TEXAS INSTRUMENTS

Improving Man's Effectiveness Through Electronics

Model 960 Computer

Assembly Language
Programmer's Reference Manual

MANUAL NO. 942779-9701
ORIGINAL ISSUE 1 SEPTEMBER 1975
REVISED 1 APRIL 1977
INCLUDES
CHANGE 1 . . . . . 2 APRIL 1977
CHANGE 2 . . . . . 15 MAY 1979

Digital Systems Division

LIST OF EFFECTIVE PAGES

INSERT LATEST CHANGED PAGES DESTROY SUPERSEDED PAGES

Note: The portion of the text affected by the changes is indicated by a vertical bar in the outer margins of the page.

Model 960 Computer Assembly Language Programmer's Reference Manual (942779-9701)

Original Issue . . . . 1 September 1975
Revised and Reissued . . . . 1 March 1976
Revised and Reissued . . . . 1 August 1976 (ECN 406941)
Revised and Reissued . . . . 1 April 1977 (ECN 419551)
Change 1 . . . . 2 April 1977 (ECN 419551)
Change 2 . . . . 15 May 1979 (ECN 449628)

Total number of pages in this publication is 218 consisting of the following:

PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.
Cover	2	Appendix A Div	0		
Effective Pages	2	A-1 - A-4	0		
iii - vi	0	Appendix B Div	0		
vii	2	B-1 - B-16	0		
viii	0	Appendix C Div	0		
ix/x	2	C-1 - C-14	0		
1-1 - 1-2	0	Appendix D Div	0		
2-1 - 2-22	0	D-1 - D-4	0		
3-1 - 3-46	0	Appendix E Div	0		
4-1 - 4-8	0	E-1 - E-2	0		
5-1 - 5-6	0	Appendix F Div	0		
5-7 - 5-8	1	F-1 - F-22	0		
5-9 - 5-12	0	Appendix G Div	0		
6-1 - 6-6	0	G-1 - G-2	0		
6-7	2	Appendix H Div	0		
6-8 - 6-10	0	H-1 - H-2	0		
7-1 - 7-2	0	Alphabetical Index Div.	0		
7-3 - 7-4	1	Index-1 - Index-4	0		
7-5 - 7-6	0	User's Response	2		
7-7 - 7-8B	2	Business Reply	2		
7-9 - 7-10	2	Cover Blank	0		
7-11 - 7-22	0	Cover	0		

PREFACE

This manual incorporates programming reference data for the Texas Instruments models 960A and 960B computers. Some of this information was previously found in the *Model 960A Computer Programmer's Reference Manual,* manual number 958360-9701. The detailed information about the Symbolic Assembly Language (SAL) used on the 960 series was formerly found in the *Model 960 Computer Assembly Language Programmer's Guide,* manual number 942769-9701. This manual supersedes both previous manuals.

This manual consists of seven sections and eight appendixes. A brief description of each element follows.

Section I General Information—This section contains general information about the equipment and the software that is available to be used with it.

Section II Hardware Features—This section contains information about the Model 960A and 960B hardware and their features.

Section III Machine Instructions—This section contains information about the machine instructions for the 960 series computers including format, operation code, mnemonic, operands, and the types of addressing used with each.

Section IV Language Requirements—This section provides a format of source statements and a description of the fields and symbols used.

Section V Assembler Directives—This section contains information about the directives that are available for use with the SAL assembler and how to use them.

Section VI Programming Techniques—This section contains programming techniques to be used by new users of the equipment. Also included is information about common subroutines and program modules.

Section VII SAL Inputs and Outputs—This section contains information about assembler input/output formats, etc. It also contains information about the loading and executing of the assembler.

Appendix A SAL Character Set—This appendix specifies the Hollerith and ASCII codes of the character set. The decimal and hexadecimal equivalent of each ASCII code is provided.

Appendix B General Tables—This appendix contains arithmetic, conversion, powers, and common mathematical constants tables and paper-tape ASCII character arrangement.

Appendix C Instruction Tables—This appendix contains tabular material about the source formats and operation codes for the 960 series machine instructions.

Appendix D Instruction Execution Timing—This appendix contains execution times for the machine instructions for the series 960 computers.

Appendix E Assembler Directive Table—This appendix contains a list of the assembler directives for the series 960 computer Symbolic Assembler Language and their formats.

Appendix F Sample Programs—This appendix contains three sample programs that are written in and assembled by SAL.

Appendix G Instruction Index—This index provides references (by paragraph number) to the machine instructions in Section III. Mnemonics are ordered alphabetically.

Appendix H – An example of Job Control statements for creating an SWJC File to assemble a program under PAM/D.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

APPENDIXES

Appendix	Title	Page
A	SAL Character Set	A-1
B	General Tables	B-1
C	Instruction Tables	C-1
D	Instruction Execution Timing	D-1
E	Assembler Directive Table	E-1
F	Sample Programs	F-1
G	Instruction Index	G-1
H	SWJC**0087	H-1

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION I

GENERAL INFORMATION

1.1 GENERAL

This section contains general information about the Texas Instruments Inc. 960 series computer hardware and the Symbolic Assembly Language (SAL).

1.2 EQUIPMENT CAPABILITIES

The 960A and 960B computers are an advanced implementation of the TI960 computer. The unique internal design allows easy and efficient application to a wide variety of industrial control and data acquisition functions. Operational software is prepared in the language normally used to describe process control functions. These functions may be discrete or continuous operations. Some typical applications are tool operation, fabrication and automatic assembly material handling, environmental control, and data acquisition. The computers can be programmed to perform inspections and issue status reports. Critical data can be displayed instantly to an on-site operator for evaluation, or it can be relayed to a central computer facility where accurate management decisions can be made.

Real-time process control requires a computer with fast efficient context switching, manipulation of bits and bit-fields, and exchange of data between the computers and external devices. The TI960 series computers solve a great many automation problems with the following features:

- *Dual Mode Operation.* The dual-mode feature permits fast context switching. While running in one mode with one set of registers and execution counter, control can be switched to a second mode with identical capabilities. This not only provides a new programming environment, but frequently avoids the need to save the status of the old environment. Mode switching can be accomplished under interrupt or programmed instruction control.

- *Real-Time Clocks.* Many process control functions are time critical. The computer's optional 1-millisecond resolution interval timers allow timing of many tight, time critical functions. These optional timers are desirable where process functions must occur at specific instants or must occur after a specific time delay.

- *Versatile Direct Data Input/Output.* The Communications Register Unit (CRU) provides a simple, program-controlled interface with low-speed and medium-speed devices. Interface modules plug into ports which are connector slots in a CRU backpanel. CRU backpanels are in standard internal expansion or external expansion configurations. The CRU direct I/O system may be expanded to a total capacity of 4096 input signal lines and 4096 output signal lines.

  The CPU backpanel provides the four standard CRU ports. An optional internal expansion backpanel provides for an additional 12 CRU modules to be mounted within the CPU chassis. External CRU expansion racks with positions for 16 modules may be added to expand the total input/output capacity of the 960 CRU to 4096 input and output lines.

A variety of CRU modules are available:

- Data Modules
  - 16 input/16 output lines (TTL levels)
  - 16 input/16 output lines (EIA levels)
  - 32 Input lines
  - 32 Output lines
- Analog-to-Digital modules
- Digital-to-Analog modules
- Interrupt modules
  - 16 optionally coupled lines
  - 8 maskable interrupt lines (TTL)
- Relay Contractor module
- Serial Interfaces
  - EIA level–asynchronous
  - Current loop–asynchronous
  - EIA level–synchronous
- Universal solder and wirewrap boards for custom interface

1.3 SYMBOLIC ASSEMBLY LANGUAGE (SAL)

The SAL Assembler is a two-pass assembler with two versions, SALD and SALM, that run on the 960 series computers. A source program can be input to SALD or SALM from punched cards, paper tape, magnetic tape (on a standard width, 800 BPI reel or in a cassette), or disc and an object format version of the program can be produced. The object can be loaded in a 960 series computer and is the executable version of the program. The object can be loaded in a 960 series computer and is the executable version of the program. The object can also be linked with other object modules into a larger executable program.

The first pass reads the source program, builds and lists a complete symbol table, and generates the identification and linkage data records of the object program. If bulk storage (magnetic tape or disc for example) is available, SAL copies the input source file to bulk storage during the first pass and reads the input for the second pass from bulk storage. Bulk storage is either assumed or specifiable through an input option, depending on which monitor (i.e., executive system) is being used. Thus, only one reading of the original source input is necessary. During pass 2, the assembler uses the symbol table of pass 1 to complete the assembly of the source statements. The output of pass 2 is the text records and the end record of the object file and the assembly listing.

SAL generates relocatable code. It allows external references, address arithmetic, and operation code definition.

SECTION II

HARDWARE FEATURES

2.1 GENERAL

The 960 series computer block diagram (figure 2-1) shows the basic internal functional relationship of the hardware.

- The standard semiconductor (MOS) memory of the 960 series has a storage capacity ranging from 4096 (8192 for 960B) to 65,536 words. Space is provided within the 960 enclosure for 32,768 (65,536 for 960B) words of semiconductor memory.
- The Central Processing Unit (CPU) can address the memory, perform arithmetic and logic functions, and sequence and control the exchange of information between memory and other elements of the computer. The CPU features an arithmetic unit and a read-only memory (ROM) controller.
- The Communication Register Unit (CRU) controls the exchange of information between the computer and external devices.
- The Direct Memory Access Channel (DMAC) interfaces the computer with high-speed automatic computer peripherals, such as disc storage units, high-speed line printers, and magnetic-tape units. By using a separate controller for each device, concurrent operation of high-speed peripherals is achieved.
- The Front Panel (Control Console) allows the contents of memory or internal registers to be displayed or changed as necessary.

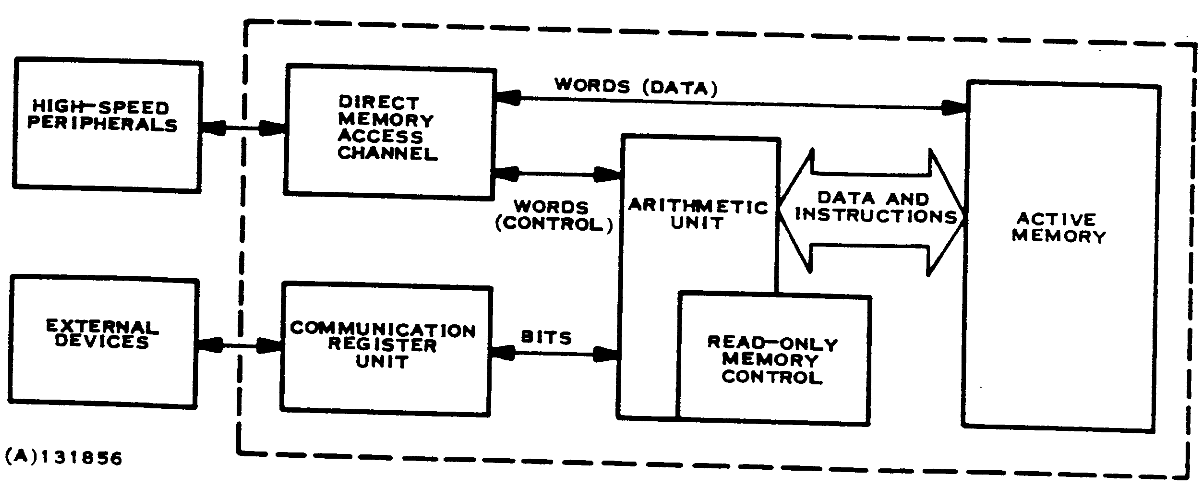


Figure 2-1. TI 960 Computer Block Diagram

2.1.1 SPECIFICATIONS.

- Organization

  Parallel Operation

  Single and double address logic

  Direct addressing of entire memory

  Indirect addressing with pre-indexing or post-indexing

  32-bit instruction word

  16-bit data word

  16 active hardware registers (16 bit) for arithmetic, index, or mask operations, and base addressing

  Supervisor and worker execution mode architecture

  Memory protect feature for variable amounts of memory

  Three levels of priority interrupts

- Performance

  4-MHz system clock rate

  750-nanosecond memory cycle time

  500-nanosecond memory access time

  Hardware multiply/divide option

  Execution times:

  - Load: 3.3 microseconds
  - Store: 3.6 microseconds
  - Add: 3.6 microseconds
  - Set CRU bit: 2.8 microseconds
  - Load register in CRU: 4.2 – 8.2 microseconds (1-16 bits)

- Memory

  Semiconductor memory using 4096 X 1-bit dynamic MOS arrays; (1024 X 1 bit dynamic for 960A).

