonic:	Name of the Instruction:	Op Code (Octal)
MME	Master Mode Entry	001

SUMMARY: Causes a fault which obtains and executes, in the Master Mode, the two instructions stored at the memory locations 4 + C and 5 + C (decimal)

MODIFICATIONS: Are without any effect on the operation.

INDICATORS: (Indicators not listed are not affected)

	The MME instruction itself does not affect any indicator. However, the execution of the two instructions from 4 + C and 5 + C may affect indicators; particularly, each one in turn will affect the Master Mode Indicator as follows:
Master Mode	If the instruction obtained actually results in a transfer of control and is not the TSS instruction, then ON If the instruction obtained is either the RET instruction with bit 28 of the RET operand = ZERO or the TSS instruction, then OFF

NOTES:
1. The value of the constant C is set up in the FAULT switches.

2. During the execution of this MME instruction and the two instructions obtained, the Processor is in the Master Mode, independent of the value of its Master Indicator. The Processor will stay in the Master Mode if the Master Indicator is set ON after the execution of these three instructions.

3. The instruction from 4 + C MUST NOT alter the memory location 5 + C, and MUST NOT be an XED instruction.

4. If the instruction from 4 + C alters the contents of the Instruction Counter, then this transfer of control is effective immediately; and the instruction from 5 + C is not executed.

5. After the execution of the two instructions obtained from Y-pair, the next instruction to be executed is obtained from C(IC) + 1. This is the instruction stored in memory right after this MME instruction unless the contents of the Instruction Counter have been changed by the execution of the two instructions obtained from 4 + C and 5 + C.

CPB-1004F

132

Mnemonic:	Name of the Instruction:	Op Code (Octal)
DRL	Derail	002

SUMMARY: Causes a fault which obtains and executes in the Master Mode the two instructions stored at the memory locations 12 + C and 13 + C (decimal)

MODIFICATIONS: Are without any effect on the operation

INDICATORS: (Indicators not listed are not affected)

	The DRL instruction itself does not affect any indicator. However, the execution of the two instructions from 12 + C and 13 + C may affect indicators; particularly, each one in turn will affect the Master Mode Indicator as follows:
Master Mode	If the instruction obtained actually results in a transfer of control and is not the TSS instruction, then ON If the instruction obtained is either the RET instruction with bit 28 of the RET operand = ZERO or the TSS instruction, then OFF

NOTES:

1. The value of the constant C is set up in the FAULT switches.

2. During the execution of this DRL instruction and the two instructions obtained, the Processor is in the Master Mode, independent of the value of its Master Indicator. The Processor will stay in the Master Mode, if the Master Indicator is ON after the execution of these three instructions.

3. The instruction from 12 + C MUST NOT alter the memory location 13 + C, and MUST NOT be an XED instruction.

4. If the instruction from 12 + C alters the contents of the Instruction Counter, then this transfer of control is effective immediately; and the instruction from 13 + C is not executed.

5. After the execution of the two instructions obtained from Y-pair, the next instruction to be executed is obtained from C(IC) + 1. This is the instruction stored in the memory right after this DRL instruction unless the contents of the Instruction Counter have been changed by the execution of the two instructions obtained from 12 + C and 13 + C.

CPB-1004F

Mnemonic:	Name of the Instruction:	Op Code (Octal)
RPT	Repeat (See page 207 for coding format)	520

SUMMARY: Execute the next instruction until an exit condition is met.

MODIFICATIONS: No modifiers are allowed.

INDICATORS: The RPT instruction itself does not affect any of the indicators; however, the execution of the repeated instruction may affect indicators.

NOTES: 1.   The RPT instruction has the following format:

0	7 8 9 10 11	17 18	26 27 28 29 30	35

Tally	$\sim$	$\sim$	C	Term. Cond.	Op Code	0	1	0	Delta

2.   If C = 1, then bits 0-17 of the RPT instruction $\Rightarrow$ X0.

3.   In the normal case, the Terminate Condition(s) and Tally from X0 control the repetition loop for the instruction following the RPT instruction.   Initial Tally = 0 is interpreted as 256.   A fault also causes an exit from the loop.

4.   The repetition loop which does _not_ contain a fault consists of the following steps:

   (a)   Execute the repeated instruction

   (b) $C(X0)_{0...7} -1 \Rightarrow C(X0)_{0...7}$

   (c)   If a Termination Condition is met (see 7b), then set Tally Runout Indicator OFF and exit.

   (d)   If $C(X0)_{0...7} = 0$ and no Terminate Condition is met, then set Tally Runout Indicator ON and exit.

   (e)   Go to (a) if (c) or (d) conditions are not met.

5.   The instructions which _cannot_ be repeated are:

   (a)   All transfer-of-control instructions

   (b)   All miscellaneous instructions except _BCD_ and _GTB_, which _are_ _permitted._

   (c)   The instructions STCA, STCQ, STBA, STBQ, SREG, LREG, DIS, CIOC,

6.   Address modification for the repeated instruction:

   For the repeated instruction, only the modifiers R and RI and only the designators specifying X1, ..., X7 are permitted.

CPB-1004F

The effective address Y (in the case of R) or the address yI of the indirect word to be referenced (in the case of RI) is:

(a) For the _first execution_ of the repeated instruction

$$y + C(R) = Y_1 \text{ or } yI_1; \; Y_1 \text{ or } yI_1 \Rightarrow C(R)$$

(b) For _any successive execution_

$$\text{Delta} + C(R) = Y_n \text{ or } yI_n \; ; \; Y_n \text{ or } yI_n \Rightarrow C(R), \text{ where } n > 1$$

In the case of RI, only one indirect reference is made per repeated execution. The Tag portion of the indirect word is not interpreted as usual but is ignored. Instead the modifier R and the designator R = N are applied.

7. The Exit Conditions:

An exit is made from the repeat loop if one of the Terminate Conditions exists or if Tally = 0 after the execution of the repeated instruction. Also, an exit is made any time a fault occurs.

The program-controlled exit conditions are:

(a) Tally = 0

(b) Terminate Conditions:

The bit configuration in bit positions 11 - 17 of the RPT instruction defines the Terminate Conditions. If more than one condition is specified, the repeat terminates if any one of them is met.

The Carry, Negative, and Zero Indicators each use 2 bits, one for the OFF condition and one for ON. A zero in both positions for one indicator causes this indicator to be ignored as a Termination Condition. A one in both positions causes an exit after the first execution of the repeated instruction.

Bit 17 = 0: any overflow is completely ignored, i.e., the respective Overflow Indicator is _not_ set ON, and an Overflow Trap does _not_ occur.

Bit 17 = 1: any overflow is treated as usual. If the Overflow Mask is ON, then exit from the repetition loop.

Bit 16 = 1: if Carry Indicator is OFF, then exit.

Bit 15 = 1: if Carry Indicator is ON, then exit.

Bit 14 = 1: if Negative Indicator is OFF, then exit.

Bit 13 = 1: if Negative Indicator is ON, then exit.

Bit 12 = 1: if Zero Indicator is OFF, then exit.

Bit 11 = 1: if Zero Indicator is ON, then exit.

(c) Overflow Fault Trap:

If bit 17 = 1 and an overflow occurs with the Overflow Mask OFF, an Overflow Fault Trap occurs and an exit is made from the repetition loop upon completion of the fault instruction.

A nonprogram-controlled exit from the repetition loop occurs if any Fault Trap other than Overflow occurs (i.e., Divide Check, Parity Error on indirect word or operand fetch, etc.).

8. At the time of exit from the repetition loops:

$X0_{0...7}$ contains the Tally Residue, i.e., the number of repeats remaining until a Tally Runout would have occurred. The Terminate Conditions in bits 11 - 17 remain unchanged.

If the exit was due to Tally = 0 or a Terminate Condition, the X_n specified by the designator of the repeated instruction will contain the contents of the designated Xn after the last execution plus delta.

If the exit was due to a Fault Trap, the Xn specified by the designator of the repeated instruction may contain either:

(a) The contents of the designated Xn at the time the Fault Trap occurred, or

(b) The contents of the designated Xn at the time the Fault Trap occurred plus delta.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
RPD	Repeat Double (See page 207 for coding format)	560

SUMMARY: The instructions from the next Y-pair are fetched and saved in the processor and are executed repeatedly until an exit condition is met.

MODIFICATIONS: No modifiers are allowed.

INDICATORS: The RPD instruction itself does not affect any of the indicators. However, the execution of the repeated instructions may affect indicators.

NOTES:

1. The RPD instruction must be stored in an odd memory location except when accessed via the XEC instruction. In this case, the RPD instruction can be either even or odd, but the XEC instruction must be in an odd location.

2. The RPD instruction has the following format:

0	7 8 9 10 11	17 18	26 27 28 29 30	35

Tally	A	B	C	Term. Cond.	Op Code	0	1	0	Delta
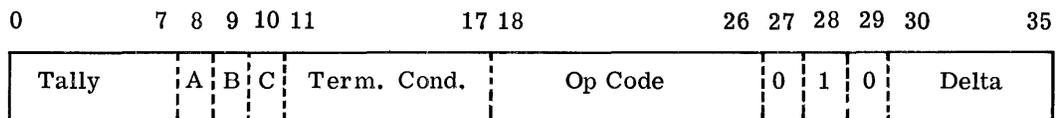
3. If C = 1, then bits 0-17 of the RPD instruction $\Rightarrow$ X0.

4. In the normal case, the Terminate Condition(s) and Tally from X0 control the repetition loop for the instructions following the RPD instruction. Initial Tally = 0 will be interpreted as 256. A fault also causes an exit from the loop.

5. The repetition loop which does not contain a fault consists of the following steps:

   (a) Execute the pair of repeated instructions

   (b) $C(X0)_{0...7} - 1 \Rightarrow C(X0)_{0...7}$

   (c) If a Termination Condition is met (see 8b), then set the Tally Runout Indicator OFF and exit.

   (d) If $C(X0)_{0...7} = 0$ and no Terminate Condition is met, then set Tally Runout Indicator ON and exit.

   (e) Go to (a) if conditions (c) or (d) are not met.

6. The instructions which cannot be repeated are:

   (a) All transfer-of-control instructions

   (b) All miscellaneous instructions except BCD and GTB, which are permitted.

   (c) The instructions STCA, STCQ, STBA, STBQ, SREG, LREG, DIS, CIOC

CPB-1004F

7. Address modification for the pair of repeated instructions:

For each of the two repeated instructions, only the modifiers R and RI and only the designators specifying X1,..., X7 are permitted.

The effective address Y (in the case of R) or the address yI of the indirect word to be referenced (in the case of RI) is:

(a) For the <u>first</u> <u>execution</u> of <u>each</u> of the two repeated instructions

$$y + C(R) = Y_1 \text{ or } yI_1; \text{ Y1 or yI1 } \Rightarrow C(R)$$

(b) For <u>any</u> <u>successive</u> <u>execution</u> of

the <u>first</u> of the two repeated instructions
if A = 1, then Delta + C(R) = Y_n or yI_n ;
Y_n or $yI_n \Rightarrow C(R)$
if A = 0, then C(R) = Y_n or yI_n, where n > 1

the <u>second</u> of the two repeated instructions
if B = 1, then Delta + C(R) = Y_n or yI_n ;
Y_n or $yI_n \Rightarrow C(R)$
if B = 0, then C(R) = Y_n or yI_n, where n > 1

where A and B are the contents of bit positions 8 and 9 of Index Register Zero (XRO).

In the case of RI, only one indirect reference is made per repeated execution. The Tag portion of the indirect word is not interpreted as usual but is ignored. Instead, the modifier R and the designator R = N are applied.

8. The Exit Conditions:

An exit is made from the repeat loop if one of the Terminate Conditions exists or if Tally = 0 after the execution of the odd instruction of the repeated pair. Also, an exit is made any time a fault occurs.

The program-controlled exit conditions are:

(a) Tally = 0

(b) Terminate Conditions

The bit configuration in bit positions 11-17 of the RPD instruction defines the Terminate Conditions. If more than one condition is specified, the repeat terminates if any one of them is met.

The Carry, Negative, and Zero Indicators each use two bits, one for the OFF condition and one for ON. A zero in both positions for one indicator causes this indicator to be ignored as a Terminate Condition. A one in both positions causes an exit after the first execution of the repeated instruction pair.

CPB-1004F

138

Bit 17 = 0: any overflow is completely ignored, i. e., the respective Overflow indicator is not set ON, and an Overflow Trap does not occur.

Bit 17 = 1: any overflow is treated as usual. If the Overflow Mask is ON, then exit from the repetition loop.

Bit 16 = 1: if Carry Indicator is OFF, then exit.

Bit 15 = 1: if Carry Indicator is ON, then exit.

Bit 14 = 1: if Negative Indicator is OFF, then exit.

Bit 13 = 1: if Negative Indicator is ON, then exit.

Bit 12 = 1: if Zero Indicator is OFF, then exit.

Bit 11 = 1: if Zero Indicator is ON, then exit.

(c) Overflow Fault Trap

If bit 17 = 1 and an overflow occurs with the Overflow Mask OFF, an Overflow Fault Trap occurs and an exit is made from the repetition loop upon completion of the fault instruction.

A nonprogram-controlled exit from the repetition loop occurs if any Fault Trap other than an Overflow occurs. Note that if any Fault Trap (i. e., Overflow, Divide Check, Parity Error on indirect word or operand fetch, etc.) occurs on the even instruction, the odd instruction will not be executed.

9. At the time of exit from the repetition loop:

$X0_{0...7}$ contains the Tally Residue, i.e., the number of repeats remaining until a Tally Runout would have occurred. The Terminate Conditions in bits 11-17 remain unchanged.

If the exit was due to Tally = 0 or a Terminate Condition, the X_n specified by the designator of each of the two repeated instructions will contain either:

(a) The contents of the designated X_n after the last execution of the repeated pair plus the delta associated with each instruction A and B (bits 8 and 9 of X_0) = 1, or

(b) The contents of the designated X_n after the last execution of the repeated pair of Z and B = 0.

If the exit was due to a Fault Trap, the X_n specified by the designator of each of the two repeated instructions may contain either:

(a) The contents of the designated X_n's at the time the Fault Trap occurred plus the delta associated with each instruction A and B = 1, or

(b) The contents of the designated X_n's at the time the Fault Trap occurred.

CPB-1004F

(b) The contents of the designated X_n's at the time the Fault Trap occurred.

10. A repeat double of instructions that have long execution times may cause a LUF, if the time involved is greater than 16 milliseconds.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
RPL	Repeat Link (See page 207 for coding format)	500

SUMMARY: Execute the next instruction until an exit condition is met.

MODIFICATIONS: No modifiers are allowed.

INDICATORS: The RPL instruction itself does not affect any of the indicators. However, the execution of the repeated instructions may affect the indicators.

NOTES: 1. The RPL instruction has the following format:

0		7 8 9 10 11	17 18	26 27 28 29 30	35

Tally	$\sim$ $\sim$ C	Term. Cond.	Op Code	0 1 0	$\sim$

2. If C = 1, then bits 0-17 of the RPL instruction $\Rightarrow$ X0.

3. In the normal case, the Terminate Condition(s) and Tally from X0 control the repetition loop for the instruction following the RPL instruction. Initial Tally = 0 will be interpreted as 256. A fault also causes an exit from the loop.

4. The repetition loop that does <u>not</u> contain a fault consists of the following steps:

(a) Execute the repeated instruction

(b) $C(X0)_{0\ldots7} - 1 \Rightarrow (CX0)_{0\ldots7}$

(c) If a Termination Condition is met (see 7c), then set Tally Runout Indicator OFF and exit.

(d) If the Tally $C(X0)_{0\ldots7} = 0$ or the Link Address $C(Y)_{0\ldots17} = 0$ and no Termination Condition is met, then set Tally Runout Indicator ON and exit.

(e) Go to (a) if conditions (c) or (d) are not met.

5. The instructions which <u>cannot</u> be repeated are:

(a) Instructions that could alter the Link Address of a linked word.

Example -

1. LDX_n type instruction with the same X_n specified as is specified in a modifying register

2. STORES or READ-ALTER-REWRITE's

(b) The instructions EAA, EAQ, EAXn, NEG, NEGL, FNO, FNEG, LREG, DIS, CIOC.

CPB-1004F

141

(c)   All miscellaneous operations instructions.

(d)   All transfer-of-control instructions.

(e)   All shift instructions.

> Note: All instructions that would normally alter the
> contents of an index register (except (LXL_n))
> result in the specified register either being
> cleared to zero or remaining unchanged when
> in the RPL mode. This is because bits 0-17
> of the operand are zero.

6.   Address modification for the repeated instruction:

For the repeated instruction, only the modifier R and the
designators specifying $R = X1, \ldots, X7$ are permitted. The
modifier is effective only for the first execution of the re-
peated instruction.

The effective address Y is:

(a)   For the <u>first execution</u> of the repeated instruction

$$Y_1 = y_1 + C(R); \quad Y_1 \Rightarrow C(R)$$

(b)   For <u>any successive execution</u> of the repeated instruction

$$Y_2 = C(Y_1)_{0\ldots17}; \quad Y_2 \Rightarrow C(R)$$
.
.
.
$$Y_n = C(Y_{n-1})_{0\ldots17}; \quad Y_n \Rightarrow C(R) \text{ if } Y_{n\,(0\ldots17)} \neq 0$$

The effective address Y is the address of the next list word.
The lower portion of the list word contains the operand to be
used for this execution of the repeated instruction. The oper-
and is:

$$\underbrace{00\ldots0,}_{\text{bits 0-17}} \begin{cases} C(Y)_{18\ldots35} \text{ for single precision} \\ C(Y)_{18\ldots71} \text{ for double precision} \end{cases}$$

The upper 18 bits of the list word contain the Link Address,
i. e., the address of the next successive list word, and thus the
effective address for the next successive execution of the
repeated instruction.

7.   The Exit Conditions:

An exit is made from the repeat loop if one of the Terminate
Conditions exists or if Tally = 0 or Link Address = 0 after the
execution of the instruction being repeated. Also an exit is
made any time a fault occurs.

The program-controlled exit conditions are:

(a) Tally = 0

(b) Link Address = 0

(c) Terminate Conditions

The bit configuration in bit positions 11-17 of the RPL instruction defines the Terminate Conditions. If more than one condition is specified, the repeat terminates if any one of them is met.

The Carry, Negative, and Zero Indicators each use two bits, one for the OFF condition and one for ON. A zero in both positions for one indicator causes this indicator to be ignored as a Termination Condition. A one in both positions causes an exit after the first execution of the repeated instruction.

Bit 17 = 0: any overflow is completely ignored, i.e., the respective Overflow Indicator is not set ON and the Overflow Trap does not occur.

Bit 17 = 1: any overflow is treated as usual. If the Overflow Mask is ON, then exit from the repetition loop.

Bit 16 = 1: if Carry Indicator is OFF, then exit.

Bit 15 = 1: if Carry Indicator is ON, then exit.

Bit 14 = 1: if Negative Indicator is OFF, then exit.

Bit 13 = 1: if Negative Indicator is ON, then exit.

Bit 12 = 1: if Zero Indicator is OFF, then exit.

Bit 11 = 1: if Zero Indicator is ON, then exit.

(d) Overflow Fault Trap

If bit 17 = 1 and an overflow occurs with the Overflow Mask OFF, an Overflow Fault Trap occurs and an exit is made from the repetition loop upon completion of the fault instructions.

A nonprogram-controlled exit from the repetition loop occurs if any Fault Trap other than Overflow occurs (i.e., Divide Check, Parity Error on indirect word or operand fetch, etc.).

8. At the time of exit from the repetition loop:

$X0_{0...7}$ contains the Tally Residue, i.e., the numbers of repeats remaining until a Tally Runout would have occurred. The Terminate Conditions in bits 11-17 remain unchanged.

The X_n specified by the designator of the repeated instruction
contains the address of the list word that contains

    in its lower half: the operand used in the last execution of
                        the repeated instruction

    in its upper half: the address of the next list word

9. An exit will <u>not</u> occur if the effective address = 0 for the first
     execution of the linked instruction. This address specifies
     the location of the first word in the Link Table and is not
     interpreted as a Link Address.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
DIS	Delay Until Interrupt Signal	616

SUMMARY: No operation takes place, and the Processor does not continue with the next instruction but waits for a program interrupt signal

MODIFICATIONS: Are without any effect on the operation

INDICATORS: None affected

NOTE: This instruction can be used in the Master Mode only. If this instruction is attempted by a Processor that is in the Slave Mode, a Command Fault Trap occurs.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LBAR	Load Base Address Register	230

SUMMARY: $C(Y)_{0...17} \Rightarrow C(BR)$

MODIFICATIONS: All except CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(BR) = 0, then ON; otherwise OFF
Negative	If $C(BR)_0$ = 1, then ON; otherwise OFF

NOTE: This instruction can be used in the Master Mode only. If its use is attempted in the Slave Mode, the instruction functions like the NOP instruction.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
LDT	Load Timer Register	637

SUMMARY: $C(Y)_{0...23} \Rightarrow C(TR)$

MODIFICATIONS: All except CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(TR) = 0, then ON; otherwise OFF
Negative	If $C(TR)_0$ = 1, then ON; otherwise OFF

NOTE: This instruction can be used in the Master Mode only. If its use is attempted in the Slave Mode, the instruction functions like the NOP instruction.

CPB-1004F

145

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SMIC	Set Memory Controller Interrupt Cells	451

SUMMARY: C(A) is used to set selected Interrupt Cells ON in the System Controller of the Memory unit selected by Y_{0-2}

MODIFICATIONS: All except DU, DL, SC, and CI

INDICATORS: None affected

NOTES:
1. The effective address Y is used in selecting a Memory module as with a normal memory access request. However, the selected module does not store the data received in a memory location, but uses it to set selected Interrupt Cells ON.

For i = 0, 1,....,15 AND $C(A)_{35} = 0$:

$\overline{\text{if } C(A)_i = 1, \text{ then set Interrupt Cell i ON}}$

For i = 0, 1,....,15 AND $C(A)_{35} = 1$:

$\overline{\text{if } C(A)_i = 1, \text{ then set Interrupt Cell (16+i) ON.}}$

2. This instruction can be used in the Master Mode only. If the use of this instruction is attempted by a Processor that is in the Slave Mode, a Command Fault Trap will occur.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
RMCM	Read Memory Controller Mask Register	233

SUMMARY: C (Memory Controller Interrupt Mask Register) $\left.\begin{array}{l} \\ \\ \\ \end{array}\right\} >$ C(AQ)
C (Memory Controller Access Mask Register)
of Memory Unit specified by Y_{0-2}

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: (Indicators not listed are not affected)

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If $C(AQ)_0 = 1$, then ON; otherwise OFF

NOTES:
1. The effective address Y is used in selecting a Memory module as with a normal memory access request. However, the selected module does not transmit the contents of an addressed memory location, but the contents of its Memory Controller Interrupt Mask Register and Memory Controller Access Mask Register.

CPB-1004F

146
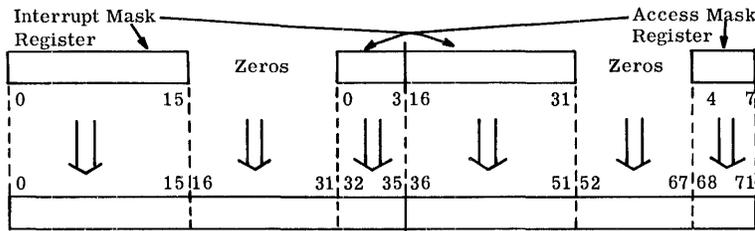
Combined AQ-register

> 2. This instruction can be used in the Master Mode only. If the use of this instruction is attempted by a Processor that is in the Slave Mode, a Command Fault Trap will occur.

Mnemonic:	Name of the Instruction:	Op Code (Octal)
SMCM	Set Memory Controller Mask Register	553

SUMMARY:

$$C(AQ) \Rightarrow \begin{cases} C \text{ (Memory Controller Interrupt Mask Register)} \\ C \text{ (Memory Controller Access Mask Register)} \\ \text{of Memory Unit specified by } Y_{0-2} \end{cases}$$

MODIFICATIONS: All except DU, DL, CI, SC

INDICATORS: None affected

NOTE:

> 1. The effective address Y is used in selecting a Memory module as with a normal memory access request. However, the selected module does not store the data received in a memory location but in its Memory Controller Interrupt Mask Register and Memory Controller Access Mask Register.

Combined AQ-register
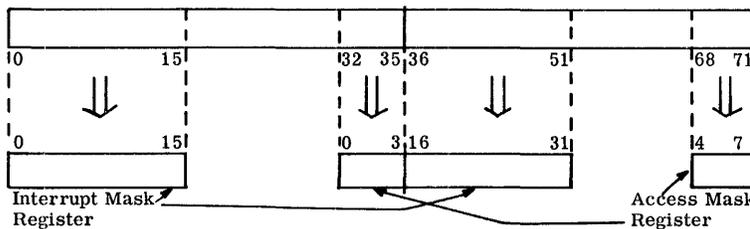
> 2. This instruction can be used in the Master Mode only. If the use of this instruction is attempted by a Processor that is in the Slave Mode, a Command Fault Trap will occur.

CPB-1004F

147

Mnemonic:	Name of the Instruction:	Op Code (Octal)
CIOC	Connect I/O Channel	015

SUMMARY: C(Y) are transferred from the Memory module via the channel that is specified by C(Y)

MODIFICATIONS: All except DU, DL, SC, and CI

INDICATORS: None affected

NOTES:

1. The effective address Y is used to access a memory location as usual. However, the Memory module does not transmit the contents of this location to the Processor that submitted the effective address; it uses $C(Y)_{33...35}$ to select one of its eight channels, sends a connect pulse to the unit on this channel, and transmits C(Y) on the data lines to this unit

2. This instruction can be used in the Master Mode only. If the use of this instruction is attempted by a Processor that is in the Slave Mode, a Command Fault Trap will occur.

CPB-1004F

III. SYMBOLIC MACRO ASSEMBLER--GMAP

GENERAL DESCRIPTION

The GE-625/635 macro assembly program is a program which will translate symbolic machine language convenient for programmer use into absolute or relocatable binary machine instructions. The symbolic language is sufficiently like machine language to permit the programmer to utilize all the facilities of the computer which would be available to him if he were to code directly in machine language.

An Assembler resembles a compiler in that it produces machine language programs. It differs from a compiler in that the symbolic language used with an Assembler is closely related to the language used by the computer, while the source language used with a compiler resembles the technical language in which problems are stated by human beings.

Compilers have several advantages over Assemblers. The language used with the compiler is easier to learn and is oriented toward the problem to be solved. The user of a compiler usually does not need an intimate knowledge of the inner workings of the computer. Programming is faster. Finally, the time required to obtain a finished, working program is greatly reduced since there is less chance for the programmer to make mistakes. The Assembler compensates for its disadvantages by offering those programmers, who need a great degree of flexibility in writing their programs, that flexibility which is not currently found in compilers.

The GE-625/635 Macro Assembler is being provided to give the professional programmers some of the conveniences of a compiler and the flexibility of an Assembler. The ability to design desired MACROs in order to provide convenient shorthand notations plus the use of all GE-625/635 machine instructions as well as a complete set of pseudo-operations provides the programmer with a very powerful and flexible tool. The output options enable him to obtain binary text in relocatable as well as absolute formats.

This Assembler is implemented in the classic format of Macro Assemblers with several variations.   There are two passes over the external text: the first pass allows for updating and/or merging of an ALTER package to a previously prepared assembly input. The ALTER package consists of changes to be made to the previous assembly under control of ALTER cards. During pass one, all symbols are collected and assigned their absolute or relocatable values relative to the current location counter.   MACRO prototypes are processed and placed in the MACRO skeleton table immediately ready for expansion. All MACRO calls, therefore, are expanded in pass one, allowing the MACRO skeleton table to be destroyed prior to pass two.

Machine operation codes, pseudo-operations, and MACRO names are all carried in the operation table during pass one.

This implies that all operation codes, machine or pseudo, along with MACROs are looked up during pass one, and that the general operation table is destroyed at the end of pass one. The literal pool is completely expanded during pass one, avoiding duplicates (except for V, M, and nH literals where n is greater than 12), which are assigned unique locations in pass one and will be later expanded in pass two. Double-precision numbers in the literal pool start at even locations.

At the end of pass one, the symbol table is sorted; and a complete readjustment of symbols by their relative location counter is performed. The preface card is then punched.

All instructions are generated during pass two. This is accomplished by performing a scan over the variable fields and address modifications. This information is then combined with the operation code from pass one by using a Boolean OR function. Apparent errors are flagged.

The symbolic cross-reference table is created as the variable fields are scanned and expanded. The final edit of the symbol table is done at the end of pass two. Generative pseudo-operations are processed with the conversion being done in pass two. Pseudo-operations are available to control punching of binary cards and printing images of source cards. Images of source cards in error will be printed, regardless of control pseudo-operations. Undefined symbols, and error conditions will be noted at the end of the printer listing.

The classic format of a variable field symbolic assembly program is used throughout the GE-625/635 Macro Assembler. Typically, a symbolic instruction consists of four major divisions; location field, operation field, variable field, and comments field.

The location field normally contains a name by which other instructions may refer to the instruction named. The operation field contains the name of the machine operation, pseudo-operation or Macro. The variable field normally contains the location of the operand. The comments field exists solely for the convenience of the programmer and plays no part in the assembly process. An identification field is provided to give a means of identifying the location of a card within a deck.

RELOCATABLE AND ABSOLUTE ASSEMBLIES

The Macro Assembler program processes inputs of several types: (1) FORTRAN IV compilations that have been translated into the Assembler language, (2) COBOL-61 compilations translated into the Assembler language, (3) source programs written originally in the Assembler language, (4) compressed source decks (COMDK) for any of items (1) through (3), and (5) correction (ALTER) cards for any of (1) through (3).

The normal operating mode of the Assembler in processing input subprograms of the types indicated above is relocatable; that is, each subprogram in a job stream is handled individually and is assigned memory locations nominally beginning with zero and extending to the upper limit required for that subprogram. Since a job stream can contain many such subprograms, it is apparent that they cannot all be loaded into a memory area starting with location zero; they must be loaded into different memory areas. Furthermore, they must be movable (relocatable) among the areas. Then for relocatable subprograms, the Assembler must provide (1) delimiters identifying each subprogram, (2) information specifying that the subprogram is relocatable, (3) the length of the subprogram, and (4) relocation control bits for both the upper and lower 18 bits of each assembled word.

Subprogram delimiters are the Assembler output cards $ OBJECT, heading the subprogram assembly, and $ DKEND, ending the assembly. An assembly is designated as relocatable on a card-to-card basis by a unique 3-bit Assembler punched code value in each binary output card. (See Binary Decks, page 228.) The subprogram length is punched in the preface card(s) which immediately follows the $ OBJECT card of each subprogram. The relocation control bits are grouped together on the binary card and are referenced by GELOAD while it is loading the subprogram into absolute memory locations.

The Assembler designates that the assembly output is absolute on a card-to-card basis by punching a unique 3-bit code value in each card. This value causes GELOAD to regard all addresses on a card as actual (physical) memory address relative to the Base Address Register and to load accordingly. Each absolute subprogram assembly begins with a $ OBJECT card and terminates with the $ DKEND card, as in the case of relocatable assemblies.

The normal Assembler operating mode is relocatable; it is set to the absolute mode by programmer use of ABS (page 183).

ASSEMBLY LANGUAGE PROGRAMMING

Location Field

In machine instruction or MACROs this location may contain a symbol or may be left blank, if no reference is made to the instruction. (With certain pseudo-operations, this field has a special use and is described later in this publication.) Associated with the location field is a one-character field which allows the programmer to specify whether this generated machine word should fall in a special memory location. If this is left blank, then the instruction will be located in the next available location. But, if there is an O in this field, the instruction will be located at the next available odd location; if an E, then at the next available even location; if the number 8, then in the next location which is a multiple of eight.

Operation Field

The operation field may contain from zero to six characters taken from the set 0-9, A-Z, and the period (.). The group of characters must be: (1) a legal GE-625/635 operation,* (2) a Macro Assembler pseudo-operation or (3) programmer macro operation code. The character group must begin in column eight (left-justified) and must be followed by at least one blank.

A blank field or the special code ARG will be interpreted as a zero operation, the operation field will be all zeros in the assembly coding. Anything appearing in the operation field which is not in (1), (2), or (3) above is an "illegal" operation and will result in an error flag in the assembly listing.

---

*All indexing instructions (LDX, STX, ADX, etc.) may be used without the index register number appended. In this case there are three subfields in the variable field. The first subfield is an expression which when evaluated will designate the proper index register. Thus,

```
    LDX          1, 5, DU
is equivalent to
    LDX1         5, DU
also, the following is admissible:
    LDX          B+A, 5, DU
```

Variable Field

The variable field contains one or more subfields that are separated by the programmer through the use of commas placed between subfields. The number and type of subfields vary depending upon the content of the operation field: (1) machine instruction, (2) Macro Assembler pseudo-operation, or (3) macro operation.

The subfields within the variable field of GE-625/635 machine instructions consist of the address and the tag (modifier). The address may be any legitimate expression or a literal. This is the first subfield of the variable field and is separated from the tag by a comma. Through address modification, as directed by the tag, a program address is defined. This program address is either (1) an instruction address used for fetching instructions, (2) a tentative address used for fetching an indirect word, or (3) an effective address used for obtaining an operand or storing a result.

The subfields used with pseudo-operations vary considerably; they are described individually in this publication under each pseudo-operation. Subfields used with macro operations are substitutable arguments which, in themselves, may be instructions, operand addresses, modifer tags, pseudo-operations, or other macro operations. All of these types of subfields are presented in the discussion on macro operations.

The first character of the variable field must begin by column 16. The end of the variable field is designated by the first blank character encountered in the variable field (except for the BCI instruction and in the use of Hollerith literals). If any subfield is null (no entry given when one is needed), it is interpreted to be zero.