  Internal storage for up to 65,536 (32,768 for 960A) words of MOS memory in standard increments of 8196 (4096 for 960A)
  Power failure protection.

- Input/Output System

  Direct Memory Access Channel (expandable to 8) with 16-bit parallel transfer, 1 million words per second burst rate, and parity checking interface.

  Communications Register Unit with up to 4096 I/O ports and 4 million bits per second burst rate.

- Instruction Set–78 instructions
  - 9-bit and field manipulating instructions
  - 36 register-memory instructions
  - 5 powerful memory-memory instructions
  - 28 flexible program control instructions
- Physical Characteristics
  - Dimension (rack-mount configuration)
    - Height–12.25 inches
    - Width–19 inches
    - Depth–24 inches
    - Weight–75 pounds
  - Power Requirements: 115 V ±10%, 47–63 Hz
  - Power Consumption: 420 Watts, average
- Operating conditions:
  - Temperature (@ sea level)
    - 0°C to 50°C
    - 32°F to 122°F
  - Humidity 10%–95%
  - Altitude 0–10,000 feet

2.2 SYSTEM ORGANIZATION

The system organization is discussed in the following paragraphs.

2.2.1 **DATA FORMAT.** The basic element of data is a 16-bit word. Bit positions are numbered from 0 through 15.

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

A machine instruction occupies two words of memory in which the bit positions are numbered 0 through 31.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
WORD 1															

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
WORD 2															

A fixed-point integer occupies one word in memory, represented in binary form with the sign bit in position 0. A positive sign is indicated by a zero. Negative integers are represented in a two's-complement form. Thus, the range of integers representable in a word of memory is from -2^{15} through $2^{15} - 1$ or from −32,768 through 32,767.

2.2.2 **MEMORY**. The basic unit of memory is a 16-bit word plus a parity bit (960A) or error correction bits (960B). The CPU can directly address all 65,536 words of memory. An optional battery assembly maintains the contents of the semiconductor memory in the event main power is interrupted.

One 750-nanosecond memory cycle is used every 32 microseconds (63 microseconds for 960A) to refresh semiconductor memory. This performance reduction should be accounted for in time-critical instruction sequences.

2.2.3 **DEDICATED MEMORY LOCATIONS**. Certain memory locations have been reserved and are assigned to interrupt and I/O status information. These locations and their corresponding functions are listed below:

Memory Address$_{16}$	**Function**
90-91	Internal Interrupt
92-93	Direct Memory Access Channel Interrupt
94-95	Communication Register Unit Interrupt
96	Direct Memory Access Channel Status
98-99	Status, Device Controller 0
9A-9B	Status, Device Controller 1
9C-9D	Status, Device Controller 2
9E-9F	Status, Device Controller 3
A0-A1	Status, Device Controller 4
A2-A3	Status, Device Controller 5
A4-A5	Status, Device Controller 6
A6-A7	Status, Device Controller 7

2.2.4 **PROTECTED MEMORY**. Locations 0-7F are protected memory locations reserved for bootstrap programs. The memory protect area can be expanded as an option (see table 2-1). Writing in protected locations can be accomplished through the use of the MPO (Memory Protect Override) switch on the operator's console (front panel). When MPO is on, protected memory can be written in through console switches or by software.

2.2.5 **ACTIVE REGISTER FILE**. A file of 16 active registers are implemented in the computer. These hardware registers do not actually reside in memory, but a special feature allows them to be addressed as memory locations 80 through 8F (the corresponding locations in actual memory are not used). Normally, eight of these registers are available to Supervisor Mode programs and the other eight are available to Worker Mode programs. All 16 registers are available in either mode using alternate mode addressing. The 16 registers and the special function assigned to each are listed in table 2-2. The register file provides 16 words of high-speed scratchpad memory. A significant increase in performance results when an instruction operand resides in the register file and particularly when an instruction resides in the register file.

Table 2-1. Optional Memory Protect Limits

Memory Address (hexadecimal) excluding memory locations 80 - A7
0-FF
0-1FF
0-3FF
0-7FF
0-FFF
0-1FFF
0-3FFF
0-7FFF
0-FFFF

2.2.6 **PROCESSING MODES.** Programs are executed in one of two modes. Instructions, in either mode, have independent access to a general file of eight registers. Programs of either mode can provide arithmetic, logical, and control functions typical of general purpose computers. Programs designed for bit or bit-field processing can also be executed in either mode. These instructions can address any bit in memory or any bit or bit-field in the Communications Register Unit.

In the Supervisor Mode, the CPU is executing instructions via the Program Counter (PC) and utilizing the Supervisor Mode register file for register referencing instructions. In the Worker Mode, instructions are executed via the Event Counter (EC) and utilize the Worker Mode register file for register referencing instructions. Any instruction can be executed in either mode. Mode changing is under program and interrupt control.

2.2.7 **PROGRAM STATUS BLOCK.** The control conditions for program execution in the CPU are defined by the PC (Program Counter) or EC (Event Counter) and the status register. Instruction addressing is controlled by the PC when the computer is in the Supervisor Mode and by the EC when in the Worker Mode. At the completion of each instruction, depending upon

Table 2-2. Register File

Register Number	Supervisor $Address_{16}$	Worker $Address_{16}$	Functional use by Arithmetic Instructions	Functional Use by Bit and Field Manipulating Instructions
0	80	88	General Arithmetic Reg. Index Register 0	General Register
1	81	89	General Arithmetic Reg. Index Register 1	General Register
2	82	8A	General Arithmetic Reg. Index Register 2	General Register
3	83	8B	General Arithmetic Reg. Index Register 3	General Register
4	84	8C	General Arithmetic Reg. Index Register 4	Data Base Address
5	85	8D	General Arithmetic Reg. Index Register 5	Procedure Base Address
6	86	8E	General Arithmetic Reg. Index Register 6	Base of Software Flag Areas
7	87	8F	General Arithmetic Reg. Index Register 7	Base of CRU Address

the mode currently executing, the PC or EC is incremented by two. Program control instructions can modify the PC or EC in other ways. The PC or EC always contains the address of the next instruction to be executed. The PC or EC are implemented by live registers that can be addressed by special instructions.

2.2.8 **STATUS REGISTER.** The status register is used to hold the condition of the computer and instruction results at any time and to enable or disable interrupts. A functional chart of the status bits follows:

0	1	2	3	4	5	6	7	8	9	10 12	13	14 15
MI	OI	MP	PF	UC	MV	IM	II	DI	CI	CB	CO	RESERVED

MI – Mode Indicator
If MI is 0, execution is in the Supervisor Mode; if MI is 1, execution is in the Worker Mode.

OI Overflow Indicator
OI is set to 1 if the results of an arithmetic operation (a signed two's complement integer) cannot be contained in 16 bits (32 bits for double-precision instruction) without truncation. In left shift instructions, it is set if the sign bit (0) is changed at any time during the shift. If no overflow occurs in an instruction that can produce overflow, then OI is set to 0.

MP – Memory Parity Indicator
If MP is 1, a parity error has occurred on a 960A or a multiple bit error has occurred on a 960B. (Only single bit errors can be eliminated by error correction.)

PF - Power Failure Indicator
If PF is 1, a power failure is indicated.

UC - Undefined Code
If UC is 1, an undefined operation code has been detected.

MV Memory Violation
If MV is 1, an attempt has been made to alter protected memory.

IM – Index Mode
If IM is 0, pre-indexing is performed; if it is a 1, post-indexing is done.

II - Internal Interrupt Mask
If II is 0, the internal interrupt is enabled.

DI DMAC Interrupt Mask
If DI is 0, the DMAC interrupt is enabled.

CI – CRU Interrupt Mask
If CI is 0, the CRU interrupt is enabled.

CB – Comparison Bits
Bits 10-12 serve as comparison indicators. They are set as follows for the compare register algebraic and compare register logical instructions (CR, CRA, CRL, and CRLA):

	ST10	ST11	ST12
(R)>EO	1	0	0
(R) = EO	0	1	0
(R) <EO	0	0	1

where

R = effective register contents
EO = effective operand (or effective address for CRA and CRLA)

In addition, these status bits indicate the results of an arithmetic comparison of the effective operand of certain instructions with zero. The instructions affected are listed in table 2-3.

For the divide instructions the remainder is tested. For the multiply and all double length instructions the most significant half of the result is tested. For all the other instructions named, the final result placed in a register or stored in memory is tested. The comparison status bits are set as follows:

	ST10	ST11	ST12
Effective operand > 0	1	0	0
Effective operand = 0	0	1	0
Effective operand < 0	0	0	1

CO – Carry Out Indicator
Indicates a carry out of bit position 0 (sign bit) for A, AA, S, SA, SAT and AMI instructions. The bit is set to one when a carry occurs and is reset otherwise.

2.2.9 PRIORITY INTERRUPT SYSTEM. The computers feature a priority interrupt system that provides added program control of I/O operations, provides immediate response to abnormal conditions, and allows immediate recognition of special external conditions.

The interrupt system gives the programmer flexible control of external devices. Three interrupt priority levels are implemented as follows: first priority, internal interrupts; second priority, CRU interrupts; third priority, DMAC interrupts.

Two consecutive, fixed memory locations are associated with each interrupt. When an interrupt is taken, the instruction in the respective interrupt location is executed. Control is then returned to the next instruction that would normally have been executed prior to the interrupt. If the instruction in the interrupt location is one that modifies the PC (or EC depending upon the mode of execution) the branch is taken. The interrupt location normally contains a Store Status Block, Transfer and Branch In Supervisor Mode (SXBS) instruction that stores either the contents of the PC or the EC and the Status Register and then causes a branch to an interrupt program sequence. After the interrupt has been serviced, a Load Status Block instruction restores the computer to the state it was in immediately prior to the interrupt (see figure 2-2). Saving and restoring general registers used in interrupt subroutines is the responsibility of the programmer.

Table 2-3. Instructions Affecting Status Register Comparison Bits

Instruction	Result Tested
L,LA	Final value loaded into the Register
ST	Final Value Stored into Memory
A,AA	Sum placed in the Register
S,SA	Difference placed in the Register
LOT,LOTA	Tally placed in the Register
N,NA	Result of 'AND' placed in the Register
OR,ORA	Result of 'OR' placed in the Register
XOR,XORA	Result of 'XOR' placed in the Register
SAT	Result after adding Tally placed in the Register
MLA,MLAX	Result placed in Memory after the Shift
MRA,MRAX	Result placed in Memory after the Shift
MRR,MRRX	Result placed in Memory after the Shift
ARB	Result placed in Register after the Addition
AMI	Result placed in Memory after the Addition
MOV	Value moved
STPS	Value stored in Memory from the data switches
STCR	Value stored in Memory from the CRU
M,MA	Most significant half of product placed in Memory
D,DA	Remainder
DAD,DS	Most significant half of result placed in Memory
DLA,DLAX	Most significant half of result placed in Memory
DRA,DRAX	Most significant half of result placed in Memory
DRL,DRLX	Most significant half of result placed in Memory
DRR,DRRX	Most significant half of result placed in Memory
CR,CRA CRL,CRLA	See section 2.2-8

2.2.9.1 **Internal Interrupt.** An internal interrupt provides immediate attention to any of the following conditions:

- Memory Parity Error. This interrupt condition protects the user against a possible data transmission error or misread instruction. When this interrupt occurs, bit 2 (MP) of the Status Register is set to 1.
- Change of Power System Status. This interrupt signal tells the computer that power has just been restored or that power loss is imminent. The power loss condition sets bit 3 (PF) of the status register to 1. The computer operates for 1.0 millisecond after power loss is sensed. When power is restored, the status register is set to $C0_{16}$.

- Undefined Code Execution. An attempted execution of an undefined operation code results in an internal interrupt and bit four (UC) of the status register being set to 1.

- Memory Violation. An attempt to alter protected memory with memory protect on (enabled) results in an internal interrupt and sets bit 5 (MV) of the Status register to 1.

When an internal interrupt occurs, the three interrupt mask bits of the status register are automatically set to 1, assuring that complete corrective response can be made for the condition by disabling any subsequent internal interrupts, CRU interrupts, or DMAC interrupts. The internal interrupt causes the CPU to execute the instruction in memory location 90_{16} where a Store Status Block, Transfer and Branch in Supervisor Mode (SXBS) instruction normally resides. The status register saved by this instruction shows the status of the interrupt mask bits just prior to the interrupt, that is, before the interrupt mask bits are set to 1 by the interrupt. The stored contents of the Status Register also show the condition(s) causing the interrupt. This allows determination of the cause(s) of the interrupt by examining bits 2, 3, 4, or 5 of the stored contents of the Status Register. The contents of the PC or the EC are also captured. The internal interrupt is re-enabled by setting bit 7 (II) of the status register to 0.

2.2.9.2 **Communications Register Unit Interrupt.** The Communications Register Unit interrupt occurs when a CRU interrupt line goes to 1. Most CRU modules can activate the CRU interrupt line. The particular module requesting program attention is identified by scanning the interrupt bit associated with each module. When the interrupt occurs, bits 8 and 9 (DI and CI) of the status register are automatically set to ones, disabling other CRU interrupts and the DMAC interrupt. The CRU interrupt causes an execution of the instruction in memory location 94_{16}.

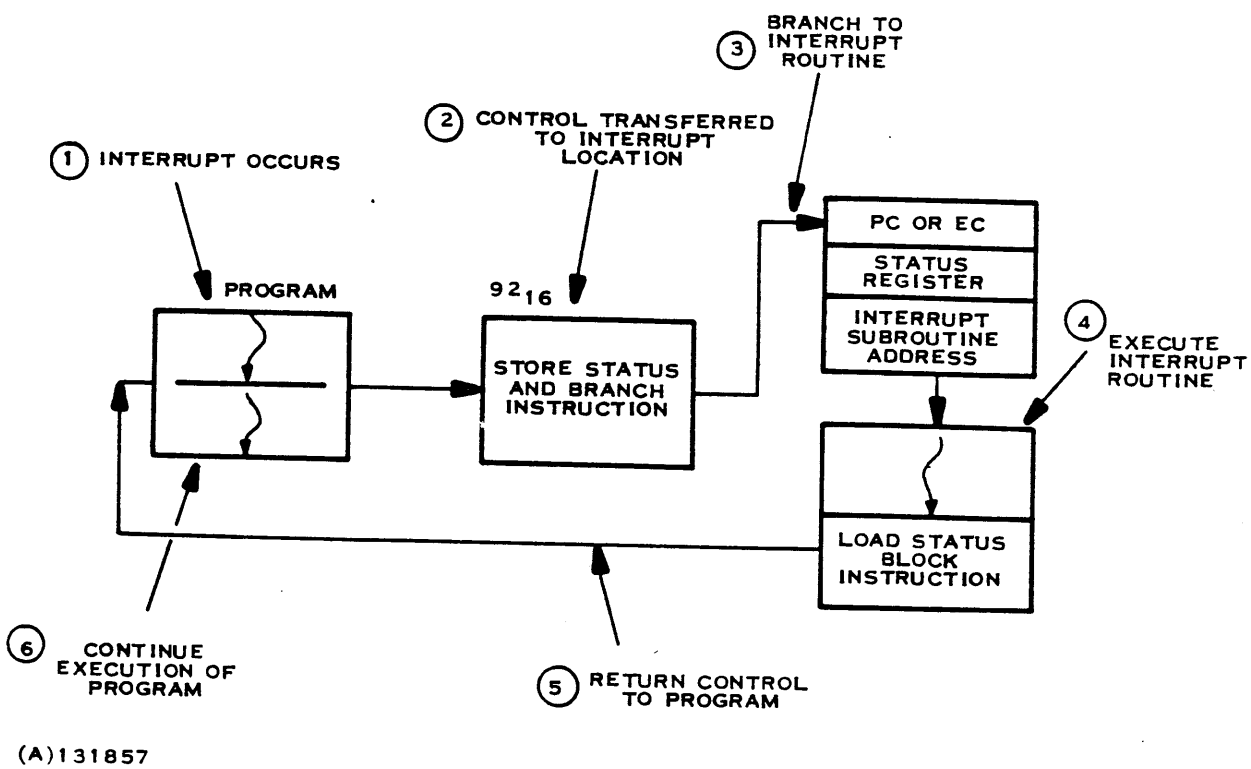


Figure 2-2. Typical Linkage to Interrupt Routine

2.2.9.3 Direct Memory Access Channel Interrupt. The Direct Memory Access Channel (DMAC) interrupt is taken when any device connected to a channel port has its interrupt enabled and changes its status storage memory location. The DMAC interrupt causes an execution of the instruction in memory location to 92_{16} where a Store Status Block, Transfer and Branch in Supervisor Mode (SXBS) instruction normally is located. To identify the particular device causing the interrupt, a special memory location reserved for DMAC status can be examined. When a DMAC interrupt is taken, bit 8 (DI) of the Status Register is set to logic 1, disabling DMAC interrupts.

When interrupt lines from device controllers to the DMAC are turned on, they set bits in the DMAC interrupt Status Register to indicate the source of the interrupt. One or more bits set in the DMAC interrupt Status Register causes the DMAC interrupt line from the DMAC to the CPU to turn on. If the CPU has the DMAC interrupt masked, further interrupts from other device controllers only cause more bits to be set in the DMAC interrupt Status Register. The device controllers keep their interrupt lines logic 1 until they receive the interrupt recognized (reset) signal.

As the CPU traps to the DMAC interrupt trap location (92_{16}), it turns the interrupt recognized signal to the DMAC on. If interrupts are enabled, the DMAC responds with status storage at 96_{16} and interrupt recognized to each controller that has set an interrupt bit in the DMAC Status Register. Figure 2-2 shows a typical interrupt routine linkage.

DMAC and device controllers status words are then stored in dedicated memory. After status words are stored the DMAC trap is executed and control is transferred to the proper interrupt service routine.

The DMAC interrupt is masked upon trapping to the trap location so that the interrupt routine is protected. Masking of individual controller interrupts is controlled by a bit in the device initialization list.

2.2.10 COMMUNICATIONS REGISTER UNIT (CRU). The Communications Register Unit provides the computer user a wide variety of Input/Output (I/O) operations for testing, monitoring, and controlling discrete events as well as processing operator messages and management reports.

To meet these varied requirements the computer is capable of I/O functions ranging from a single bit to a 16-bit word. This I/O field width flexibility is a unique feature of the CRU.

2.2.10.1 CRU Input/Output Devices. The CRU is capable of I/O with the following standard medium or slow-speed devices. A typical CRU configuration is shown in figure 2-3.

Card Reader–up to 400 cards per minute

TI Model 733 ASR Electronic Data Terminal (30 cps), TI Model 743 KSR (30 cps) or Teletype Model ASR-33 (10 cps)

TI Model 912 Video Display Terminal

Data Set–up to 9600 baud

Paper Tape Reader–up to 300 frames per second

Paper Tape Punch–up to 75 frames per second

Line Printer–up to 330 characters per second

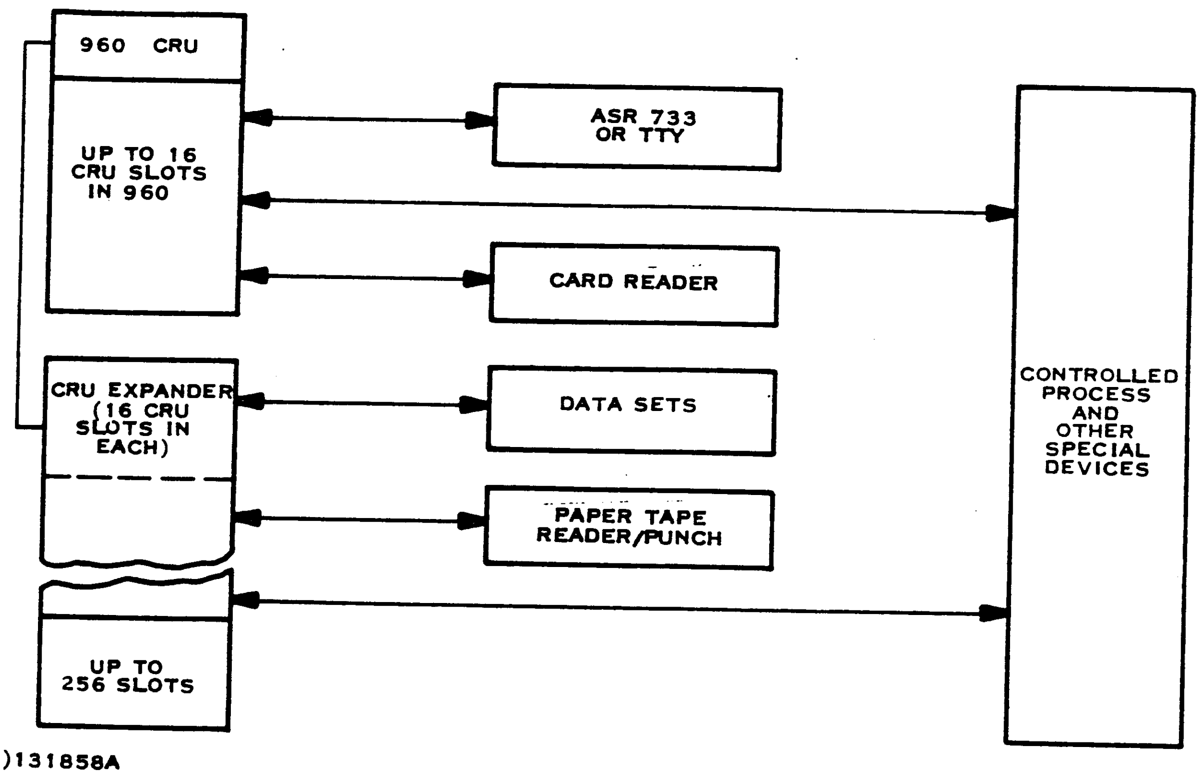


Figure 2-3. Typical CRU Configuration

In addition, the CRU presents an interface which can accommodate practically any specialized application-oriented device.

All CRU I/O operations are based on the set/reset state of individual I/O lines. Special bit oriented machine instructions are implemented in the computer instruction set to operate the CRU interfaces.