Comments Field

The comments field exists solely for the convenience of the programmer; it plays no part in the assembly process. Programmer comments follow the variable field and are separated from that field by at least one blank column.

Identification Field

This field is used or not used according to programmer option. Its intended use is for instruction identification and sequencing.

Symbolic Card Format

Symbolic instructions are punched one per card, each card representing one line of the coding sheet (Figure 6). The following is a breakdown of the card columns normally used.

Columns	1 - 6	Location field
Column	7	Even/odd/eight subfield
Columns	8 - 13	Operation field (left justified)
Column	14 - 15	Blank
Columns	16 - Blank*	Variable field
Column Blank - 72		Comments field (separated from variable field by at least one blank)
Columns	73 - 80	Identification field

When columns 1-16 are all blank, the symbolic card is treated as a remarks card. The first blank column encountered within an expression terminates the processing of the variable field.

Figure 6. GE-625/635 Macro Assembler Coding Form

Symbols

A symbol is a string of from one to six nonblank characters, at least one of which is non-numeric, and the first of which is non-zero. The characters must be taken from the set made up of 0-9, A-Z and the period (.). Symbols can appear in the location and variable fields of the Assembler coding form. (Symbols are also known as location symbols and symbolic addresses.)

Symbols are defined by:

1. Their appearance in the location field of an instruction, pseudo-operation, or MACRO.

2. Their use as the name of a subprogram in a CALL pseudo-operation.

3. Their appearance in the SYMREF pseudo-operation.

Every symbol used in a program must be defined exactly once, except for those symbols which are initially defined and redefined by the SET pseudo-operation. An error will be indicated by the Assembler if any symbol is referenced but never defined, or if any symbol is defined more than once.

The following are examples of permissible symbols:

A	A1000	E1XP3	A.....
Z	FIRST	.XP3	B.707
B 1	ALOG10	ADDTO	1234X
ERR	BEGIN	ERROR	3.141P

Types of Symbols

Symbols are classified into four types:

1. Absolute--A symbol which refers to a specific number.

2. Common--A symbol which refers to a location in common storage. These locations are defined by the use of the BLOCK pseudo-operation.

3. Relocatable--A symbol which appears in the location field of an instruction. Symbols that appear in the location field of symbol defining pseudo-operations are defined as the same type as the symbol in the variable field.

4. SYMREF--A symbol which appears in the variable field of a SYMREF pseudo-operation; it is considered to be defined external to the subprogram being assembled and is to be considered specially by the Loader.

Expressions In General

In writing symbolic instructions, the use of symbols only in the allowable subfields presents the programmer with too restrictive a language. Therefore, in the notation of subfields of machine instructions and in the variable fields of pseudo-operations (and by following specific rules), the use of expressions as well as symbols is permitted. Before discussing expressions, it is necessary to describe the building blocks used to construct them. These building blocks are elements, terms, and operators.

Elements

The smallest component of a complete expression is an element. An element consists of a single symbol or an integer less than 2^{35}.(The asterisk may also be used as an element.)

Terms and Operators

A term is a string composed of elements and operators. It may consist of one element or, generally speaking, n elements separated by n - 1 operators of the type * and / where * indicates multiplication and / indicates division. If a term does not begin with an element or end with an element, then a null element will be assumed. It is not permissible to write two operators in succession or to write two elements in succession.

Examples of terms are:

M	MAN*T	7*Y
436	BETA/3	A*B*C/X*Y*Z
START	4*AB/ROOT	ONE*TWO/THREE

Asterisk Used as an Element

An asterisk (*) may be used as an element in addition to being used as an operator. When it is used as an element, it refers to the location of the instruction in which it appears. For example, the instruction

	A10	TRA	*+2

is equivalent to

	A10	TRA	A10+2

and represents a transfer to the second location following the transfer instruction. There is no ambiguity between this usage of the asterisk as an element and its use as the operator for multiplication since the position of the asterisk always makes clear what is meant. Thus, **M means "the location of this instruction multiplied by the element M", and the ** means "the location of this instruction times the null element" and would be equal to zero. The notation *-* means "the location of this instruction minus the location of this instruction." (See description of + and - operators below.)

Algebraic Expressions

An algebraic expression is a string composed of terms separated by the operators + (addition) and - (subtraction). Therefore, an expression may consist of one term or, more generally speaking, n terms separated by n - 1 operators of the type + and -. It is permissible to write two operators, plus and minus, in succession and the Assembler will assume a null element between the two operators. If no initial term or final term is stated, it will be assumed to be zero, except when the divisor is zero, in which case the divisor is assumed to be 1. An expression may begin with the operator plus or minus but if not explicitly given + will be assumed. Examples of permissible algebraic expressions are:

A	B+4		CX*DY+EX/FY-100
SINE	7		-EXP*FUNC/LOGX+XYZ/10-SINE
XYZ	+99	-X/Y	*+5*X (Note: the first asterisk refers to the instruction location)
A-3	-88	X*Y	--(Note: equivalent to zero minus zero minus zero)

Evaluation of Algebraic Expressions

An algebriac expression is evaluated as follows: first, each symbolic element is replaced by its numerically-defined value; then, each term is computed from left-to-right in the order of its occurrence. In division, the integral part of the quotient is retained; the remainder is immediately discarded. For example, the value of the term 7/3 * 3 is 6. In the evaluation of an expression, division by zero is equivalent to division by one and is not regarded as an error. After the evaluation of terms, they are combined in a left-to-right order with the initial term of the expression assumed to be zero followed by a plus operator. If there is no final term, a null term will be used. At the completion of the expression evaluation, the Assembler reduces the result by modulo 2^n where n is the number of binary bits in the field being defined, 18 for address field evaluations and variable according to specified field size for the VFD pseudo-operation, (page 199). Grouping by parentheses is not permitted, but this restriction may often be circumvented.

Boolean Expressions

A Boolean expression is defined similarly to an algebraic expression except that the operators *, /, +, or - are interpreted as Boolean operators. The meaning of these operators is defined below:

1. The expression that appears in the variable field of a BOOL pseudo-operation uses Boolean operators.

2. The expression that appears in the octal subfield of the variable field of a VFD pseudo-operation uses Boolean operators.

Evaluation of Boolean Expressions

A Boolean expression is evaluated following the same procedure used for an algebraic expression except that the operators are interpreted as Boolean.

In a Boolean expression, the form operators +, -, *, and / have Boolean meanings, rather than their normal arithmetic meanings, as follows:

Operator	Meaning	Definition
+	OR, INCLUSIVE OR, union	0 + 0 = 0 0 + 1 = 1 1 + 0 = 1 1 + 1 = 1
-	EXCLUSIVE OR symmetric difference	0 - 0 = 0 0 - 1 = 1 1 - 0 = 1 1 - 1 = 0
*	AND, intersection	0 * 0 = 0 0 * 1 = 0 1 * 0 = 0 1 * 1 = 1
/	1's complement, complement, NOT	/0 = 1 /1 = 0

Although / is a unary operation involving only one term, by convention A/B is taken to mean A*/B; and the A is ignored. This is not regarded as an error by the Assembler. Thus, the table for / as a two-term operation is:

$$0/0 = 0 \qquad 1/0 = 1$$
$$0/1 = 0 \qquad 1/1 = 0$$

other conventions are:

$$+A = A+ = A$$
$$-A = A- = A$$
$$*A = A* = 0 \qquad \text{(possible error--operand missing)}$$
$$A/ = A/0 = A$$

156

Relocatable and Absolute Expressions

Expression evaluation can result in either relocatable or absolute values. There are three types of relocatable expressions; program relocatable (R), BLANK COMMON relocatable (C), and LABELED COMMON relocatable (L). The rules by which the Assembler determines the relocation validity of an expression are of necessity a little complex, and the presence of multiple location counters compounds the problem somewhat. Certain of the principal pseudo-operations impose restriction as to type of expression that is permissible; these are described separately under each of the affected pseudo-operations. These are:

EQU	MAX	BFS	DUP
SET	BOOL	ORG	FEQU
MIN	BSS	BEGIN	

The following rules summarize the conditions and restrictions governing the admissibility of relocation:

1. Division involving a relocatable element(s) is not valid.

2. Multiplication of two relocatable elements is not valid.

3. The asterisk(*) symbol (implying current location counter) is a relocatable element.

4. When the result of the evaluation of an expression is an absolute element, the expression is absolute.

5. When the result of the evaluation of an expression is a relocatable element, the expression is relocatable.

6. When the result of the evaluation of an expression is the sum or difference of a relocatable element and an absolute element, the expression is relocatable.

7. When the result of the evaluation of an expression is the difference between two relocatable elements, the expression is absolute.

As the result of the evaluation of an expression:

1. The sum of two or more relocatable elements is not valid.

2. The product of an absolute element and a relocatable element is not valid.

3. A negative relocatable element is not valid.

4. The difference of two different types of relocatable elements is not valid.

These rules are not a complete set of determinants but do serve as a basis for establishing a method of defining relocation admissibility of an expression.

CPB-1004F

157

Let R_r denote a program-text relocatable element, R_c denote a BLANK COMMON element, and R_1 denote a LABELED COMMON element. Next, take any expression and process it as follows:

1. Replace all absolute elements with their respective values.

2. Replace any relocatable element with the proper R_i, where i = r, c, or l. This yields a resulting expression involving only numbers and the terms R_r, R_1, and R_c.

3. Discard all terms in which all elements are absolute.

4. Evaluate the resulting expression. If it is zero or numeric, the original expression is absolute; if it is explicitly R_r, R_c, or R_1, then the original exprsssion is normal relocatable. BLANK COMMON, relocatable, or LABELED COMMON reloctable, respectively.

5. If the resulting expression is not as given in 4 above, it is a relocation error and/or an invalid expression.

In the illustrative examples following, assume ALPHA and BETA to be normal relocatable elements (R_r), GAMMA and DELTA to be BLANK COMMON relocatable elements (R_c), and EPSILON and ZETA to be LABELED COMMON relocatable elements (R_1). Let N and K be absolutely equivalent to 5 and 8, respectively.

1. 4*ALPHA-7-4*BETA
   reduces to
   $4*R_r - 4*R_r = 0$,
   thus indicating a valid absolute expression.

2. N*ALPHA+8*GAMMA+21 - K*DELTA
   reduces to
   $5*R_r + 8*R_c - 8*R_c = 5R_r$,
   thus indicating an <u>invalid</u> expression.

3. EPSILON+N-ZETA
   reduces to
   $R_1 + 5 - R_1 = 5$,
   thus indicating a valid absolute expression.

4. ALPHA-GAMMA+DELTA+7
   reduces to
   $R_r - R_c + R_c = R_r$,
   thus indicating a valid relocatable expression.

Special Relocatable Expressions

Since all symbols defined as other than equal to some number (A EQU 4), are defined relative to some explicit or implied location counter (USE, BLOCK), and are subject to adjustment at the end of pass 1, they are considered to be relocatable in pass 1, even in an absolute assembly.

CPB-1004F

158

Thus, special action must be taken, if they are to be referenced and used in pass 1 by certain pseudo-operations--those which call for an expression evaluation for the determination of some count subfield, the result of which must be absolute. As an example, consider

```
BCI        3,HOLLERITH TEXT
DUP        5,2
```

Normally, the count fields in the above are nonvariant and there is no problem. Consider however

```
M      BCI    N,HOLLERITH TEXT
       DUP    N,M-1
```

The Assembler is equipped to handle expressions in these count fields, provided the result is absolute. But, since M in the above example is a location symbol, and its value relative to the origin of the USE is all that is known in pass 1, a relocation error would result. The solution to this problem is simply to define some symbol at the first available location of the counter in question. It has a value of zero relative to the origin of that counter and may be used as follows:

```
          USE    CTR
FIRST  NULL
             .
             .
             .
M      BCI    N, HOLLERITH TEXT
       DUP    N, M-FIRST-1
```

The result of this expression is now absolute, and truly represents the pass 1 value of the symbol M (less 1).

Literals

A literal in a subfield is defined as being the data to be operated on rather than an expression which points to a location containing the data.

A programmer must refer frequently to a memory location containing a program constant. For example, if the constant 2 is to be added to the accumulator, the number 2 must be somewhere in memory. Data generating pseudo-operations in the Macro Assembler enable the programmer to introduce data words and constants into his program; but often the introduction is more directly accomplished by the use of the literal that serves as the operand of a machine instruction. Thus, the literal is data itself.

The Assembler retains source program literals by means of a table called a literal pool. When a literal appears, the Assembler prepares a constant which is equivalent in value to the data in the literal subfield. This constant is then placed in the literal pool, providing an identical constant has not already been so entered. If the constant is placed in the literal pool, it is assigned an address; and this address then replaces the data in the literal subfield, the constant being retained in the pool. If the constant is already in the literal pool, the address of the identical constant replaces the data in the literal subfield.

The Assembler processes five types of literals: decimal, octal, alphanumeric, instruction, and variable field. The appearance of an equal sign (=) in columns 16 of the variable field[1] instructs the Assembler that the subfield immediately following is a literal. The instruction and variable-field literal are placed in the literal pool. Because they cannot be evaluated until pass two of the assembly, no attempt is made to check for duplicate entries into the pool. Literals on the CALL and TALLY pseudo-operation are restricted to decimal, octal, and alphanumeric where the character count is less than 13.

<u>Decimal Literals</u>

1. Integers
    A decimal integer is a signed or unsigned string of digits. It is differentiated from the other decimal types by the absence of a decimal point, the letter B, the letter E, and the letter D.

2. Single-Precision Floating-Point

    A floating-point number is distinguished by the presence of an E, a decimal point, or both. A floating-point number consists of two parts: a principal part and an exponent. The presence of the exponent is optional. The principal part is a signed or unsigned decimal number with a decimal point in any position of the number or with an assumed decimal point at the right-hand end of the number. If there is no exponent part, the decimal point may not be assumed, but must be present.

    The exponent part follows the principal part and consists of the letter E followed by a signed or unsigned decimal integer.

3. Double-Precision Floating-Point

    The format of the double-precision floating-point number is identical to the single-precision format with two exceptions:

    1. There must always be an exponent
    2. The letter E must be replaced by the letter D

The Assembler will ensure that all double-precision numbers begin in even memory locations. Ambiguity of storage assignment as to even or odd will always cause the Assembler to force double-precision word pairs to even locations; it will then issue a warning in the printout listing.

4. Fixed-Point

    A fixed-point quantity possesses the same characteristics as the floating-point-- with one exception: it must have a third part present. This is the binary scale factor denoted by the letter B, followed by a signed or unsigned integer. The binary point is initially assumed at the left-hand end of the word between bit position 0 and 1. It is then adjusted by the binary scale factor, designated with plus implying a shift to the right and with minus, a shift to the left. Double-precision fixed-point follows the rules of double-precision floating-point with addition of the binary scale factor.

---

[1] The equal sign preceding a literal may appear in any column (following the left parenthesis) of the variable field of a CALL pseudo-operation. This allows the specification in a CALL pseudo-operation of one or more literal arguments.

Examples of decimal literals are:

```
=-10                 Integer
=26.44167E-1         Single-precision floating-point
=1.27743675385D0     Double-precision floating-point
=22.5B5              Fixed-point
```

Octal Literals

The octal literal consists of the character O followed by a signed or unsigned octal integer. The octal integer may be from one to twelve digits in length plus the sign. The Assembler will store it in a word, right-justified. The word will be stored in its real form and will not be complemented if there is the presence of a minus sign. The sign applies to bit 0 only.

Examples of octal literals are:

```
=O1257
=O-377777777742
```

Alphanumeric Literals

The alphanumeric, or Hollerith literal consists of the letters H or kH, where k is a character count followed by the data. If there is no count specified, a literal of exactly six 6-bit characters including blanks is assumed to follow the letter H. If a count exists, the k characters following the character H are to be used as the literal. If the value k is not a multiple of six, the last partial word will be left-justified and filled in with blanks. The value k can range from 1 through 53. (Embedded blanks do not terminate scanning of the cards by the Assembler.)

Examples of alphanumeric literals are:

```
=HALPHA1
=HGONE               (ƀ represents a blank)
=4HGONEƀƀ
=7HTHEƀEND
```

Instruction Literals

The instruction literal consists of the character =followed by the letter M. This is followed in turn by an operation code, one blank, and a variable field. (The embedded blank does not terminate scanning of the card in this instance.) Only the machine instructions and one pseudo-operation (ARG) are legal in an instruction literal.

Examples of instruction literals are:

```
=MARGƀBETA
=MLDAƀ5,1
```

Instructions containing instruction literals cannot make use of any of the forms of a tag modifier, since if a modifier is encountered it is assumed to be part of the instruction literal.

Variable Field Literals

The variable field literal begins with the letter V. Reference should be made to the description of the VFD pseudo-operation for the detailed description of using variable field data description. The subfields of a variable field literal may be one of three types: Algebraic, Boolean, and Alphanumeric.

Examples of variable field literals are:

    =V10/895,5/37,H6/C,15/ALPHA
    =V18/ALPHA,O12/235,6/0

Instructions containing variable field literals cannot make use of any of the forms of a tag modifier.

Literals Modified by DU or DL

When a literal is used with the modifier variations DU or DL, the value of the literal is not stored in the literal pool but is truncated to an 18-bit value, and is stored in the address field of the machine instruction. Normally, a literal represents a 36-bit number. For the DU or DL modifier variations, if the literal is a floating-point number or Hollerith, then bits 0-17 of the literal will be stored in the address field. In the case of all other literals, bits 18-35 of the literal will be stored in the address field.

Examples of literals modified by DU and DL are:

CODED LITERAL	RESULTANT ADDRESS FIELD (OCTAL)
=100,DL	000144
=-1.0,DU	001000
=320.,DU	022500
=0.,DU	400000
=O77,DU	000077
=2B25,DU	004000
=3H00A,DL	000021

OPERATIONS AND OPERATION CODING

Processor Instructions

Processor instructions written for the Assembler consist of a symbol (or blanks) in the location field, a 3- to 6-character alphanumeric code representing a GE-625/635 operation in the operation field, and an operand address, (symbolic or numeric), plus a possible modifier tag in the variable field. (Legal symbols used in the location field and as operand addresses in the variable field are described on page 153 and following.)

Standard machine mnemonics are entered left-justified in the operation field. These are any instruction mnemonic, as presented in the listings comprising Appendices A and C.

Several Assembler pseudo-operations are closely related to machine instructions. These are:

1. OPSYN (operation synonym)--redefinition of a machine instruction by equating a new mnemonic to one already existing in the Assembler operation table.

2. OPD (operation definition)--definition of a new machine instruction to the Assembler.

3. MACRO (macro instruction definition)--define a mnemonic operation code to cause one or more standard operations to be generated by the Assembler.

The operand address and modifier tag of most machine instructions comprise the subfield entries of the variable field. The address portion may be any legitimate expression, described earlier. The address is the first subfield in the variable field and begins in column 16. The modifier tag subfield is separated from the address subfield by a comma. Coding of the modifier tag subfield entries is described on the pages following.

Address Modification Features

● <u>Summary.</u> The GE-625/635 performs address modification in four basic ways: Register modification (R), Register then Indirect modification (RI), Indirect then Register modification (IR), and Indirect then Tally modification (IT). Each of these basic types has associated with it a number of variations in which selectable registers can be substituted for R in R, RI, and IR and in which various tallying or other substitutions can be made for T in IT. I always indicates indirect address modification and is represented by the asterisk * placed in the variable field of the <u>Macro Assembler coding sheet as</u> *R or R* when IR or RI is specified. To indicate IT modification, only the substitution for T appears in the coding sheet variable field; that is, the asterisk is not used.

● <u>Indirect Addressing.</u> In indirect addressing, the contents of the instruction address y are treated as another address, rather than as the operand of the instruction code. In the GE-625/635, indirect address modification is handled automatically as a hardware function whenever called for by program instruction. This form of modification precedes directly address modification for IR and IT: for RI, it follows. When the I modification is called for by a program instruction, an indirect word is always obtained from memory. This indirect word may call for continued I modification, or it may specify the effective address Y to be used by the original instruction. Indirect addressing for RI, IR and IT is performed by the Processor whenever a binary 1 appears in either position of the t_m field (bit position 30 and 31) of an instruction or an applicable indirect word. The four basic modification types, their mnemonic substitutions as used in the variable field of the coding sheet, and the binary forms presented to the Processor by the Assembler are as follows:

MODIFICATION TYPE	CODING SHEET MNEMONIC	BINARY FORMS

The binary forms shown:

- (top, no label) z | i | z | t_m | t_d — Tag, positions 30,31,32 ... 35

- R — BETA,(R) — z | i | z | 0 0 — Tag, 30,31,32 ... 35

- RI — BETA,(R)* — z | i | z | 0 1 — Tag, 30,31,32 ... 35

- IR — BETA,*(R) — z | i | z | 1 1 — Tag, 30,31,32 ... 35

- IT — BETA,(T) — z | i | z | 1 0 — Tag, 30,31,32 ... 35

The parentheses in (R) and (T) indicate that substitutions are made by the programmer for R and T; these are explained under the separate discussions of R, IR, RI, and IT modification. Binary equivalents of the substitution are used in the t_d subfield.

Register (R) Modification

Simple R-type address modification is performed by the Processor whenever the programmer codes an R-type variation (listed below) and causes the Assembler to place binary zeros in both positions of the modifier subfield t_m of the general instruction. Accordingly, one among 16 variations under R will be performed by the Processor, depending upon bit configurations generated by the Assembler and placed in the designator subfield (t_d) of the general instruction. The 16 variations, their mnemonic substitutions used on the Assembler coding sheet, the t_d field binary forms presented to the Processor, and the effective address Y generated by the Processor are indicated in the following table.

A special kind of address modification variation is provided under R modification. The use of the instruction address field as the operand is referred to as direct operand address modification, of which there are two types; (1) Direct Upper and (2) Direct Lower. With the Direct Upper variation, the address field of the instruction serves as bit positions 0-17 of the operand and zeros serve as bit positions 18-35 of the operand. With the Direct Lower variation, the address field of the instruction serves as bit positions 18-35 of the operand and zeros serve as bit positions 0-17 of the operand.

MODIFICATION VARIATION	MNEMONIC SUBSTITUTION	BINARY FORM (t_d FIELD)	EFFECTIVE ADDRESS
(R)=X0	0	1000	$Y=y+C(X0)_{0-17}$
=X1	1	1001	$Y=y+C(X1)_{0-17}$
=X2	2	1010	$Y=y+C(X2)_{0-17}$
=X3	3	1011	$Y=y+C(X3)_{0-17}$
=X4	4	1100	$Y=y+C(X4)_{0-17}$
=X5	5	1101	$Y=y+C(X5)_{0-17}$
=X6	6	1110	$Y=y+C(X6)_{0-17}$
=X7	7	1111	$Y=y+C(X7)_{0-17}$
$=A_{0-17}$	AU	0001	$Y=y+C(A)_{0-17}$
$=A_{18-35}$	AL	0101	$Y=y+C(A)_{18-35}$
$=Q_{0-17}$	QU	0010	$Y=y+C(Q)_{0-17}$
$=Q_{18-35}$	QL	0110	$Y=y+C(Q)_{18-35}$
$=IC_{0-17}$	IC	0100	$Y=y+C(IC)_{0-17}$
$=IR_{0-17}$	DU	0011	$C(Y)_{0-17}=y$
$=IR_{0-17}$	DL	0111	$C(Y)_{18-35}=y$
=None	Blank or N	0000	$Y=y$
=Any symbolic index register	Any defined symbol*		

* Symbol must be defined as one of the index registers X0-X7 by use of an applicable pseudo-operation. (See discussion of EQU, page 191, and BOOL, page 192.)

The examples following show how R-type modification variations are entered in the variable field and their resultant control effects upon Processor development of effective addresses.

	LOCATION	OPERATION	VARIABLE FIELD (ADDRESS, TAG)	COMMENTS MODIFICATION TYPE	COMMENTS EFFECTIVE ADDRESS
1.			B, 0	(R)	$Y=B+C(X0)_{0-17}$
2.			C, AL	(R)	$Y=C+C(A)_{18-35}$
3.			M, QU	(R)	$Y=M+C(Q)_{0-17}$
4.			-2, IC	(R)	$Y=C(IC)-2$
5.			*, DU	(R)	$Operand_{0-17}=IC$
6.			1, 7	(R)	$Y=1+C(X7)$
7.			2, DL	(R)	$Operand_{18-35}=2$
8.			B	(R)	$Y=B$
9.			B, N	(R)	$Y=B$
10.			C, ALPHA	(R)	$Y=C+C(X2)$
	ALPHA	EQU	2		

Register Then Indirect (RI) Modification

Register then Indirect address modification in the GE-625/635 is a combination type in which both indexing (register modification) and indirect addressing are performed. For indexing modification under RI, the mnemonic substitutions for R are the same as those given under the discussion of Register (R) modification with the exception that DU or DL cannot be substituted for R. For indirect addressing (I), the Processor treats the contents of the operand address associated with the original instruction or with an indirect word as described on page 163.

Under RI modification, the effective address Y is found by first performing the specified Register modification on the operand address of the instruction; the result of this R modification under RI obtains the address of an indirect word which is then retrieved.

After the indirect word has been accessed from memory and decoded, the Processor carries out the address modification specified by this indirect word. If the indirect word specifies RI, IR, or IT modification (any type specifying indirection), the indirect sequence is continued. When an indirect word is found that specifies R modification, the Processor performs R modification, using the register specified by the t_d field of this last encountered indirect word and the address field of the same word, to form the effective address Y.

It should be observed again that the variations DU and DL of Register modification (R) cannot be used with Register then Indirect modification (RI).

If the programmer desires to reference an indirect word from the instruction itself without including Register modification, he specifies the "no modification" variation; under RI modification, this is indicated on the coding form by an asterisk alone placed in the variable field tag position.

The examples below illustrate the use of R combined with RI modification, including the use of (R) = N (no register modification). The asterisk (*) appearing in the modifier subfield is the Assembler symbol for I (Indirect). The address subfield, single-symbol expressions shown are not intended as realistic coding examples but rather to show the relation between operand addresses, indirect addressing, and register modification.

	LOCATION	OPERATION	VARIABLE FIELD (ADDRESS, TAG)	COMMENTS MODIFICATION TYPE	EFFECTIVE ADDRESS
1.		--	Z,AU*	(R)*	$Y=B+C(X1)_{0-17}$
	$Z+C(A)_{0-17}$	--	B,1	(R)	
2.		--	Z,*	(R)*	$Y=B+C(Q)_{0-17}$
	Z	--	B,QU	(R)	
3.		--	Z,*	(R)*	Y=M
	Z	--	B,5*	(R)*	
	$B+C(X5)_{0-17}$	--	C,3*	(R)*	
	$C+C(X3)_{0-17}$	--	M	(R)	

Indirect Then Register (IR) Modification

Indirect then Register address modification is a combination type in which both indirect addressing and indexing (register modification) are performed. IR modification is not a simple inverse type of RI; several important differences exist.

Under IR modification, the Processor first fetches an indirect word (obtained via I or IR) from the core storage location specified by the address field y of the machine instruction; and the C(R) of IR are safe-stored for use in making the final index modification to develop Y.

Next, the address modification, if any, specified by this first indirect word is carried out. If this modification is again IR, another indirect word is retrieved from storage immediately; and the new C(R) are safe-stored, replacing the previously safe-stored C(R). If an IR loop develops, the above process continues, each new R replacing the previously safe-stored R, until something other than IR is encountered in the indirect sequence--R, IT, or RI.

If the indirect sequence produces an RI indirect word, the R-type modification is performed immediately to form another address; but the I of this RI treats the contents of the address as an indirect word. The chain then continues with the R of the last IR still safe-stored, awaiting final use. At this point the new indirect word might specify IR-type modification, possibly renewing the IR loop noted above; or it might initiate an RI loop. In the latter case, when this loop is broken, the remaining modification types are R or IT.

When either R or IT is encountered, it is treated as type R where R is the last safe-stored C(R) of an IR modification. At this point the safe-stored C(R) are combined with the y of the indirect word that produced R or IT, and the effective address Y is developed.

If an indirect modification without Register modification is desired, the no-modification variation (N) of Register modification should be specified in the instruction. This normally will be entered on the coding sheets as *N in the modifier part of the variable field. (The entry * alone is equivalent to N* under RI modification and must be used in this way.) The mnemonic substitutions for (R) are listed under the Register modification description.

CPB-1004F

167

The examples below illustrate the use of IR-type modification, intermixed with R and RI types, under the several conditions noted above.

	LOCATION	OPERATION	VARIABLE FIELD (ADDRESS, TAG)	COMMENTS	
				MODIFICATION TYPE	EFFECTIVE ADDRESS
1.		--	Z,*QL	*(R)	$Y=M+C(Q)_{18-35}$
	Z	--	M	(R)	
2.		--	Z, *3	*(R)	$Y=C+C(X3)_{0-17}$
	Z	--	B,5*	(R)*	
	$B+C(X5)_{0-17}$	--	C, IC	(R)	
3.		--	Z, *3	*(R)	$Y=M+C(Q)_{0-17}$
	Z	--	B, *5	*(R)	
	B	--	C,*QU	*(R)	
	C	--	M, 7	(R)	
4.		--	Z, *DL	*(R)	$C(Y)_{18-35}=M$
	Z	--	B, 3*	(R)*	
	$B+C(X3)_{0-17}$	--	M, QL	(R)	
5.		--	Z, *AL	*(R)	$Y=B+C(A)_{18-35}$
	Z	--	B, AD	(T)	
6.		--	Z, *N	*(R)	$Y=B$
	Z	--	B, 3	(R)	
7.		--	Z, *N	*(R)	$Y=M+C(X5)_{0-17}$
	Z	--	B, *5	*(R)	
	B	--	M, DU	(R)	
8.		--	Z, *	(R)*	$Y=M+C(X5)_{0-17}$
	Z	--	B, *5	*(R)	
	B	--	M, DU	(R)	

CPB-1004F

168

Indirect Then Tally (IT) Modification

● Summary. Indirect then Tally address modification in the GE-625/635 is a combination type in which both indirect addressing and indexing (register modification) are performed. In addition automatic incrementing/decrementing of fields in the indirect word are done as hardware features, thus relieving the programmer of these responsibilities. The automatic tallying and other functions of the IT type modification greatly enhance the processing of tabular data in memory, provide the means for working upon character data, and allow termination on programmer-selectable numerical tally conditions. (Refer to page 206 for the special word formats TALLYB, TALLYD, and TALLYC for Assembler coding of the indirect words used with IT; and refer to Figure 5B for Tally Runout status.)

The ten variations under IT modification are summarized in the following table. It should be noted that the mnemonic substitution for IT on the Macro Assembler coding sheet is simply (T); the designator I for indirect addressing in IT is not represented. (Note that one of the substitutions for T is I.)

NAME OF THE VARIATION	CODING FORM SUBSTITUTION FOR I(T)	BINARY FORM (t_d FIELD)	EFFECT UPON THE INDIRECT WORD
Indirect	I	1001	None.
Increment address, Decrement tally	ID	1110	Add one to the address; subtract one from the tally.
Decrement address, Increment tally	DI	1100	Subtract one from the address; add one to the tally.
Sequence Character	SC	1010	Add one to the character position number; subtract one from the tally; add one to the address when the character count crosses a word boundary.
Character from Indirect	CI	1000	None.
Add Delta	AD	1011	Add an increment to the address; decrement the tally by one.
Subtract Delta	SD	0100	Subtract an increment from the address; increase the tally by one.
Fault	F	0000	None; the Processor is forced to a fault trap starting at a predetermined, fixed location
Increment address Decrement tally, and Continue	IDC	1111	Same as ID variation except that further address modification can be performed.
Decrement address, Increment tally, and Continue	DIC	1101	Same as DI except that further address modification can be performed.

169

● Indirect (T) = I Variation. The Indirect (I) variation of IT modification is in effect a subset of the ID and DI variations described below in that all three--I, ID, and DI--make use of one indirect word in order to reference the operand. The I variation is functionally unique, however, in that the indirect word referenced by the program instruction remains unaltered-- no incrementing/decrementing of the address field. Since the t_m and t_d subfields of the indirect word under I are not interrogated, this word will always terminate the indirect chain.

The following differences in the coding and effects of *, *N, and I should be observed:

1. RI modification is coded as R* for all cases, excluding R=N.

   For R=N under RI, the modifier subfield can be written as N* or as * alone, according to programmer preference.

   When N* or just * is coded, the Assembler generates a machine word with 20_8 in positions 30-35; 20 causes the Processor to add 0 to the address y of the word containing the N* or * and then to access the indirect word at memory location y of the N* or * word.

2. IR modification is coded as *R for all cases, including R=N.

   For R=N under IR, the modifier subfield must be written as *N.

   When *N is coded, the Assembler generates 60_8 in positions 30-35 of the associated machine word; 60_8 causes the Processor to (1) retrieve the indirect word at location y of the machine word, and (2) effectively safe-store zeros (for possible final index modification of the last indirect word--to develop the effective address Y).

3. IT modification is coded using only a variation designator (I, ID, DI, SC, CI, AD, SD, F, IDC, DIC); that is, the asterisk (*) is not written (for I). Thus, a written IT address modification appears as ALPHA, DI; BETA, AD; etc.

   For the variation I under IT, the Assembler generates a machine word with 51_8 in bit positions 30-35; 51 causes the Processor to perform one and only one indirect word retrieved from memory location y (of the word with I specified) to obtain the effective address Y. For example:

LOCATION	OPERATION	VARIABLE FIELD (ADDRESS, TAG)	COMMENTS MODIFICATION TYPE	EFFECTIVE ADDRESS
	--	Z, I	(T)	Y=B
Z	--	B, *5	*(R)	

● Increment Address, Decrement TALLY (T) = ID Variation. The ID variation under IT modification provides the programmer with automatic (hardware) incrementing/decrementing of an indirect word that is best used for processing tabular operands (data located at consecutive memory addresses). The indirect word always terminates the indirect chain.