Mnemonic	Operation
LDCR	Load CRU Register (1-16 bits) from memory
STCR	Store CRU Register (1-16 bits) in memory
SETB	Set CRU Output Bit
TSBX	Test Input Bit and Set Output Bit or Switch Mode
XBNE	Switch Mode if Bit not Equal
BBNE	Branch if Bit not Equal

The CRU can be expanded to a total of 4096 I/O lines. Figure 2-4 shows an example of this. Each I/O line can be addressed independently or up to 16 lines can be addressed at the same time as a register. The external function of each group of 16 lines is determined by the type of

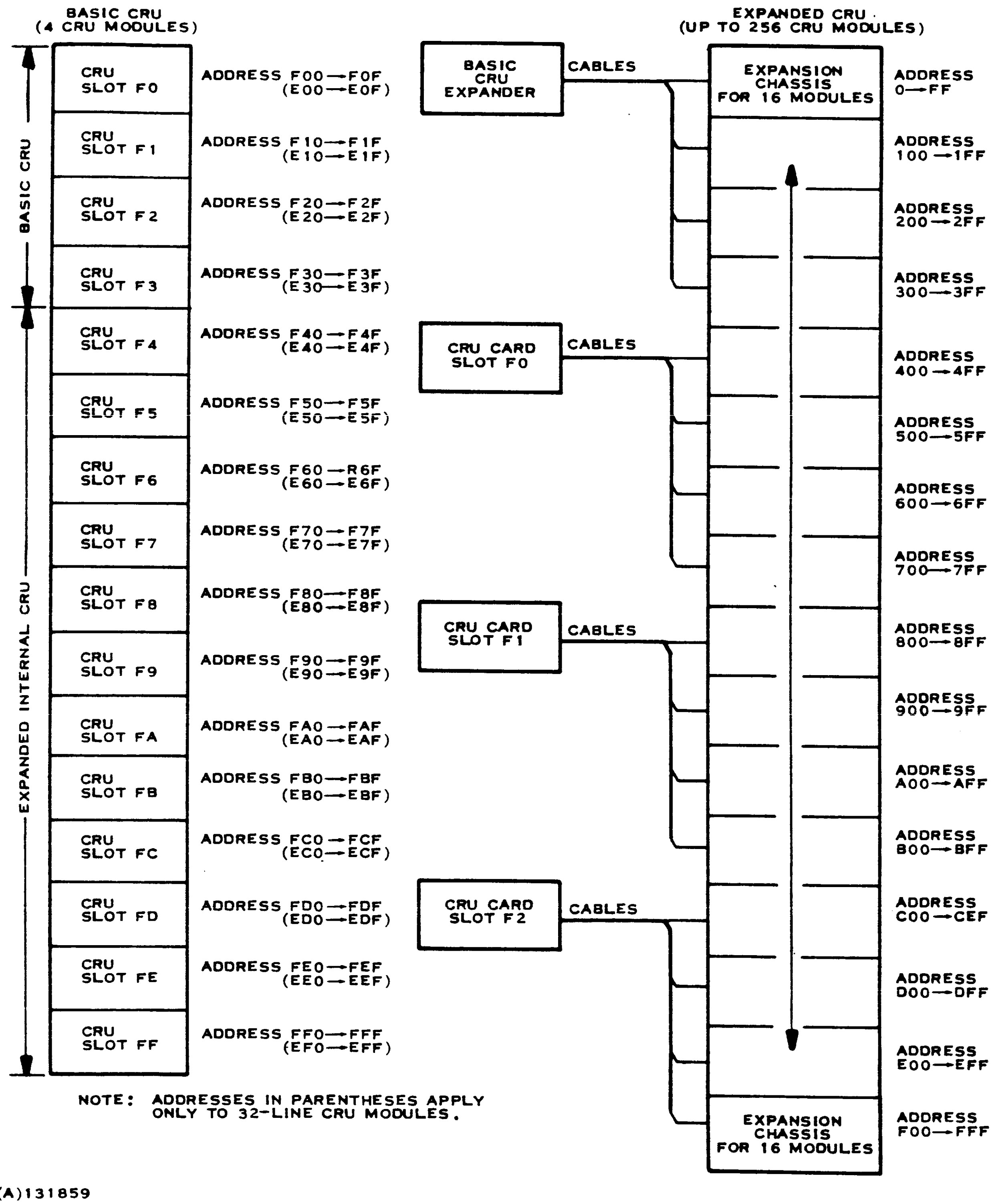


Figure 2-4. CRU Expansion Addressing

interface circuit board used. A group of 16 lines can act as a 16-bit data bus, an interval timer, external interrupts, or as address lines and input data for a multiplexer-A/D converter circuit board. Many other functions such as stepping motor control, teletypewriter interface, modem interface, D/A conversion, and pulse generation can be performed easily with the CRU Digital I/O interface. Detailed programming information for standard interface modules is found in *Model 960 Computer Communications Register Unit Manual*, manual number 966313-9701.

2.2.10.2 **CRU Description.** The CRU port of the basic 960 series computers consists of a CRU Interface/Expander card and four card connectors for CRU modules. A connection from a CRU module to an external device is made by a top-edge connector on the CRU module. The CRU Interface/Expander card provides CRU module select signals for the four basic modules plus 12 internal expansion modules. Select signals for both 16 and 32 line CRU modules are generated.

CRU modules installed in the four basic locations are addressed at F00, F10, F20, and F30 for 16-line modules. The addresses are given in hexadecimal notation. When 32-line modules are used, the starting addresses of the two 16-line groups on a card are separated by 100_{16}. Thus, the addresses for 32-line modules installed in the four basic locations are E00-F00, E10-F10, E20-F20, and E30-F30, respectively. External racks connected to the Interface/Expander card are addressed as 0, 100, 200, and 300. Expander cards installed in the first three module connectors address racks 400 through 700, 800 through B00, and C00 through F00, respectively.

2.2.11 **DIRECT MEMORY ACCESS CHANNEL (DMAC).** The computer provides high-speed Input/Output through the Direct Memory Access Channel. A single DMAC port is included in the basic DMAC for I/O through one peripheral controller. A Direct Memory Access Port Expander (DMAPE) can be added to the DMAC allowing up to eight high-speed I/O peripheral devices.

The DMAC is activated under program control and, once started, performs the designated I/O task independently of the program. An interrupt can be generated when the I/O task is complete and the status of the task is automatically stored in memory.

DMAC is capable of I/O through the following standard optional devices. (See figure 2-5).

1. Magnetic-Tape Transports (3 per controller)
2. Magnetic Disc (up to 4 DS31 or DS44 Moving-Head Discs)
3. Line Printer

All DMAC I/O is accomplished through the single command Activate Direct Memory Access Channel (ADAC) which is described in Section IV.

2.2.11.1 **DMAC Data Handling.** When transferring data between memory and a single high-speed device the maximum transfer rate is 10^6 words per second. When memory access is requested by more than one device (however, none requiring successive memory cycles), four system clock cycles must elapse between servicing of the first device and the granting of access to the second. Maximum data transfer rate under this condition is 8×10^5 words per second. The DMAC services access requests from multiple devices on a priority basis. Priority for data transfers from the devices is selectable by the user. However, an interrupt from any device has priority over access requests for data transfer. The DMAC has priority over the CPU when both require memory access at the same time.

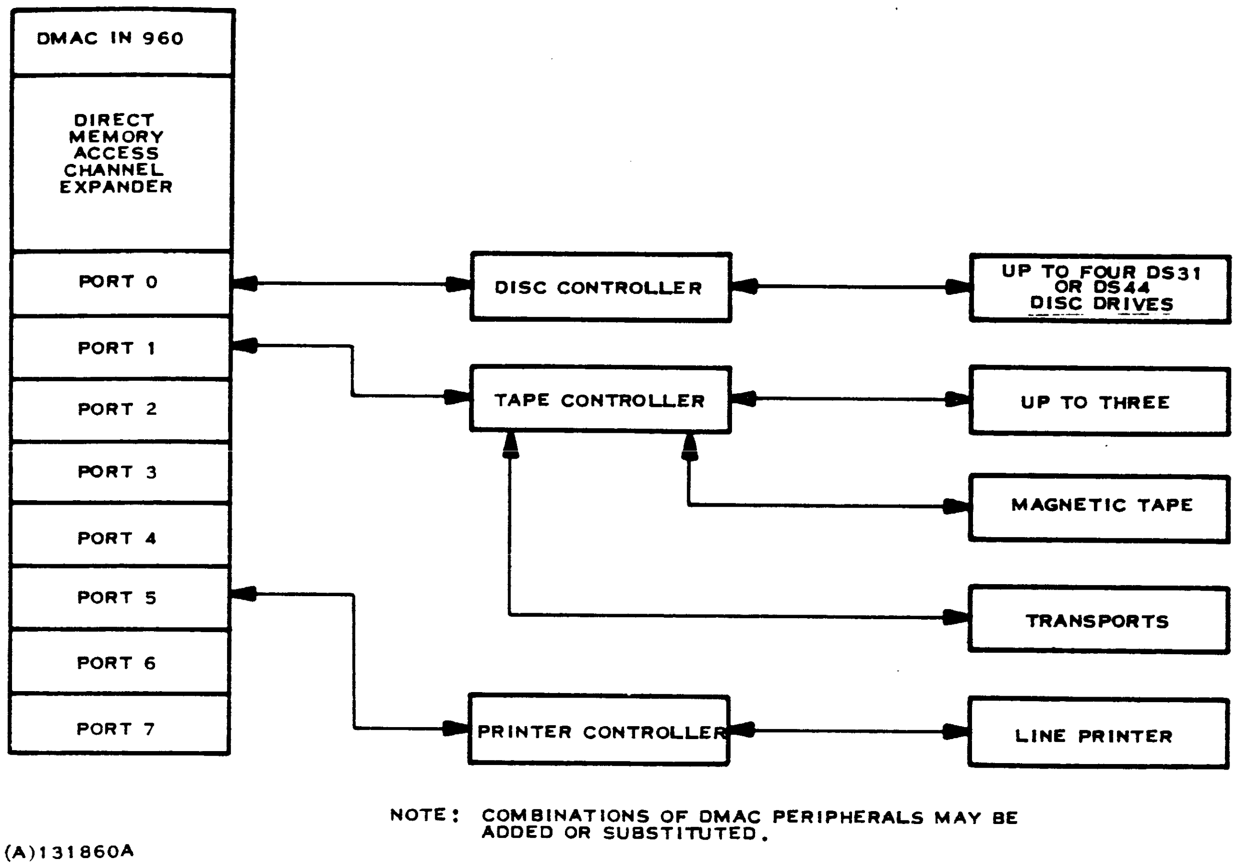


Figure 2-5. Typical DMAC I/O Configuration

2.2.12 STANDARD FRONT PANEL. Figure 2-6 shows the control and display panel for the 960 series of computers. All controls and indicators and their respective functions are described in the following paragraphs and table 2-4.

2.2.12.1 Panel LOCK/UNLOCK. The Panel LOCK/UNLOCK switch must be in the UNLOCK position for complete front panel use.

In the LOCK position, all front panel controls are disabled except the DATA switches.

In the UNLOCK position, all front panel switches are operational.

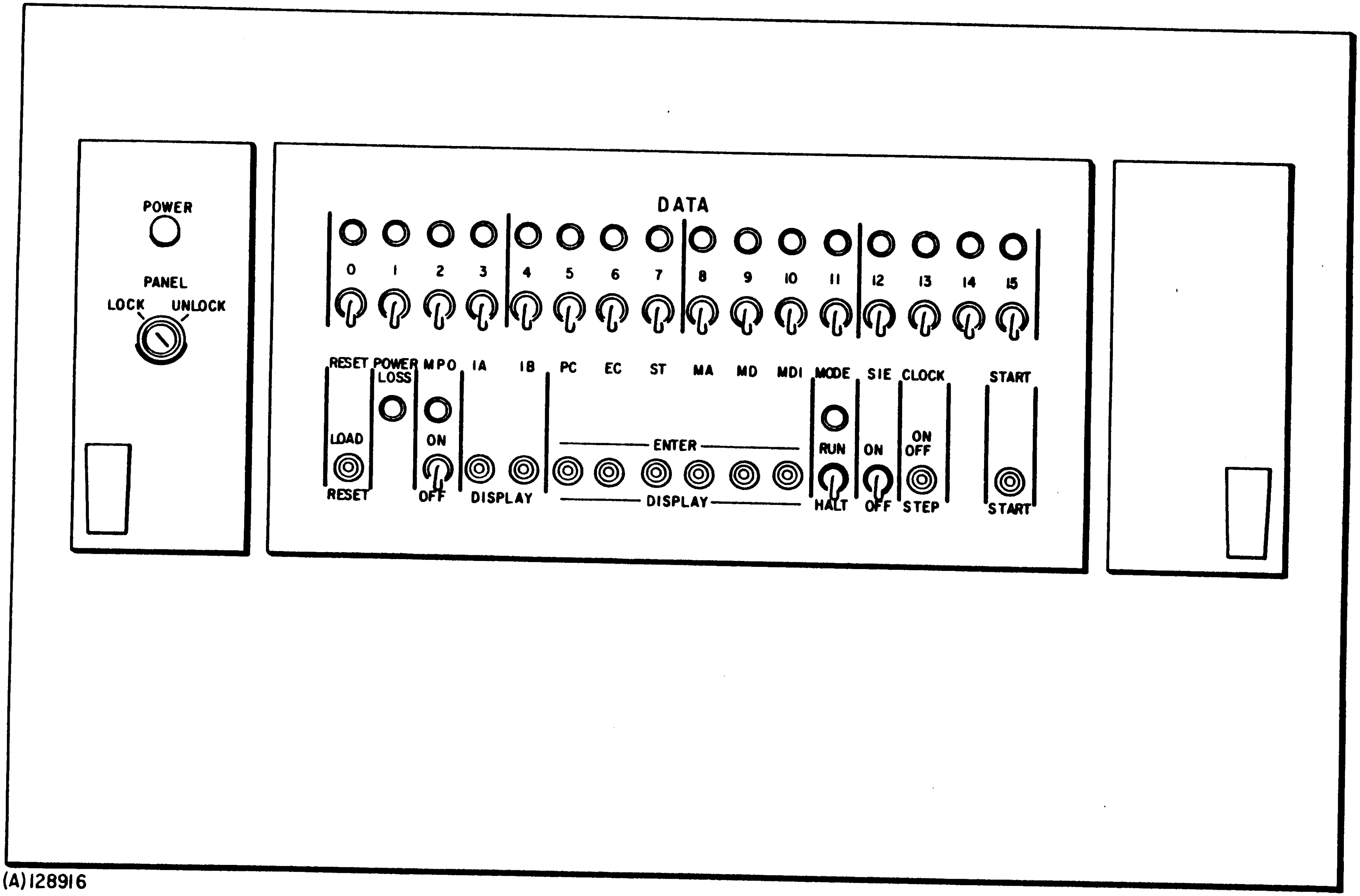


Figure 2-6. 960 Series Computer Standard Front Panel

Table 2-4. Front Panel Indicators

Indicator Name	State	Meaning
DATA (16 lights at top)	On	The value of the associated bit is a logic one.
	Off	The value of the associated bit is a logic zero. In the RUN or SIE mode, the PC or EC address is continually displayed in the DATA lights.
POWER	On	System ac power is on.
	Off	Power off.
POWER LOSS	On	Memory power lost and ac power subsequently restored.
	Off	The RESET switch has been moved to the down position since last memory power loss.
MPO	On	The memory protect override is on and data may be stored in protected memory locations.
	Off	Data cannot be stored in the protected memory locations.
MODE	On	RUN and START have been selected, and the computer is executing instructions.
	Off	The computer is not in the RUN mode.

2.2.12.2 **Data Switches.** The sixteen data switches are two-position toggle switches that are used for data entry. Each switch is placed in the up position for a logic one, or down for a logic zero.

The switches are arranged in groups of four by lines on the front panel. Each group of four bits corresponds to a hexadecimal digit. For example, when a group of four bits is 1010, this is interpreted as A_{16}. Also, when all 16 switches are down, the display is interpreted as 0000_{16}. To set a hexadecimal value with the DATA switches, the binary equivalent of each hexadecimal digit determines the positions of the switches. Refer to table 2-5 for the binary equivalents of hexadecimal numbers.

Each switch in a 4-place switch group carries the binary value shown in figure 2-7. Refer to figure 2-8 for an example of setting a hexadecimal value in memory with the DATA switches.

Table 2-5. Hexadecimal-to-Binary Equivalents

Hex	Binary	Hex	Binary	Hex	Binary	Hex	Binary
0	0000	4	0100	8	1000	C	1100
1	0001	5	0101	9	1001	D	1101
2	0010	6	0110	A	1010	E	1110
3	0011	7	0111	B	1011	F	1111

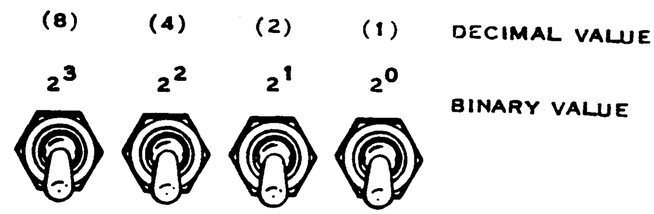


Figure 2-7. Switch Binary Values

2.2.12.3 **IA and IB Switches.** The IA and IB switches are used to display instruction register data on the DATA indicators. The switches have a normal center position, a momentary up position, and a momentary down position. When pressed to the DISPLAY (down) position, the content of that instruction register is displayed. The up position of the switch is nonfunctional. The display function is only active while the computer is in the HALT mode.

2.2.12.4 **PC, EC, ST, MA, MD Display and Enter Switches.** These switches control the following data DISPLAY and ENTER functions, respectively: Program Counter (PC); Event Counter (EC); Status Register (ST); Memory Address register (MA); and Memory Data (MD). Each switch has a normal center position, a momentary up position, and a momentary down position. When any of these switches is pressed to the ENTER (up) position, the current setting of the 16 DATA switches is loaded into the associated register or memory location. The entered value is displayed by the DATA indicators. Memory Data (MD) is entered into the address specified by the contents of the Memory Address (MA) regsiter. If the switch is set to the DISPLAY (down) position, the contents of the associated register or memory location is displayed. All switches in this group function only when the computer is in the HALT mode.

2.2.12.5 **MDI Switch.** The Memory Data and Increment Address (MDI) switch functions like the MD switch described in the preceding paragraph for entering or displaying memory data except that each time the MDI switch is actuated to either ENTER or DISPLAY, the Memory Address Register is incremented to the next consecutive address. New data is entered in the register before incrementing takes place. The switch is used for loading or displaying consecutive memory locations.

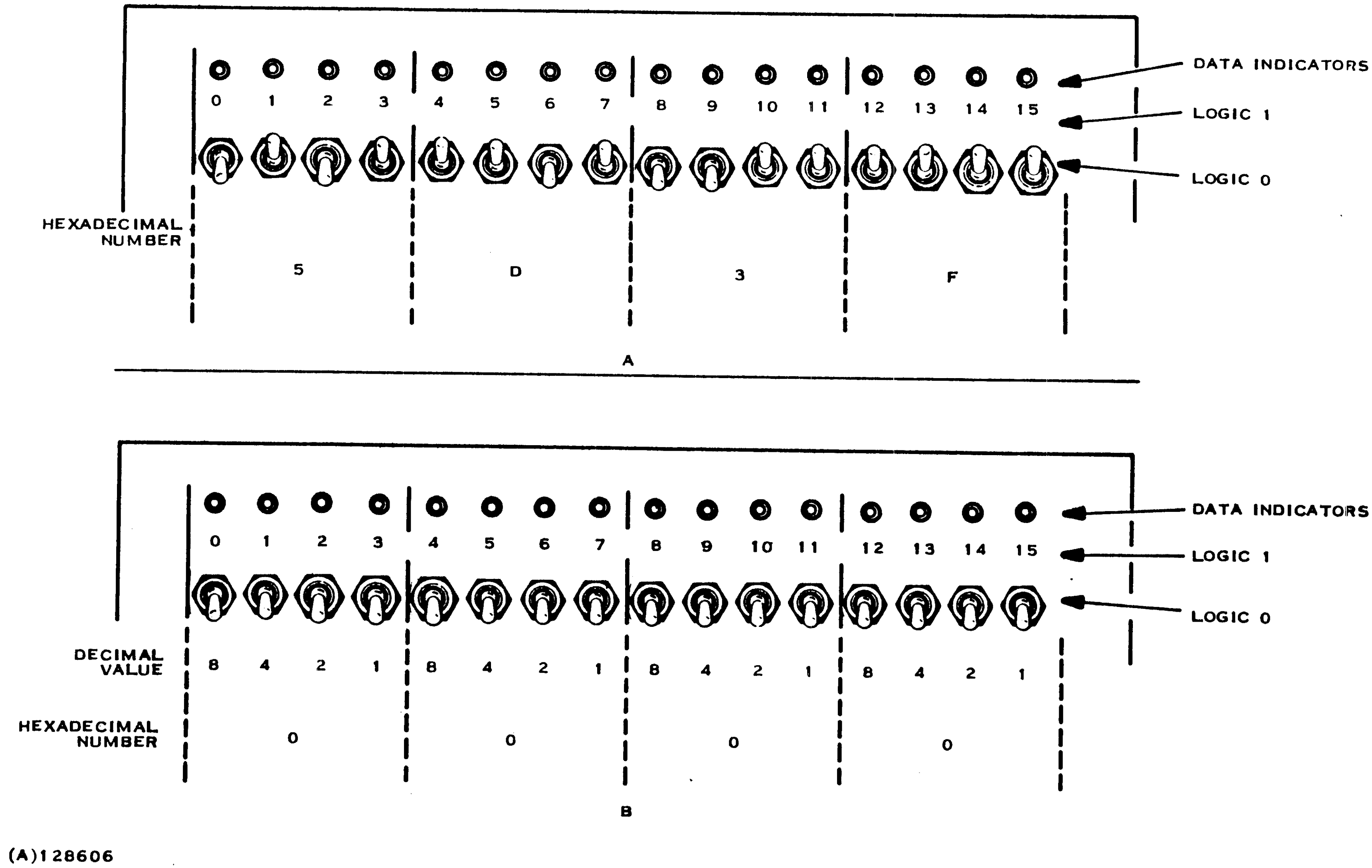


Figure 2-8. Data Switches

2.2.12.6 RESET Switch. The system RESET switch is used to reset all system registers except the instruction registers (IA and IB). This switch has a normal center position and a momentary down position. When the switch is pushed to the RESET (down) position, the program counter, status register, memory address register and display register are cleared. This function can occur only if the computer is in the HALT mode. The LOAD position of this switch is used to load the contents of the optional ROM bootstrap loader into the computer memory. When the switch is pushed to the LOAD (up) position, the 256-word bootstrap program is loaded into memory starting at address zero plus the contents of the MA register.

2.2.12.7 MODE Switch. The MODE control switch is a two-position toggle switch. To start program execution, the MODE switch is placed in the RUN (up) position and the START switch is actuated (placed in the momentary down position). The RUN mode is entered and the RUN indicator is lighted when the START switch is actuated. When the MODE switch is placed in the HALT position, the system halts when the instruction in progress is completed.

2.2.12.8 SIE Switch. The SIE (Single-Instruction Execution) is a two-position toggle switch that allows single instruction execution. To execute a single instruction, the MODE switch is placed in the HALT position, the SIE switch is set to ON, and the START switch is actuated. An instruction is executed each time the START switch is activated.

2.2.12.9 CLOCK Switch. The CLOCK switch is a three-position toggle switch with a normal center position (OFF), an up position (ON) and momentary down position (STEP). For normal system operation, this switch is in the ON (up) position and the system clock is free running. When the CLOCK switch is placed in the OFF (center) position, the arithmetic-unit clock is stopped. Each time the CLOCK switch is pushed to the momentary STEP (down) position, a single system-clock pulse is generated for the arithmetic unit. The operation of this switch is normally bypassed by a jumper wire on the console printed circuit board. Normal shipments are made with the switch disabled.

The clock switch is also used to initialize the memory correction feature on the 960B computer. Each time the clock switch is pushed to the momentary STEP position (down), the error indicator latches are reset.

2.2.12.10 START Switch. The START switch is a two-position toggle switch with a normal center position and a momentary down position. If the system is in the HALT mode and this switch is pushed to the START (down) position, one of the following will happen:

- If the SIE switch is in the ON position, single instructions are executed.
- If the SIE switch is in the OFF position, the system will remain in the HALT mode.

When the mode control is in the RUN position and the START switch is depressed, the RUN mode is entered.

2.2.13 Memory Initialization. When power is first turned on, and the battery is either disconnected or turned off, memory needs to be initialized. To do this, enter the following values into memory:

Address (hexadecimal)	Value (hexadecimal)
0080	4881
0081	0090
0082	4C81
0083	0001
0084	7082
0085	0080

Then set the PC to 80_{16} ; set the status register to $01C0_{16}$; turn the Memory Protect Override (MPO) switch on; and select RUN and START. This should put the computer in the RUN mode and, after execution (less than a second) each memory location should contain the address of that location.

2.2.14 **Optional ROM Loader.** The optional ROM (hardware primitive loader option) loader for the Model 960 computers is a small printed circuit board. It reduces to a minimum the manual operation require to "cold start" or "warm start" the computer system. Five different programs totaling 256 words can be stored in memory beginning with any memory location. The programs are:

- Card media primitive loader
- Paper tape primitive loader–High Speed Paper Tape Reader
- Paper tape/cassette primitive loader–Full Duplex ASR-33/733 ASR
- Warm start initialization for PAM/D (ALPHA)–Fixed Head Disc
- Warm start initialization for PAM/D (ALPHAM)–Moving Head Disc

The first three programs would otherwise have to be loaded by hand. In their respective media, they load a relocating bootstrap loader into protected memory. The last two programs are otherwise read into memory by the resident bootstrap loader. Their purpose is to bring the disc resident monitor, PAM/D, into memory. Since there are two different discs available for use with the computer, there must be two such programs. The steps required to load and execute any of these programs is:

1. Set MA to the desired load address for all five programs and ST to $01C0_{16}$.
2. Set MPO to ON.
3. Move RESET to LOAD
4. Set PC and memory location 0085_{16} to the starting address for the desired program as shown in table 2-6. The location 85_{16} does not have to be set for ALPHA or ALPHAM.
5. (Primitives only) set locations 0086_{16} to 0 and 0087_{16} to the CRU address of the input device.
6. Set ST to $01C0_{16}$, turn MODE to RUN, and push START.
7. Turn MPO to OFF.