In the ID variation the effective address is the address field of the indirect word obtained via the tentative operand address of the instruction or preceding indirect word, whichever specified the ID variation. Each time such a reference is made to the indirect word, the address field of the indirect word is incremented by one; the tally portion of the indirect word is decremented by one. The incrementing and decrementing are done after the effective address is provided for the instruction operation. When the tally reaches zero, the Tally Runout indicator is set.

CPB-1004F

The example following shows the effect of ID.

| | | | COMMENTS | | |
LOCATION	OPERATION	VARIABLE FIELD ADDRESS, TAG	MODIFICATION TYPE	EFFECTIVE ADDRESS	REFERENCE
	--	Z, ID	(T)	B	1
Z	--	B		B+1	2
				.	.
				.	.
				.	.
Assuming an initial tally of j, the tally runout indicator is set on the jth reference.				B+n	n+1
				.	.
				.	.
				.	.
				.	.

● **Decrement Address, Increment Tally (T) = DI Variation.** The DI variation under IT modification provides the programmer with automatic (hardware) incrementing/decrementing of an indirect word that is best used for processing tabular operands (data located at consecutive memory addresses). The indirect word always terminates the indirect chain.

In the DI variation the effective address is the address field minus one of the indirect word obtained via the tentative operand address of the instruction or preceding indirect word, whichever one specified the DI variation. Each time a reference is made to the indirect word, the address field of the indirect word is decremented by one; and the tally portion is incremented by one. The incrementing and decrementing is done prior to providing the effective address for the current instruction operation.

The effect of DI when writing programs is shown in the example following.

| | | | COMMENTS | | |
LOCATION	OPERATION	VARIABLE FIELD ADDRESS,TAG	MODIFICATION TYPE	EFFECTIVE ADDRESS	REFERENCE
	--	Z, DI	(T)	B-1	1
Z	--	B		B-2	2
				.	.
				.	.
				.	.
Assuming an initial tally of 4096-j the tally runout is set on the jth reference.				B-n	n
				.	.
				.	.

● **Sequence Character (T) = SC Variation.** The Sequence Character (SC) variation is provided for programmed operations involving 6-bit or 9-bit characters that are accessed sequentially in memory. Processor instructions that exclude character operations are indicated in the individual instruction description. For the SC variation, the effective operand address is the address field of the indirect word obtained by the tentative operand address of the instruction or preceding indirect word that specified the SC variation.

Characters are operated on in sequence from left to right within the machine word. The character position field of the Tally indirect word is used to specify the character to be involved in the operation. This variation is intended for use only with those operations that involve the A-and Q-registers. The tally – runout indicator is set when the tally field of the indirect word reaches zero. The following is an example of the coding sequence for the SC variation:

```
    ADD1      LDA      ADDR, SC
    ADDR      TALLY    ADD,12,3      6-bit characters
or  ADDR      TALLYB   ADD,12,3      9-bit characters
              TTF      ADD1
```

The effective address is ADD. The character in character position 3 is loaded into the A-register in position 5 for 6-bit characters and position 3 for 9-bit characters.

The tally field of the indirect word is used to count the number of times a reference is made to a character in the tally indirect word. Each time an SC reference is made to the tally indirect word, the tally is decremented by one and the character position is incremented by one to show the next character position. When C=5 (or 3 for 9-bit characters), it is changed to zero and the address field of the tally indirect word is incremented by one. All incrementing and decrementing is done <u>after</u> the effective address has been provided for the instruction execution.

● Character From Indirect (T) = CI Variation. The Character from Indirect (CI) variation is provided for programmer operations on 6-bit or 9-bit characters in any situation where repeated reference to a single character in memory is required.

For this variation substitution, the effective address is the address field of the CI indirect word obtained via the tentative operand address of the instruction or preceding indirect word that specified the CI variation. The character position field of the indirect word is used to specify the character to be involved in the operation and is intended for use only with the operations that involve the A- or Q-register.

This variation is similar to the SC variation except that no incrementing or decrementing of the address or character position is performed.

A CI example is:

		VARIABLE FIELD	COMMENTS		
LOCATION	OPERATION	ADDRESS, TAG	MODIFICATION TYPE	EFFECTIVE ADDRESS	REFERENCE
	--	Z, CI	(T)	Y=B	
Z	--	B			

● __Add Delta (T) = AD Variation.__ The Add Delta (AD) variation is provided for programming situations where tabular data to be processed is stored at equally spaced locations, such as data words, each occupying two or more consecutive memory addresses. It functions in a manner similar to the ID variation, but the incrementing (delta) of the address field is selectable by the programmer.

Each time such a reference is made to the indirect word, the address field of the indirect word is increased by delta and the tally portion of the indirect word is decremented by one. The addition of delta and decrementing is done __after__ the effective address is provided for the instruction operation.

The example following shows the effect of AD.

		VARIABLE FIELD	COMMENTS		
LOCATION	OPERATION	ADDRESS, TAG	MODIFICATION TYPE	EFFECTIVE ADDRESS	REFERENCE
	--	Z, AD	(T)	B	1
Z	--	B	(R)	B+δ	2
				B+2δ	3
			.	.	.
			.	.	.
			.	B+nδ	n+1
			.	.	.
			.	.	.
			.	.	.

Assuming an initial tally of j, the tally runout is set on the jth reference.

● __Subtract Delta (T) = SD Variation.__ The Subtract Delta (SD) variation is useful in processing tabular data in a manner similar to the AD variation except that the table can easily be scanned from back to front using a programmer specified increment. The effective address from the indirect word is decreased by delta and the tally is increased by one each time the indirect word is used. This applies to the first reference to the indirect word, making the SD variation analogous to the DI variation.

● __Fault (T) = F Variation.__ The fault variation enables the programmer to force program transfers to Comprehensive Operating Supervisor routines or to his own corrective routines during the execution of an address modification sequence. (This will usually be an indication of some abnormal condition against which the programmer wishes to protect himself. For an explanation of how faults are handled in the GE-625/635, refer to the reference manual on the Comprehensive Operating Supervisor.)

CPB-1004 F

173

● Increment Address, Decrement Tally and Continue (T) = IDC Variation. The IDC variation under IT modification functions in a manner similar to the ID variation except that, in addition to automatic incrementing/decrementing, it permits the programmer to continue the indirect chain in obtaining the instruction operand. Where the ID variation is useful for processing tabular data, the IDC variation permits processing of scattered data by a table of indirect pointers. More specifically, the ID portion of this variation gives the sequential stepping through a table; and the C portion (continuation) allows indirection through the tabular items. The tabular items may be data pointers, subroutine pointers or possibly a transfer vector.

The address and tally fields are used as described under the ID variation. The tag field uses the set of GE-625/635 instruction address modification variations under the following restrictions: No variation is permitted which requires an indexing modification in the IDC cycle since the indexing adder is in use by the tally phase of the operation. Thus, permissible variations are any form of I(T) or I(R); but if (R)I or (R) is used, R must equal N.

The effect of IDC is indicated in the following example:

		VARIABLE FIELD	COMMENTS		
			MODIFICATION	EFFECTIVE	
LOCATION	OPERATION	ADDRESS, TAG	TYPE	ADDRESS	REFERENCE
				Effective Address	Character Position
	--	Z, IDC	(T)	B	1
Z	--	B	(R)	B+1	2
				.	.	.
				.	.	.
				.	.	.
Assuming an initial tally of j, the tally runout indicator is set on the jth reference.				B+n	n+1
				.	.	.
				.	.	.
				.	.	.

● Decrement Address, Increment Tally, and Continue (T) = DIC Variation. The DIC variation under IT modification works in much the same way as the DI variation except that in addition to automatic decrementing/incrementing it allows the programmer to continue the indirect chain in obtaining an instruction operand. The continuation function of DIC operates in the same manner and under the same restrictions as IDC except that (1) it increments in the reverse direction, and (2) decrementing/incrementing is done prior to obtaining the effective address from the tally word. (Refer to the example under IDC; work from the bottom of the table to the top.) DIC is especially useful in processing last-in, first-out lists.

CPB-1004F

174

LOCATION	OPERATION	VARIABLE FIELD ADDRESS, TAG	COMMENTS MODIFICATION TYPE	EFFECTIVE ADDRESS	REFERENCE
	--	Z, DIC	(T)		
Z	--	B, *3	*(R)	C+C(X3)	1
B-1	--	C, QU	(R)	A+C(X3)	2
B-2	--	M,5*	(R)*	$Q+C(AR)_{0-17}$	3
B-3	--	D,*AU	*(R)	.	.
.			.	.	.
$M+C(X5)_{0-17}$ -		A	(R)		
D	--	Q	(R)		

Assuming an initial tally of 4096-j, the tally runout indicator is set on the jth reference.

PSEUDO-OPERATIONS

Pseudo-operations are so-called because of their similarity to machine operations in an object program. In general, however, machine operations are produced by computer instructions and perform some task, or part of a task, directly concerned with solving the problem at hand. Pseudo-operations work indirectly on the problem by performing machine conditioning functions, such as memory allocating, and by directing the Macro Assembler in the preparation of machine coding. A pseudo-operation affecting the Assembler may generate several, one, or no words in the object program.

All pseudo-operations for the Macro Assembler are grouped according to function and described (in this chapter) as to composition and use. The pseudo-operation functional groups and their uses are:

FUNCTIONAL GROUP	PRINCIPAL USES
Control pseudo-operations	Selection of printout options for the assembly listing, direction of punchout of absolute/relocatable binary program decks, selection of format for the absolute binary deck.
Location counter pseudo-operations	Programmer control of single or multiple instruction counters.
Symbol defining pseudo-operations	Definition of Assembler source program symbols by means other than appearance in the location field of the coding form
Data generating pseudo-operations	Production of binary data words for the assembly program.
Storage allocation pseudo-operations	Provision of programmer control for the use of memory.

FUNCTIONAL GROUP	PRINCIPAL USES
Special pseudo-operations	Generation of zero operation code instructions, of binary words divided into two 18-bit fields, and of continued subfields for selected pseudo-operations.
MACRO pseudo-operations	Begin and end MACRO prototypes; Assembler generation of MACRO-argument symbols; and repeated substitution of arguments within MACRO prototypes.
Conditional pseudo-operations	Conditional assembly of variable numbers of input words based upon the subfield entries of these pseudo-operations.
Program linkage pseudo-operations	Generation of standard system subroutine calling sequences and return (exit) linkages.
Address, tally pseudo-operations	Control of automatic address, tally, and character incrementing/decrementing.
Repeat mode coding formats	Control of the repeat mode of instruction execution (coding of RPT, RPD, and RPL instructions).

The above pseudo-operation functional groups, together with their pseudo-operations, are given as a complete listing with page references in Appendix D.

Control Pseudo-Operations

The On/Off switch type pseudo-operation

The subset of the control pseudo-operations consisting of those operations which may best be described as switches (which current state may be 'on' or 'off') are comprised of the following:

DETAIL, LIST, PCC, REF, PMC, INHIB, PUNCH, EDITP, CRSM

Provisions have been made to allow the user to treat these switches in a push-down pull-up manner so that he may recall prior states of a switch and retrieve that state at some later point. The depth to which this may be accomplished is 35; a switch may therefore have a current state plus 35 "remembered" states.

The mnemonic representing the push-down feature is SAVE; pull-up or retrieve prior is designated by the mnemonic RESTORE. The mnemonic for turning the current state of a switch on is ON; its counterpart is OFF. If a switch alteration is implied but not explicitly given, its current state will be alternated (i.e., if off, turn on); if alteration is not implied, its current state will be unchanged (see example 4 on the following page).

The eight possible variable field representations are:

1. CRSM* ON   (turn switch on)
2. CRSM* OFF   (turn switch off)
3. CRSM*   (alternate current status of switch)
4. CRSM* SAVE   (push down--remember current state and leave unchanged)
5. CRSM* SAVE, ON   (push down, and set switch on)
6. CRSM* SAVE, OFF   (push down, and set switch off)
7. CRSM* SAVE,   (push down, and alternate current switch setting)
8. CRSM* RESTORE   (pull up prior state of switch)

*CRSM is used for illustrative purposes only.

The Assembler has been preset with a 'current' state for each switch, and 35 remembered states which are the same. Restores past this point will pull up an ON state for all switches. The initial setting is given in the discussion of each of the pseudo-operations in question, and corresponds to normal mode of operation.

DETAIL ON/OFF (Detail Output Listing)

1 2	8	16	32
Blanks	DETAIL	ON	Normal mode
Blanks	DETAIL	OFF	

Some pseudo-operations generate no binary words; however, several of them generate more than one. The generative pseudo-operations are; OCT, DEC, BCI, DUP, CALL, SAVE, RETURN, ERLK, LIT, and VFD. The DETAIL pseudo-operation provides control over the amount of listing detail generated by the generative pseudo-operations.

The use of the DETAIL OFF pseudo-operation causes the assembly listing to be abbreviated by eliminating all but the first word generated by any of the above pseudo-operations. In the case of the DUP pseudo-operation, only the first iteration will be listed. The DETAIL ON pseudo-operation causes the Assembler to resume the listing which had been suspended by a DETAIL OFF pseudo-operation.

If at the end of the listing the Assembler is in the DETAIL OFF mode, the literal pool will not be printed, but a notation will be made as to its presence.

If the Assembler is already in a specified ON/OFF mode, then the pseudo-operation requesting the same ON/OFF mode is ignored.

LIST ON/OFF (Control Output Listing)

1 2	8	16	32
Blanks	LIST	ON	Normal mode

The use of LIST in the operation field with OFF in the variable field causes the normal listing to change as follows: the instruction LIST OFF will appear in the listing; thereafter, only instructions which are flagged in error will appear. If the assembly ends in the LIST OFF mode, only the error messages will appear.

The use of LIST in the operation field with ON in the variable field causes the normal listing, which was suspended by a LIST OFF pseudo-operation, to be resumed. If the Assembler is already in a specified ON/OFF mode, then the pseudo-operation requesting the same ON/OFF mode is ignored.

PCC ON/OFF (Print Control Cards)

1	8	16	32
Blanks	PCC	OFF	Normal Mode

The PCC pseudo-operation affects the listing of the following pseudo-operations:

DETAIL	LIST	*TTL	PMC
EJECT		*TTLS	PUNCH
*LBL	REF	CRSM	IDRP
INE	IFE	IFG	IFL

PCC ON causes the affected pseudo-operations to be printed. PCC OFF causes the affected pseudo-operations to be suppressed; this is the normal mode at the beginning of the assembly. If the Assembler is already in a specified ON/OFF mode, then the pseudo-operation requesting the same ON/OFF mode is ignored.

REF ON/OFF (References)

1	8	16	32
Blanks	REF	ON	Normal mode

The REF pseudo-operation controls the Assembler in making entries into the symbol reference table and controls the listing of nonreferenced symbols. REF ON (the normal mode) causes the Assembler to begin making entries into the symbol reference table. REF OFF causes the Assembler to suppress making entries into the symbol reference table. If the Assembler is already in a specified ON/OFF mode, another request for the same mode is ignored.

The entry LNRSM (list nonreferenced symbols) can also be used as a subfield of the variable field, to cause listing of nonreferenced symbols when the Assembler is in the REF ON mode. The variable field scan is terminated when either an ON, OFF or RESTORE subfield is encountered. Therefore, these entries should always be last when used in a series of subfields.

EXAMPLES: REF ON or the absence of a REF pseudo-operation causes a listing of only referenced symbols and references to those symbols.

REF LNRSM,ON or REF LNRSM causes listing of all symbols and references.

REF OFF causes listing of all symbols, but no references. (REF LNRSM, OFF has the same effect because the LNRSM entry is only effective when the assembler is in the REF ON mode.)

---

* Not affected if alter number is three or less (1, 2, or 3).

CPB-1004F

178

PMC ON/OFF (Print MACRO Expansion)

1	8	16	32
Blanks	PMC	OFF	Normal mode

The PMC pseudo-operation causes the Assembler to list or suppress all instructions generated by a MACRO call.

PMC ON causes the Assembler to print all generated instructions. PMC OFF causes the Assembler to suppress all but the MACRO call.

If the Assembler is already in a specified ON/OFF mode, then the pseudo-operation requesting the same ON/OFF mode is ignored.

INHIB ON/OFF (Inhibit Interrupts)

1	8	16	32
Blanks	INHIB	OFF	Normal mode

The instruction INHIB ON causes the Assembler to set the program interrupt inhibit bit in bit position 28 of all machine instructions which follow the pseudo-operation. The setting of the instruction interrupt inhibit bit continues for the remainder of the assembly, unless the pseudo-operation INHIB OFF is encountered.

The INHIB OFF causes the Assembler to stop setting the inhibit bit in each instruction, if used when the Assembler is in the INHIB ON mode.

If the Assembler is already in a specified ON/OFF mode, then the pseudo-operation requesting the same ON/OFF mode is ignored.

PUNCH ON/OFF (Control Card Output)

1	8	16	32
Blanks	PUNCH	ON	Normal mode

Subject to the DECK/NDECK option of the GMAP call card, the normal mode of the Assembler is to punch binary cards for everything it assembles. If PUNCH is used in the operation field with OFF in the variable field, the binary deck will not be punched, beginning at the point the Assembler encounters the pseudo-operation.

CPB-1004F

These conventions hold true for both the output binary deck, and the load file counterpart, in the case of assemble and execute activities.

If the Assembler is already in a specified ON/OFF mode, then the pseudo-operation requesting the same ON/OFF mode is ignored.

EDITP (Edit Print Lines)

1	8	16	32
	EDITP	OFF	Normal mode

This pseudo-operation has a special application. It is for the program which includes the character ? (17) and/or ! (77) punched somewhere on a symbolic card. In normal operation these characters have special meaning to the printer subsystem and may cause character shifting, line suppression, slewing, or buffer overflow. As such, an EDITP ON instruction, causes the output routine to issue printer commands which will treat these as non-special characters. The Assembler will then remain in this mode until an EDITP OFF instruction is encountered.

EJECT (Restore Output Listing)

1	8	16	32
Blanks	EJECT		Column 16 must be blank

The EJECT pseudo-operation causes the Assembler to position the printer paper at the top of the next page, to print the title(s), and then print the next line of output on the second line below the title(s).

REM (Remarks)

1	8	16	32
Blanks or remarks	REM		Remarks and comments in the variable field start at column 12 or later

The REM pseudo-operation causes the contents of this line of coding to be printed on the assembly listing (just as the comments appear on the coding sheet). However, for purposes of neatness, columns 8-10 are replaced by blanks before printing.

REM is provided for the convenience of the programmer; it has no other effect upon the assembly.

* (In Column One--Remarks)

1	8	16	32
*			Remarks and comments in columns 2-80

A card containing an asterisk (*) in column 1 is taken as a remark card. The contents of columns 2-80 are printed on the assembly listing (just as they appear on the coding sheet); the asterisk has no other effect on the assembly program.

LBL (Label)

1	8	16	32
Blanks	LBL	X, Y	X=null or up to 8 alphabetic and numeric characters. Y=null or up to 42 alphabetic and numeric characters.

LBL causes the Assembler to serialize the binary cards using columns 73-80, except when punching full binary cards by use of the FUL pseudo-operation. The LBL pseudo-operation allows the programmer to specify a left-justified alphabetic label for the identification field and begin serialization with some initial serial number other than zero. The LBL pseudo-operation also allows the programmer to specify up to 42 characters of comments on the $ OBJECT card of the binary deck. The comment, if present, begins in column 16 of the $ OBJECT card. The following conditions apply:

1. If the first sub-field is null, the Assembler discontinues serialization of the binary deck.

2. If the first sub-field is not blank, serialization begins with the characters appearing in the first sub-field; the characters are left-justified and filled in with terminating zeros up to the position(s) used for the sequence number. Serialization is incremented until the rightmost nonnumeric character is encountered, at which time the sequence recycles to zero.

3. If no LBL pseudo-operation appears in the symbolic deck, the Assembler begins serializing with 00000000.

4. If the second sub-field is blank, the Assembler inserts blanks in the variable field of the $ OBJECT card.

5. If the second sub-field is not blank, the characters in this sub-field are inserted on the $ OBJECT card in column 16 through column 57.

CPB-1004F

181

TTL (Title)

1	8	16	32
Blanks or an integer	TTL		Title in the variable field

The TTL pseudo-operation causes the printing of a title at the top of each page of the assembly listing. In addition, when the Assembler encounters a TTL card, it causes the output listing to be restored to the top of the next page and the new title to be printed. The information punched in columns 16-72 is interpreted as the title.

The title may be redefined by use of repeated TTL pseudo-operations as often as the programmer desires. The title may be deleted by a TTL pseudo-operation with a blank variable field. If a decimal integer appears in the location field, the page count is re-numbered beginning with the specified integer.

TTLS (Subtitle)

1	8	16	32
Blanks or an integer	TTLS		Subtitle in the variable field

The TTLS pseudo-operation is identical in function to the TTL pseudo-operation except that it causes subtitling to occur. When a TTLS pseudo-operation is encountered, the subtitle provided in columns 16-72 replaces the current subtitle; the output listing is restored to the top of the next page. The title and new subtitle are then printed.

The maximum number of subtitles that may follow a title is one.

DATE (Current Date)

1	8	16	32
Blanks	DATE		Column 16 must be blank

The DATE pseudo-operation is used to enter the current date into a program. The 6-character current date in the form mmddyy is inserted into an assembled program at this point.

Example:

Location	Contents	Relocation	
001021	000601050607	000	DATE

This example shows the results of a DATE pseudo-operation assembled on 6/15/67.

CPB-1004F

182

ABS (Output Absolute Text)

1	8	16	32
Blanks	ABS		Column 16 must be blank

The ABS pseudo-operation causes the Assembler to produce an output of absolute binary text.

The normal mode of the Assembler is relocatable; however, if absolute text is required for a given assembly, the ABS pseudo-operation should appear in the deck before any instructions or data. It may be preceded only by listing pseudo-operations. It may, however, appear repeatedly in an assembly interspersed with the FUL pseudo-operation. It should be noted that the pseudo-operations affecting relocation are considered errors in an absolute assembly.

Pseudo-operations that will be in error if used in an absolute assembly are:

    BLOCK   SYMDEF
    ERLK   SYMREF

(Refer to the descriptions of binary punched card formats in this chapter for details of the absolute binary text.)

FUL (OUTPUT Full Binary Text)

1	8	16	32
Blanks	FUL		Column 16 must be blank

The FUL pseudo-operation is used to specify absolute assembly and the FUL format for absolute binary text.

The FUL pseudo-operation has the same effect and restrictions on the Assembler as ABS, except for the format of the binary text output. The format of the text is of continuous information with no address identification; that is, the absolute binary cards are punched with program instructions in columns 1-78 (26 words). Such cards can be used in self-loading operations or other environments where control words are not required on the binary card.

TCD (Punch Transfer Card)

1	8	16	32
Blanks or a symbol	TCD		An expression in the variable field

In an absolute assembly, the binary transfer card, produced at the end of the deck as a result of the END card, directs the loading program to cease loading and turn control over to the program at the point specified by the transfer card. Sometimes it is desirable to cause a transfer card to be produced before encountering the end of the deck. This is the purpose of the TCD pseudo-operation. Thus, a binary transfer card is produced generating a transfer address equivalent to the value of the expression in the variable field.

TCD is an error in the relocatable mode.

CPB-1004F

HEAD (Heading)

1	8	16	32
Blanks	HEAD		From 1 to 7 subfields in the variable field, each containing a single, nonspecial character used as a heading character

In programming, it is sometimes desirable to combine two programs, or sections of the same program, that use the same symbols for different purposes. The HEAD pseudo-operation makes such a combination possible by prefixing each symbol of five or fewer characters with a heading character. This character must not be one of the special characters; that is, it must be one of the characters A-Z, 0-9, or the period(.). Using different heading characters, in different program sections later to be combined for assembly, removes any ambiguity as to the definition of a given symbol.

The effect of the HEAD pseudo-operation is to cause every symbol of five or less characters, appearing in either the location field or the variable field, to be prefixed by the current HEAD character. The current HEAD character applies to all symbols appearing after the current HEAD pseudo-operation and before the next HEAD or END pseudo-operation.

Deheading is accomplished by a zero or blanks in the variable field. To understand more thoroughly the operation of the heading function, it is necessary to know that the Assembler internally creates a six-character symbol by right-justifying the characters of the symbol and filling in leading zeros. Thus, if the Assembler is within a headed program section and encounters a symbol of five or fewer characters, it inserts the current HEAD character into the high-order, leftmost character position of the symbol. Each symbol, with its inserted HEAD character, then can be placed in the Assembler symbol table as unique entries and assigned their respective location values.

It is also possible to head a program section with more than one character. This is done by using the pseudo-operation HEAD in the operation field with from two to seven heading characters in the variable field, separated by commas. The effect of a multiple heading is to define each symbol of that section once for each heading character. Thus, for example, if the symbols SHEAR, SPEED, and PRESS are headed by

	HEAD	X,Y,Z

nine unique symbols

XSHEAR	XSPEED	XPRESS
YSHEAR	YSPEED	YPRESS
ZSHEAR	ZSPEED	ZPRESS

are generated and placed in the Assembler symbol table. This allows regions by HEADX, HEADY, or HEADZ to obtain identical values for the symbols SHEAR, SPEED, and PRESS.

Cross-referencing among differently headed sections may be accomplished by the use of six-character symbols or by the use of the dollar sign ($). Six character symbols are immune to HEAD; therefore, they provide a convenient method of cross-referencing among differently headed regions.

CPB-1004F

184

When a symbol within a headed section is also to be a SYMDEF symbol, it must be a six-character symbol (immune to HEAD).

To allow the programmer more flexibility to cross-referencing, the Assembler language includes the use of the dollar sign ($) to denote references to an alien-headed region.

If the programmer wishes to reference a symbol of less than six characters in another program section, he merely prefixes the symbol by the HEAD character for that respective section, separating the HEAD character from the body of the symbol by a dollar sign ($).

To reference from a headed region into a region that is not headed (zero heading), the programmer can use either the heading character zero and the dollar sign (0$) preceding the symbol; or, if the symbol is the initial value of the variable field, then the appearance of only the leading dollar sign will cause the zero heading to be attached to the symbol.

<u>EXAMPLE OF HEAD PSEUDO-OPERATION</u>

```
START    LDA       A                    Initial instruction (no heading)
         ---
         ---
         TRA       B$SUM                Transfer to new headed section
A        BSS       1
         HEAD      B            ⎫
SUM      LDA       $A           ⎪
         ---                    ⎬  Section headed B
         ---                    ⎪
         TRA       0$START + 2  ⎪
         END                    ⎭
```

The LDA $A could have been written as LDA 0$A, as they both mean the same.

<u>DCARD (Punch BCD Card)</u>

1	8	16	32
Blanks	DCARD	N, M	Two subfields in the variable field

The first subfield contains a decimal integer N (limited only by the size of available memory), and the second subfield (M) contains a single BCD character used as a decimal data identifier. The Assembler punches the next N cards after the DCARD instruction with the specified BCD identifier in column one of each of these N cards and with the BCD information taken from the corresponding source cards on a one-for-one basis.

There are no restrictions on the BCD information that can be placed in columns 2-72 of the source cards. (One of the significant uses of DCARD is to generate Operating Supervisor (GECOS) $ control cards.)

The DCARD has the further effect of suppressing the normal automatic generation of a $ OBJECT and $ DKEND card.

CPB-1004F

185

END (End of Assembly)

1	8	16	32
Blanks or a symbol	END		Blanks or an expression in the variable field

The END pseudo-operation signals the Assembler that it has reached the end of the symbolic input deck; it must be present as the last physical card encountered by the Assembler.

If a symbol appears in the location field, it is assigned the next available location.

In a relocatable assembly, the variable field must be blank; in an absolute assembly, the variable may contain an expression. In relocatable decks, the starting location of the program will be an entry location and the location specified is given to the General Loader (GELOAD) by a special control card used with the GELOAD. (Refer to the GELOAD manual.) Absolute programs require a binary transfer card which is generated by the END pseudo-operation. The Transfer address is obtained from the expression in the variable field of the END card.

OPD (Operation Definition)

1	8	16	32
New operation code	OPD		One or more subfields, separated by commas, in the variable field. The subfields define the bit configuration of the new operation code

The OPD pseudo-operation may be used to define or redefine machine instructions to the Assembler. This allows programmers to add operation codes to the Assembler table of operation codes during the assembly process. This is extremely useful and powerful in defining new instructions or special bit configurations, unique in a particular program, to the Assembler.

The variable field subfields are bit-oriented and have the same general form as described under the VFD pseudo-operation. In addition, the variable field, considered in its entirety, requires the use of either of two specific 36-bit formats for defining the operation.

1. The normal instruction format

2. The input/output operation format

The normal instruction-defining format and subfields are shown below:

op		1	0	m_1	m_2	m_3	m_4	m_5	m_6	a_1	a_2	a_3	a_4	a_5	a_6	r_1		p	p
0	11 12	17 18								26					30	31	32 33	34	35
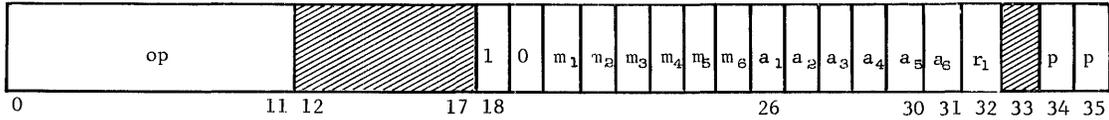
op--new operation code (bits 18 through 29 of instruction)
m--modifier tag type (0=allowed; 1=not allowed)
    m_1: register modification (R)
    m_2: indirect addressing (*)
    m_3: indirect and tally (T)
    m_4: Direct Upper (DU)
    m_5: Direct Lower (DL)
    m_6: Sequence Character (SC) and Character from Indirect (CI)
    r_1: instruction(s) in a repeat loop
a--address field conditions (0=not required; 1=required)
    a_1: address required/not required
    a_2: address required even
    a_3: address required absolute
    a_4: symbolic index required
    a_5: 2-octal digit tag field required
    a_6: address required mod 8
p--octal assembly listing format (x represents one octal digit)
    00: xx xxxx xxxxxx
    01: xxxxxxxxxxxx
    10: xxxxxx xxxxxx
    11: xxxxxx xxxx xx

The assembly listing types 00, 01, 10, and 11 are used for input/output commands, data-generating pseudo-operations (OCT, DEC, BCI, etc.), special word-generating pseudo-operations (such as ZERO), and machine instructions.

To illustrate the use of OPD, assume one wished to define the current machine instruction, Load A (LDA). Using the preceding format and the octal notation (as described under the VFD pseudo-operation), one could code OPD as

	LDA	OPD	O12/2350, 6/, O2/2, 6/, O3/4, 5/, O2/3
or	LDA	OPD	O18/235000, O2/2, 6/, O3/4, 5/, O2/3
	LDA	OPD	O36/235000401003

or in other forms, providing the bit positions of the instruction-defining format are individually specified to the Assembler.

The input/output operation defining format and subfields for types 00, 01, and 10 are as follows:

op (bit positions 18-35)		1	1	op (bits 0-5)	a_1	a_2	a_3	i		p	p
0		17 18	19 20	25	26	27	28	29 30	31 33	34	35
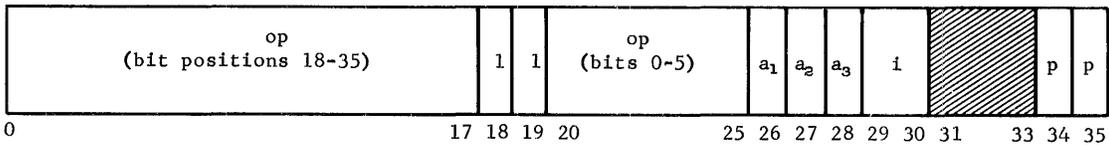
The input/output operation-defining format and subfields for type 11 are as follows:

zeros		1	1	op (bits 18-23)	a_1	a_2	a_3	i		p	p
0 - - - - - - - - - - - - - - - - 0		17 18	19 20	25	26	27	28	31		33 34	35
0											

CPB-1004F

187
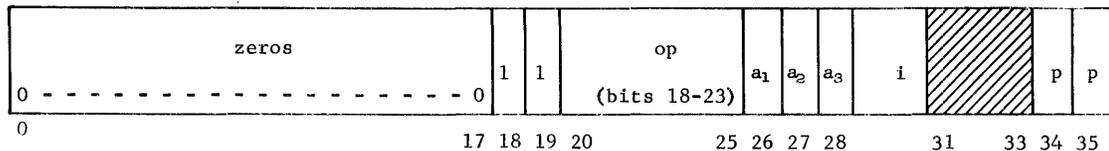
op--new operation code for bit positions 18-35 and 0-5 (or bit positions 18-23 for type 11), see Appendix E

a--address field conditions (0=not required; 1=required)

a_1: address required/not required

a_2: address required even

a_3: address required absolute

i--type of input/output command (see Appendix E, I/O Command Formats)

00:	OP da,ca	kkdacakkkkkk
01:	OP nn,da,ca	kkdacakkkknn
10:	OP cc,da,ca	kkdacakkcckk
11:	OP a,c	aaaaaakkcccc

p--see preceding normal instruction format

Input/output operation types 00, 01, and 10 are the formats for the commands; type 11 is the format for a Data Control Word (DCW).

As an example of the use of OPD to generate an input/output command (using the above format for the variable field and defining the bits according to the rules for VFD), assume one wanted to generate the extant command, Write Tape Binary (WTB--Appendix E). This could be written as

        WTB   OPD   18/,O2/3,O6/15,10/0

or in various other bit-oriented forms.