If it is desired to load ALPHAM into protected memory, set MA to 0 and turn MPO on at step 1. In order to load ALPHA into protected memory, set MA to $FFB0_{16}$ and turn MPO on at step 1.

Table 2-6. Starting Address—Contents of ROM Loader

Program	Starting Address
ALPHAM	(contents of MA)
ALPHA	(contents of MA) +50
Card Primitive	(contents of MA) +A0
H.S.R. Primitive	(contents of MA) +C0
ASR 33/733 ASR Primitive*	(contents of MA) +E0

*The ROM assembly boards (226861-0001), revision * and A, contain the ASR-33 primitive loader. Revision B and later contain the ASR-33/733 ASR teletypewriter or cassette primitive loader.

SECTION III

MACHINE INSTRUCTIONS

3.1 GENERAL

Each machine instruction is divided into distinct fields of one or more bits each. The formats for these instruction fields are classified into three basic groups. Most of the instructions have the same fundamental format and belong to format group I. The other two groups include instructions that require manipulation of data bits and words and are dependent on base registers.

This section describes the formats within each group and lists the relevant machine instructions for each format. Each instruction requires 32 bits in two consecutive memory word locations. The instruction bit locations are designated bit 0 (most significant) to bit 31 (least significant).

A source statement contains a label field, an operation field, an operand field, and a comment field. In addition, it can contain the sequence number of the source line after the comment field. The label field, comment field and sequence number can appear in the statement but are not required. In the typical source statements shown for each format, these fields are indicated by <label>, <comments>, and <seq>. In addition, the "at" symbol (@), number symbol (#), and <rq>, which appear in the operand field of some source statements, are optional. The asterisk (*) and <xr> may or may not be optional, depending on the instruction. They appear in the operand field of some of the source statements. In the typical source statements shown for each format, entries that are permitted but not required are enclosed in brackets ([ ]). User-supplied items are enclosed in angle brackets (< >).

In this section the machine instruction descriptions are in order of their internal hardware formats, they are found listed by available functions (arithmetic, shifting, etc..) in Appendix C along with a brief description of the necessary operands.

3.2 ADDRESSING MODES

Seven distinct addressing modes are available for instruction operands. Each can be used with one or two of the instruction format groups. These addressing modes are summarized in table 3-1 and are described in the following paragraphs. The table includes an example of the assembly language source statement operand for each mode.

Elements of source statements may be relocatable; that is, they may be placed in available memory locations depending on the loading address (load bias).

3.2.1 DIRECT OPERAND. Each machine instruction has one or two fields that contain a memory address location. A direct operand, the most general type of operand, is located at the memory address specified by the address field of an instruction.

The following assembly language source statement, an example of a direct operand, loads register number 2 with the contents of address N:

```
L   2,N
```

L is the mnemonic for the Load Register instruction. N is the address operand, and is a direct operand.

Table 3-1. Addressing Modes

Addressing Mode	Example of Source Statement Operands	Applicable Instruction Format Group
Direct Operand	2,N	I
Indirect Operand	2,*N	I
Indexed Operand	2,N,3	I
Indirect Indexed Operand	2,*N,3	I
Immediate Addressing	2,N	I
Base Register Relative Addressing	2,@N,5	I,II
Alternate Mode Registers	#2,N	I,III

3.2.2 **INDIRECT AND INDEXED OPERANDS.** Operand addresses in machine instructions may be modified by means of indirect addressing and indexing. These modifications are discussed in the following paragraphs.

3.2.2.1 **Indirect Operand.** If the memory location specified by the address field of the instruction contains another address (the operand address) instead of an operand, the instruction has an indirect operand. A one-bit field in the machine instruction word specifies whether indirect addressing is to be done. An asterisk (*) preceding the address operand indicates indirect addressing in a source statement, as in this example, which loads register 2 with the value whose address is found in the location specified by N:

```
L   2,*N
```

3.2.2.2 **Indexed Operand.** Indexing is a means of using an index number to represent a memory address, where the memory address is the sum of a base number and an index number. An index register contains the index number. The contents of the address field in the machine instruction is the base number. In a source statement, the address operand is the base number or a symbol that represents the base number.

Some machine instructions include an index bit that indicates whether indexing is to be performed. If an indirect operand is indexed (paragraph 3.2.2.3), the Index Mode bit in the status register specifies the order in which the indexing and indirect addressing are carried out.

An indexed operand is represented by a register number (0 through 7) or symbol positioned after the address operand, as in the following source statement:

```
L   2,N,3
```

In this example, register 3 is the index register. Register 2 will be loaded with the value found in the address specified by N plus the contents of register 3.

3.2.2.3 Indirect Indexed Operand. If an operand is both indirect and indexed, the indexing may be performed either before or after the indirect addressing. The Index Mode, bit in the status register specifies either pre-indexing or post-indexing and is discussed later.

In indirect addressing, the address field of a machine instruction contains a memory address (the instruction operand address). The memory location specified by the instruction operand address contains another memory address (the referenced address). If pre-indexing is specified, the instruction operand address is indexed before accessing the referenced address. If post-indexing is specified, the referenced address is indexed.

An example of the source statement for an indirect indexed operand:

```
L    2,*N,3
```

N is an indirect address and register 3 is the index register.

The source statement example given above can be used to explain pre-indexing and post-indexing. Figure 3-1 shows a machine instruction, index register, and selected memory locations. The addresses and operands are hexadecimal numbers. The first word of the machine instruction contains the operation code for L (Load Register instruction). Register 3 is the index register, and register 2 is to be loaded. The second word is the N field, which contains the value 0150_{16} (instruction operand address), the value of N.

The first and second words of the machine instruction are placed in the internal CPU instruction decoding registers, IA and IB. At the start of both pre- and post-indexing, the IB register contains the value N. In pre-indexing, the indexing is done before the indirect addressing. The contents of the index register, 0004, is added to the contents of the IB register, 0150, and the result, 0154, is placed in the IB register. (The register contents, memory addresses, and memory word contents in this discussion are hexadecimal numbers.) The contents of memory address 154 is now placed in the IB register, so that the register contains 03C7. This number is in turn a memory address (the effective address), which contains the desired data. This data, 026F, is placed in register 2.

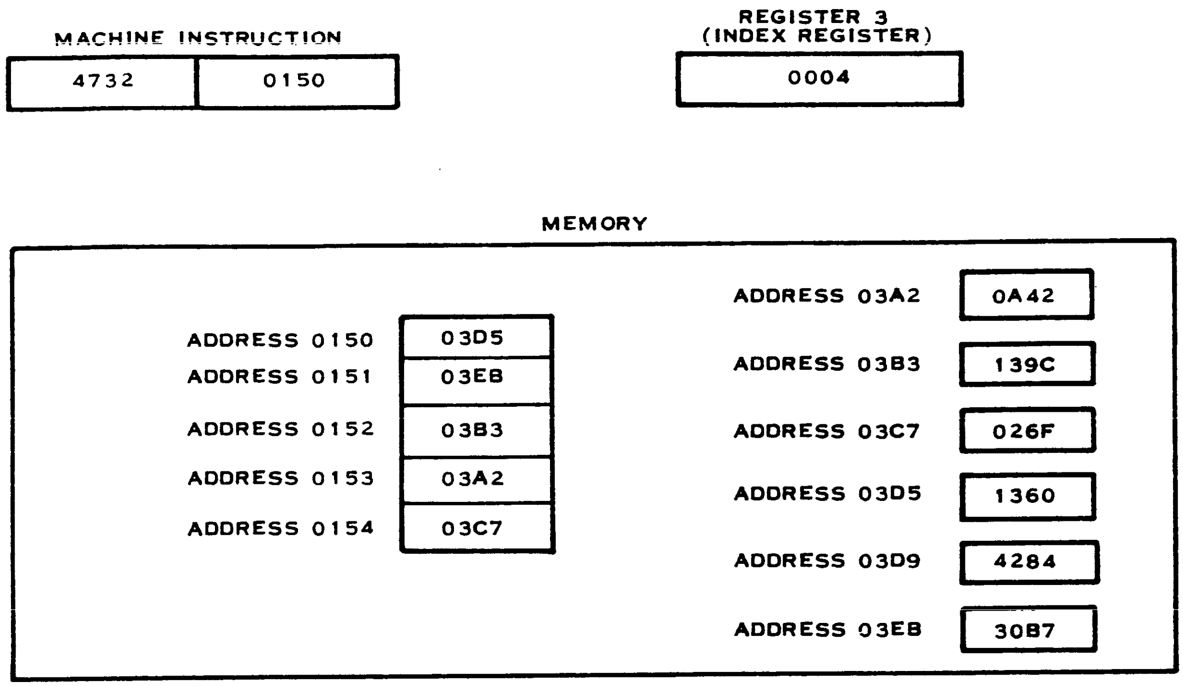
The first step in post-indexing is indirect addressing, so that the contents of memory address 150 replaces the address 0150 in the IB register. The IB register contents therefore becomes 03D5. This number is indexed by the index register contents, 0004, giving the result 03D9. This number is the effective address of the desired data. The data, 4284, is placed in register 2. Note that this number is different from the data placed in register 2 when pre-indexing is specified.

3.2.3 IMMEDIATE ADDRESSING. When immediate addressing is used, the machine instruction operates on the effective address in the address field as if it were an operand. The instruction modifies the numeric value of the instruction operand address in the same manner as the effective address calculated in the example in paragraph 3.2.2.3.

In a source statement, immediate addressing is accomplished by using a different mnemonic for the operation. The immediate counterpart of the Load Register (L) instruction, for example, would be Load Register with Address (LA). The source statement that parallels the example in paragraph 3.2.1 is therefore:

```
LA   2,N
```

Register 2 would be loaded with the value (address) of N.



(A)128925

Figure 3-1. Typical Example of Machine Instruction, Index Register, and Memory Word Contents

3.2.4 **BASE REGISTER RELATIVE ADDRESSING.** In base register relative addressing, one or more registers are designated as base registers. Depending on the instruction, the base registers may be chosen by the programmer, or may be predetermined by the CPU and implied in the instruction. Certain instructions make use of one or two of these base registers. The sum of the contents of the base register and the address in the instruction is the absolute address of the operand. The contents of the instruction address is called the relative address.

The base register numbers are defined in the CPU as follows:

Register No.	Function
4	Data base register
5	Procedure base register
6	Flag base register
7	Communication Register Unit (CRU) base register

SAL is structured so that a user can construct programs from four basic types of program segments. These segment types are procedure segment (PSEG), data segment (DSEG), flag segment (FSEG), and Communication Register Unit (CRU) symbolic address segment (BSEG). The segment types are described in Section V. In some of the machine instructions (format groups II and III) automatic base registers are used for specified purposes, typically with one of the program segment types. For example, when a software flag instruction is used to reference a software flag in the software flag segment, the value of the symbol is biased with (i.e., added with) the contents of the Software Flag base register during execution of the instruction. This action yields the absolute bit address of the software flag. In other words, the displacement of the symbol relative to the origin of the segment in which it was defined (the relative address) is specified rather than the absolute address of the symbol.

A relative address is indicated in source statements by an "at" symbol (@) preceding the address operand. The following statement is an example:

```
L    2,@N,5
```

In this statement, the address N is a relative address and register 5 is used as a base register. It is the programmer's duty to load the base registers as required by the application.

In format group III machine instructions, the "at" symbol is usually not necessary and is considered illegal syntax. The address operand is automatically handled as a relative address by the assembler. With format group II, the base register must be specified explicitly or implicitly. Assignment is explicit if the user specifies the base register in the instruction. Implicit assignment results when the assembler defaults assignment to the base register of the segment in which the symbol is defined.