OPSYN (Operation Synonym)

1	8	16	32
A sym-bol or opera-tion code	OPSYN		A mnemonic operation code in the variable field.

The OPSYN pseudo-operation is used for equating either a newly defined symbol or a presently defined operation to some operation code already in the operation table of the Assembler. The operation code may have been defined by a prior OPD or OPSYN pseudo-operation; in any case, it must be in the Assembler operation table. The new symbol to be defined is entered in the location field and the operation code that must be in the Assembler operation table is entered in the variable field. The new symbol must be defined (and so entered into the operation table) by the OPSYN pseudo-operation code before it is used as an operation code.

REFMA ON/OFF (Reference Macros)

1 2	8	16	32
Blanks	REFMA	OFF	Normal Mode

The use of the REFMA ON psuedo-operation causes the Assembler to create a separate symbol reference table for MACRO's. Each entry of this table consists of a MACRO name and the alter number(s) at which the name is referenced. If a MACRO is present but not referenced, it will not appear in the table.

For a MACRO to be referenced, REFMA ON must be specified prior to defining the MACRO. However, since the GMAP MACRO's are loaded automatically by the Assembler before this pseudo-operation appears, the LODM pseudo-operation must be used to load these MACRO's again if it is required to reference them. REFMA OFF causes the Assembler to stop referencing MACRO's.

Examples

    1.   To reference GECOS System MACROS:

```
REFMA      ON
LODM       .G3MAC
```

       All MACRO's under the name .G3MAC will be referenced until REFMA OFF is encountered.

    2.   To reference GMAP System MACROS:

```
REFMA      ON
LODM       .JMAC
```

       All MACRO's under the name .JMAC (GMAP MACRO's) will be referenced until REFMA OFF is encountered.

    3.   To reference program MACROS:

```
        REFMA      ON
SPLL    MACRO
        #1         1,DU
        STA        #3
        LDQ        0,DU
        ENDM
```

The symbolic name of the macro (in the location field of the MACRO identification) must be unique for the program in which the REFMA pseudo-operation is used. The use of this name in the location field at any other instruction, pseudo-operation, or macro-operation will result in a multidefined symbol error.

Location Counter Pseudo-Operations

USE (Use Multiple Location Counters)

1	8	16	32
Blanks	USE		A single symbol, blanks, or the word PREVIOUS in the variable field

The Assembler provides the ability to employ multiple location counters via the USE pseudo-operation. The use of this pseudo-operation causes the Assembler to place succeeding instructions under control of the location counter represented by the symbol in the variable

field. Each location counter begins with the value of zero, and its size is determined as being the highest value assumed by it (that is, occupied by some instruction assembled under it). This is not always the last instruction under the USE, as an ORG may have occurred within it. At the completion of the first pass through the symbolic program, the length of each USE will be a known value, and the order of their memory allocation will be implied by the order of their first presentation to the Assembler. Thus, the origin of each location counter may be computed based on the origin and size of the one preceding it. There is an assumed location counter, called the blank USE, implied in all assemblies, which has a natural origin of zero.

Automatic determination of a counter origin may be overridden with the BEGIN pseudo-operation. In this case, the chain of location counters will be made, completely ignoring those counters which had an associated BEGIN. In more general terms, then, the origin of a non-begin location counter is taken as one more than the highest value taken by the next prior non-begin counter. The first of these non-begin counters has an origin of zero, by definition. The location counter which is in control at the time that a USE is encountered is suspended at its current value and is preserved as the PREVIOUS counter. It may be called back into operation at any later point in the program without confusion as to its current state, and will begin counting at the address which is one higher than the last location used under it.

If the word PREVIOUS appears in the variable field, the Assembler reactivates the location counter which appeared just before the present one. It is not possible to go back more than one level via the USE PREVIOUS command, as the one in control when the USE PREVIOUS is encountered is made previous.

BEGIN (Origin of a Location Counter)

1	8	16	32
Blanks	BEGIN		Two subfields in the variable field

The BEGIN pseudo-operation is used to arbitarily specify the origin of a given location counter. As such, it will not be tied into the chain of location counters as described in USE. Its origin, however, may be an expression involving some symbol or symbols defined under another location counter, in which case it will be linked to the chain at the specified point. The user must beware of overlaying code with this pseudo-operation. It is primarily intended for the more sophisticated user. Under normal programming circumstances its power is not needed.

The location counter symbol is specified in the first subfield and is given the value specified by the expression found in the second subfield. Any symbol appearing in the second subfield must have been previously defined and must appear under one location counter. The BEGIN pseudo-operation may appear anywhere in the deck. It does not invoke the counter, however. A USE must be given to bring a location counter into effect.

ORG (Origin Set by Programmer)

1	8	16	32
Blanks or a symbol	ORG		An expression in the variable field

The ORG pseudo-operation is used by the programmer to change the next value of a counter, normally assigned by the Assembler, to a desired value. If ORG is not used by the programmer, the counter is initially set to zero.

All symbols appearing in the variable field must have been previously defined. If a symbol appears in the location field, it is assigned the value of the variable field. If the result of the evaluation of a variable field expression is absolute, the instruction counter will be reset to the specified value relative to the current location counter. If an expression result is relocatable, the current location counter will be suspended, and the counter to which the expression is relocated will be invoked with the value given by the expression.

LOC (Location of Output Text)

1	8	16	32
Blanks	LOC		An expression in the variable field

The LOC pseudo-operation functions identically to the ORG pseudo-operation, with one exception; it has no effect on the loading address when the Assembler is punching binary text. That is, the value of the location counter will be changed to that given by the variable field expression, but the loading will continue to be consecutive. This provides a means of assembling code in one area of memory while its execution will occur at some other area of memory.

All symbols appearing in the variable field of this pseudo-operation must have been previously defined.

The sole purpose of this pseudo-operation is to allow program coding to be loaded in one section of memory and then to be subsequently moved to another section for execution.

Symbol-Defining Pseudo-Operations

Increased facility in program writing frequently can be realized by the ability to define symbols to the Assembler by means other than their appearance in the location field of an instruction or by using a generative pseudo-operation. Such a symbol definition capability is used for (1) equating symbols, or (2) defining parameters used frequently by the program but which are subject to change. The symbol-defining pseudo-operations serve these and other purposes.

It should be noted that they do not generate any machine instructions or data but are available merely for the convenience of the programmer.

EQU (Equal To)

1	8	16	32
Symbol	EQU		An expression in the variable field

The purpose of the EQU pseudo-operation is to define the symbol in the location field to have the value of the expression appearing in the variable field. The symbol in the location field will assume the same mode as that of the expression in the variable field, that is, absolute or relocatable. (See Relocatable and Absolute Expressions.)

All symbols appearing in the variable field must have been previously defined and must fall under the same location counter. SYMDEF or SYMREF symbols cannot appear in the variable field.

If the asterisk (*) appears in the variable field denoting the current location counter value, it will be given the value of the next sequential location not yet assigned by the Assembler with respect to the unique location counter presently in effect.

FEQU (Special FORTRAN Equivalence)

1	8	16	32
Symbol	FEQU		A symbol in the variable field

The purpose of the FEQU pseudo-operation is to equate the symbol in the location field with the symbol in the variable field, the latter of which is as yet undefined. It was initially implemented to allow the FORTRAN IV compiler of the GE-600 Series software to generate more efficient code in certain cases where the value of a certain symbol was not immediately known. It was known that it would be defined before the compilation was complete, and as such, offers one advantage over the EQU pseudo-operations though it does carry restrictions as well.

The most stringent restriction is that the variable field may not contain an expression. Secondly, the symbol in the variable field may not subsequently appear in either field of another FEQU pseudo-operation. A third restriction is that if HEAD characters are in effect, both symbols (or neither symbol) must be able to be headed.

As implemented, both symbols are essentially held in abeyance until the variable field symbol is defined. At that point, both symbols take on the same value and characteristics, and are available for normal functions.

It should be noted that the symbol in the variable field does not have to be undefined. Nor does it have to be a symbol. It could be a number, or the current location counter value symbol (*). However, in these cases FEQU acts just as EQU, and the location symbol will be immediately defined with the indicated value.

CPB-1004F

191

BOOL (Boolean)

1	8	16	32
Symbol	BOOL		A Boolean expression in the variable field

The BOOL pseudo-operation defines a constant of 18 bits and is similar to EQU except that the evaluation of the expression in the variable field is done assuming Boolean operators. By definition, all integral values are assumed in octal and are considered to be in error otherwise. The symbol in the location field will always be absolute, and the presence of any expression other than an absolute one in the variable field will be considered an error. (See Relocatable and Absolute Expressions.)

All symbols appearing in the variable field must have been previously defined.

SET (Symbol Redefinition)

1	8	16	32
Symbol	SET		An expression in the variable field

The SET pseudo-operation permits the redefinition of a symbol previously defined to the Assembler. This ability is useful in Macro expansions where it may be undesirable to use created symbols (CRSM).

All symbols entered in the variable field must have been previously defined and must fall under the same location counter. SYMDEF or SYMREF symbols cannot be used in the variable field.

The symbol in the location field is given the value of the expression in the variable field. The SET pseudo-operation may not be used to define or redefine a relocatable symbol. (See Relocatable and Absolute Expressions.)

When the symbol occurring in the location field has been previously defined by a means other than a previous SET, the current SET pseudo-operation will be ignored and flagged as an error.

The last value assigned to a symbol by SET affects only subsequent in-line coding instructions using the redefined symbol.

MIN (Minimum)

1	8	16	32
Symbol	MIN		A sequence of expressions, separated by commas, in the variable field -- all of the same type; that is, relocatable or absolute

The MIN pseudo-operation defines the symbol in the location field as having the minimum value among the various values of all relocatable or all absolute expressions contained in the variable field.

All symbols appearing in the variable field must have been previously defined and must fall under the same location counter. SYMDEF or SYMREF symbols cannot be used in the variable field.

MAX (Maximum)

The MAX pseudo-operation is coded in the same format as MIN above. It defines the symbol in the location field as having the maximum value of the various expressions contained in the variable field.

All symbols appearing in the variable field must have been previously defined and must fall under the same location counter. SYMDEF or SYMREF symbols cannot be used in the variable field.

SYMDEF (Symbol Definition)

1	8	16	32
Blanks	SYMDEF		Symbols separated by commas in the variable field

The SYMDEF pseudo-operation is used to identify symbols which appear in the location field of a subprogram when these symbols are referred to from outside the subprogram (by SYMREF). Also, the programmer must provide a unique SYMDEF for use by the Loader to denote each subprogram entry point for the loading operations. The symbols used in the variable field of a SYMDEF instruction will be called SYMDEF symbols. Multiple defined SYMDEF symbols cannot occur since the Assembler ignores the current definition if it finds the same symbol previously entered in the SYMDEF table.

The appearance of a symbol in the variable field of a SYMDEF instruction indicates that:

1. The symbol must appear in the location field of only one of the instructions within the subroutine in which SYMDEF occurs.

2. The Assembler will place each such SYMDEF symbol along with its relative address in the preface card.

3. At load time, the Loader will form a table of SYMDEF symbols to be used for linkage with SYMREF symbols.

CPB-1004F

193

It is possible to classify SYMDEF symbols as primary and secondary. A secondary SYMDEF symbol is denoted by a minus sign in front of the symbol. The Loader will provide linkage for a secondary SYMDEF symbol only after linkage has been required to a primary SYMDEF within the same subprogram. The use of secondary SYMDEF symbols is intended for programmers who are specifically concerned with using the system subroutine library and generating routines for accessing the library. Secondary SYMDEF symbols are normally thought of as secondary entries to subroutines contained within a subprogram library package that will be used as an entire package. (The use of primary and secondary SYMDEF symbols is further described in the General Loader--GELOAD--manual.)

SYMREF (Symbol Reference)

1	8	16	32
Blanks	SYMREF		A sequence of symbols separated by commas entered in the variable field

The SYMREF pseudo-operation is used to denote symbols which are used in the variable field of a subprogram but are defined in a location field external to the subprogram. Symbols used in the variable field of a SYMREF instruction will be called SYMREF symbols.

When a symbol appears in the variable field of a SYMREF instruction, the following items apply:

1. The symbol should occur in the variable field of at least one instruction within the subroutine.

2. At assembly time the Assembler will enter the SYMREF symbol in the preface card of the assembled deck and place a special entry number (page 230) in the variable fields of all instructions in the referenced subroutine which contain the symbol.

3. At load time the Loader will associate the SYMREF symbol with a corresponding SYMDEF symbol and place the appropriate address in all instructions that have been given the special entry number.

Symbols appearing in the variable field of a SYMDEF instruction must not appear in the location field of any instruction within the subroutine in which SYMREF is used.

EXAMPLE OF SYMDEF AND SYMREF PSEUDO-OPERATIONS

Base Program or Subprogram			Referencing Subprogram		
	SYMDEF	ATAN,ATAN2		SYMREF	ATAN,ATAN2
ATAN2	STC2	INDIC		:	
ATANS	SAVE	0,1		:	
	SZN	INDIC		:	
	TZE	START	POLYX	FLD	X
	.			:	
	.			.	
ATAN	STZ	INDIC		TSX1	ATAN
	TRA	ATANS		:	
	:			TSX1	ATAN2
				:	

CPB-1004F

194

NULL (Null)

1	8	16	32
Symbol	NULL		The variable field is not interpreted.

The NULL pseudo-operation acts as an NOP machine instruction to the Assembler in that no actual words are assembled. A symbol on a NULL will be defined as current value of the location counter.

EVEN (Force Location Counter Even)

1	8	16	32
Symbol or blanks	EVEN		The variable field is not interpreted

The EVEN pseudo-operation accomplishes the same end result as the E in column 7. If the location counter is odd, a NOP is generated, thereby making it even. If there is a symbol in the location field it will be defined at the even address.

ODD (Force Location Counter Odd)

1	8	16	32
Symbol or blanks	ODD		The variable field is not interpreted

The ODD pseudo-operation acts as if an O has been punched in column 7. If the location counter is even, a NOP is generated, thereby making it odd. If there is a symbol in the location field it will be defined at the odd address.

CPB-1004F

195

EIGHT (Force Location Counter to a Multiple of 8)

1	8	16	32
Symbol or blanks	EIGHT		The variable field is not interpreted

The EIGHT pseudo-operation behaves as an <u>8</u> punched in column 7. If the location counter is not a multiple of 8, a TRA *+n is generated, where the value of *+n is the next location which is a multiple of 8, and the location counter is bumped by n. If there is a symbol in the location field it will be defined at the mod-8 address.

NOTE: In each of the 3 pseudo-operations, (EVEN, ODD, and EIGHT) the origin of the location counter will also be forced to a related address. For EVEN and ODD, it will be forced even, and for EIGHT, it will be forced to a multiple of eight.

Data Generating Pseudo-Operations

The Assembler language provides six pseudo-operations which can be used to generate data in the program at the time of assembly. These are BCI, OCT, DEC, ASCII, UASCI and VFD. The first five, BCI, OCT, ASCII, UASCI and DEC, are word-oriented while VFD is bit-oriented. There exists a fifth pseudo-operation, DUP, which in itself does not generate data, but through its repeat capability causes symbolic instruction and pseudo-operations to be iterated.

OCT (Octal)

1	8	16	32
Symbol or blanks	OCT		One or more subfields separated by commas appearing in the variable field, each one containing a signed or unsigned octal integer.

The OCT pseudo-operation is used to introduce data in octal integer notation into an assembled program. The OCT pseudo-operation causes the Assembler to generate n locations of OCT data where the variable field contains n subfields (n-1 commas). Consecutive commas in the variable field cause the generation of a zero data word, as does a comma followed by a terminal blank. Up to 12 octal digits plus the leading sign may make up the octal number.

The OCT configuration is considered true and will not be complemented on negatively signed numbers. The sign applies only to bit 0. All assembly program numbers are right-justified, retaining the integer form.

EXAMPLE OF OCT PSEUDO-OPERATION

OCT   1,-4,7701,+3,,-77731,04

If the current location counter were set at 506, the above would be printed out as follows (less the column headings):

Location	Contents	Relocation	
000506	000000000001	000	OCT 1,-4,7701,+3,,-77731,04
000507	400000000004	000	
000510	000000007701	000	
000511	000000000003	000	
000512	000000000000	000	
000513	400000077731	000	
000514	000000000004	000	

DEC (Decimal)

1	8	16	32
Symbol or blanks	DEC		One or more subfields in the variable field, separated by commas, each containing a decimal entry.

The Assembler language provides four types of decimal information which the programmer may specify for conversion to binary data to be assembled. The various types are uniquely defined by the syntax of the individual subfields of the DEC pseudo-operation. The basic types are single-precision, fixed-point numbers; single-precision, floating-point numbers; double-precision fixed point number. All fixed-point numbers are right-justified in the assembly binary words; floating-point numbers are left-justified to bit position eight with the binary point between positions 0 and 1 of the mantissa. (The rules for forming these numbers are described under Decimal Literals, page 160.)

EXAMPLES OF SINGLE-PRECISION DEC PSEUDO-OPERATION

GAMMA   DEC   3,-1,6.,.2E1,1B27,1.2E1B32,-4

The above would print out the following data words (without column headings), assuming that GAMMA is located at 1041.

Location	Contents	Relocation	
001041	000000000003	000	GAMMA DEC 3,-1, 6.,.2E1, 1B27, 1.2E1B32, -4
001042	777777777777	000	
001043	006600000000	000	
001044	004400000000	000	
001045	000000000400	000	
001046	000000000140	000	
001047	777777777774	000	

CPB-1004F

The presence of the decimal point and/or the E scale factor implies floating-point, while the added B (binary scale) implies fixed-point binary numbers. The absence of all of these elements implies integers. Several more examples follow (see decimal literals for further explanation):

DEC   -1B17,-1.,1000

With the location counter at 1050, the above would generate:

Location	Contents	Relocation	
001050	777777000000	000	DEC -1B17,-1.,1000
001051	001000000000	000	
001052	000000001750	000	

EXAMPLE OF DOUBLE-PRECISION DEC PSEUDO-OPERATION

BETA   DEC   .3D0,0.D0,1.2D1B68,1D-1

The location counter is at the address BETA (1060); the above subfields generate the following double words:

Location	Contents	Relocation	
001060	776463146314	000	BETA DEC .3D0,0.D0,
			1.2D1B68,1D-1
001061	631463146314	000	
001062	400000000000	000	
001063	000000000000	000	
001064	000000000000	000	
001065	000000000140	000	
001066	772631463146	000	
001067	314631463146	000	

BCI (Binary Coded Decimal Information)

1	8	16	32
Symbol or blanks	BCI		Two subfields in the variable field: a count subfield and a data subfield

The BCI pseudo-operation is used by the programmer to enter binary-coded decimal (BCD) character information into a program.

The first subfield is numeric and contains a count that determines the length of the data subfield. The count specifies the number of 6-character machine words to be generated; thus, if the count field contains n, the data subfield contains 6n characters of data. The maximum value which n can be is 9. The minimum value for n is 1.

The second subfield contains the BCD characters, six per machine word.

CPB-1004F

EXAMPLE OF BCI PSEUDO-OPERATION

BETA   BCI   3,NO ERROR CONDITION

Again assume the location counter set at 506 (location of BETA); the above would print out (less column headings):

Location	Contents	Relocation	
000506	454620255151	000	BETA BCI 3,NO ERROR CONDITION
000507	465120234645	000	
000510	243163314645	000	

ASCII, UASCI (ASCII Coded Information)

1	8	16	32
Symbol or Blanks	ASCII or UASCI		Two subfields in the variable field: a count subfield and a data subfield.

The ASCII and UASCI pseudo-operations are used by the programmer to enter lower case (ASCII pseudo-operation) and upper case ASCII character information into a program.

Appendix F contains the standard GE-625/635 conversion character set and the code generated by these pseudo-operations.

The first subfield is numeric and contains a count that determines the length of the data subfield. This count specifies the number of 4-character machine words to be generated. If the count is n, the data field contains 4 n characters. The maximum value for n is 14 and the minimum is 1.

The second subfield contains the ASCII characters, four per machine word.

EXAMPLE OF ASCII PSEUDO-OPERATION

BETA   ASCII 2, NO ERROR

Again assume the location counter set at 506 (location of BETA); the above would print out (less column headlings):

Location	Contents	Relocation	
000506	156157040145	000	BETA ASCII 2,
000507	162162157162	000	NO ERROR

VFD (Variable Field Definition)

1	8	16	32
Symbol or blanks	VFD		One or more subfields in the variable field separated by commas.

The VFD pseudo-operation is used for generation of data where it is essential to define the data word in terms of individual bits. It is used to specify by bit count certain information to be packed into words.

In considering the definition of a subfield, it is understood that the unit of information is a single bit (in contrast with the unit of information in the BCI pseudo-operation which is six bits). Each VFD subfield is one of three types: an algebraic expression, a Boolean expression, or alphanumeric (H or R). Each subfield contains a conversion type indicator and a bit count, the maximum value of which is 36. The bit count is an unsigned integer which defines the length of the subfield; it is separated from the data subfield by a slash (/). If the bit count is immediately preceded by an O or H, the variable-length data subfield is either Boolean or alphanumeric, respectively. In the absence of both the type indicators, O and H, the data subfield is an algebraic field. A Boolean subfield contains an expression that is evaluated using the Boolean operators (*,/,+,-).

R is an alphanumeric indicator which specifies right adjustment of the argument. Unused bit positions are zero filled. R can be used only in a VFD pseudo operation.

The data subfield is evaluated according to its form: algebraic, Boolean, or alphanumeric. A 36-bit field results. The low-order n bits of the algebraic or Boolean expression determine the resultant field value; whereas for the alphanumeric subfield the high-order n bits are used for H, and low-order n bits are used for R.

If the required subfields cannot be contained on one card, they must be continued by the use of the ETC pseudo-operation. This is done by terminating the variable field of the VFD pseudo-operation with a comma. The next subfield is then given as the beginning expression in the variable field of an ETC card. If necessary, subsequent subfields may be continued onto following ETC cards in the same manner. Except for the H type alphanumeric, the scanning of the variable field is terminated upon encountering the first blank character.

The VFD may generate more than one machine word; if the sum of the bit counts is not a multiple of a discrete machine word, the last partial string of bits will be left-justified and the word completed with zeros.

EXAMPLES OF VFD PSEUDO-OPERATION

Assume one would like to have the address ALPHA packed in the first 18 bits of a word, decimal 3 in the next 6 bits, the literal letter B in the next 6 bits, and an octal 77 in the last 6 bits. One could easily define it as follows:

VFD   18/ALPHA,6/3,H6/B,O6/77

With the location counter at 1053 and the location 731_8 assigned for ALPHA, this would print out (without column headings):

Location	Contents	Relocation
001053	000731032277	000	VFD 18/ALPHA,6/3,H6/B,06/77

NOTE: Relocation digits 000 refer to binary code data for A, BC, and DE of the relocation scheme. (Page 229.)

If ALPHA had been a program relocatable element, the relocation bits would have been 010; that is, the relocation scheme would have specified the left half of the word as containing a relocatable address. The relocation is only assigned if the programmer specifies a field width of 18 bits and has it left- or right-justified; in all other cases the fields are considered absolute. The total number of bits under a VFD need not be a multiple of full words nor is the total field (sum of all subfields) restricted to one word. The total field width, however, for a single subfield is 36 bits.

Consider a program situation where one wishes to generate a three-word identifier for a table. Assume n is the word length of the table and is equal to 12. You wish to place twice the length of the table in the first 12 bits, the name of the table in the next 60 bits, the location of the table (where TABLE is a program relocatable symbol equal to 2351_8) in the next 18 bits, zero in the next 8 bits, and -1 in the next 6 bits--all in a three-word key.

With the location counter at 1054.

VFD   12/2*12,H36/PRESSU,H24/RE,18/TABLE,8/,6/-1

will generate

Location	Contents	Relocation
001054	003047512562	000	VFD 12/2*12,H36/PRESSU,H24 /RE,18/TABLE,8/,6/-1
001055	626451252020	000
001056	002351001760	010

where 010 specifies the relocatability of TABLE.

DUP (Duplicate Cards)

1	8	16	32
Symbol or blanks	DUP		Two subfields in the variable field, separated by a comma

The DUP pseudo-operation provides the programmer with an easy means of generating tables and/or data. It causes the Assembler to duplicate a sequence (range) of instructions or pseudo-operations a specified number of times.

The first subfield in the variable field is an absolute expression which defines the count. The value of the count field specifies the number of cards, following the DUP pseudo-operation, that are included in the group to be duplicated. The value in the count field must be a decimal integer less than or equal to ten.

The second subfield of the pseudo-operation is an absolute expression which specifies the number of iterations. The value in the iteration field specifies the number of times the group of cards, following the DUP pseudo-operation, is to be duplicated. This value can be any positive integer less than 2^{18}-1. The groups of duplicated cards appear in the assembled listing immediately behind the original group.

If either the count field or the iteration field contains 0 (zero) or is null, the DUP pseudo-operation will be ignored.

If a symbol appears in the location field of the pseudo-operation, it is given the address of the next location to be assigned by the Assembler.

If an odd/even address is specified for an instruction within the range of a DUP pseudo-operation, the instruction will be placed in odd/even address and a filler used when needed. The filler will be an NOP instruction.

All symbols appearing in the variable field of the DUP pseudo-operation must have been previously defined. Any symbols appearing in the location field of the instructions being duplicated are defined only on the first iteration, thus avoiding multiply-defined symbols. SET would of course be the exception to this rule.

The only instructions or pseudo-operations which may not appear in the range of a DUP instruction are END, MACRO, and DUP. ETC may not appear as the first card after the range of a DUP.

Storage Allocation Pseudo-Operations

These pseudo-operations are used to reserve specified core memory storage areas within the coding sequence of a program for use as storage areas or work areas.

CPB-1004F

201

BSS (Block Started by Symbol)

1	8	16	32
Symbol or blanks	BSS		A permissible expression in the variable field defines the amount of storage to be reserved.

The BSS pseudo-operation is used by the programmer to reserve an area of memory within his assembled program for working and for data storage. The variable field contains an expression that specifies the number of locations the Assembler must reserve in the program.

If a symbol is entered in the location field, it is assigned the value of the first location in the block of reserved storage. If the expression in the variable field contains symbols, they must have been previously defined and must yield an absolute result. No binary cards are generated by this pseudo-operation.

BFS (Block Followed by Symbol)

1	8	16	32
Symbol or blanks	BFS		A permissible expression in the variable field defines the amount of storage to be reserved

The BFS pseudo-operation is identical to BSS with one exception. If a symbol appears in the location field, it is assigned the value of the first location after the block of reserved storage has been assigned.

BLOCK (Block Common)

1	8	16	32
Blanks	BLOCK		A symbol in the variable field

The purpose of the BLOCK pseudo-operation is to specify that program data following the BLOCK entry is to be assembled in the LABELED COMMON region of the user program under the symbol appearing in the variable field. BLOCK is, in effect, another location counter external to the text of the program.

The symbol in the variable field specifies the label of the COMMON area to be assembled. If the variable field is left blank, the normal FORTRAN BLANK COMMON is specified; and data following the BLOCK pseudo-operation will be assembled relative to the unlabeled (BLANK COMMON) memory area of the user program. It is not possible to assemble data or instructions into BLANK COMMON. Storage labeling and reservation is all that is permitted.

The pseudo-operations which take the program out of BLOCK mode and into some other mode are:

1. BLOCK (for some other LABELED COMMON)
2. USE
3. ORG/LOC, where the value of the expression is relocatable
4. END

It should be noted that BLOCK does not cause the Assembler to make the current USE location counter PREVIOUS. As such, a USE PREVIOUS following a BLOCK will cause the location counter which was in effect prior to the last USE to be invoked. A maximum of 63 labeled commons are permitted in a program.

LIT (Literal Pool Origin)

1	8	16	32
Symbol or blanks	LIT		Column 16 must be blank

The LIT pseudo-operation causes the Assembler to punch and print out all the previously developed literals. If the LIT instruction occurs in the middle of the program, the literals up to that point are output and printed out starting with the first available location after LIT; the literal pool is reinitialized as if the assembly had just begun.

If there are literals remaining in the pool when the END card is encountered, the origin of the literal pool will be one location past the final word defined by the program. The maximum number of LIT pseudo-operations that can occur in a program is 63.

Conditional Pseudo-Operations

The pseudo-operations INE, IFE, IFL, and IFG, which follow, are useful within MACRO prototypes to add flexibility to variable-length or conditional expansions of the MACRO prototype. When used within a MACRO, the conditional pseudo-operation can only be used to affect cards within the MACRO itself. The use of these pseudo-operations, however, is not limited to MACRO's; they can be used elsewhere in coding a subprogram to effect conditional assembly of segments of the program.

The programmer must avoid using noncomparable elements within these pseudo-operations. He must remember that the first comma encountered in the variable field is considered as separating the first subfield from the second subfield (the fields to be compared). Symbols used in the variable field will normally have been previously defined. On the other hand, one of the primary uses of conditionals is to test whether or not a symbol has been defined at a given point in an assembly. Consequently, undefined symbols within a conditional are not flagged in the left margin of the listing. If the symbol is never defined within the assembly, the symbol will be listed as undefined at the end of the listing; if the symbol is defined later in the assembly, it is not listed as undefined.

INE (If Not Equal)

1	8	16	32
Blanks	INE	X, Y, n	Two or three subfields in the variable field

The INE pseudo-operation provides for conditional assembly of the next n cards depending on the relationship of the first two subfields of the variable field. The value of the expression in the first subfield is compared to the value of the expression in the second subfield. If they are not equivalent, the next n cards are assembled, where n is specified in the third subfield; otherwise, the next n cards are bypassed, resumption beginning at the (n+1)th card. If the third subfield is not present, n is assumed to be one.

Two types of comparisons are possible in the subfields of the INE pseudo-operation. The first is an algebraic comparison after the expression has been evaluated. The second is alphanumeric comparison and the relation is the collating sequence. Alphanumeric strings in the variable field of INE are denoted by placing the subfield within apostrophe marks. If either the first or second subfield is designated as an alphanumeric string, the other will automatically be classified as such. Each alphanumeric subfield is right justified (with zero fill) within a 12-character field before comparison is made.

IFE (If Equal)

1	8	16	32
Blanks	IFE	X, Y, n	Two or three subfields in the variable field

The IFE pseudo-operation provides for conditional assembly of the next n cards depending on the relationship of the first two subfields of the variable field. The next n cards are assembled if and only if the expression or alphanumeric string in the first subfield is equal to the expression or alphanumeric string in the second subfield. If the compared subfields are not equal, the next n cards are bypassed. Resumption begins at card n+1. The n is specified in the third subfield and is assumed to be one if not present.

Two types of comparisons are possible in the subfields of the IFE pseudo-operation. The first is an algebraic comparison after the expression has been evaluated. The second is an alphanumeric comparison and the relation is the collating sequence. Alphanumeric strings in the variable field of IFE are denoted by placing the subfield within apostrophe marks. If either the first or the second subfield is designated as an alphanumeric string, the other is automatically classified as such. Each alphanumeric subfield is right justified (with zero fill) within a 12-character field before comparison is made.

IFL (If Less Than)

1	8	16	32
Blanks	IFL	X, Y, n	Two or three subfields in the variable field

The IFL pseudo-operation provides for conditional assembly of the next n cards, depending on the value of the first two subfields of the variable field. The next n cards are assembled if and only if the expression or alphanumeric string in the first subfield is less than the expression or alphanumeric string in the second subfield. If the first subfield is not less, the next n cards are bypassed. Resumption begins at card n+1. The n is specified in the third subfield and is assumed to be one if not present.

Two types of comparisons are possible in the subfields of the IFL pseudo-operation. The first is a straight numeric comparison after the expression has been evaluated. The second is an alphanumeric comparison, using the relation of the collating sequence. Alphanumeric strings in the variable field of IFL are denoted by placing the subfield within apostrophe marks. If either the first or second subfield is designated as an alphanumeric string, the other is automatically classified as such. Each alphanumeric subfield is right justified (with zero fill) within a 12-character field before comparison is made.

IFG (If Greater Than)

1	8	16	32
Blanks	IFG	X, Y, n	Two or three subfields in the variable field

The IFG pseudo-operation provides for conditional assembly of the next n cards, depending on the value of the first two subfields of the variable field. The next n cards are assembled if and only if the expression or alphanumeric string in the first subfield is greater than the expression or alphanumeric string in the second subfield. If the first subfield is not greater, the next n cards are bypassed. Resumption begins at card n+1. The n is specified in the third subfield and is assumed to be one if not present.

Two types of comparisons are possible in the subfields of the IFG pseudo-operation. The first is a straight numeric comparison after the expression has been evaluated. The second is an alphanumeric comparison, using the relation of the collating sequence. Alphanumeric strings in the variable field of the IFG are denoted by placing the subfield within apostrophe marks. If either the first or the second subfield is designated as an alphanumeric string, the other is automatically classified as such. Each alphanumeric subfield is right justified (with zero fill) within a 12-character field before comparison is made.

Special Word Formats

ARG A, M (Argument--Generate Zero Operation Code Computer Word)

1	8	16	32
Symbol	ARG		Two subfields in the variable field

The use of ARG in the operation field causes the Assembler to generate a binary word with bit configuration in the general instruction format. The operation code 000 is placed in the operation field. The variable field is interpreted in the same manner as a standard machine instruction.

NONOP (Undefined Operation)

When an undefined operation is encountered, NONOP is looked up in the operation table and used in place of the undefined operation. NONOP is initially set as an error routine, but the programmer through the use of OPD, OPSYN OR MACRO may redefine NONOP to his own purpose. For example, NONOP could be redefined by the use of a MACRO to be a MME to GECHEK with a dump sequence, or it could be made equivalent to the ARG pseudo-operation.

CPB-1004F

205

ZERO B, C (Generate One Word With Two Specified 18-bit Fields)

1	8	16	32
Symbol or blanks	ZERO		Two subfields in the variable field

The pseudo-operation ZERO is provided primarily for the definition of values to be stored in either or both the high- or low-order 18-bit halves of a word. The Assembler will generate the binary word divided into the two 18-bit halves; bit positions 0-17 and 18-35. The equivalent binary value of the expression in the first subfield will be in bit positions 0-17. The equivalent binary value of the expression in the second subfield will be in bit positions 18-35. Literals are not allowed in the variable field of the ZERO pseudo-operation.

MAXSZ (Maximum Size of Assembly)

1	8	16	32
Blank	MAXSZ		A decimal number in the variable field

The decimal number represents the programmer's estimate of the largest number of assembled instructions and data in his program or subprogram. The variable field number is evaluated, saved, and printed out at the end of the assembly listing. It can then be compared with the actual size of the assembly.

MAXSZ is provided as a programmer convenience and can be inserted anywhere in his coding.

Address Tally Pseudo-Operations

The Indirect then Tally (IT) type of address modification in several cases requires special word formats which are not instructions and do not follow the standard word format. The following pseudo-operations are for this purpose. (Refer to page 169 and following.)

● TALLY A,T,C (Tally)   Used for ID, DI, SC, and CI type of tally modification, where SC and CI are for 6 bit characters. The first subfield is the address for the indirect reference, T is the tally count, and C is the character position ($0 \leq C \leq 5$). When used with the CI modifier the contents of the tally count subfield (T) is not interpreted.

● TALLYB A,T,B (Tally Byte)   Used for SC and CI type of tally modification, where 9 bit bytes (characters) are desired. A and T are the same as for TALLY and B indicates the byte position ($0 \leq B \leq 3$).

● TALLYD A,T,D, (Tally and Delta)   Used for Add Delta (AD) and Subtract Delta (SD) modification. A is the address, T the tally, and D the delta of incrementing.

● TALLYC A,T,mod (Tally and Continue)   Used for Address, Tally, and Continue. A is the address, T the tally count, and mod the address modification as specified under normal instructions.

CPB-1004F

206

Repeat Instruction Coding Formats

The Repeat (RPT), Repeat Double (RPD), and Repeat Link (RPL) machine instructions and variations of these instructions use special formats and have special tally, terminate repeat, and other conditions associated with them. The machine instructions describing these conditions appear on pages 134 through 144. There is no address modification for the repeat instructions (see pages 134 through 144). Address modifications for the repeated instructions are limited to R and RI with designators specifying X1, $\cdots$, X7. Index register zero is used to control terminate conditions and tally. The coding formats for this family of instructions are as follows:

RPT N, I, k_1, k_2, $\cdots$, k_7 The command generated by the Assembler from this format will cause the instruction immediately following the command to be iterated N times and that instruction's effective address to be incremented by the value I for each of N iterations. The range for N is O-225. If N=O, the instruction will be iterated 256 times. If N is greater than 256, the instruction will cause an error flag (A) to be produced in the assembly listing. The fields k_1, k_2, . . ., k_7 may or may not be present. They represent conditions for termination which, when needed, are declared by the conditional transfer symbols TOV, TNC, TRC, TMI, TPL, TZE, and TNZ. These symbols affect the termination condition bits in positions 11 through 17 of the repeat instruction.

It is also possible to use an octal number rather than the special symbols to denote termination conditions. Thus, if the field for k_1, k_2, . . ., k_7 is found to be numeric, it will be interpreted as octal and the low order seven bits will be ORed into bit positions 11 through 17 of the repeat instruction. The variable field scan will be terminated with the octal field.

RPTX ,I This instruction behaves just as the RPT instruction with the exception that N and the conditions for termination are loaded by the programmer into bit positions 0 through 7 and 11 through 17, respectively, of index register zero (instead of embedded in the instruction).

RPD N, I, k_1, k_2, . . ., k_7 The command generated by the Assembler from this format will cause the two instructions immediately following the RPD instruction to be iterated N times and the effective address of those two instructions to be incremented by the value I for each of N iterations. The meaning of k_1, k_2, . . ., k_7 is the same as for the RPT instruction. Since the double repeat must fall in an odd location, the Assembler will force this condition and use a NOP instruction for a filler when needed.

RPDX ,I This instruction behaves just as the RPD instruction with the exception that N and the conditions for termination are loaded by the programmer into the index register zero.

RPDA N, I, k_1, k_2, $\cdots$, k_7 This instruction behaves just as the RPD instruction with the exception that only the effective address of the first instruction following the RPDA instruction will be incremented by the value of I for each of N iterations.

RPDB N, I, k_1, k_2, $\cdots$, k_7 This instruction behaves just as the RPD instruction with the exception that only the effective address of the second instruction following the RPDB instruction will be incremented by the value I for each of N iterations.

RPL N, k_1, k_2, $\cdots$, k_7 This format will cause the instruction immediately following it to be repeated N times or until one of the conditions specified in k_1, . . ., k_7 is satisfied. The address effectively used by the repeated instruction is the linked address described on pages 141 through 144.

RPLX This instruction behaves just as the RPL instruction with the exception that N and the conditions for termination are loaded by the programmer into index register zero.

MACRO-OPERATIONS

Introduction

Programming applications frequently involve (1) the coding of a repeated pattern of instructions that within themselves contain variable entries at each iteration of the pattern and (2) basic coding patterns subject to conditional assembly at each occurrence. The macro-operation gives the programmer a shorthand notation for handling (1) and (2) through the use of a special type of pseudo-operation referred to in the GE-625/635 Macro Assembler as a MACRO. Having once determined the iterated pattern, the programmer can, within the MACRO, designate selectable fields of any instruction of the pattern as variable. Thereafter, by coding a single MACRO instruction, he can use the entire pattern as many times as needed, substituting different parameters for the selected subfields on each use.

When he defines the iterated pattern, the programmer gives it a name, and this name then becomes the operation code of the MACRO instruction by which he subsequently uses the macro-operation.

As a generative operation, the macro-operation causes n card images (where n is normally greater than one) to be generated; these may have substitutable arguments. The MACRO is known as the prototype or skeleton, and the card images that may be defined are relatively unrestricted as to type.

They can be:

1. Any Processor instruction
2. Almost any Assembler pseudo-operations
3. Any previously defined macro-operation

Card images of these types are subject to the same conditions and restrictions when generated by the macro processor as though they had been produced directly by the programmer as in-line coding.

To use the MACRO prototype, once named, the programmer enters the macro-operation code in the operation field and arguments in the variable field of the MACRO instruction. (The arguments comprise variable field subfields and refer directly to the argument pointers specified in the fields of the card images of the prototype.) By suitably selecting the arguments in relation to their use in the prototype, the programmer causes the Assembler to produce in-line coding variations of the n card images defined within the prototype.

The effect of a macro-operation is the same as an open subroutine in that it produces in-line code to perform a predefined function. The in-line code is inserted in the normal flow of the program so that the generated instructions are executed in-line with the rest of the program each time the macro-operation is used.

An important feature in specifying a prototype is the use of macro-operations within a given prototype. The Assembler processes such "nested" macro-operations at expansion time only. The nesting of one macro definition within another prototype is not permitted. If macro-operation codes are arguments, they must be used in the operation field for recognition. Thus, the MACRO must be defined before its appearance as an argument; that is, the prototype must be available to the Assembler before encountering a demand for its usage.

CPB-1004F

208

Definition of the Prototype

The definition of a MACRO prototype is made up of three parts:

1. Creation of a heading card that assigns the prototype a name
2. Generation of the prototype body of n card images with their substitutable arguments
3. Creation of a prototype termination card

These parts are described in the following three paragraphs.

MACRO (MACRO Identification)

1	8	16	32
Symbol	MACRO		Blanks in the variable field

The MACRO pseudo-operation code is used to define a macro-operation by symbolic name. The symbol in the location field conforms to standard symbol formation rules and defines the name of a MACRO whose prototype is given on the next n lines. (The prototype definition continues until the Assembler encounters the proper ENDM pseudo-operation.) The name of the MACRO is a required entry. If the symbol is identical to an operation code already in the table, the macro-operation will be used as a new definition for that operation code. It is entered in the Assembler operation table with a pointer to its associated prototype that is entered in the MACRO prototype table.

ENDM (End MACRO)

1	8	16	32
Blanks	ENDM		A symbol in the variable field

The symbol in the variable field is the symbolic name of the MACRO instruction as defined in the location field of the corresponding MACRO heading card. Every MACRO prototype must contain both the terminal ENDM pseudo-operation and the MACRO pseudo-operation.

Thus, every prototype will have the form

Heading card	{	OPNAME	MACRO
Prototype body	{		-------- -------- -------- -------- . . . -------- --------
Terminal card	{		ENDM OPNAME
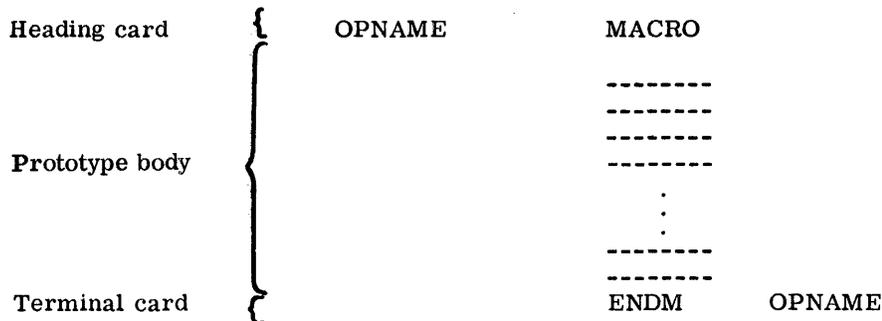
where OPNAME represents the prototype name that is placed in the Assembler operation table.

CPB-1004F

209

● Prototype Body. The prototype body contains a sequence of standard source-card images (of the types listed earlier) that otherwise would be repeated frequently in the source program. Thus, for example, if the iterated coding pattern

LOCATION	E O	OPERATION		ADDRESS, MODIFIER	COMMENTS
1 2 6	7	8 14	15	16	32
		:			
		LDA		5, DL	
		LDQ		13, DL	
		CWL		ALPHA, 2	
		TZE		FIRST	
		:			
		:			
		:			
		LDA		U	
		LDQ		V	
		CWL		BETA, 4	
		TZE		SCND	
		:			
		:			
		:			
		LDA		W+X	
		LDQ		Y+Z	
		CWL		GAMMA	
		TZE		NEXT1	

appeared in a subprogram, it could be represented by the following prototype body (preceded by the required prototype name):

1	8	16	32
CMPAR	MACRO		MACRO prototype with substitutable
	LDA	#1	arguments in the variable field
	LDQ	#2	
	CWL	#3	
	TZE	#4	
	ENDM	CMPAR	

Then the previous coding examples could be represented by the macro-operation CMPAR as follows:

```
CMPAR                    (5,DL),(13,DL),(ALPHA,2),FIRST
---
---
CMPAR                    U,V,(BETA,4),SCND
---
---
CMPAR                    W+X,Y+Z,GAMMA,NEXT1
```

The Assembler recognizes substitutable arguments by the presence of the number-sign identifer (#). Having sensed this identifier, it examines the next one or two digits. (Sixty-three is the maximum number of arguments usable in a single prototype.)

MACRO prototype arguments can appear in the location field, in the operation field, in the variable field, and coincidentally in combinations of these fields within a single card image. Substitutions that can be made in these fields are:

1. Location field--any permissible location symbol (see comments below)

2. Operation field--all machine instructions, all pseudo-operations (except the MACRO pseudo-operation) and previously defined macro operations

3. Variable field--any allowable expression followed by an admissible modifier tag and separated from the expression by a delimiting comma.

In general, anything appearing to the right of the first blank in the variable field will not be copied into the generated card image. For example, a substitutable argument appearing in the comments field of a card image--that is, separated from the variable field by one or more blanks--will not be interpreted by the Assembler (except in the case of the ASCII, BCI, REM, TTL, TTLS, and UASCI pseudo-operations). This means that only pertinent information in the location, operation and variable fields is recognized, that internal blanks are not allowed in these fields, and that the first blank in these fields causes field termination.

When specifying a symbol in a location field of an instruction within a prototype the programmer must be aware that this MACRO can be used only once since on the second use the same symbol will be redefined, causing a multiple-defined symbol. Consequently, the use of location symbols within the prototype is discouraged. Alternatively, for cases where repeated use of a prototype is necessary, two techniques are available: (1) use of Created Symbols and (2) placement of substitutable argument in the location field and use of a unique symbol in the argument of the macro operation each time the prototype is used. These techniques are described under Using a MACRO operation, on the following page.

The location field, operation field, and variable field may contain text and arguments which can be linked by simply entering the substitutable argument (for example, AB#3) directly in the text with no blanks or special symbols preceding or following the entry. Linking is especially useful in the operation field and in the partial subfields of the variable field. (Refer to the discussion of ASCII, BCI, REM, TTL, TTLS, and UASCI immediately following.) As an example of the first use, consider a machine instruction such as LD(R) where R can assume the designators A, Q, AQ, and X0-X7.

The prototype NAME

```
NAME          MACRO
              -------
              -------
              LD#2
              -------          A,#1
              -------
              ENDM            NAME
```

contains a partial operation field argument; and when the in-line coding is generated, LD#2 becomes LDA, LDQ, etc., as designated by the argument used in the macro operation.

The ASCII, BCI, REM, TTL, TTLS, and UASCI pseudo-operations used within the prototype are scanned in full for substitutable arguments. The variable field of these pseudo-operations can contain blanks and argument pointers. The following illustrates a typical use:

```
ALPHA         MACRO
              -------
              -------
              -------
NOTE#1        REM          IGNOREⱶ #2ⱶERRORSⱶONⱶ#3
              -------
              -------
              ENDM         ALPHA
```

An asterisk (*) type comment card cannot appear in a MACRO prototype.

Using a MACRO Operation

Use of a MACRO operation can be divided into two basic parts; definition of the prototype and writing the MACRO operation. The first part has been described on the preceding pages; writing the macro operation to call upon the prototype is the process of using the MACRO and is described in the following paragraphs.

The macro operation card is made up of two basic fields; the operation field that contains the name of the prototype being referenced and the variable field that contains subfield arguments relating to the argument pointers of the prototype on a sequential, one-to-one basis. For example, the defined prototype CMPAR, mentioned earlier, could be called for expansion by the MACRO instruction

```
CMPAR              U,V,(BETA,4),SCND
```

where the variable field arguments, separated by commas and taken left-to-right, correspond with the prototype pointers #1 through #4. These arguments are then substituted in their corresponding positions of the prototype to produce a sequence of instructions using these arguments in the assigned location, operation, and variable fields of the prototype body. (The above MACRO instruction expands to the coding shown on page 210.)

The maximum number of MACRO call arguments is 63; arguments greater than 63 are treated modulo 64. For example, the 70th argument is the same as the 6th argument and would be so recognized by the Assembler. Each such argument can be a literal, a symbol, or an expression (delimited by commas) that conforms to the restrictions imposed upon the field of the machine instruction or pseudo-operation within the prototype where the argument will be inserted.

The following conditions and restrictions apply to the expansion of MACROs:

1. Anything appearing in the location field of a prototype card image, whether text or a substitutable argument, causes generation to begin in column 1 for that text or argument.

2. Location field text generated from an argument pointer (in a prototype location field) so as to produce a resultant field extending beyond column 8 causes the operation field to begin in the next position after the generated text. Normally, the operation field will begin in column 8.

3. Operation field text generated from an argument pointer (in a prototype operation field) so as to produce a resultant field extending beyond column 16 causes the variable field to start in the next position after the generated text. Normally, the variable field will begin in column 16.

4. The variable field may begin after the first blank that terminates the operation field but not later than column 16 in the absence of the condition in 3 above.

5. No generated card image can have more than 72 characters recorded; that is, the capacity of one card image cannot be exceeded (columns 73-80 are not part of the card image).

6. No argument string of alphanumeric characters can exceed 57 characters.

7. Up to 63 levels of MACRO nesting are permitted.

An argument can also be declared null by the programmer when writing the MACRO instruction; however, it must be declared explicitly null. Explicitly null arguments of the MACRO instruction argument list can be specified in either of two ways; by writing the delimiting commas in succession with no spaces between the delimiters or by terminating the argument list with a comma with the next normal argument of the list omitted. (Refer to the CRSM description, following.) A null argument means that no characters will be inserted in the generated card image wherever the argument is referenced. When a macro operation argument relates to an argument pointer and the pointer requires the argument to have multiple entries or contains blanks, the corresponding argument must be enclosed within parentheses with the parenthetical argument set off by the normal comma delimiters. The parenthetical argument can contain commas as separators. Examples of prototype card images that require the use of parentheses in the MACRO call are pseudo-operations such as IDRP, VFD, BCI, and REM, as well as the variable field of an instruction where the address and tag may be one argument. In these cases the elements of the arguments contained within the parentheses are called subarguments.

It is also possible to enclose an argument within brackets, making them subarguments, in which case blanks are ignored as part of the argument. For example the MACRO call of the MACRO named ABC can be written as

```
ABC       [A,
ETC       24,
ETC       2*D]
```

and is equivalent to

```
ABC       (A, 24, 2*D)
```

even though numerous blanks occur after the arguments A, and 24,. Thus, the Assembler packs everything it finds within brackets and suppresses all blanks therein. The above manner of writing the MACRO call permits the programmer additional flexibility in placing one subargument per card by means of using ETC, the blanks no longer being significant.

It can happen that the argument list of a macro operation extends beyond the capacity of one card. In this case, the ETC pseudo-operation is used to extend the list on to the next card. In using ETC, the last argument entry of the macro operation is delimited by a following comma, and the first entry of the ETC card is the next argument in the list. Within the prototype, as many ETC cards as required can be used for internal MACROs or VFD pseudo-operations.

Pseudo-Operations Used Within Prototypes

● Need for Prototype Created Symbols. In case of a MACRO prototype in which an argument pointer is used in the location field, the programmer must specify a new symbol each time the prototype is called. In addition, for those cases where a nonsubstitutable symbol is used in a prototype location field, the programmer can use the macro operation only once without incurring an Assembler error flag on the second and all subsequent calls to the prototype (multiply-defined symbol). Primarily to avoid the former task (having to repeatedly define new symbols on using the macro operation) and to enable repeated use of a prototype with a location field symbol (nonsubstitutable), the created symbol concept is provided.

● Use of Created Symbols. Created symbols are of the type .xxx. where xxx runs from 001 through 999, thus making possible up to 999 created symbols for an assembly. The periods are part of the symbol. The Assembler will generate a created symbol only if an argument in the macro operation is implicitly null; that is, only if the macro operation defines fewer arguments than given in the related MACRO prototype or if the designator # is used as an argument. Explicitly null arguments will not cause created symbols to be generated. The example given clarifies these ideas.

Assume a MACRO prototype of the form

```
NAME        MACRO
            -------     #1,#2
#4          -------     X
#5          -------     ALPHA,#3
            -------     #4
            TMI         #5
            ENDM        NAME
```

with five arguments, 1 through 5. The macro operation NAME in the form

```
NAME            A,7,,,B
```

specifies the third and fourth arguments as explicitly null; consequently, no created symbols would be provided. The expansion of the operation would be

```
            -------     A,7
            -------     X
B           -------     ALPHA,      (Unless a specified modification is given,
            -------                  X_0 will be assumed.)
            TMI         B
```

The macro operation card

```
NAME        A,7,
```

indicates the third argument is explicitly null, while arguments four and five are implicitly

null. Consequently, created symbols would be provided for arguments four and five but not for three. This is shown in the expansion of the macro operation as follows:

```
           -------        A,7
.011.      -------        X
.012.      -------        ALPHA,      (Unless a specified modification is given,
           -------        .011.       XRO will be assumed.)
           TMI           .012
```

A created symbol could be requested for argument three simply by omitting the last comma. The programmer can conveniently change an explicitly null argument to an implicitly null one by inserting the # designator in an explicitly null position. Thus, for the preceding example

```
      NAME            A,7,,#,B
```

the fourth argument becomes implicitly null and a created symbol will be generated.

CRSM ON/OFF (Created Symbols)

1	8	16	32
Blanks	CRSM	ON	Normal mode

Created symbols are generated only within MACRO prototypes. They can be generated for argument pointers in the location, operation, and variable fields of instructions or pseudo-operations that use symbols. Accordingly, the created symbols pseudo-operation affects only such coding as is produced by the expansion of MACROs. CRSM ON causes the Assembler to initiate or resume the creation of symbols; CRSM OFF terminates the symbol creation if CRSM ON was previously in effect. If the Assembler is already in the specified mode, the pseudo-operation is ignored.

ORGCSM (Origin Created Symbols)

1	8	16	32
Blanks	ORGCSM		One expression in the variable field.

The variable field is evaluated and becomes the new starting value between the decimal points of the created symbols.

IDRP (Indefinite Repeat)

1	8	16	32
Blanks	IDRP	#3	An argument number or blanks in the variable field, depending on the IDRP of the IDRP pair

The purpose of the IDRP is to provide an iteration capability within the range of the MACRO prototype by letting the number of grouped variables in an argument pointer determine the iteration count.

The IDRP pseudo-operation must occur in pairs, thus delimiting the range of the iteration within the MACRO prototype. The variable field of the first IDRP must contain the argument number that points to the particular argument used to determine the iteration count and the variables to be affected. The variable field of the second IDRP must be blank.

At expansion time, the programmer denotes the grouping of the variables (subarguments) of the iteration by placing them, contained in parentheses, as the nth argument where n was the argument value contained in the initial IDRP variable field entry.

IDRP is limited to use within the MACRO prototype, and nesting is not permitted. However, as many disjoint IDRP pairs may occur in one MACRO as the programmer wishes.

For example, given the MACRO skeleton

```
NAME            MACRO
                  .
                  .
                  .
                IDRP      #2
                ADA       #2
                IDRP
                  .
                  .
                ENDM      NAME
```

the MACRO call (with variables X1, X2, and X3)

```
A               NAME    Q+2, (X1, X2, X3), B
```

would generate

```
A                         .
                          .
                          .
                ADA       X1
                ADA       X2
                ADA       X3
                          .
                          .
```

In the example, arguments #1 and #3, Q+2, and B respectively, are used in the skeleton ahead of and after the appearance of the IDRP, range-iteration pair.

DELM (Delete MACRO)

1	8	16	32
Symbol or Blanks	DELM		A symbol in the variable field

The function of this pseudo-operation is to delete the MACRO named in the variable field from the MACRO prototype area, and disable its corresponding operation table entry. Through the use of this pseudo-operation, systems which require many, or large MACRO prototypes, or which have minimal storage allocation at assembly time, can re-use storage in the prototype area for redefining or defining new MACROs. Redefinition of a deleted MACRO will not produce an M multiple defined flag on the assembly listing.

Implementation of System MACRO's

GMAP can load a unique set (or sets) of MACRO's under control of a pseudo-operation. This permits the various langauge processors to uniquely identify the standard system MACRO's required for the assembly of their generated code.

GMAP itself has a set of system MACRO's which it loads as part of its initialization procedures. This includes FILCB, the GEFRC File Control Block MACRO (see GE-625/635 File and Record Control, CPB-1003), SORT and MERGE (see GE-625/635 Sort/Merge Program, CPB-1005) and the DEBUG Symbol Table MACRO's VTAB and LTAB (see GE-625/635 General Loader, CPB-1008). Loading of these MACRO's is dependent upon the elected option on the $ GMAP control card. The option GMAC/NGMAC instructs GMAP to load or not load its own system MACRO's in initializing for assembly. The absence of either option is equivalent to having elected GMAC, hence the normal user of GMAP does not need to be aware of the fact that GMAP MACRO's are optionally loaded.

System MACRO's are, by definition, located on the System File on the high speed drum. They are put there by the System Editor, in System Loadable Format, as a freestanding system program. Their catalog name is that which is to be used by GMAP in the loading operation. For proper implementation, the MASTER option of the System Editor parameters card must be elected. It may be in absolute or relocatable System Loadable Format.

This implementation technique permits any unit, or functionally related group of users of GMAP to define and implement a unique set of System MACRO's; or on a larger scale, it allows various GE-600 installations to install local standard sets of MACRO's, without changing the Assembler.

PUNM (Punch MACRO Prototypes and Controls)

1	8	16	32
Blanks	PUNM		The variable field is not examined

This pseudo-operation causes the Assembler, in pass one, to scan the operation table for all MACRO's defined. It then appends their definitions to the end of the prototype table and constructs a control word specifying the length of this area and the number of MACRO's defined therein.

At the beginning of pass two, this information is punched onto relocatable binary instruction cards, along with $ OBJECT, preface, and $ DKEND cards. The primary SYMDEF of this deck will arbitrarily be .MACR. .

CPB-1004F

217

In the normal preparation of System MACRO's, it would not be desirable to include the GMAP System MACRO's. For this reason, the assembly of a set of System MACRO's should have NGMAC elected on its $ GMAP card.

LODM (Load System MACROs)

1	8	16	32
Blanks	LODM		A symbol in the variable field

This pseudo-operation causes the Assembler to issue an MME GECALL for a set of System MACROs. The name used in the GECALL sequence is the symbol taken from the variable field of the LODM pseudo-operation. MACROs thus loaded will be appended to (not overlay) the MACRO prototype table. They will be defined and made available for immediate use. If a MACRO is redefined by this operation the LODM instruction will be flagged with an M.

Notes and Examples on Defining a Prototype

The examples following show some of the ways in which MACROs can be used.

●   Field Substitution

    Prototype definition:

```
        ADDTO           MACRO
                        LDA         #1
                        ADA         #2
                        STA         #3
                        ENDM        ADDTO
Use:
                        ADDTO       A,(1,DL),B+5
```

●   Linkage of Text and Arguments

    Prototype definition:

```
        INCX            MACRO
                        ADLX#2      #3,DU
                        INE         #1,'*+1'
                        TRA         #1
                        ENDM        INCX
Use:
                        INCX        LOCA,4,1
    or
                        INCX        *+1,4,1
```

- **Argument in a BCI Pseudo-Operation**

    Prototype definition:

ERROR	MACRO	
	TSX1	DIAG
	ARG	#1
	BCI	5,ERRORⱡ#1ⱡCONDITIONⱡIGNORED
	ENDM	ERROR

    Use:

	ERROR	5

- **MACRO Operation in a Prototype**

    Prototype definition:

TEST	MACRO	
	LDA	#1
	CMPA	#2
	#3	#4
	ERROR	#5
	ENDM	TEST

    Use:

	TEST	A,B,TZE,ALPHA,3

- **Indefinite Repeat**

    Prototype definition (for generating a symbol table):

SYMGEN	MACRO	
	IDRP	#1
#1	BCI	1,#1
	IDRP	
	ENDM	SYMGEN

    Use:

	SYMGEN	(LABEL,TEST,ERROR,MACRO)

- **Subroutine Call MACRO**

    Prototype definition:

DOO	MACRO	
K	SET	0
	IDRP	#2
K	SET	K+1
	IDRP	
	TSX1	#1
	TRA	*+1+K
	IDRP	#2
	ARG	#2
	IDRP	
	ENDM	DOO

    Use:

	DOO	SRT,(ARG1,ARG2,ARG3)

CPB-1004F

219

PROGRAM LINKAGE PSEUDO-OPERATIONS

CALL (Call--Subroutines)

1	8	16	32
Symbol or blanks	CALL		Subfields in the variable field with contents and delimiters as described below

The CALL pseudo-operation is used to generate the standard subroutine calling sequence.

The first subfield in the variable field of the instruction is separated from the next n subfields by a left parenthesis. This subfield contains the symbol which identifies the subroutine being called. It is possible to modify this symbol by separating the symbol and the modifier with a comma. (In a Relocatable Assembly the symbol entered in this subfield is treated as if it were entered in the variable field of a SYMREF instruction.)

The next n subfields are separated from the first subfield by a left parenthesis and from subfield n+1 by a right parenthesis. Thus the next n subfields are contained in parentheses and are separated from each other by commas. The contents of these subfields are arguments which will be used in the subroutine being called.

The next m subfields are separated from the previous subfields by a right parenthesis and from each other by commas. These subfields are used to define locations for error returns from the subroutine. If no error returns are needed, then m=0.

The last subfield is used to contain an identifier for the instruction. This identifier is used when a trace of the path of the program is made. The identifier may be an expression contained in apostrophes. Thus the last subfield is separated from the previous subfields by an apostrophe. If the last subfield is omitted, the assembly program will provide an identifier which is the assigned alter number of the CALL pseudo-operation itself.

In the examples following, the calling sequences generated by the pseudo-operation are listed below the CALL pseudo-operation. For clarification AAAAA defines the location the CALL instruction; SUB is the name of the subroutine called; MOD is an address modifier; A1 through An are arguments; E1 through Em define error returns; E.I. is an identifier; and .E.L.. defines a location where error linkage information is stored. The number sequences 1,2,...,n and 1,2,...,m designate argument positions only.

```
AAAAA     CALL      SUB,MOD(A1,A2,....,An)E1,E2,.........Em'E.I.'

AAAAA     TSX1      SUB,MOD
          TRA       *+2+n+m
          ZERO      .E.L..,E.I.
          ARG       A1
          ARG       A2
            .
            .
            .
          ARG       An
          TRA       Em
            .
            .
            .
          TRA       E2
          TRA       E1
```

The preceding example of instructions generated by the CALL pseudo-operation was in the relocatable mode. The following example is in the absolute mode.

AAAAA	CALL	SUB,MOD(A1,A2,....,An)E1,E2,.....,Em'E.I.'

AAAAA	TSX1	SUB,MOD
	TRA	*+2+n+m
	ZERO	0,E.I.
	ARG	A1
	ARG	A2
	.	
	.	
	.	
	ARG	An
	TRA	Em
	.	
	.	
	.	
	TRA	E2
	TRA	E1

If the variable field of the CALL cannot be contained on a single line of the coding sheet, it may be continued onto succeeding lines by use of the ETC pseudo-operation. This is done by terminating the variable field of the CALL instruction with a comma (,). The next subfield is then placed as the first subfield of the ETC pseudo-operation. Subsequent subfields may be continued onto following lines in the same manner.

When a CALL to an external subprogram appears within a headed section, the external subprogram must be identified by a six-character symbol (immune to HEAD).

If a CALL is being used to access an internally defined subroutine, the subroutine must be placed ahead of the CALL in the program deck. Also, a SYMDEF pseudo-operation with the symbol identifying the subroutine in its variable field must be placed ahead of the CALL in the program deck. Starting the subroutine with a SAVE pseudo-operation automatically provides the SYMDEF.

SAVE (Save--Return Linkage Data)

1	8	16	32
Symbol	SAVE		Blanks or subfields separated by commas in the variable field--as described below

The SAVE pseudo-operation is used to produce instructions neccessary to save specified index registers and the contents of the error linkage index register.

The symbol in the location field of the SAVE instruction is used for referencing by the RETURN instruction. (This symbol is treated by the Assembler as if it had been coded in the variable field of a SYMDEF instruction when the Assembler is in the relocatable mode.)

The subfields in the variable field, if present, will each contain an integer 0-7. Thus each subfield specifies one index register to be saved.

When the SAVE variable field is blank, the following coding is generated:

```
NAME        TRA      *+2
            RET      .E.L. .
            STI      .E.L. .
            STX1     .E.L. .
```

The instructions generated by the SAVE pseudo-operation are listed below. The symbols i_1 through i_n are integers 0-7. .E.L..defines the location provided for the contents of the error linkage register.

BBBBB is a symbol that must be present; it is always a primary SYMDEF.

Example one is in the relocatable mode, and example two is in the absolute mode.

	EXAMPLE ONE				EXAMPLE TWO	
BBBBB	SAVE	$i_1, i_2, ...i_n$		BBBBB	SAVE	$i_1, i_2, ...i_n$
BBBBB	TRA	*+2+n		BBBBB	TRA	*+3+n
	LDX(i_1)	**,DU			ZERO	
	.				LDX(i_1)	**,DU
	.				LDX(i_2)	**,DU
	.				.	
	LDX(i_n)	**,DU			.	
	RET	.E.L..			.	
	STI	.E.L..			LDX(i_n)	**,DU
	STX1	.E.L..			RET	BBBBB+1
	STX(i_1)	BBBBB+1			STI	BBBBB+1
	STX(i_2)	BBBBB+2			STX1	BBBBB+1
	.				STX(i_1)	BBBBB+2
	.				STX(i_2)	BBBBB+3
	.				.	
	STX(i_n)	BBBBB+n			.	
					.	
					STX(i_n)	BBBBB+n+1

RETURN (Return--From Subroutines)

1	8	16	32
Symbol or blanks	RETURN		One or two subfields in the variable field

CPB-1004F

222

The RETURN pseudo-operation is used for exit from a subroutine. The instructions generated by a RETURN pseudo-operation must make reference to a SAVE instruction within the same subroutine. This is done by the first subfield of RETURN. The first subfield in the variable field must always be present. This subfield must contain a symbol which is defined by its presence in the location field of a SAVE pseudo-operation.

The second subfield is optional and, if present, specifies the particular error return to be made; that is, if the second subfield contains the value k, then the return is made to the kth error return.

In the examples following, the assembled instructions generated by RETURN are listed below the RETURN instruction. For both examples the group of instructions on the left are generated when the Assembler is in the relocatable mode, and the instructions on the right when the Assembler is in the absolute mode.