For example:

```
LABEL   DSEG
TV      RES     64
        END
P       PSEG
TVRO    EQU     TV+10
        AMI     (TVRO,4),2
```

The value calculated for TVRO in the AMI instruction is relative to register 5, not register 4. At execution time, the value 2 is added to the effective address calculated incorrectly by TVRO (relative to register 5) plus the contents of register 4.

In format groups II and III, the offset value calculated in the machine instruction is relative to the segment in which the symbol is defined, regardless of the base register specified in the source statement.

3.2.5 ALTERNATE MODE REGISTERS. Either the supervisor or the worker mode is the active mode for the execution of an instruction. Normally the instruction uses the general register of the active mode, but it is possible to specify use of the general register of the inactive mode. A number symbol (#) preceding the operand list causes the register of the inactive mode to be used in the execution of the instruction, as in the following source statement example:

```
L    #2,N,3
```

This statement loads inactive register 2 from the address calculated by N plus the contents of active register 3. An alternate mode register can not be used for indexing.

3.2.6 COMBINATIONS OF ADDRESSING MODES. The address modification attributes described in the preceding paragraphs may be used in combination, as shown in this example:

```
L    #2,*@N,4
```

ALTERNATE MODE REGISTER #2 FOR DESTINATION, INDIRECT, RELATIVE OPERAND N, INDEXING BY REGISTER 4 OF THE ACTIVE MODE.

This statement loads register 2 of the inactive mode with the relative address operand, N. The operand is both indirect and indexed, with register 4 of the active mode used as an index register. Register 4 is also the base register for relative addressing.

The same register can be used as both a general register and an index register as shown in the following example:

L 2,N,2

This statement fetches the operand at the memory address specified by N plus the contents of register number 2 and then loads this operand into register 2.

3.3 GENERAL FORMAT DESCRIPTION

Machine instructions are implemented in three basic formats, each of which has some subsets. Each instruction requires two consecutive memory locations. The basic formats are defined as:

0–5	6	7	8	9–11	12	13–15	16–31
op	x	*	k	xr	#	r	n

GROUP I

0–5	6–15	16–18	19–21	22–31
op	m	m_b	n_b	n

GROUP II

0–5	6–15	16–19	20	21	22–31
op	m	b	v_1	#	n

GROUP III

A description of each of the fields is:

op The basic operation code field of an instruction.

* The bit of the instruction used to specify indirect addressing.

x The bit of the instruction used to specify that indexing is to be done.

k Immediate operand indicator.

xr The field of the instruction used to specify an index register.

\# The field of the instruction used to specify alternate mode registers.

r The field of the instruction that specifies a general register in the register file or a count.

n The field of an instruction used as an address field.

m The field of an instruction used as an address field in two address instructions.

m_b The field of an instruction used to specify a base register to be used with the M address field.

n_b The field of an instruction used to specify a base register to be used with the N address field.

v_I A bit used as an immediate value in flag and bit instructions.

b The field used to specify a flag address within a memory word or the number of bits in a communication register.

In addition to the above some of the subformats might contain one of the following additional fields:

b Output bit data

q Additional device address data

rq Relative address control bit

The x and * fields, used to specify address modification, have the following meanings:

X — Index bit - specifies indexing

* — Indirect bit - specifies indirect

X	*	
0	0	Direct operand
0	1	Indirect operand
1	0	Indexed operand
1	1	Indirect, indexed operand

3.3.1 **PRE-INDEXING AND POST-INDEXING.** Address modifications, as specified by the x and * fields, are performed using the xr and the n fields. If indirect addressing and indexing are both specified, the indexing is done before the indirect addressing if bit 6 (im) of the status register is zero (pre-indexing). If it is a one, the indexing is done after the indirect addressing (post-indexing). Any of the eight registers of each Mode Register File can be used as an index file.

The usual conventions in the Operating System and user programs utilize pre-indexing rather than post-indexing. See section 6-11 for a more detailed discussion.

3.3.2 **EFFECTIVE OPERANDS AND ADDRESSES.** In the machine instructions descriptions that follow the word "effective operand (or address)" should be taken to mean the operand or address that results from any indirect, indexed (pre-index or post-index), base-relative or other mode of addressing, after the required adjustment has been accomplished in the process of decoding the instruction for hardware execution.

3.3.3 **OPERAND SYMBOL DEFINITIONS.** In the following paragraphs the operand symbols are defined as:

* Indirect addressing

\# Alternate mode addressing

@ Base register relative addressing

RQ Relative address control bit

3.4 FORMAT GROUP I–GENERAL INSTRUCTIONS

This format group includes six subsets, designated I-A through I-F. Each of the six formats is described in a separate paragraph. Format group I instructions are structured as:

0–5	6	7	8	9–11	12	13–15	16–31
op	x	*	k	xr	#	r	n

3.4.1 **FORMAT I-A.** Format I-A is a general format structure that accommodates a large number of instructions with common field requirements. Table 3-2 lists these instructions.

A typical source statement for this format is: typical source statement for this format is:

[<label>] <oper> [#]<gr>,[*][@]<address>[,<xr>] [<comments>] [<seq>]

In this statement, <oper>, <gr>, and <address> are required. The other entries are permitted, but not required. <oper> stands for an instruction mnemonic. The number symbol (#), asterisk (*), and "at" symbol (@) are memory addressing attributes. <gr> represents the general register, and <seq> the sequence number of the source line.

The machine instruction format is identical to that shown for format group I.

A given source statement results in a 32-bit string of binary digits in the corresponding machine instruction. To see how this occurs, consider the following source statement:

LA 2,15 LOAD EFFECTIVE ADDRESS

LA is the mnemonic for the Load Register with Address instruction, which has the operation code 4480 0000_{16}. The first six bits of the operation code, which correspond to the first hexadecimal digit, and the first two bits of the second digit are placed in the OP field of the instruction. Therefore, the OP field contains 010001_2. Note that bits not designating the machine operation are included as zeros in the operation code.

The LA instruction loads the effective address (15) into register 2. In the machine instruction, the R field contains 2 (010_2), and the N field contains 15 (1111_2). The N field is right-justified with leading zeros.

Table 3-2. Format I-A Instructions

Mnemonic	Instruction Name	Hexadecimal Operation Code	
A	Add to Register	4C00	0000
AA	Add Address to Register	4C80	0000
BL	Branch and Link	7480	0000
*BL	Branch Indirect and Link	7400	0000
CR	Compare Register with Memory	C000	0000
CRA	Compare Register with Effective Address	C080	0000
CRL	Compare Register wtih Memory (Logical)	C400	0000
CRLA	Compare Register with Effective Address (Logical)	C480	0000
D	Divide	D000	0000
DA	Divide Immediate	D080	0000
DAD	Double Add	E800	0000
DLAX	Shift Memory Double Left Arithmetic, Count in Register r	C880	0000
DRAX	Shift Memory Double Right Arithmetic, Count in Register r	D480	0000
DRLX	Shift Memory Double Right Logical, Count in Register r	D880	0000
DRRX	Double Right Rotate, Count in Register r	DC80	0000
DS	Double Subtract	EC00	0000
L	Load Register	4400	0000
LA	Load Register with Address	4480	0000
LOT	Load Ones Tally	5400	0000
LOTA	Load Ones Tally of Address	5480	0000
M	Multiply	CC00	0000
MA	Multiply Immediate	CC80	0000
MLAX	Shift Memory Left Airthmetic, Count in Register r	6080	0000
MRAX	Shift Memory Right Arithmetic, Count in Register r	6480	0000
MRRX	Rotate Memory Right Logical, Count in Register r	6880	0000
N	Logical AND	5800	0000
NA	Logical AND with Address	5880	0000
OR	Logical OR	5C00	0000
ORA	Logical OR with Address	5C80	0000
S	Subtract from Register	5000	0000
SA	Subtract Address from Register	5080	0000
SAT	Shift and Add Tally of Leading Zeros	6C00	0000
ST	Store Register	4880	0000
XOR	Exclusive OR	4000	0000
XORA	Exclusive OR with Address	4080	0000

The x, *, xr, and # fields are all set to zero. Because LA is an immediate operand, the k field contains 1. The entire machine instruction in object format is:

0–5	6	7	8	9–11	12	13–15	16–31
0100 01	0	0	1	000	0	010	0000 0000 0000 1111

$(4482000F_{16})$

The following paragraphs contain the description of the format I-A instructions and coding information.

3.4.1.1 **A (ADD to Register).**
Operand: [#] <r>, [*] [@] <n> [,<xr>]

Overflow: Yes
Carry: Yes
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
010011	x	*	0	xr	#	r	n

The effective operand is added to the contents of register r and the result is placed in register r.

3.4.1.2 **AA (Add Effective Address to Register).**
Operand: [#] <r> [*] [@] <n> [,<xr>]

Overflow: Yes
Carry: Yes
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
010011	x	*	1	xr	#	r	n

The effective address is added to the contents of register r and the result is placed in register r.

3.4.1.3 **BL (Branch and Link to Subroutine).**
Operand: [#] <r> [*] [@] <n> [,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: No

0–5	6	7	8	9–11	12	13–15	16–31
011101	x	*	1	xr	#	r	n

The nonupdated contents of either the PC or the EC, depending on the mode of execution, are placed in register r. The effective address is loaded into either the PC or the EC.

3.4.1.4 *BL (Indirect Branch and Link to Subroutine).

Operand:[#]<r>,[*][@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: No

0–5	6	7	8	9–11	12	13–15	16–31
011101	x	*	0	xr	#	r	n

The nonupdated contents of either the PC or the EC, depending on execution mode, are placed in register r. The effective operand is loaded into either the PC or the EC.

3.4.1.5 CR (Compare Register With Memory).

Operand:[#]<r>,[*][@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
110000	x	*	0	xr	#	r	n

The value of effective register r is compared arithmetically with the effective operand and the appropriate compare status bit will be set accordingly.

	ST10	ST11	ST12
(R) > EO	1	0	0
(R) = EO	0	1	0
(R) < EO	0	0	1

3.4.1.6 CRA (Compare Register With Effective Address).

Operand: [#]<r>,[*][@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
110000	x	*	1	xr	#	r	n

The value of effective register r is compared arithmetically with the effective address and the appropriate compare status bit will be set accordingly.

	ST10	ST11	ST12
(R) > EA	1	0	0
(R) = EA	0	1	0
(R) < EA	0	0	1

3.4.1.7 CRL (Compare Register With Memory (Logical).)

Operand: [#]<r>,[*][@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
110001	x	*	0	xr	#	r	n

The contents of effective register r is compared logically (as unsigned 16-Bit Intergers) with the effective operand and the appropriate compare status bit will be set accordingly.

	ST10	ST11	ST12
(R) > EO	1	0	0
(R) = EO	0	1	0
(R) < EO	0	0	1

3.4.1.8 CRLA (Compare Register With Effective Address (Logical).)

Operand: [#]<r>,[*][@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
110001	x	*	1	xr	#	r	n

The contents of effective register r is compared logically (as unsigned 16-Bit Integers) with the effective address and the appropriate compare status bit will be set accordingly.

	ST10	ST11	ST12
(R) > EA	1	0	0
(R) = EA	0	1	0
(R) < EA	0	0	1

3.4.1.9 D(Divide).

Operand: [#]<r>,[*][@]<n>[,<xr>]

Overflow: Yes
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
110100	x	*	0	xr	#	r	n

The concatenated contents of effective registers r and r+1 is divided by the effective operand. The quotient is placed in register r+1 and the remainder is placed in register r. The quotient if multiplied by the divisor and added to the remainder will result in the original dividend. Overflow will be set if the quotient, a signed two's-complement integer, cannot be contained without truncation in a 16-bit word or if the divisor is zero.

This is an optional instruction.

CAUTION

When supervisor mode register mode register 7 is specified, the registers used are supervisor mode register 7 and worker mode register 0. When worker mode register 7 is specified, worker mode register 7 and the contents of memory location 90_{16} are used.

3.4.1.10 DA (Divide By Effective Address).

Operand: [#]<r>,[*][@]<n>[,<xr>]

Overflow: Yes
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
110100	x	*	1	xr	#	r	n

The concatenation of effective registers r and r+1 is divided by the effective address. The quotient is placed in register r+1 and the remainder is placed in register r. The quotient if multiplied by the divisor and added to the remainder will result in the original dividend. Overflow will be set if the quotient, a signed two's-complement integer, cannot be contained without truncation in a 16-bit word or if the divisor is zero.