<u>EXAMPLE ONE</u>

RETURN		BBBBB	
TRA BBBBB+1 } Generated Instructions		TRA BBBBB+2 } Generated Instructions	

<u>EXAMPLE TWO</u>

RETURN		BBBBB,k	
LDX1 .E.L..,* SBX1 k,DU STX1 .E.L.. TRA BBBBB+1 } Generated Instructions		LDX1 BBBBB+1,* SBX1 k,DU STX1 BBBBB+1 TRA BBBBB+2 } Generated Instructions	

<u>ERLK (Error Linkage--to Subroutines)</u>

1	8	16	32
Blanks	ERLK		Column 16 must be blank

The normal operation of the Assembler is to assign a location for error linkage information, as referenced by .E.L..in the examples of the CALL, SAVE, and RETURN pseudo-operations. If the programmer wishes to specify the location for error linkage information, he must do so by using ERLK since the symbol .E.L..may not appear to the right of an EQU pseudo-operation. The appearance of ERLK causes the Assembler to generate two words of the following form:

```
.E.L..      ZERO
            BCI       1,NAME
```

These words will be placed in the assembly at the point the Assembler encountered ERLK. Note that if the programmer has placed all program data under the BLOCK pseudo-operation, he must use ERLK since in this case automatic error linkage is suppressed.

NAME, as selected by the Assembler, will be the first SYMDEF defined in the routine. This may have been accomplished explicitly through use of the SYMDEF pseudo-operation, or implicitly through SAVE.

Error linkage will be generated for all relocatable assemblies, except in the case mentioned above, where all assembling has been relative to BLOCK counters.

SYSTEM (BUILT-IN) SYMBOLS

It is possible to include additional permanently defined system symbols in the Assembler. This is done by a reassembly of the Macro Assembler and by placing the proper information in the required tables.

SOURCE PROGRAM INPUT

Activity Definition

The input job stream managed by the Comprehensive Operating Supervisor (GECOS) can comprise assembled object programs, Macro Assembler language source programs, and FORTRAN or COBOL compiler-language source programs. Such programs of a job are referred to as activities. Comments to follow in this section pertain to an Assembler language input activity.

The Assembler language activity is composed of the following parts, in order:

1. $ GMAP control card (calls the Assembler into Memory from external storage and provides Assembler output options)

2. Text of the subprogram

3. END pseudo-operation card (terminates the input subprogram)

The $ GMAP control card is prepared as shown below:

Card Column	1	8	16	32
Symbolic Example	$	GMAP	Option 1, Option 2,...	
Actual Example	$	GMAP	NDECK, LSTOU, NCOMDK	

The operand field specifies the system options which may be listed in any random order. When an option, or its converse, does not appear in the operand field, the standard option is assumed. (The standard entries are underlined.)

CPB-1004F

The options available with GMAP are as follows:

LSTOU   Prepare a listing of the GMAP output
NLSTOU   Do not prepare a listing of the GMAP output

DECK   Prepare a program deck as part of the output of this processor
NDECK   Do not prepare a program deck as part of the output of this processor

COMDK   Prepare a compressed deck of the source program
NCOMDK   No not prepare a compressed deck of the source program

GMAC   Use GMAP System Macros for this assembly
NGMAC   Do not use GMAP System Macros for this assembly

DUMP   Dump all of slave core if a GMAP activity terminates abnormally
NDUMP   Dump only program registers if a GMAP activity terminates abnormally

NXEC   Turn off the execution bit (bit 5) in the Program Switch Word if any fatal errors are encountered in the assembly.

SYMTAB   Prepare a listing of the Symbol Reference Table (if one has been built) even though NLSTOU is specified.

ON5   Print all source images regardless of any pseudo-operations that might otherwise result in their not being printed.

The contents of columns 73-80 are used as an identifier to uniquely identify the binary object programs resulting from the assembly.

Compressed Decks

The Assembler program contains routines and tables for compressing source subprogram cards from a one-instruction-per-card input to a multiple-instruction-per-card input. This Assembler feature is provided primarily for reducing the size of input source decks as concerns handling and correcting (altering) the input subprogram. (For details of the compression and the compressed deck card format, refer to the next paragraph and the GE-625/635 File and Record Control reference manual, CPB-1003.)

The compressed deck (COMDK) option is specified in the operand field of the $ GMAP control card. The normal mode of Assembler operation is NCOMDK; that is, no compressed deck is produced. To use the Assembler COMDK feature, the $ GMAP control card would appear as

$ GMAP   COMDK

and be placed as the first card of the deck. When combined with the standard output options, the above control card would cause the Assembler to produce:

1. An output listing containing in its format a complete listing of the source card image (See the listing and symbolic reference table formats, page 236.)

2. A compressed deck of the source card images, column-binary, alphanumeric.

CPB-1004F

The COMDEK format is produced by a procedure which compresses any Hollerith-coded card image by removing sequences of 3 or more blanks and packing the information in standard column binary form.

To accomplish the compression, the Hollerith card is considered as being made up of a series of fields and strings. A field is defined as a segment of the card containing no sequences of more than 2 blanks except at the beginning. A string is that portion of a field obtained by deleting any leading blanks.

Each field specification starts with the octal value of $A(0<A\leq67_8)$ followed by the octal value of $B(0<A\leq67_8)$ followed by the B characters constituting the string. (A=the number of characters in the field; B=the number of characters in the string.)

The size of A and B is limited, as indicated above, in order to reserve a set of codes to serve as flags when found in a position in which a count had been expected. If a given length exceeds the maximum length, it is segmented into separate fields. For example, given 70 (decimal) consecutive nonblank characters, it is necessary to treat this as two fields with:

Field 1   A = 67,   B = 67 (octal values)
Field 2   A = 17,   B = 17 (octal values)

The field specifications (A,B,string) are packed sequentially on a binary card in the format indicated below.   A field specification may be started on a COMDEK card (X) and may be completed on the following card (X+1).

The following codes for A are used to designate specific conditions. The B character is not present in such cases.

A = 0   End of a compressed card; continue decoding on the next card

A = 77_8 End of encoded string for a given Hollerith card image

A = 76_8 End of the compressed deck segment

A = 70_8 Available for extension

The COMDEK card layout consists of:

Word 1:	0-2	Column binary card type 5
	3-8	Zeros
	9-11	101 (7-9 punches)
	12-35	Binary sequence number
Word 2:		Checksum of word 1 or words 3-24
Words 3-24:		Compressed card image
Words 25-27:		Hollerith-coded label or zeros

The binary sequence number is maintained when a COMDEK output is produced and is checked when the deck is used as input. When a sequence error is found in an input COMDEK file, the activity will be terminated.

The label words of the card are supplied in uncompressed form by the I/O Editor and give identification data from columns 73-80 of the standard binary deck cards.

Source Deck Corrections

Corrections to an Assembler language source deck are made by the use of $ ALTER control cards. A source program correction deck consists of the following parts in order:

1. $ GMAP control card

2.   Text of the subprogram in either of two forms:

   a.   Standard one-instruction-per-card deck
   b.   Compressed deck

3. $ UPDATE control card (notifies the Comprehensive Operating Supervisor that the cards to follow are to be placed on the A* (alter) file for use by the Assembler

4. ALTER Information

    a. ALTER cards (contain the updating delimiting information)

    b. New source cards which are to be inserted into the source deck as additions or replacement instructions

The operand field of the ALTER card uses alter numbers that are obtained from the previous assembly listing of the deck now being processed. (See page 235.) The format of the ALTER card is:

Card Column	1	8	16	32
Symbolic Example	$	ALTER	n, m	
Actual Example	$	ALTER	07364, 07464	

The entries define whether the cards following are to be added or to replace cards in the primary input file. These numbers are simply consecutive card numbers starting with 00001 and increasing by one for each source input card.

When it is desired to insert cards into a deck the m subfield is not used. In this case, the cards following this ALTER card, up to but not including the next ALTER card will be inserted just prior to the card corresponding to alter number n.

When it is desired to delete and/or replace one or more cards from a deck, the m subfield is given as shown above. When n and m are equal card n will be deleted. When m identifies a card following n all cards n through m will be deleted. In addition, any cards following this ALTER card up to but not including the next ALTER card will be inserted in place of the deleted cards.

The end of an alter file is designated by the normal end-of-file convention appropriate to the media containing the file.

The $ UPDATE control card is prepared as indicated below.

Card Column	1	8	16	32
Symbolic Example	$	UPDATE	List Option	
Actual Example	$	UPDATE		

The UPDATE control card is used when supplying alter input to a compiler or the Assembler. In the input sequence for a job the $ UPDATE control card and associated ALTER card with its alter statements must follow and be contiguous to the source program to which the alter statements apply.

The operation field contains the word UPDATE. The variable field may contain the word LIST, in which case a listing of the Alter input will be included with the output.

ASSEMBLY OUTPUTS

Binary Decks

When the $ GMAP control card specifies the DECK option, the Assembler punches a binary assembly output deck. Since the normal mode of the Assembler is relocatable or is implied as a standard option, all addresses punched in the output cards are relative to zero. Alternatively, still considering the DECK option the Assembler can operate in the absolute mode and punch only absolute addresses in the output cards.

The first card generated by GMAP for every subprogram object deck is a $ OBJECT card. The format of the $ OBJECT card is as follows:

1	8	16	59	60	61	66	67	72	73
$	OBJECT	Optional Comment, Sequence Option			Time of Assembly		Date of Assembly		Optional Label

Source Identification
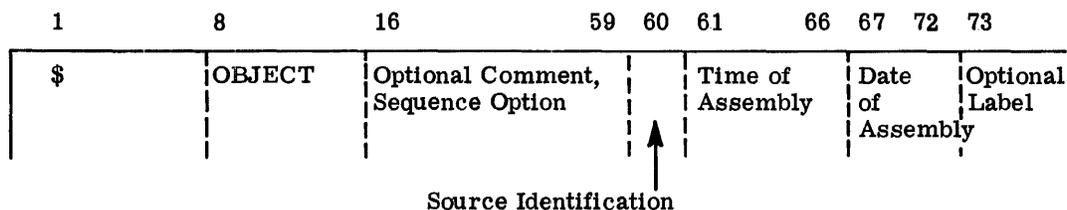
The Optional Comment and Sequence Option subfield (columns 16-59) are either a product of the second subfield of the LBL pseudo-operation or they can be added by the programmer. When a sequence checking option is not specified, the Optional Label subfield of all cards in a $ OBJECT deck will be sequence checked and the activity deleted in case of an error. When an error is detected, a message will be printed on the execution report. The following sequence checking options may be specified. The standard option, SEQ, is assumed if no option is specified.

SEQ   Check sequence and delete the activity if an error occurs.

CKSEQ   Sequence check and flag errors but do not delete the activity.

NSEQ   No sequence check.

Source Identification is the source of the object deck as follows:

A = ALGOL
C = COBOL
F = FORTRAN
G = GMAP
I = IDS

Time of Assembly in columns 61-66 is in hours and thousandths of hours (thousandths are in columns 64-66) in the form XX.XXX. This time appears in the page heading of the associated listing.

Date of Assembly in columns 67-72 is of the form mm dd yy.

The source identification, time and date of assembly are provided by the system.

The Optional Label in columns 73-80 is a product of the first subfield of the LBL pseudo-operation. It is an alphanumeric identification number designating the object program or subprogram. If not specified, it is produced as starting at 00000000.

This binary information may be represented on four types of binary cards. These cards and their uses are summarized below. GE-625/635 Loader functions performed by using the information from these cards are described in the Loader Manual. In addition, that manual describes the memory map layouts applicable to each user subprogram. The user subprogram memory map blocks are (1) the subprogram region (2) the LABELED COMMON region and (3) the BLANK COMMON region.

CARD TYPE	USE
Preface	Provides the Loader with (1) the length of the subprogram text region; (2) the length of the BLANK COMMON region; (3) the total number of SYMDEF, SYMREF, and LABELED COMMON symbols; (4) the type identification of each symbol in (3); and (5) the relative entry value or the region length for each symbol in (3).
Relocatable Binary Text	Supplies the Loader with relocatable binary text by using preface card information and relocation identifiers where the relocation identifiers specify whether the 18-bit field refers to a subprogram, LABELED COMMON, or BLANK COMMON regions (of the assembly core-storage area) and will allow the loader to relocate these fields by an appropriate value.
Absolute Binary Text	Provides the Loader with absolute binary text and the absolute starting-location value for Loader use in assigning core-storage addresses to all words on the card.
Transfer	Can be generated only in an absolute assembly and causes the Loader to transfer control to the routine at the location given on the card. (The transfer card is generated automatically as the last card of an absolute subprogram assembly by the END pseudo-operation; however, use of the TCD pseudo-operation can cause the card to appear anywhere in the assembly.)

The formats in which the Assembler punches the above cards are described in the paragraphs to follow.

CPB-1004F

229

Preface Card Format

Preface card symbolic entries are primary SYMDEF symbols secondary SYMDEF symbols, SYMREF symbols, LABELED COMMON symbols (from the BLOCK pseudo-operation), and the .SYMT. LABELED COMMON symbol. These symbols appear on the card in a precise order. All SYMDEF symbols appear before any other symbol. Following the SYMDEF symbols are any LABELED COMMON symbols. The SYMREF symbols are then recorded.

The format and content of the preface card are summarized as follows:

Word One:

100	n_1	101	n_2	n_3

0   2 3   8 9   11 12   17 18   35
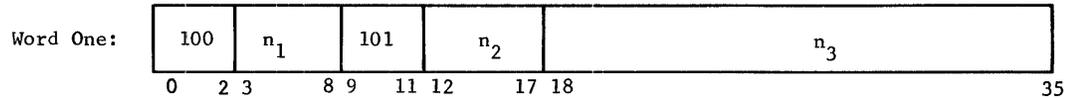
n_1--V is a value within the range $5 < V \le 17$ and represents the size of the field within a special relocation entry needed to point to the specific preface card entry. Thus, $V = \log_2 N + 1$, where N is the number of LABELED COMMON and SYMREF entries.

n_2--Word count of the preface card text

n_3--Length of the subprogram

Word Two: Checksum of columns 1-3 and 7-72

Word Three:

A	M	N

0   17 18 19   35
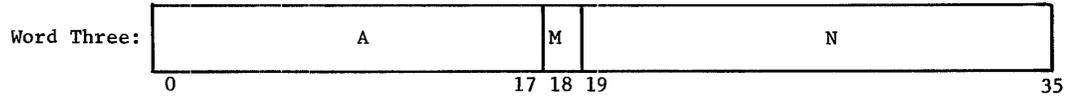
The value A is the length of BLANK COMMON; and N is two times the total number of SYMDEFs, SYMREFs, and LABELED COMMONs. The M bit indicates, when set to 1, that the subprogram must be loaded beginning at a location which is a multiple of eight.

Words Four,
    Five: $Symbol_1$; A_1, K_1

Words Six,
    Seven: $Symbol_2$; A_2, K_2

Words 2n+2,
2n+3

Symbol$_n$;A$_n$, K$_n$

Char. 1	Char. 2	Char. 3	Char. 4	Char. 5	Char. 6
0 5	6 11	12 17	18 23	24 29	30 35

A	K
0 17	18 35
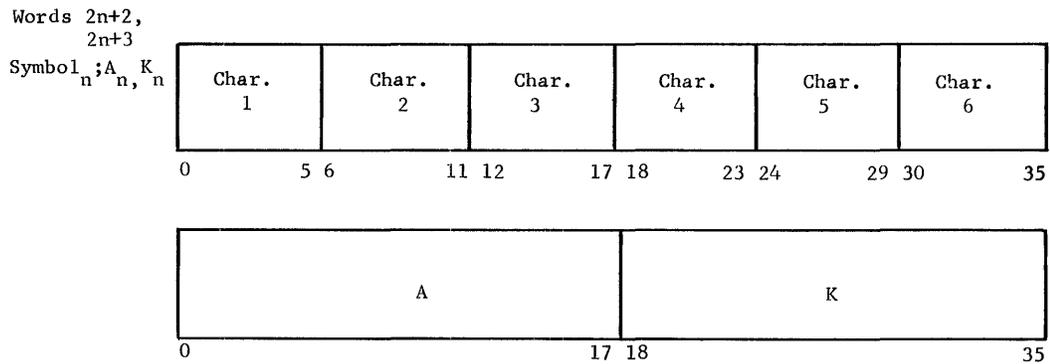
The even-numbered word contains the symbol in BCD. The value K defines the type symbol in the even-numbered word; A is a value associated with K as explained in the following list.

If K equals zero, then the symbol is a primary SYMDEF symbol; A is the entry value relative to the subprogram region origin.

If K equals one, then the symbol is a secondary SYMDEF symbol; A is the entry value relative to the subprogram region origin.

If K equals five, then the symbol is a SYMREF symbol; A is zero.

If K equals six, then the symbol is a LABELED COMMON symbol; A is the length of the region.

If K equals seven, then the symbol is a .SYMT. LABELED COMMON symbol; A is the length of the region reserved for debug information.

NOTE: If preface continuation cards are necessary, word three will be repeated unchanged on all continuation cards.

Relocatable Card Format

A relocatable assembly card has the format and contents summarized in the following comments.

Word One:

010	n_1	101	n_2	n_3
0 2	3 8	9 11	12 17	18 35
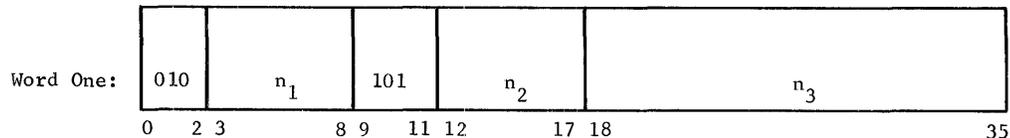
n_1--0 indicates that loading is within the subprogram region of the user subprogram core-storage area

n_2--Word count of the data words to be loaded using the origin and relative address in this control word

n_3--Loading address, relative to the subprogram region origin.

CPB-1004F

231

or for the alternative cases:

n_1--i, where $i \neq 0$ indicates that the ith entry (beginning with the first LABELED COMMON entry in the preface card text) has been used and that n_3 is relative to the origin of that entry.

Word Two: Checksum of columns 1-3 and 7-72
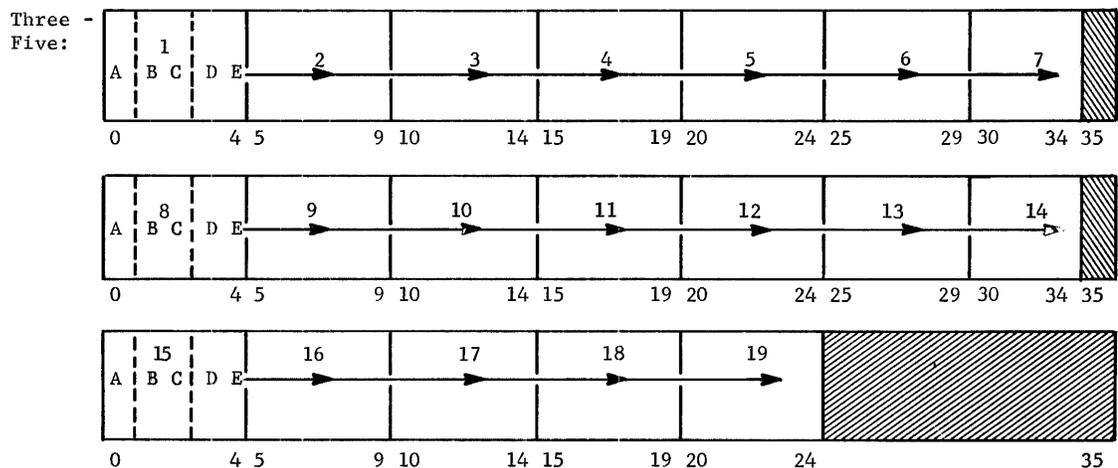
Three -
Five:

Relocation data--words three and four comprise seven 5-bit relocation identifiers, while word five holds 5 such identifiers. The five bits of each identifier carry relocation scheme data for each of the card words (7+7+5=19, or fewer). The identifiers are placed in bit positions 0-34 of words three and four and in 0-24 of word five. (Refer to the Relocation Scheme description in the paragraph following.)

Words Six-
Twenty-Four: Instructions and data (up to 19 words per card). If the card is not complete and at least two words are left vacant, then after the last word entered, word one may be repeated with a new word count and loading address. The loading is then continued with the new address, and the relocation bits are continuously retrieved from words three through five. This process may be repeated as often as necessary to fill a card.

Relocation Scheme

For each binary text word in a relocatable card, the five bits--A, BC, and DE--of each relocation scheme identifier are interpreted by the Loader as follows:

Bit A--0 (reserved for future use)

Bits BC--Left half-word

Bits DE--Right half-word

CPB-1004F

To every 18-bit half-word one of four code values apply; these are:

CODE VALUE	MEANING
xx = 00	Absolute value that is not to be modified by the Loader.
= 01	Relocatable value that is to be added to the origin of the subprogram region by the Loader.
= 10	BLANK COMMON, relative value that is to be added to the origin of the BLANK COMMON region by the Loader.
= 11	Special entry value (to be interpreted as described in the next paragraph.

apply where xx stands for BC or DE.

If special entry is required, the Loader decodes and processes the text and bits of the 18-bit field (left/right half of each relocatable card word) as follows:

Bit 1	--This is the sign of the addend; 0 implies a plus (+) and 1 implies a minus (-).
Bits $2 \to V+1$	--The value V that was specified in word 1 of the preface card dictates the length of the field. The contents of the field is a relative number which points to a LABELED COMMON region or a SYMREF that appeared in the preface card. The value one in this field would point to the first symbol entry after the last SYMDEF.
Bits $V+2 \to 18$	--The value in this field is the addend value that appeared in the expression. If the field is all bits then the corresponding 18 bits of the next data word are interpreted as the addend. In this special case there will be no relocation bits for the addend word.

All references to each undefined special symbol are chained together. When the symbol is defined, the Loader can rapidly insert the proper value of the symbol in all relocatable fields that were specified in the chain.

Absolute Card Format

The absolute binary text card appears as shown below.

Word One:

001	n_1	101	n_2	n_3
0 2	3 8	9 11	12 17	18 35
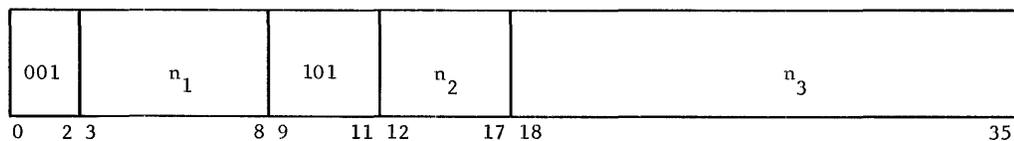
n_1--0

n_2--Word count of the card text

n_3--Loading address relative to the absolute core-storage origin zero (of allocated memory).

CPB-1004F

233

Word Two:	Checksum of columns 1-3 and 7-72
Words Three- Twenty-Four:	Instructions and text (22 words per card, maximum). If the card is not complete and at least two words are left vacant, then after the last word entered, word one may be repeated with a new word count and loading address.

Transfer Card Format

The transfer card is generated by the Assembler only in an absolute assembly deck. Its format and contents are:

Word One:

000	n_1	101	n_2	n_3
0 2 3	8 9	11 12	17 18	35

n_1--0

n_2--0

n_3--Transfer address (in absolute only).

Words Two- Twenty-Four:	Not used

Assembly Listing

Each Assembler subprogram listing is made up of the following parts:

1. Execution Report (See GE-625/635 Comprehensive Operating Supervisor Reference Manual)

2. The contents of all preface cards (primary SYMDEF symbols, secondary SYMDEF symbols, SYMREF symbols, LABELED COMMON symbols--from the BLOCK pseudo-operation--and the .SYMT. LABELED COMMON symbol). This section is omitted from an absolute assembly.

3. The sequence of instructions in order of input to the Assembler.

4. The symbolic reference table.

Full Listing Format

Each instruction word produced by the Assembler is individually printed on a 120-character line. The line contains the following items for each such word of all symbolic cards:

1. Error flags--one character for each error type (see "Error Codes" page 237).

2. Octal location of the assembled word.

3. Octal representation of the assembled word

4. Relocation bits for the assembled word (see the topic, Relocation Scheme, Loader manual)

5. Reproduction of the symbolic card, including the comments and identification fields, exactly as coded

The exact format of the full listing is as follows:

Fields	A	B	C	D	E	F	G	H
Print line Columns	1-6	7-12	15-20	22-25	27,28	31-33	35-39	41-120

Machine Instruction (C, D, E)   Source Card Image (H)

A--Error flags
B--Relative/absolute location
C--Operand address
D--Operation code

E--Tag field modifier
F--Relocation bits
G--Alter statement number
H--Card image

Several variations appear for bit positions 15 through 28. (The six, four, two subfield groups C, D, and E shown above is the octal configuration for machine instructions.) These are summarized in the table below in which the X represents one octal digit.

	Type of Machine Word	Listing Format	Source Program Instruction
1.	Processor instruction and indirect address	XXXXXX XXXX XX	Processor instruction and indirect address word
2.	Data	XXXXXXXXXXXX	Data generating pseudo-operations (OCT, DEC, BCI, etc.)
3.	Data Control	XXXXXX XX XXXX	Data Control Word (DCW)
4.	Special 18-bit field data	XXXXXX XXXXXX	ZERO pseudo-operation
5.	Input/output command	XX XXXX XXXXXX	Input/output pseudo-operation (See Appendix E.)

Error flags are summarized at the end of this section. The interpretation of the relocation bits is described in the Loader manual. That field (F) will be blank in an absolute assembly.

Preface Card Listing

The listing of the preface information is in a self-explanatory format, with each major subdivision of preface symbols preceded by a heading. The order is the same as that of the card(s) produced.

Primary SYMDEFs, secondary SYMDEFs, LABELED COMMON, and SYMREFs. The LABELED COMMONs and SYMREFs are numbered sequentially 1 through n, where this number represents the special relocation entry number employed in referencing these special symbols.

BLANK COMMON Entry

Prior to the listing of the special symbols, the Assembler enters a statement of the amount of BLANK COMMON storage requested by the subprogram. The statement format is self-explanatory.

Symbolic Reference Table

The symbol table listing contains all symbols used, their octal values (normally, the location value), and the alter numbers of all instructions that referenced the symbol. The table format is as follows:

OCTAL	SYMBOL	REFERENCES BY ALTER NO.
364	BETA	103 103 1027 1761 3767 7954

The above sample indicates that the symbol BETA has been assigned the value 364_8 and is referenced in five places: namely, at alter number positions 103, 1027, 1761, 3767, and 7954 in the listing of instructions. The first alter number is the point in the instruction listing where the symbol was defined. If an instruction contains a symbol twice, the alter number for that point in the instruction listing is given twice. The alter numbers are assigned sequentially in the subprogram listing, one per instruction. Because of this fact, it is easy for the programmer to locate in the listing those card images that referenced any particular symbol as well as locate the card image that caused the symbol to be defined.

The symbolic reference table will contain symbols referenced in the DUP pseudo-operation.

A separate symbolic reference table is generated for MACRO's (when the REFMA ON pseudo-operation is specified) that contains an entry of the MACRO name and alter number for each MACRO reference.

<u>Error Codes</u>

The following list comprises the error flags for individual instructions and pseudo-operations.

<u>ERROR</u>	<u>FLAG</u>	<u>CAUSE</u>
Undefined	U	Undefined symbol(s) appear in the variable field.
Multidefined	M	Multiple-defined symbol(s) appear in the location field and/or the variable field.
Address	A	Illegal value or symbol appears in the variable field. Also used to denote lack of a required field.
Index	X	Illegal index or address modification.
Relocation	R	Relocation error; expression in the variable field will produce a relocatable error upon loading.
Phase	P	Phase error; this implies undetected machine error or symbols becoming defined in Pass two with a different value from Pass one.
Even	E	Inappropriate character in column 7.
Conversion	C	Error in conversion of either a literal constant or a subfield of a data-generative pseudo-operation. Illegal character.
Location	L	Error in the location field.
Operation	O	Illegal operation.
Table	T	An assembly table overflowed not permitting proper processing of this card completely. Table overflow error information will appear at the end of listing.

GMAP also prints out the following error messages which are self explanatory:

Symbol Table Overflow

Macro Expansion Table Overflow

Macro Prototype Table Overflow

No END Card on Input File

Symbol Reference Table Overflow

Execution not Possible, no SYMDEFS.

Too Many Cards to be Duplicated

Operation Table Overflow

Unexpected EOF on Intermediate File

NXEC Option Specified. Fatal Errors ... Execution Deleted

Not Enough Cards to be Skipped

IV. CODING EXAMPLES

PRELIMINARY

This chapter contains examples of coding techniques for performing typical program functions. These examples:

1. Indicate how certain very efficient Processor instructions can be used

2. Illustrate the use of address modification variations for indexing, indirection and automatic tallying

3. Demonstrate operations performed on characters

4. Show operations on fixed- and floating-point numbers

5. Present the use of the BCD instruction

The list of examples is by no means complete in that it does not present all of the processor capabilities; however, the examples provided can serve as convenient references for programmers newly acquainted with the GE625/635.

Each example is self-contained and self-explanatory. In most cases, questions that may be raised can be answered by referring to the descriptions of particular instruction or pseudo-operations. Convenient references are contained in Appendixes A through D.

EXAMPLES

Fixed Point to Floating Point (Integer)

The following example illustrates the conversion of a fixed-point integer to floating point (float an integer). The integer to be converted is in the location M.

Step 01 resets the Overflow Indicator.

Step 02 places the binary integer to be converted in the accumulator.

Step 03 places zeros in the quotient register.

Step 04 sets the exponent register to 35_{10}.

Step 05 converts the number in the accumulator to floating point.

For example, if the contents of M equal 000000000002_8, then the contents of the floating-point register will be $E = 2_{10}$, and $AQ = 200000000000000000000000_8$ at the completion of step 05.

```
01      TOV      1,IC
02      LDA      M              FLOAT AN INTEGER M
03      LDQ      ,DL            C(AQ) = M AT B35.
04      LDE      =35B25,DU      C(E) = 35.
05      FNO                     NORMALIZE  M
```

Floating Point to Fixed Point (Integer)

The following example illustrates the conversion of a double-precision, floating-point number to a fixed-point number, binary point 71. The result will be only the integral part of the number. The number to be converted must lie between -2^{71} and $2^{71}-1$ inclusive.

Step 01 loads the floating-point number to be converted into the floating-point register.

Step 02, an unnormalized floating add of zero (exponent of 71), causes the contents of AQ to be shifted right a number of places equal to the difference between 71 and the exponent of the number to be converted. This will leave in AQ the binary integer (binary point 71) equal to the integral part of the floating-point number in X and X+1.

For example, if prior to executing step 02, the floating-point register contained -2, that is, if the exponent register contained 2_{10} and AQ contained $600000000000000000000000_8$, then the result in AQ after the addition of zero (exponent 71) would be $777777777777777777777776_8$.

```
.
.
.
01      DFLD     X              COMPUTE THE INTEGER PART OF
                                A FLOATING-POINT NUMBER CON-
                                TAINED IN X AND X+1.

02      UFA      =71B25,DU      FIX THE RESULT IN AQ, BINARY
                                POINT 71.
```

<u>Real Logarithm</u>

Purpose:

Compute log X for ALOG(X) or ALOG10(X) in an expression.

Method:

1. $\log_2 X = \log_2 (2^I{*}F) = I + \log_2 F$, where $X=2^I{*}F$.

2. $\log_e X = \log_e 2^{(\log_2 X)} = (\log_2 X) * (\log_e 2)$, and similarly $\log_{10} X = (\log_2 X) * (\log_{10} 2)$.

3. $\log_2 X = Z* \left(A + \left(\dfrac{B}{Z^2-C} \right) \right) = \log_2 F+\frac{1}{2}$, where $Z = \dfrac{F - \left(\frac{\sqrt{2}}{2} \right)}{F + \left(\frac{\sqrt{2}}{2} \right)} = \dfrac{F-.707}{F+.707}$

   A = 1.2920070987
   B = -2.6398577031
   C = 1.6567626301

4. X and log X are real numbers, with values of X from 2^{-129} to $2^{127} - 2^{100}$ inclusive.

5. log X is accurate to 8 decimal places.

Use:

Calling Sequence -- CALL ALOG(X)   for $\log_e X$

   CALL ALOG10(X) for $\log_{10} X$

CPB-1004F

241

```
         SYMDEF       ALOG10,ALOG
LOGS     SAVE                                REAL LOGARITHM FUNCTIONS
         FLD          2,1*                   X = (2**I) * F = ARGUMENT
         FNO
         TZE          ERR1                   ERROR IF X=0
         TMI          ERR2                   ERROR IF X NEGATIVE
BEGIN    FCMP         =1.0,DU
         TZE          UNITY                  LOG(1) = 0
         STE          I                      STORE I AT BINARY POINT 7
         LDE          0,DU                   OBTAIN F
         DFAD         SRHLF
         DFST         Z
         DFSB         SRTWO
         DFDV         Z
         DFST         Z                      Z = (F - SQRT(1/2))/(F + SQRT(1/2))
         DFMP         Z                      Z²
         DFSB         C                      Z²-C
         DFDI         B                      B/(Z²-C)
         DFAD         A                      A+B/(Z²-C)
         DFMP         Z                      Z(A+B/(Z²-C))
         DFST         Z                      Z = Z*(A + B/(Z**2-C)) = LOG2(F) + 1/2
I        LDA          *-*,DU
         LDQ          0,DU
         LDE          =7B25,DU               FLOAT I
         FSB          =0.5,DU
         DFAD         Z                      LOG2(X) = I + LOG2(F)
INDIC    DFMP         *                      CONVERT TO BASE 10 OR E
         RETURN       LOGS
ERR1     CALL         .FXEM.(EALN1)          ERROR EXIT NUMBER 1 (X=0)
UNITY    FLD          =0.0,DU
         RETURN       LOGS
ERR2     CALL         .FXEM.(EALN2)          ERROR EXIT NUMBER 2 (X IS NEGATIVE)
         FNEG
         TRA          BEGIN

ALOG10   ESTC2        INDIC                  REAL COMMON LOGARITHM
         TRA          LOGS
         DEC          .301029996D0
ALOG     ESTC2        INDIC                  REAL NATURAL LOGARITHM
         TRA          LOGS
         DEC          6.93147180559D-1
EALN1    DEC          9
EALN2    DEC          10
A        DEC          .12920070987D1
B        DEC          -.26398577031D1
C        DEC          .16567626301D1
SRHLF    DEC          .707106781187D0        SQUARE ROOT OF TWO DIVIDED BY TWO
SRTWO    DEC          .1414213562374D1       SQUARE ROOT OF TWO
Z        BSS          2
         END
```

<u>BCD Addition</u>

The following example illustrates the addition of two words containing BCD integers. The example limits the result to 999999.

Step 01 places the number in A into the accumulator.

Step 02 adds the number in B to the accumulator. Column V in the table, following, shows the possible results for any digit. It should be noted that there are 19 possible results, indicated by lines 0-18.

Step 03 forces any carries into the units position of the next digit. Lines 10-18 of Column V contain the sums that will carry into the next digit. Column W contains the 20 possible results for each digit position. The additional possibility (line 19) arises from the fact that there can be a carry of one into a digit.

Step 04 stores the intermediate result in C.

Step 05 extracts an octal 60 from each non-carry digit. The results are indicated in column X. The digits that did not force a carry (lines 0-9) result in an octal 60, the digits that had a carry into the next digit (lines 10-18) result in 00.

Step 06 performs an exclusive OR of the contents of the accumulator with the contents of C. This in effect subtracts octal 60 from each digit that did not have a carry (lines 0-9). The results are indicated in Column Y.

Step 07 shifts the octal 60s to the right three places.

Step 08 negates the contents of the accumulator.

Step 09 is an add to storage the contents of the accumulator to the contents of C. This in effect subtracts a 06 from each digit that did not have a carry, the results of which are indicated in Column Z.

```
01   LDA  ⎫   A              ⎰ TO ADD C = A+B IN BCD.
02   ADLA ⎬   B              ⎱ COMPUTE A+B
03   ADLA ⎭   =0666666666666    ADD OCTAL 66 TO EACH DIGIT TO FORCE CARRIES
04   STA      C
05   ANA      =0606060606060    EXTRACT OCTAL 60 FROM EACH NON-CARRY
06   ERSA     C                 SUBTRACT OCTAL 60 FROM EACH NON-CARRY
07   ARL  ⎫   3              ⎧ SUBTRACT OCTAL
08   NEG  ⎬                  ⎨ 06 FROM EACH
09   ASA  ⎭   C              ⎩ NON-CARRY
```

ADDITION RESULTS

LINE	V	W	X	Y	Z
0	00	66	60	6	00
1	01	67	60	7	01
2	02	70	60	10	02
3	03	71	60	11	03
4	04	72	60	12	04
5	05	73	60	13	05
6	06	75	60	14	06
7	07	75	60	15	07
8	10	76	60	16	10
9	11	77	60	17	11
10	12	00	00	0	00
11	13	01	00	1	01
12	14	02	00	2	02
13	15	03	00	3	03
14	16	04	00	4	04
15	17	05	00	5	05
16	20	06	00	6	06
17	21	07	00	7	07
18	22	10	00	10	10
19	-	11	00	11	11

BCD Subtraction

The following is an example of subtracting one BCD number from another BCD number. The contents of A must be equal to or greater than the contents of B.

Step 01 loads the accumulator with the contents of A.

Step 02 subtracts the contents of B from the accumulator. The possible results for each digit are indicated in Column W of the table that is included with this example.

Step 03 stores the intermediate result in C.

Step 04 extracts an octal 60 from each digit that required a borrow. This will leave an octal 60 in each digit position where there was a borrow. The possible results of this instruction are indicated in Column X, lines 0-19 (10-19 refer to those which result in octal 60.)

Step 05, an exclusive OR to storage, in effect subtracts the octal 60's in the accumulator from the corresponding digit in C. The possible results for each digit are displayed in Column Y.

Step 06 shifts the octal 60's in the accumulator right three places.

Step 07 negates the contents of the accumulator.

Step 08, an add to storage, is in effect a subtraction of 06 from each digit that required a borrow, the result being placed in C. Column Z of the table reflects the possible results for each digit.

CPB-1004F

244

```
01      LDA  ⎤   A              ⎧ TO SUBTRACT C = A-B IN BCD.
02      SBLA ⎦   B              ⎨ COMPUTE A-B
03      STA      C              ⎩
04      ANA      =0606060606060   EXTRACT OCTAL 60 FROM EACH BORROW
05      ERSA     C               SUBTRACT OCTAL 60 FROM EACH BORROW
06      ARL  ⎤   3              ⎧ SUBTRACT OCTAL
07      NEG  ⎥                  ⎨ 06 FROM EACH
08      ASA  ⎦   C              ⎩ BORROW
```

SUBTRACTION RESULTS

LINE	W	X	Y	Z
0	11	0	11	11
1	10	0	10	10
2	07	0	07	07
3	06	0	06	06
4	05	0	05	05
5	05	0	04	04
6	03	0	03	03
7	06	0	02	02
8	01	0	01	01
9	00	0	00	00
10	77	60	17	11
11	76	60	16	10
12	75	60	15	07
13	74	60	14	06
14	73	60	13	05
15	72	60	12	04
16	71	60	11	03
17	70	60	10	02
18	67	60	7	01
19	66	60	6	00

Character Transliteration

The following example illustrates a method of transliterating each character of a card image that has been punched in the FORTRAN Character Set to the octal value of the corresponding character in the General Electric Standard Character Set. There are 48 characters in the FORTRAN Set and 64 characters in the General Electric Standard Character Set. Each character that is punched invalidly (not a standard punch combination in the FORTRAN Set) is converted to a blank. The card is origined at IMAGE.

Steps 01 and 02 initialize the indirect word TALLY2.

Step 03 picks up the character to be transliterated by referencing the word TALLY2 with the Character from Indirect (CI) modifier. This will place the character specified by bits 33-35 of TALLY2 from a location specified by bits 0-17 of TALLY2 into the accumulator, bits 29-35. Bits 0-28 of the accumulator will be set to zero. Step 03 is forced even so as to place the four-step loop (step 03-06) in two even/odd pairs. This decreases run time.

Step 04 picks up the corresponding General Electric standard character from the address TABLE modified by the contents of accumulator, bits 18-35.

Step 05 places the transliterated character back in the card image where it was originally picked up. The Sequence Character (SC) modifier increments the character specified in bits 33-35 of the word TALLY2.

Each time the character position becomes greater than 5, it is reset to zero; and the address specified in bits 0-17 of TALLY2 is incremented by one. The tally in bits 18-29 of the same word is decremented by 1 with each SC reference. Whenever a tally reaches zero, the Tally Runout Indicator is set ON. Otherwise, it is set OFF.

Step 06 tests the Tally Runout Indicator. If it is OFF, the program transfers to LOOP; if not, the next sequential instruction is taken.

The table, TABLE, is 64 locations long. The character in each location is a General Electric standard character that corresponds to a FORTRAN character in the following manner. The relative location of a particular character to the start of the table is equal to the binary value of the corresponding FORTRAN character. For example, an A punched in the FORTRAN Character Set has the octal value $21(17_{10})$. The relative location 17 to TABLE contains an A in the General Electric Standard Character Set. A 3-8 punch in the FORTRAN Set represents an = character. The 3-8 punch would be read as an octal $13(11_{10})$. The relative location 11 to TABLE contains an octal 75 (see line 21) which represents the = character in the General Electric Standard Character Set.

.
.
.

01		LDA	TALLY1	INITIALIZE TALLY WORD
02		STA	TALLY2	
03	LOOP	ELDA	TALLY2,CI	PICK UP CHARACTER TO BE TRANSLITERATED
04		LDQ	TABLE,AL	LOAD OR WITH TRANSLITERATED CHARACTER
05		STQ	TALLY2,SC	STORE BACK ON CARD IMAGE
06		TTF	LOOP	IF TALLY HAS NOT RUN OUT CONTINUE LOOP

.
.
.

07	TALLY1	TALLY	IMAGE,80,0	
08	TALLY2	ZERO		
09	IMAGE	BSS	14	
10	TABLE	OCT	0	
11		OCT	1	
12		OCT	2	
13		OCT	3	
14		OCT	4	
15		OCT	5	
16		OCT	6	
17		OCT	7	
18		OCT	10	
19		OCT	11	
20		OCT	20	
21		OCT	75	3-8 PUNCH = IN FORTRAN SET
22		OCT	57	4-8 PUNCH ' IN FORTRAN SET
23		OCT	20	
24		OCT	20	
25		OCT	20	
26		OCT	20	
27		OCT	21	
28		OCT	22	
29		OCT	23	
30		OCT	24	
31		OCT	25	
32		OCT	26	
33		OCT	27	

CPB-1004F

34	OCT	30	
35	OCT	31	
36	OCT	60	12 PUNCH + IN FORTRAN SET
37	OCT	33	12-3-8 PUNCH . IN FORTRAN SET
38	OCT	55	12-4-8 PUNCH) IN FORTRAN SET
39	OCT	20	
40	OCT	20	
41	OCT	20	
42	OCT	20	
43	OCT	41	
44	OCT	42	
45	OCT	43	
46	OCT	44	
47	OCT	45	
48	OCT	46	
49	OCT	47	
50	OCT	50	
51	OCT	51	
52	OCT	52	11 PUNCH - IN FORTRAN SET
53	OCT	53	11-3-8 PUNCH $ IN FORTRAN SET
54	OCT	54	11-4-8 PUNCH * IN FORTRAN SET
55	OCT	20	
56	OCT	20	
57	OCT	20	
58	OCT	20	
59	OCT	61	0-1 PUNCH / IN FORTRAN SET
60	OCT	62	
61	OCT	63	
62	OCT	64	
63	OCT	65	
64	OCT	66	
65	OCT	67	
66	OCT	70	
67	OCT	71	
68	OCT	20	
69	OCT	73	0-3-8 PUNCH , IN FORTRAN SET
70	OCT	35	0-4-8 PUNCH (IN FORTRAN SET
71	OCT	20	
72	OCT	20	
73	OCT	20	

Table Lookup

The following example illustrates a method of searching an unordered table for a value equal to the value in the accumulator. Prior to entering the routine given below, the user must load the accumulator with the search argument, load the quotient register with the size of the table to be searched (the size should be scaled at binary point 25), and initialize index register 1 with the first location of the table to be searched. The user enters the routine by executing a transfer and set index register 2 (TSX2) to the symbolic location TLU (see step 05, below). Return from the routine is to the instruction following the TSX2. The Zero Indicator will tell the user whether or not a match has occurred. Zero Indicator ON indicates a match; Zero Indicator OFF indicates no match. If a match was made, the contents of index register 1 will be W locations (W being the increment specified in the RPTX command, step 15) higher than the location of the equal argument.

CPB-1004F

Steps 01-11 are comment cards.

Step 12 places the contents of the lower half (bits 18-35) of the quotient register plus 64, in index register 0. The number 64, in effect, set the TZE terminate repeat condition on. The instruction also places the last 8 bits of the size of the table in index register 0, bits 0-7. Thus, if the size of the table is a multiple of 256 words, zeros will be loaded into bits 0-7 of index register 1. Zeros in those bit positions will cause the repeat to execute 256 times. If, however, the size of the table to be searched is of the form 256n+m, where n≥0, and 0 < m < 256, then m would be placed in bits 0-7 of index register 0. This will cause the repeat instruction to be executed a maximum of m times on the first pass through.

Step 13 subtracts 1024 from the quotient register. This, in effect, subtracts 1 from the size of the table to be searched. The subtracting of 1 becomes meaningful in two places: (1) it provides a test to be sure the table is not zero words long (see step 14) and (2) if the table is a multiple of 256 words long, it effectively subtracts 1 from bits 0-17 (a look-ahead to steps 18 and 19 points out the importance of this).

Step 14 causes the routine to return to the main program if the size of the table was zero.

Step 15, an RPTX, executes step 16 a number of times equal to the contents of index register 0, bits 0-7, at the start of the instruction execution. Each time step 16 is executed, the contents of the accumulator (the search argument) are compared with the contents of the location specified by index register 1. At the same time, index register 1 is incremented by W as is specified in the repeat instruction; and the contents of index register 0, bits 0-7, are decremented by 1. The repeat sequence terminates when the compare causes the Zero Indicator to be set or when bits 0-7 of index register 0 are set to zero.

Step 17 tests the Zero Indicator and returns to the main program if it is set. It should be noted that index register 1 will be set W locations higher than when the equal argument was found because of the sequence of events described above.

Step 18. If the Zero Indicator was not set by step 16, then step 18 will be executed. This instruction subtracts 1 from bits 0-17 of the quotient register. In effect, this is subtracting 256 from the size of the table. The size of the table can be expressed in the form 256n+m. If m=0 and n=1, then the contents of the quotient register would also go zero at this point. This is because step 13 would have caused a borrow of 1 from n when m equals zero. Further inspection of these instructions will reveal that positive values of n and m, other than those expressed above, will only cause the routine to loop until the contents of the quotient register are reduced to a negative value.

Step 19 transfers control to step 15 if the contents of quotient register remained positive. If the quotient register became negative, step 20 is executed and the routine returns to the main program.

It should be noted that when control is transferred back to step 15, index register 0, bits 0-7, contains zeros (causes the repeat to be executed a maximum of 256 times); and index register 1 contains the address of the next location in the table that is to be searched.

```
01   *      CALLING SEQUENCE IS
02   *      LDA     ITEM            SEARCH ITEM.
03   *      LDQ     SIZE            NUMBER OF TABLE ENTRIES--AT B25.
04   *      LDX1    FIRST,DU        LOCATION OF FIRST SEARCH WORD IN TABLE.
05   *      TSX2    TLU             CALL TABLE LOOKUP SUBROUTINE.
06   *      TZE     FOUND           TRANSFER IF SEARCH ITEM IS IN TABLE, OR
07   *      TNZ     ABSENT          TRANSFER IF SEARCH ITEM IS NOT IN TABLE.
08   *      USE ONE OF THE TWO INSTRUCTIONS IMMEDIATELY ABOVE.
09   *      IF IN TABLE, C(X1)-W WILL BE THE LOCATION OF THE MATCHING SEARCH
10   *      WORD.    OTHERWISE, C(X1)-W WILL BE THE LOCATION OF THE LAST
11   *      SEARCH WORD IN THE TABLE. W IS THE NUMBER OF WORDS PER ENTRY.
12   TLU    EAX0    64,QL           PICKUP SIZE (MOD 256) AND TZE-BIT
13          SBLQ    1024,DL         SIZE = SIZE-1.
14          TMI     ,2              EXIT IF SIZE WAS 0--EMPTY TABLE.
15   TLU1   RPTX    ,W              NOTE THAT 0 REPRESENTS 256 (MOD 256).
16          CMPA    ,1              PERFORM TABLE LOOKUP
17          TZE     ,2              EXIT IF SEARCH ITEM IS IN TABLE.
18          SBLQ    1,DU            SIZE = SIZE-256.
19          TPL     TLU1            CONTINUE TABLE LOOKUP IF MORE ENTRIES.
20          TRA     ,2              EXIT--SEARCH ITEM IS NOT IN TABLE.
```

Binary to BCD

The following example illustrates a method of converting a number from binary to BCD. The example converts a number that is in the range of -10^6+1 to $+10^6-1$, inclusive.

Step 01 places zeros in index register 2.

Step 02 loads the accumulator with the binary number that is to be converted.

Steps 03 and 04 perform the conversion of the binary number in the accumulator to the Binary-Coded Decimal equivalent. Step 03 will repeat step 04 six times. It will also increment the contents of index register 2 by one after each execution.

The BCD instruction, step 04, is designed to convert the magnitude of the contents of the accumulator to the Binary-Coded Decimal equivalent. The method employed is to effectively divide this number by a constant, place the result in bits 30-35 of the quotient register and leave the remainder in the accumulator. The execution of the BCD instruction allows the user to convert a binary number to BCD, one digit at a time, with each digit coming from the high-order part of the number. The address of the BCD instruction refers to a constant to be used in the division; a different constant is needed for each digit. In the process of the conversion, the number in the accumulator is shifted left three positions. The $C(Q)_{0-35}$ are shifted left 6 positions before the new digit is stored.

In this example, the constants used for dividing are located at TAB, TAB+1, TAB+2,..., TAB+5. If the value in X were 00000052224_8, the quotient register would contain 01070302010_7 at the completion of the repeat sequence. Step 05 stores the quotient register in Y.

The values in the table below are the conversion constants to be used with the Binary to BCD instruction. Each vertical column represents the set of constants to be used depending on the initial value of the binary number to be converted to its decimal equivalent. The instruction is executed once per digit, using the constant appropriate to the conversion step with each execution.

An alternate use of the table for conversion involves the use of the constants in the row corresponding to conversion step 1. If after each conversion, the contents of the accumulator are shifted right 3 positions, the constants in the conversion step 1 row may be used one at a time in order of decreasing value until the conversion is complete.

BINARY TO BCD CONVERSION CONSTANTS

Conversion Step	Starting Range of C(AR): $-10^{10}+1 \rightarrow 10^{10}-1$	$-10^{9}+1 \rightarrow 10^{9}-1$	$-10^{8}+1 \rightarrow 10^{8}-1$	$-10^{7}+1 \rightarrow 10^{7}-1$	$-10^{6}+1 \rightarrow 10^{6}-1$	$-10^{5}+1 \rightarrow 10^{5}-1$	$-10^{4}+1 \rightarrow 10^{4}-1$	$-10^{3}+1 \rightarrow 10^{3}-1$	$-10^{2}+1 \rightarrow 10^{2}-1$	$-10^{1}+1 \rightarrow 10^{1}-1$
1	$8^{1}\mathrm{x}10^{9}$	$8\,\mathrm{x}10^{8}$	$8\,\mathrm{x}10^{7}$	$8\,\mathrm{x}10^{6}$	$8\,\mathrm{x}10^{5}$	$8\,\mathrm{x}10^{4}$	$8\,\mathrm{x}10^{3}$	$8\,\mathrm{x}10^{2}$	$8\,\mathrm{x}10^{1}$	8
2	$8^{2}\mathrm{x}10^{8}$	$8^{2}\mathrm{x}10^{7}$	$8^{2}\mathrm{x}10^{6}$	$8^{2}\mathrm{x}10^{5}$	$8^{2}\mathrm{x}10^{4}$	$8^{2}\mathrm{x}10^{3}$	$8^{2}\mathrm{x}10^{2}$	$8^{2}\mathrm{x}10^{1}$	8^{2}	
3	$8^{3}\mathrm{x}10^{7}$	$8^{3}\mathrm{x}10^{6}$	$8^{3}\mathrm{x}10^{5}$	$8^{3}\mathrm{x}10^{4}$	$8^{3}\mathrm{x}10^{3}$	$8^{3}\mathrm{x}10^{2}$	$8^{3}\mathrm{x}10^{1}$	8^{3}		
4	$8^{4}\mathrm{x}10^{6}$	$8^{4}\mathrm{x}10^{5}$	$8^{4}\mathrm{x}10^{4}$	$8^{4}\mathrm{x}10^{3}$	$8^{4}\mathrm{x}10^{2}$	$8^{4}\mathrm{x}10^{1}$	8^{4}			
5	$8^{5}\mathrm{x}10^{5}$	$8^{5}\mathrm{x}10^{4}$	$8^{5}\mathrm{x}10^{3}$	$8^{5}\mathrm{x}10^{2}$	$8^{5}\mathrm{x}10^{1}$	8^{5}				
6	$8^{6}\mathrm{x}10^{4}$	$8^{6}\mathrm{x}10^{3}$	$8^{6}\mathrm{x}10^{2}$	$8^{6}\mathrm{x}10^{1}$	8^{6}					
7	$8^{7}\mathrm{x}10^{3}$	$8^{7}\mathrm{x}10^{2}$	$8^{7}\mathrm{x}10^{1}$	8^{7}						
8	$8^{8}\mathrm{x}10^{2}$	$8^{8}\mathrm{x}10^{1}$	8^{8}							
9	$8^{9}\mathrm{x}10^{1}$	8^{9}								
10	8^{10}									

```
01      LDX2    0,DU              PLACE ZEROS IN X2

02      LDA     X                 LOAD ACCUMULATOR WITH VALUE TO BE CONVERTED

03      RPT     6,1               REPEAT 6 TIMES, INCREMENT BY 1

04      BCD     TAB,2             DIVIDE BY TAB, TAB+1, ETC.

05      STQ     Y                 STORE CONVERTED NUMBER IN Y
        .
        .
        .
        .
06TAB   DEC     800000, 640000, 512000, 409600, 327680

        DEC     262144
```

CPB-1004F

DATA MOVEMENT Load			GE-615 Timing (μsec)$\neq$	GE-635 Timing (μsec)$\neq$	Reference (Page)
LDA	235	Load A	4.0	1.9	48
LDQ	236	Load Q	4.0	1.9	48
LDAQ	237	Load AQ	4.3	1.9	48
LDXn	22n	Load Xn from Upper	4.3	1.9	49
LXLn	72n	Load Xn from Lower	4.0	1.9	49
LREG	073	Load Registers	14.3	6.7	49
LCA	335	Load Complement A	4.0	1.9	50
LCQ	336	Load Complement Q	4.0	1.9	51
LCAQ	337	Load Complement AQ	4.3	1.9	51
LCXn	32n	Load Complement Xn	4.0	1.9	52
EAA	635	Effective Address to A	2.6	1.6	52
EAQ	636	Effective Address to Q	2.6	1.6	53
EAXn	62n	Effective Address to Xn	2.6	1.6	53
LDI	634	Load Indicator Register	4.0	1.9	54

Store

STA	755	Store A	3.2	2.1	55
STQ	756	Store Q	3.2	2.1	55
STAQ	757	Store AQ	4.2	3.0	55
STXn	74n	Store Xn into Upper	3.2	2.1	55
SXLn	44n	Store Xn into Lower	3.2	2.1	56
SREG	753	Store Register	14.2	9.5	56
STCA	751	Store Character of A (6 Bit)	3.2	2.1	57
STCQ	752	Store Character of Q (6 Bit)	3.2	2.1	57
STBA	551	Store Character of A (9 Bit)	3.2	2.1	58
STBQ	552	Store Character of Q (9 Bit)	3.2	2.1	59
STI	754	Store Indicator Register	3.5	2.5	60
STT	454	Store Timer Register	3.0	2.1	61
SBAR	550	Store Base Address Register	3.5	2.5	61
STZ	450	Store Zero	3.5	2.5	61
STC1	554	Store Instruction Counter plus 1	3.5	2.5	62
STC2	750	Store Instruction Counter plus 2	3.5	2.5	62

Shift

ARS	731	A Right Shift	6.5	2.1	63
QRS	732	Q Right Shift	6.5	2.1	63
LRS	733	Long Right Shift	6.5	2.1	63

$\neq$See Calculation of Instruction Execution Times, page 42.

DATA MOVEMENT Shift				GE-615 Timing (μsec)$\neq$	GE-635 Timing (μsec)$\neq$	Reference (Page)
ALS	735	A Left Shift		6.5	2.1	64
QLS	736	Q Left Shift		6.5	2.1	64
LLS	737	Long Left Shift		6.5	2.1	65
ARL	771	A Right Logic		6.5	2.1	65
QRL	772	Q Right Logic		6.5	2.1	65
LRL	773	Long Right Logic		6.5	2.1	66
ALR	775	A Left Rotate		6.5	2.1	66
QLR	776	Q Left Rotate		6.5	2.1	66
LLR	777	Long Left Rotate		6.5	2.1	67

FIXED-POINT ARITHMETIC

Addition

			GE-615	GE-635	Ref
ADA	075	Add to A	4.0	1.9	68
ADQ	076	Add to Q	4.0	1.9	68
ADAQ	077	Add to AQ	4.3	1.9	69
ADXn	06n	Add to Xn	4.0	1.9	69
ASA	055	Add Stored to A	4.0	1.9	70
ASQ	056	Add Stored to Q	4.9	3.3	70
ASXn	04n	Add Stored to Xn	4.9	3.3	71
ADLA	035	Add Logic to A	4.0	1.9	71
ADLQ	036	Add Logic to Q	4.0	1.9	72
ADLAQ	037	Add Logic to AQ	4.0	1.9	72
ADLXn	02n	Add Logic to Xn	4.0	1.9	73
AWCA	071	Add with Carry to A	4.0	1.9	73
AWCQ	072	Add with Carry to Q	4.0	1.9	74
ADL	033	Add Low to AQ	4.0	1.9	75
AOS	054	Add One to Storage	4.9	3.3	75

Subtraction

			GE-615	GE-635	Ref
SBA	175	Subtract from A	4.0	1.9	76
SBQ	176	Subtract from Q	4.0	1.9	76
SBAQ	177	Subtract from AQ	4.0	1.9	77
SBXn	16n	Subtract from Xn	4.0	1.9	77
SSA	155	Subtract Stored from A	4.9	3.3	78
SSQ	156	Subtract Stored from Q	4.9	3.3	78
SSXn	14n	Subtract Stored from Xn	4.9	3.3	79

$\neq$See Calculation of Instruction Execution Times, page 42.

FIXED-POINT ARITHMETIC			GE-615 Timing (μsec)$\neq$	GE-635 Timing (μsec)$\neq$	Reference (Page)
Subtraction					
SBLA	135	Subtract Logic from A	4.0	1.9	79
SBLQ	136	Subtract Logic from Q	4.0	1.9	80
SBLAQ	137	Subtract Logic from AQ	4.0	1.9	80
SBLXn	12n	Subtract Logic from Xn	4.0	1.9	81
SWCA	171	Subtract with Carry from A	4.0	1.9	81
SWCQ	172	Subtract with Carry from Q	4.0	1.9	82
Multiplication					
MPY	402	Multiply Integer	19.2	7.6	83
MPF	401	Multiply Fraction	19.2	7.6	84
Division					
DIV	506	Divide Integer	29.4	15.1	85
DVF	507	Divide Fraction	29.4	15.1	86
Negate					
NEG	531	Negate A	2.6	1.6	87
NEGL	533	Negate Long	2.6	1.6	87

BOOLEAN OPERATIONS

AND					
ANA	375	AND to A	4.0	1.9	88
ANQ	376	AND to Q	4.0	1.9	88
ANAQ	377	AND to AQ	4.0	1.9	88
ANXn	36n	AND to Xn	4.0	1.9	89
ANSA	355	AND to Storage A	4.9	3.3	89
ANSQ	356	AND to Storage Q	4.9	3.3	89
ANSXn	34n	AND to Storage Xn	4.9	3.3	90

OR					
ORA	275	OR to A	4.0	1.9	90
ORQ	276	OR to Q	4.0	1.9	90
ORAQ	277	OR to AQ	4.0	1.9	91
ORXn	26n	OR to Xn	4.0	1.9	91
ORSA	255	OR to Storage A	4.9	3.3	91
ORSQ	256	OR to Storage Q	4.9	3.3	92
ORSXn	24n	OR to Storage Xn	4.9	3.3	92

$\neq$ See Calculation of Instruction Execution Times, page 42.

		BOOLEAN OPERATIONS	GE-615 Timing $(\mu sec)\neq$	GE-635 Timing $(\mu sec)\neq$	Reference (Page)

EXCLUSIVE OR

ERA	675	EXCLUSIVE OR to A	4.0	1.9	92
ERQ	676	EXCLUSIVE OR to Q	4.0	1.9	93
ERAQ	677	EXCLUSIVE OR to AQ	4.0	1.9	93
ERXn	66n	EXCLUSIVE OR to Xn	4.0	1.9	93
ERSA	655	EXCLUSIVE OR to Storage A	4.9	3.3	94
ERSQ	656	EXCLUSIVE OR to Storage Q	4.9	3.3	94
ERSXn	64n	EXCLUSIVE OR to Storage Xn	4.9	3.3	94

COMPARISON

Compare

CMPA	115	Compare with A	4.0	1.9	95
CMPQ	116	Compare with Q	4.0	1.9	96
CMPAQ	117	Compare with AQ	4.0	1.9	97
CMPXn	10n	Compare with Xn	4.0	1.9	98
CWL	111	Compare with Limits	4.8	1.9	99
CMG	405	Compare Magnitude	4.0	1.9	100
SZN	234	Set Zero and Negative Indicators from Memory	4.0	1.9	100
CMK	211	Compare Masked	4.8	1.9	101

Comparative AND

CANA	315	Comparative AND with A	4.0	1.9	102
CANQ	316	Comparative AND with Q	4.0	1.9	102
CANAQ	317	Comparative AND with AQ	4.0	1.9	102
CANXn	30n	Comparative AND with Xn	4.3	1.9	103

Comparative NOT

CNAA	215	Comparative NOT with A	4.0	1.9	103
CNAQ	216	Comparative NOT with Q	4.0	1.9	103
CNAAQ	217	Comparative NOT with AQ	4.0	1.9	104
CNAXn	20n	Comparative NOT with Xn	4.3	1.9	104

FLOATING POINT

Load

FLD	431	Floating Load	4.0	1.9	105
DFLD	433	Double-Precision Floating Load	4.3	1.9	105
LDE	411	Load Exponent Register	4.0	1.9	105

$\neq$ See Calculation of Instruction Execution Times, page 42.

FLOATING POINT			GE-615 Timing (μsec)$\neq$	GE-635 Timing (μsec)$\neq$	Reference (Page)
Store					
FST	455	Floating Store	3.2	2.1	106
DFST	457	Double-Precision Floating Store	4.2	3.0	106
STE	456	Store Exponent Register	3.2	2.1	106
FSTR*	470	Floating Store Rounded	4.8	2.9	106.1
Addition					
FAD	475	Floating Add	6.5	2.8	107
UFA	435	Unnormalized Floating Add	6.5	2.8	107
DFAD	477	Double-Precision Floating Add	6.2	2.8	108
DUFA	437	Double-Precision Unnormalized Floating Add	6.2	2.8	108
ADE	415	Add to Exponent Register	4.0	1.9	109
Subtraction					
FSB	575	Floating Subtract	6.5	2.8	109
UFS	535	Unnormalized Floating Subtract	6.5	2.8	110
DFSB	577	Double-Precision Floating Subtract	6.5	2.8	110
DUFS	537	Double-Precision Unnormalized Floating Subtract	6.5	2.8	111
Multiplication					
FMP	461	Floating Multiply	16.2	6.5	111
UFM	421	Unnormalized Floating Multiply	16.0	6.3	112
DFMP	463	Double-Precision Floating Multiply	31.0	12.7	112
DUFM	423	Double-Prec. Unnormal. Floating Multiply	31.0	12.4	113
Division					
FDV	565	Floating Divide	31.0	15.6	114
FDI	525	Floating Divide Inverted	31.0	15.1	115
DFDV	567	Double-Precision Floating Divide	48.0	25.1	116
DFDI	527	Double-Precision Floating Divide Inverted	47.0	24.7	117
Negate, Normalize					
FNEG	513	Floating Negate	3.4	2.3	118
FNO	573	Floating Normalize	3.4	1.9	118

$\neq$ See Calculation of Instruction Execution Times, page **42.**

*When normalization does not take place, subtract 0.3 microsecond from the listed time.

FLOATING POINT			GE-615 Timing (μsec)$\neq$	GE-635 Timing (μsec)$\neq$	Reference (Page)
Compare					
FCMP	515	Floating Compare	5.1	2.1	119
FCMG	425	Floating Compare Magnitude	5.1	2.1	120
DFCMP	517	Double-Precision Floating Compare	5.1	2.1	121
DFCMG	427	Double-Precision Floating Compare Magnitude	4.0	1.9	122
FSZN	430	Floating Set Zero and Negative Indicators from Memory	4.0	1.9	123

TRANSFER OF CONTROL

Transfer

TRA	710	Transfer Unconditionally	2.0	1.7	124
TSXn	70n	Transfer and Set Xn	3.0	1.7	124
TSS	715	Transfer and Set Slave Mode	2.0	1.7	124
RET	630	Return	4.0	3.3	125

Conditional Transfer

TZE	600	Transfer on Zero	2.0	1.7	126
TNZ	601	Transfer on Not Zero	2.0	1.7	126
TMI	604	Transfer on Minus	2.0	1.7	126
TPL	605	Transfer on Plus	2.0	1.7	126
TRC	603	Transfer on Carry	2.0	1.7	127
TNC	602	Transfer on No Carry	2.0	1.7	127
TOV	617	Transfer on Overflow	2.0	1.7	127
TEO	614	Transfer on Exponent Overflow	2.0	1.7	128
TEU	615	Transfer on Exponent Underflow	2.0	1.7	128
TTF	607	Transfer on Tally-Runout Indicator OFF	2.0	1.7	128

MISCELLANEOUS OPERATIONS

NOP	011	No Operation	2.0	1.4	129
BCD	505	Binary to Binary-Coded Decimal	9.5	4.1	129
GTB	774	Gray to Binary	31.3	11.2*	130
XEC	716	Execute	2.0	1.7	131
XED	717	Execute Double	2.0	1.7	131
MME	001	Master Mode Entry	3.0	2.3	132
DRL	022	Derail	3.0	2.3	133

$\neq$See Calculation of Instruction Execution Times, page **42.**

*Operations unit execution time only

MISCELLANEOUS OPERATIONS			GE-615 Timing (μsec)≠	GE-635 Timing (μsec)≠	Reference (Page)
RPT	520	Repeat	4.0	1.6	134
RPD	560	Repeat Double	4.0	1.6	137
RPL	500	Repeat Link	4.0	1.6	141

MASTER MODE OPERATIONS

Master Mode

DIS**	616	Delay Until Interrupt Signal	Indefinite		145
LBAR*	230	Load Base Address Register	4.1	1.9	145
LDT**	637	Load Timer Register	4.1	2.5	145
SMIC*	451	Set Memory Controller Interrupt Cells	4.0	1.9	146

Master Mode and Control Processor

RMCM*	233	Read Memory Controller Mask Registers	4.1	1.9	146
SMCM*	553	Set Memory Controller Mask Registers	4.0	3.0	147
CIOC*	015	Connect I/O Channel	4.0	2.5	148

≠See Calculation of Instruction Execution Times, page 42.
*Functions as NOP in slave mode
**Covers command fault if executed in slave mode

Mnemonic:	Page:	Mnemonic:	Page:	Mnemonic:	Page:	Mnemonic:	Page:
ADA	68	DFCMG	122	LDAQ	48	SBXn	77
ADAQ	69	DFCMP	121	LDE	105	SMCM	147
ADE	109	DFDI	117	LDI	54	SMIC	146
ADL	75	DFDV	116	LDT	145	SREG	56
ADLA	71	DFLD	105	LDQ	48	SSA	78
ADLAQ	72	DFMP	112	LDXn	49	SSQ	78
ADLQ	72	DFSB	110	LLR	67	SSXn	79
ADLXn	73	DFST	106	LLS	65	STA	55
ADQ	68	DIS	145	LREG	49	STAQ	55
ADXn	69	DIV	85	LRL	66	STBA	58
ALR	66	DRL	133	LRS	63	STBQ	59
ALS	64	DUFA	108	LXLn	49	STC1	62
ANA	88	DUFM	113			STC2	62
ANAQ	88	DUFS	111	MME	132	STCA	57
ANQ	88	DVF	86	MPF	84	STCQ	57
ANSA	89			MPY	83	STE	106
ANSQ	89	EAA	52			STI	60
ANSXn	90	EAQ	53	NEG	87	STQ	55
ANXn	89	EAXn	53	NEGL	87	STT	61
AOS	75	ERA	92	NOP	129	STXn	55
ARL	65	ERAQ	93			STZ	61
ARS	63	ERQ	93	ORA	90	SWCA	81
ASA	70	ERSA	94	ORAQ	91	SWCQ	82
ASQ	70	ERSQ	94	ORQ	90	SXLn	56
ASXn	71	ERSXn	94	ORSA	91	SZN	100
AWCA	73	ERXn	93	ORSQ	92		
AWCQ	74			ORSXn	92	TEO	128
		FAD	107	ORXn	91	TEU	128
BCD	129	FCMG	120			TMI	126
		FCMP	119	QLR	66	TNC	127
CANA	102	FDI	115	QLS	64	TNZ	126
CANAQ	102	FDV	114	QRL	65	TOV	127
CANQ	102	FLD	105	QRS	63	TPL	126
CANXn	103	FMP	111			TRA	124
CIOC	148	FNEG	118	RET	125	TRC	127
CMG	100	FNO	118	RMCM	146	TSS	124
CMK	101	FSB	109	RPD	137	TSXn	124
CMPA	95	FST	106	RPL	141	TTF	128
CMPAQ	97	FSTR	106.1	RPT	134	TZE	126
CMPQ	96	FSZN	123				
CMPXn	98	GTB	130	SBA	76	UFA	107
CNAA	103			SBAQ	77	UFM	112
CNAAQ	104	LBAR	145	SBAR	61	UFS	110
CNAQ	103	LCA	50	SBLA	79		
CNAXn	104	LCAQ	51	SBLAQ	80	XEC	131
CWL	99	LCQ	51	SBLQ	80	XED	131
		LCXn	52	SBLXn	81		
DFAD	108	LDA	48	SBQ	76		

APPENDIX C. GE-625/635 INSTRUCTION MNEMONICS CORRELATED WITH THEIR OPERATION CODES

GE-625/635 Mnemonics and Operation Codes **GENERAL ⑧ ELECTRIC**

	000	001	002	003	004	005	006	007	010	011	012	013	014	015	016	017
000		MME	DRL							NOP				CIØC		
020	ADLX0	ADLX1	ADLX2	ADLX3	ADLX4	ADLX5	ADLX6	ADLX7				ADL		ADLA	ADLQ	ADLAQ
040	ASX0	ASX1	ASX2	ASX3	ASX4	ASX5	ASX6	ASX7					AOS	ASA	ASQ	
060	ADX0	ADX1	ADX2	ADX3	ADX4	ADX5	ADX6	ADX7		AWCA	AWCQ	LREG		ADA	ADQ	ADAQ
100	CMPX0	CMPX1	CMPX2	CMPX3	CMPX4	CMPX5	CMPX6	CMPX7		CWL				CMPA	CMPQ	CMPAQ
120	SBLX0	SBLX1	SBLX2	SBLX3	SBLX4	SBLX5	SBLX6	SBLX7						SBLA	SBLQ	SBLAQ
140	SSX0	SSX1	SSX2	SSX3	SSX4	SSX5	SSX6	SSX7						SSA	SSQ	
160	SBX0	SBX1	SBX2	SBX3	SBX4	SBX5	SBX6	SBX7		SWCA	SWCQ			SBA	SBQ	SBAQ
200	CNAX0	CNAX1	CNAX2	CNAX3	CNAX4	CNAX5	CNAX6	CNAX7		CMK				CNAA	CNAQ	CNAAQ
220	LDX0	LDX1	LDX2	LDX3	LDX4	LDX5	LDX6	LDX7	LBAR			RMCM	SZN	LDA	LDQ	LDAQ
240	ØRSX0	ØRSX1	ØRSX2	ØRSX3	ØRSX4	ØRSX5	ØRSX6	ØRSX7						ØRSA	ØRSQ	
260	ØRX0	ØRX1	ØRX2	ØRX3	ØRX4	ØRX5	ØRX6	ØRX7						ØRA	ØRQ	ØRAQ
300	CANX0	CANX1	CANX2	CANX3	CANX4	CANX5	CANX6	CANX7						CANA	CANQ	CANAQ
320	LCX0	LCX1	LCX2	LCX3	LCX4	LCX5	LCX6	LCX7						LCA	LCQ	LCAQ
340	ANSX0	ANSX1	ANSX2	ANSX3	ANSX4	ANSX5	ANSX6	ANSX7						ANSA	ANSQ	
360	ANX0	ANX1	ANX2	ANX3	ANX4	ANX5	ANX6	ANX7						ANA	ANQ	ANAQ
400		MPF	MPY			CMG				LDE				ADE		
420		UFM		DUFM		FCMG		DFCMG	FSZN	FLD		DFLD		UFA		DUFA
440	SXL0	SXL1	SXL2	SXL3	SXL4	SXL5	SXL6	SXL7	STZ	SMIC			STT	FST	STE	DFST
460		FMP		DFMP					FSTR					FAD		DFAD
500	RPL					BCD	DIV	DVF				FNEG		FCMP		DFCMP
520	RPT					FDI		DFDI		NEG		NEGL		UFS		DUFS
540									SBAR	STBA	STBQ	SMCM	STC1			
560	RPD					FDV		DFDV				FNØ		FSB		DFSB
600	TZE	TNZ	TNC	TRC	TMI	TPL		TTF					TEØ	TEU	DIS	TØV
620	EAX0	EAX1	EAX2	EAX3	EAX4	EAX5	EAX6	EAX7	RET				LDI	EAA	EAQ	LDT
640	ERSX0	ERSX1	ERSX2	ERSX3	ERSX4	ERSX5	ERSX6	ERSX7						ERSA	ERSQ	
660	ERX0	ERX1	ERX2	ERX3	ERX4	ERX5	ERX6	ERX7						ERA	ERQ	ERAQ
700	TSX0	TSX1	TSX2	TSX3	TSX4	TSX5	TSX6	TSX7	TRA					TSS	XEC	XED
720	LXL0	LXL1	LXL2	LXL3	LXL4	LXL5	LXL6	LXL7		ARS	QRS	LRS		ALS	QLS	LLS
740	STX0	STX1	STX2	STX3	STX4	STX5	STX6	STX7	STC2	STCA	STCQ	SREG	STI	STA	STQ	STAQ
760										ARL	QRL	LRL	GTB	ALR	QLR	LLR
	000	001	002	003	004	005	006	007	010	011	012	013	014	015	016	017

PSEUDO-OPERATIONS

PSEUDO-OPERATION MNEMONIC	FUNCTIONS	PAGE

CONTROL PSEUDO-OPERATIONS

DETAIL ON/OFF	(Detail output listing)	177
LIST ON/OFF	(Control output listing)	177
PCC ON/OFF	(Print control cards)	178
INHIB ON/OFF	(Inhibit interrupts)	179
PMC ON/OFF	(Print MACRO expansion)	179
REF ON/OFF	(References)	178
PUNCH ON/OFF	(Control card output)	179
EDITP	(Edit Print Lines)	180
EJECT	(Restore output listing)	180
REM	(Remarks)	180
*	(* in column one -- remarks)	181
LBL	(Label)	181
TTL	(Title)	182
TTLS	(Subtitle)	182
DATE	(Current date)	182
ABS	(Output absolute text)	183
FUL	(Output full binary text)	183
TCD	(Punch transfer card)	183
HEAD	(Heading)	184
DCARD	(Punch BCD Card)	185
END	(End of assembly)	186
OPD	(Operation definition)	186
OPSYN	(Operation synonym)	188
REFMA ON/OFF	(Reference Macros)	188

LOCATION COUNTER PSEUDO-OPERATIONS

USE	(Use multiple location counters)	188
BEGIN	(Origin of a location counter)	189
ORG	(Origin set by programmer)	190
LOC	(Location of output text)	190

SYMBOL DEFINING PSEUDO-OPERATIONS

EQU	(Equal to)	191
FEQU	(Equal to symbol as yet undefined)	191
BOOL	(Boolean)	192
SET	(Symbol redefinition)	192
MIN	(Minimum)	193
MAX	(Maximum)	193
SYMDEF	(Symbol definition)	193
SYMREF	(Symbol reference)	194
NULL	(Symbol EQU*)	195
EVEN	(Force Location Counter Even)	195
ODD	(Force Location Counter Odd)	195
EIGHT	(Force Location Counter to Multiple of	196

PSEUDO-OPERATIONS

CPB-1004F

SYSTEM SYMBOLS

The Assembler recognizes the following group of system symbols when the programmer enters any of them in the variable field of the Master Mode Entry (MME) machine instruction. (See Chapter II.) These MME instructions then serve as interfaces between the user and modules of the Comprehensive Operating Supervisor for special purposes (suggested in the meanings in the list following).

The table below indicates the system mnemonic symbol, its meaning, and the associated decimal value substituted in the MME address field by the Assembler.

SYMBOL	MEANING	DECIMAL VALUE
GEINOS	Input/Output Initiation	1
GEROAD	Roadblock	2
GEFADD	Physical File Address Request	3
GERELS	Component Release	4
GESNAP	Snapshot Dump	5
GELAPS	(Elapsed) Time Request	6
GEFINI	Terminal Transfer to Monitor	7
GEBORT	Aborting of Programs	8
GEMORE	Additional Memory of Peripherals	9
GEFCON	File Control Block Request	10
GEFILS	File Switching Request	11
GESETS	Set Switch Request	12
GERETS	Reset Switch Request	13
GEENDC	Terminate Courtesy Call	14
GERELC	Relinquish Control	15
GESPEC	Special Interrupt Courtesy Call Request	16
GETIME	Date and Time-of-Day Request	17
GECALL	System Loader	18
GESAVE	Write File in System Format	19
GERSTR	Read File In System Format	20
GEMREL	Release Memory	21
GESYOT	Write on SYSOUT	22
GECHEK	Check Point	23
GEROUT	Output to Remote Terminal	24
GEROLL	Reinitiate or Rollback Program	25
GEUSER	User-Supplied MME	26
GELOOP	Loop Protection	27
GEWAKE	Call Me Later	28
GEIDSE	Journalization and Subfile Page Range	29
.EMM	Enter Master Mode	30
GELBAR	Load Base Address Register	31
GEFRCE	GEFRC Entry	32
GEFSYE	File System Entry Point	33
GEPRIO	I/O Priority	34
GENEWS	Spawn New Job	35

INPUT/OUTPUT COMMAND FORMATS

The following listing of input/output commands is for use when coding directly to the Input/Output Supervisor within the Comprehensive Operating Supervisor.

Designators used in the listing below are:

xxxx = 0000 for Slave Mode programs
xxxx = physical device code for Master Mode programs
da = Device Address (Used only in Master Mode programs; see input/output select sequence coding, Operating Supervisor reference manual.)
ca = Channel Address (used only in Master Mode programs; see input/output select sequence coding, Operating Supervisor reference manual.)
nn = number of records (01-63)
 = 01 when subfield for nn is blank
cc = octal character to be used as file mark

COMMAND DESCRIPTION	PSEUDO-OPERATION	VARIABLE FIELD	OCTAL REPRESENTATION
Request Status	REQS	da, ca	00 xxxx 020001
Reset Status	RESS	da, ca	40 xxxx 020001
Read Card Binary	RCB	da, ca	01 xxxx 000000
Read Card Binary Continuous	RCBC	nn, da, ca	01 xxxx 0600nn
Read Card Decimal	RCD	da, ca	02 xxxx 000000
Read Card Decimal Continuous	RCDC	nn, da, ca	02 xxxx 0600nn
Read Card Mixed	RCM	da, ca	03 xxxx 000000
Read Card Mixed Continuous	RCMC	nn, da, ca	03 xxxx 0600nn
Write Card Binary	WCB	da, ca	11 xxxx 040014
Write Card Binary Continuous	WCBC	nn, da, ca	11 xxxx 0600nn
Write Card Decimal	WCD	da, ca	12 xxxx 040014
Write Card Decimal Continuous	WCDC	nn, da, ca	12 xxxx 0600nn
Write Card Decimal Edited	WCDE	da, ca	13 xxxx 040014
Write Card Decimal Edited Continuous	WCDEC	nn, da, ca	13 xxxx 0600nn
Write Printer	WPR	da, ca	10 xxxx 000000
Write Printer Continuous	WPRC	nn, da, ca	10 xxxx 0600nn
Write Printer Edited	WPRE	da, ca	30 xxxx 000000
Write Printer Edited Continuous	WPREC	nn, da, ca	30 xxxx 0600nn

COMMAND DESCRIPTION	PSEUDO-OPERATION	VARIABLE FIELD	OCTAL REPRESENTATION
Read Tape Binary	RTB	da, ca	05 xxxx 000000
Re-Read Tape Binary	RRTB	da, ca	07 xxxx 000000
Read Tape Decimal	RTD	da, ca	04 xxxx 000000
Re-Read Tape Decimal	RRTD	da, ca	06 xxxx 000000
Read Tape 9 Channel	RT9	da, ca	03 xxxx 000000
Write Tape 9 Channel	WT9	da, ca	13 xxxx 000000
Write Tape Binary	WTB	da, ca	15 xxxx 000000
Write Tape Decimal	WTD	da, ca	14 xxxx 000000
Write End-of-File	WEF	da, ca	55 xxxx 101700
Write File Mark	WFM	cc, da, ca	15 xxxx 10cc00
Write File Mark Decimal	WFMD	cc, da, ca	14 xxxx 10cc00
Erase	ERASE	da, ca	54 xxxx 020001
Backspace Record(s)	BSR	nn, da, ca	46 xxxx 0200nn
Backspace File	BSF	da, ca	47 xxxx 020001

COMMAND DESCRIPTION	PSEUDO-OPERATION	VARIABLE FIELD	OCTAL REPRESENTATION
Forward Space Record(s)	FSR	nn, da, ca	44 xxxx 0200nn
Forward Space File	FSF	da, ca	45 xxxx 020001
Rewind	REW	da, ca	70 xxxx 020001
Rewind and Standby	REWS	da, ca	72 xxxx 020001
Set Low Density	SLD	da, ca	61 xxxx 020001
Set High Density	SHD	da, ca	60 xxxx 020001
Seek Disc Address	SDIA	da, ca	34 xxxx 000002
Read Disc Continuous	RDIC	da, ca	25 xxxx 002400
Write Disc Continuous	WDIC	da, ca	31 xxxx 002400
Write Disc Continuous and Verify	WDICV	da, ca	33 xxxx 002400
Select Drum Address	SDRA	da, ca	34 xxxx 000002
Read Drum	RDR	da, ca	25 xxxx 000000
Write Drum	WDR	da, ca	31 xxxx 000000
Write Drum and Verify	WDRV	da, ca	33 xxxx 000000
Drum Compare and Verify	DRCV	da, ca	11 xxxx 000000
Read Perforated Tape	RDPT	da, ca	02 xxxx 000000
Write Perforated Tape	WPT	da, ca	11 xxxx 000000
Write Perforated Tape Edited	WPTE	da, ca	31 xxxx 000000
Write Perforated Tape--Single Character	WPTSC	da, ca	16 xxxx 000000
Write Perforated Tape--Double Character	WPTDC	da, ca	13 xxxx 000000
Read Typewriter	RTYP	da, ca	03 xxxx 000000
Write Typewriter	WTYP	da, ca	13 xxxx 000000
Write Typewriter and Return to Read	WTYPR	da, ca	13 xxxx 000002
Read DATANET-30	RDN	da, ca	01 xxxx 000000
Write DATANET-30	WDN	da, ca	10 xxxx 000000

DATA CONTROL WORD FORMATS

The Data Control Word format listing below contains designators as follows:

a = address of the data block
c = word count of data to be transferred per block
$xxxx$ = ignored by the Assembler

DESCRIPTION	PSEUDO-OPERATION	VARIABLE FIELD	OCTAL REPRESENTATION
Transmit and Disconnect	IOTD	a, c	aaaaaa00cccc
Transmit and Proceed	IOTP	a, c	aaaaaa01cccc
Non-Transmit and Proceed	IONTP	a, c	aaaaaa03cccc
Transfer to Data Control Word	TDCW	a	aaaaaa02xxxx

APPENDIX F. GE-615/635 STANDARD CHARACTER SET

STANDARD CHARACTER SET	GE-INTERNAL MACHINE CODE	OCTAL	HOLLERITH CARD CODE	ASCII CODE	UASCI CODE
0	000000	00	0	060	060
1	000001	01	1	061	061
2	000010	02	2	062	062
3	000011	03	3	063	063
4	000100	04	4	064	064
5	000101	05	5	065	065
6	000110	06	6	066	066
7	000111	07	7	067	067
8	001000	10	8	070	070
9	001001	11	9	071	071
[	001010	12	2-8	133	133
#	001011	13	3-8	043	043
@	001100	14	4-8	100	100
:	001101	15	5-8	072	072
>	001110	16	6-8	076	076
?	001111	17	7-8	077	077
b	010000	20	(blank)	040	040
A	010001	21	12-1	141	101
B	010010	22	12-2	142	102
C	010011	23	12-3	143	103
D	010100	24	12-4	144	104
E	010101	25	12-5	145	105
F	010110	26	12-6	146	106
G	010111	27	12-7	147	107
H	011000	30	12-8	150	110
I	011001	31	12-9	151	111
&	011010	32	12	046	046
.	011011	33	12-3-8	056	056
]	011100	34	12-4-8	135	135
(	011101	35	12-5-8	050	050
<	011110	36	12-6-8	074	074
\	011111	37	12-7-8	134	134
↑	100000	40	11-0	136	136
J	100001	41	11-1	152	112
K	100010	42	11-2	153	113
L	100011	43	11-3	154	114
M	100100	44	11-4	155	115
N	100101	45	11-5	156	116
O	100110	46	11-6	157	117
P	100111	47	11-7	160	120
Q	101000	50	11-8	161	121
R	101001	51	11-9	162	122
-	101010	52	11	055	055
$	101011	53	11-3-8	044	044
*	101100	54	11-4-8	052	052
)	101101	55	11-5-8	051	051
;	101110	56	11-6-8	073	073
'	101111	57	11-7-8	047	047
+	110000	60	12-0	053	053
/	110001	61	0-1	057	057
S	110010	62	0-2	163	123
T	110011	63	0-3	164	124
U	110100	64	0-4	165	125
V	110101	65	0-5	166	126
W	110110	66	0-6	167	127
X	110111	67	0-7	170	130
Y	111000	70	0-8	171	131
Z	111001	71	0-9	172	132
←	111010	72	0-2-8	137	137
,	111011	73	0-3-8	054	054
%	111100	74	0-4-8	045	045
=	111101	75	0-5-8	075	075
"	111110	76	0-6-8	042	042
!	111111	77	0-7-8	041	041

APPENDIX G. CONVERSION TABLE
OCTAL-DECIMAL INTEGERS AND FRACTIONS

Octal	10000	20000	30000	40000	50000	60000	70000
Decimal	4096	8192	12288	16384	20480	24576	28672

Octal	100000	200000	300000	400000	500000	600000	700000	1000000
Decimal	32768	65536	98304	131072	163840	196608	229376	262144

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	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	0649	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

CPB-1004F

Octal	10000	20000	30000	40000	50000	60000	70000
Decimal	4096	8192	12288	16384	20480	24576	28672

Octal	100000	200000	300000	400000	500000	600000	700000	1000000
Decimal	32768	65536	98304	131072	163840	196608	229376	262144

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	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	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	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

CPB-1004F

Octal	10000	20000	30000	40000	50000	60000	70000
Decimal	4096	8192	12288	16384	20480	24576	28672

Octal	100000	200000	300000	400000	500000	600000	700000	1000000
Decimal	32768	65536	98304	131072	163840	196608	229376	262144

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	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	2537	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

CPB-1004F

Octal	10000	20000	30000	40000	50000	60000	70000
Decimal	4096	8192	12288	16384	20480	24576	28672

Octal	100000	200000	300000	400000	500000	600000	700000	1000000
Decimal	32768	65536	98304	131072	163840	196608	229376	262144

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	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

CPB-1004F

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	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	.000919
.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

CPB-1004F

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

CPB-1004F

APPENDIX H. TABLES OF POWERS OF TWO AND BINARY DECIMAL EQUIVALENTS

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

TABLE OF POWERS OF 2

CPB-1004F

281

Maximum Decimal Integral Value	Number of Decimal Digits	Number of Bits	Maximum Decimal Fractional Value
1		1	.5
3		2	.75
7		3	.875
15	1	4	.937 5
31		5	.968 75
63		6	.984 375
127	2	7	.992 187 5
255		8	.996 093 75
511		9	.998 046 875
1 023	3	10	.999 023 437 5
2 047		11	.999 511 718 75
4 095		12	.999 755 859 375
8 191		13	.999 877 929 687 5
16 383	4	14	.999 938 964 843 75
32 767		15	.999 969 482 421 875
65 535		16	.999 984 741 210 937 5
131 071	5	17	.999 992 370 605 468 75
262 143		18	.999 996 185 302 734 375
524 287		19	.999 998 092 651 367 187 5
1 048 575	6	20	.999 999 046 325 683 593 75
2 097 151		21	.999 999 523 162 841 796 875
4 194 303		22	.999 999 761 581 420 898 437 5
8 388 607		23	.999 999 880 790 710 449 218 75
16 777 215	7	24	.999 999 940 395 355 244 609 375
33 554 431		25	.999 999 970 197 677 612 304 687 5
67 108 863		26	.999 999 985 098 838 806 152 343 75
134 217 727	8	27	.999 999 992 549 419 403 076 171 875
268 435 455		28	.999 999 996 274 709 701 538 085 937 5
536 870 911		29	.999 999 998 137 354 850 769 042 968 75
1 073 741 823	9	30	.999 999 999 068 677 425 384 521 484 375
2 147 483 647		31	.999 999 999 534 338 712 692 260 742 187 5
4 294 967 295		32	.999 999 999 767 169 356 346 130 371 093 75
8 589 934 591		33	.999 999 999 883 584 678 173 065 185 546 875
17 179 869 183	10	34	.999 999 999 941 792 339 086 532 592 773 437 5
34 359 738 367		35	.999 999 999 970 896 169 543 266 296 386 718 75
68 719 476 735		36	.999 999 999 985 448 034 771 633 148 193 359 375
137 438 953 471	11	37	.999 999 999 992 724 042 385 816 574 096 679 687 5
274 877 906 943		38	.999 999 999 996 362 021 192 908 287 048 339 843 75
549 755 813 887		39	.999 999 999 998 181 010 596 454 143 524 169 921 875
1 099 511 627 775	12	40	.999 999 999 999 090 505 298 227 071 762 084 960 937 5
2 199 023 255 551		41	.999 999 999 999 545 252 649 113 535 881 042 480 468 75
4 398 046 511 103		42	.999 999 999 999 772 626 324 556 767 940 521 240 234 375
8 796 093 022 207		43	.999 999 999 999 886 313 162 278 383 970 260 620 117 187 5
17 592 186 044 415	13	44	.999 999 999 999 943 156 581 139 191 985 130 310 058 593 75
35 184 372 088 831		45	.999 999 999 999 971 578 290 569 595 992 565 155 029 296 875
70 368 744 177 663		46	.999 999 999 999 985 789 145 284 797 996 282 577 514 648 437 5
140 737 488 355 327	14	47	.999 999 999 999 992 894 572 642 398 998 141 288 757 324 218 75
281 474 976 710 655		48	

This chart provides the information necessary to determine:

a. The number of bits needed to represent a given decimal number. Use columns one and three or four and three.

b. The number of bits needed to represent a given number of decimal digits (all nines). Use columns two and three.

c. The maximum decimal value represented by a given number of bits, use columns one and three or three and four.

CPB-1004F

282

APPENDIX I. THE TWO'S COMPLEMENT NUMBER SYSTEM

Let us first consider a simple example of two's complement numbers, namely integers of three bits each, numbering the bits 0, 1, and 2, respectively, from left to right. Then the integer "xyz" represents the decimal quantity "-4x+2y+z":

hence	011	represents	+3
	010	represents	+2
	001	represents	+1
	000	represents	+0
	111	represents	-1
	110	represents	-2
	101	represents	-3
and	100	represents	-4

Thus each decimal integer from -4 to 3 has a unique representation as a two's complement number. Bit 0 also serves as the sign-bit, since it is 0 for all positive numbers and 1 for all negative numbers. Note that "000" is a positive number.

We perform the addition "abc+xyz" as though "abc" and "xyz" were signless binary integers from 0 to 7, ignoring any carry out of bit 0 of the sum. If the true sum is not an integer from -4 to 3, then we have an overflow. We observe that the carry out of bit 0 = the carry out of bit 1 if, and only if, there is no overflow. In the case when a $\neq$ x, we cannot have an overflow, since the sum ranges from -4 to 2. It follows that a + x =1 and that the carries must be equal, since we have 0+1 = 1 with carry 0 and 1+1 = 0 with carry 1. In the case when a = x, we have no overflow if, and only if, bit 0 of the sum = x. We have this equality if, and only if, the carries are equal, since we have 0+0+0 = 0 with carry 0 and 1+1+1 = 1 with carry 1. We conclude that our overflow test is a valid one. The following examples are illustrations of two's complement addition:

CARRIES	00	11	00	01	11	10
abc	110=-2	110=-2	010=+2	010=+2	110=-2	110=-2
xyz	001=+1	011=+3	001=+1	011=+3	111=-1	101=-3
abc+xyz	111=-1	001=+1	011=+3	101=-3	101=-3	011=+3
REMARKS	NO OVF.	NO OVF.	NO OVF.	OVF.	NO OVF.	OVF.

We say that "uvw" is the one's complement of "xyz" (and vice versa) if uvw+xyz = 111. Hence u+x = v+y = w+x = 1. We say that the quantity "uvw+001" is the two's complement of "xyz", observing that its decimal value is:

$$-4u+2v+w+1 = -4(1-x)+2(1-y)+(1-z)+1$$
$$= -(-4x+2y+z),$$

or minus the value of "xyz". For this reason we call "xyz" a two's complement number. We perform the subtraction "abc-xyz" by the triple addition "abc+uvw+001" (in effect, by

adding "abc" and "uvw" with a forced carry of 1 into the low order bit 2). We use the same overflow test as for addition. Note that 000-000 = 000 (no overflow) and that 000-100 = 100 (overflow). Hence "000" is its own two's complement, and "100" does not have a proper two's complement. We note the conspicuous absence of a -0 from the two's complement system on the previous page.

We may generalize the above discussion to include two's complement integers of N bits each. The integer "$x_0 x_1 x_2 \cdots x_{N-2} x_{N-1}$" represents the decimal quantity below:

$$-2^{N-1}x_0 + 2^{N-2}x_1 + 2^{N-3}x_2 + \ldots + 2x_{N-2} = x_{N-1}$$

The same rules as above hold for addition, overflow, complementation, and subtraction. In the GE-600 hardware, we may have several choices for N:

         N=8 for exponent fields,
         N=18 for address fields,
         N=36 for single-precision integers,
and   N=72 for double-precision integers.

The use of two's complement numbers offers many advantages:

1. It eliminates housekeeping before and after addition and subtraction in the computer hardware.

2. It permits addition and subtraction modulo 2^N, since we may always consider a number to be signless.

3. It permits addition of a quantity to a field of a word, without any need to worry about the sign-bit. (In the sign-magnitude system, one would add the quantity if the sign were positive, and subtract the quantity if the sign were negative.)

4. It makes zero a unique positive number.

5. It is compatible with index register arithmetic.

Of course, the GE-600 programmer must always be aware of the fact that the computer is a two's complement machine, especially when converting programs that were originally written for a machine with sign-magnitude or one's complement arithmetic. For example, the sign magnitude convention of "changing sign" corresponds to the two's complement convention of "negation" (or "complementation"). In FORTRAN systems, the quantity -0 often indicates a blank card field. There is no such quantity in the GE-600 system, whether in fixed or floating point.

A two's complement floating point number in the Floating Point Register consists of two parts:

1. An integral exponent field of eight bits.

2. A fractional mantissa field of seventy-two bits. The mantissa "$x_0 x_1 x_2 \cdots x_{71}$" represents the decimal quantity below:

$$-x_0 + 2^{-1}x_1 + 2^{-2}x_2 + \ldots + 2^{-71}x_{71}$$

We say that a floating point number is normalized if either:

1. The exponent field is 10000000 and the mantissa field is zero, or

2. The first two mantissa bits are different: $x_0 \neq x_1$

The value of a floating point number is mantissa $*2^{\text{exponent}}$. Hence the normal form of +1 is exponent 00000001 and mantissa 0100...0, and the normal form of -1 is exponent 00000000 and mantissa 1000...0. If "f" is a floating point number that is not a power of two, however, then both +f and -f have the same exponent fields in normal form, and their mantissa fields are two's complements of each other. For f=10, the normal form of +f is exponent 00000100 and mantissa 010100...0. The normal form of -f is exponent 00000100 and mantissa 101100...0. Note that the first bit of the mantissa is the sign-bit of the number.

Since $\quad -x_0 + 2^{-1}x_0 + 2^{-2}x_1 + \ldots + 2^{-71}x_{70} = \frac{1}{2}(-x_0 + 2^{-1}x_1 + \ldots + 2^{-71}x_{71})$,

ignoring the remainder, the GE-600 hardware retains the value of bit 0 during each right shift cycle prior to a floating point addition or subtraction. For the same reason, there is a numeric right shift as well as a logical right shift for A, Q, and AQ.

The representation of mixed numbers illustrates a feature of the two's complement number system. Consider the case of f=1.25. Then the normal form of +f is exponent 00000001 and mantissa 010100...0. The normal form of -f is exponent 00000001 and mantissa 101100...0. The integral part of +f is +1, and the fractional part of +f is +.25. The integral part of -f is -2, and the fractional part of -f is +.75. Hence the integral parts are one's complements of each other, and the fractional parts are two's complements of each other. In general, this condition holds whenever we divide a two's complement number into a pair of disjoint fields, where the right field is not zero. The reason for the condition is that the sign-bit of a two's complement number is the only bit with a negative value. The condition is desirable in some mathematical applications where we wish to compute the greatest integer less than or equal to a given number. However, the condition raises a compatibility problem when converting programs originally coded on sign-magnitude or one's complement machines. The solution to the problem is the addition of +1 to the integral part of nonwhole negative numbers. The problem arises noticeably in the implementation of FORTRAN built-in functions.

KEYWORD INDEX

This Keyword Index is formed by permuting - that is, by shifting the normal sequence of - the words appearing in equipment and program titles, chapter and paragraph headings, figure titles, significant words, and descriptive phrases found in this manual.

Each keyword appears in the index position in the center of the page, and the other words associated with it in the title, heading, or phrase appear on either side of the index word position. Thus, the desired keyword is first located in the center index position by the user of the manual; the nature of the context in which the keyword appears on the page shown is then given by the explanatory words appearing on either side of the keyword.

When a phrase is too long to be printed at the left of the keyword, a truncated portion is placed at the right, preceded by an asterisk. Information is occasionally added in brackets to amplify the meaning of the phrase.

CPB-1004F

CPB-1004F

CPB-1004F

CPB-1004F

INDEX 17

CPB-1004F

CPB-1004F

CPB-1004F

INDEX 35

CPB-1004F

INDEX 36

CPB-1004F

CPB-1004F

DOCUMENT REVIEW SHEET

TITLE: GE-625/635 Programming Reference Manual

CPB #: 1004F

From:

Name: _____

Position: _____

Address: _____

_____

Comments concerning this publication are solicited for use in improving future editions. Please provide any recommended additions, deletions, corrections, or other information you deem necessary for improving this manual. The following space is provided for your comments.

COMMENTS: _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Please cut along this line

NO POSTAGE NECESSARY IF MAILED IN U.S.A.
Fold on two lines shown on reverse
side, staple, and mail.

FOLD

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

GENERAL ELECTRIC COMPANY

INFORMATION SYSTEMS EQUIPMENT DIVISION
13430 NORTH BLACK CANYON HIGHWAY
PHOENIX, ARIZONA   85029

ATTENTION: DOCUMENTATION, LARGE SYSTEMS DEPARTMENT, C-77

FOLD