This is an optional instruction.

CAUTION

When supervisor mode register mode register 7 is specified, the registers used are supervisor mode register 7 and worker mode register 0. When worker mode register 7 is specified, worker mode register 7 and the contents of memory location 90 are used.

3.4.1.11 DAD (Double Add).

Operand: [#]<r>,[*][@]<n>[,<xr>]

Overflow: Yes
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
111010	x	*	0	xr	#	r	n

The concatenated contents of the effective address and the effective address plus one are added to the concatenated contents of effective registers r and r+1. The overflow indicator (OI) may be set by this instruction. The most significant half of the result is placed in effective register r and the least significant half in effective register r+1.

This is an optional instruction.

CAUTION

When supervisor mode register mode register 7 is specified, the registers used are supervisor mode register 7 and worker mode register 0. When worker mode register 7 is specified, worker mode register 7 and the contents of memory location 90 are used.

3.4.1.12 DLAX (Shift Memory Double Left Arithmetic, Count in Register R).

Operand: [#]<r>,[*][@]<n>[,<xr>]

Overflow: Yes
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
110010	x	*	1	xr	#	r	n

The concatenated contents of the effective address and the effective address plus one are shifted left and stored in the effective address and the effective address plus one. If the sign bit is changed during the shifting, the overflow indicator is set. Zeros fill vacated Bit positions.

The shift count is taken as the least significant 5 bits of register r; therefore, one to 32 place shifts can be performed. If a shift count of zero is specified a 32 place shift is performed.

This is an optional instruction.

3.4.1.13 DRAX (Shift Memory Double Right Arithmetic, Count In Register R).

Operand: [#]<r>,[*][@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
110101	x	*	1	xr	#	r	n

The concatenated contents of the effective address and the effective address plus one are shifted right and stored in the effective address, and the effective address plus one. The sign is propagated during the shift.

The shift count is taken as the least significant 5 bits of register r; therefore, one to 32 place shifts can be performed. If a shift count of zero is specified, a 32 place shift is performed.

This is an optional instruction.

3.4.1.14 DRLX (Shift Memory Double Right Logical, Count In Register R).

Operand: [#]<r>,[*] [@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
110110	x	*	1	xr	#	r	n

The concatenated contents of the effective address and the effective address plus one are shifted right and stored in the effective address and the effective address plus one.

The shift count is taken as the least significant 5 bits of register r; therefore, one to 32 shifts can be performed. If a shift count of zero is specified a 32 place shift is performed.

This is an optional instruction.

3.4.1.15 DRRX (Double Right Rotate, Count In Register R).

Operand:[#]<r>,[*] [@]<n>[,<xr>]

Overflow: No
Carry; No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
110111	x	*	1	xr	#	r	n

The concatenated contents of the effective address and the effective address plus one are shifted right the number of places specified in bits 11-15 of register r. The result is stored in the effective address and effective address plus one. Bit fifteen of the least significant half replaces bit zero of the most significant half. One to thirty-two place shifts can be performed. If a shift count of zero is specified a thirty-two place shift is performed.

This is an optional instruction.

3.4.1.16 DS (Double Subtract).

Operand:[#]<r>,[*] [@]<n>[,<xr>]

Overflow: Yes
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
111011	x	*	0	xr	#	r	n

The concatenated contents of the effective address and the effective address plus one are subtracted from the concatenated contents of effective registers r and r+1. The overflow indicator (OI) may be set by this instruction. The most significant half of the result is placed in effective register r and the least significant half in effective register r+1.

This is an optional instruction.

CAUTION

When supervisor mode register mode register 7 is specified, the registers used are supervisor mode register 7 and worker mode register 0. When worker mode register 7 is specified, worker mode register 7 and the contents of memory location 90 are used.

3.4.1.17 L (Load Register).

Operand: [#]<r>,[*][@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
010001	x	*	0	xr	#	r	n

The effective operand is loaded into the specified register.

3.4.1.18 LA (Load Register With Effective Address).

Operand: [#]<r>,[*][@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
010001	x	*	1	xr	#	r	n

The effective address is loaded into the specified register.

3.4.1.19 LOT (Load Ones Tally)

Operand: [#]<r>,[*][@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
010101	x	*	0	xr	#	r	n

The binary ones in the effective operand are counted and the result is placed in the specified register.

3.4.1.20 LOTA (Load Ones Tally of Effective Address).

Operand: [#]<r>,[*] [@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
010101	x	*	1	xr	#	r	n

The binary ones in the effective address are counted and the result is placed in the specified register.

3.4.1.21 M(Multiply)

Operand: [#]<r>,[*] [@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
110011	x	*	0	xr	#	r	n

The value in effective register r is multiplied by the effective operand. The produce is placed in effective registers r and r+1. The most significant part of the product is in register r and least significant is in register r+1. Overflow cannot occur.

This is an optional instruction.

CAUTION

When supervisor mode register mode register 7 is specified, the registers used are supervisor mode register 7 and worker mode register 0. When worker mode register 7 is specified, worker mode register 7 and the contents of memory location 90 are used.

3.4.1.22 MA (Multiply By Effective Address)

Operand:[#]<r>,[*] [@]<n>[,<xr>]

MULTIPLY IMMEDIATE

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
110011	x	*	1	xr	#	r	n
			K				

The value in effective register r is multiplied by the effective address. The product is placed in effective registers r and r+1. The most significant part of the product is in register r and least significant is in register r+1. Overflow cannot occur.

This is an optional instruction.

CAUTION

When supervisor mode register mode register 7 is specified, the registers used are supervisor mode register 7 and worker mode register 0. When worker mode register 7 is specified, worker mode register 7 and the contents of memory location 90 are used.

3.4.1.23 MLAX (Shift Memory Left Arithmetic, Count in Register R).

Operand: [#]<r>[*][@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
011000	x	*	1	xr	#	r	n

The effective operand is shifted left the number of places specified by the contents of bits 12-15 of register r. The shifted operand is stored in the effective address. If the sign bit is changed during shifting, the overflow indicator is set. Zeros fill vacated bit positions. If a shift count of zero is specified, 16 places are shifted.

3.4.1.24 MRAX (Shift Memory Right Arithmetic, Count in Register R).

Operand: [#]<r>,[*][@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
011001	x	*	1	xr	#	r	n

The effective operand is shifted right the number of places specified by the contents of bits 12-15 of register r. The shifted operand is stored in the effective address. The sign bit is propagated during the shift. If a shift count of zero is specified, 16 places are shifted.

3.4.1.25 MRRX (Memory Rotate Right, Count In Register R).

Operand: [#]<r>,[*][@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
011010	x	*	1	xr	#	r	n

The effective operand is shifted right the number of places specified by the contents of bits 12-15 of register r. Bits shifted out of bit position 15 are entered in bit position 0. The shifted operand is stored in the effective address. If a shift count of zero is specified, 16 places are shifted.

3.4.1.26 N (Logical And).

Operand: [#]<r>,[*] [@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
010110	x	*	0	xr	#	r	n

The contents of register r are logically ANDed bit-by-bit with the effective operand. The results are placed in register r.

3.4.1.27 NA (Logical And With Effective Address).

Operand: [#]<r>,[*] [@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
010110	x	*	1	xr	#	r	n

The contents of register r are logically ANDed bit-by-bit with the effective address. The results are placed in register r.

3.4.1.28 OR (Logical OR).

Operand: [#]<r>,[*] [@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
010111	x	*	0	xr	#	r	n

The contents of register r are logically ORed bit-by-bit with the effective operand. The result is placed in register r.

3.4.1.29 ORA (Logical OR With Effective Address).

Operand: [#]<r>,[*] [@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
010111	x	*	1	xr	#	r	n

The contents of register r are logically ORed bit-by-bit with the effective address. The results are placed in register r.

3.4.1.30 S (Subtract From Register).

Operand: [#]<r>,[*] [@] <n>[,<xr>]

Overflow: Yes
Carry: Yes
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
010100	x	*	0	xr	#	r	n

The effective operand is subtracted from the contents of register r. The result is placed in register r.

3.4.1.31 SA (Subtract Effective Address From Register).

Operand: [#]<r>,[*] [@] <n>[,<xr>]

Overflow: Yes
Carry: Yes
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
010100	x	*	1	xr	#	r	n

The effective address is subtracted from the contents of register r. The result is placed in register r.

3.4.1.32 SAT (Shift And Add Tally).

Operand: [#]<r>,[*] [@] <n>[,<xr>]

Overflow: No
Carry: Yes
Mode Switching: No
COMPARISON FLAGS SET → Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
011011	x	*	0	xr	#	r	n

The effective operand is shifted left until bit 0 is logic 1. If the effective operand is 0, 16 places are shifted. The count of the number of positions shifted is added to register r. The shifted effective operand is stored in the effective address with bit position 0 forced to logic 0.

NOTE

If bit 0 is initially logic 1, no shifting is done. However, bit position 0 is still forced to a logic 0.

3.4.1.33 ST (Store Register)

Operand: [#] <r>,[*] [@] <n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
010010	x	*	1	xr	#	r	n

The contents of the specified register are stored in the memory location specified by the effective address.

3.4.1.34 XOR (Exclusive OR).

Operand: [#] <r>,[*] [@] <n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
010000	x	*	0	xr	#	r	n

The contents of register r are logically exclusive-ORed bit-by-bit with the effective operand. The result is placed in register r. Where bits in register r are equal to bits in the effective operand zeros are placed in register r. Otherwise ones are placed in register r.

3.4.1.35 XORA (Exclusive OR With Effective Address).

Operand: [#] <r>,[*] [@] <n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13–15	16–31
010000	x	*	1	xr	#	r	n

The contents of register r are logically exclusive-ORed bitwise with the effective address. The result is placed in register r. Where bits in register r are equal to corresponding bits in the effective address zeros are placed in register r. Otherwise ones are placed in register r.

3.4.2 FORMAT I-B. Format I-B instructions include the memory shift instructions. These instructions are listed in table 3-3. A typical source statement in this format has the following form:

[<label>] <oper> <count>, [*] [@] <address>[,<xr> [<comment>] [<seq>]

Table 3-3. Format I-B Instructions

Mnemonic	Instruction Name	Hexadecimal Operation Code
BC	Branch on Condition	E080 0000
*BC	Branch on Condition Indirect	E000 0000
DLA	Shift Memory Double Left Arithmetic	C800 0000
DRA	Shift Memory Double Right Arithmetic	D400 0000
DRL	Shift Memory Double Right Logical	D800 0000
DRR	Double Right Rotate	DC00 0000
MLA	Shift Memory Left Arithmetic	6000 0000
MRA	Shift Memory Right Arithmetic	6400 0000
MRR	Rotate Memory Right Logical	6800 0000

<oper> is the instruction mnemonic. <count> represents the shift count. In machine instruction format, the shift count is contained in a 4-bit variable field, r. The machine instruction format is:

0 1 2 3 4 5	6	7	8	9 10 11	12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
op	x	*	k	xr	r	n

The following paragraphs contain the descriptions of the format I-B instructions and coding information.

3.4.2.1 BC (Branch On Condition).

Operand: <r>,[*] [@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: No

0 1 2 3 4 5	6	7	8	9 10 11	12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
111000	x	*	1	xr	r	n

The r field specifies a bit in the status register to be examined. If the status bit selected by the r field is logic 1, PC or EC is loaded with the effective address. If not, the next instruction is executed.

r = 0000 Selects Status Register Bit 0

.
.
.

r = 0111 Selects Status Register Bit 7

.
.
.

r = 1111 Selects Status Register Bit 15

3.4.2.2 *BC (Indirect Branch On Condition).

Operand: <r>,[*][@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: No

0–5	6	7	8	9–11	12–15	16–31
111000	x	*	0	xr	r	n

The r field specifies a bit in the status register to be examined. If the status register bit selected by the r field is logic 1, PC or EC is loaded with the effective operand. If not, the next instruction is executed.

r = 0000 Selects Status Register Bit 0

.
.
.

r = 0111 Selects Status Register Bit 7

.
.
.

r = 1111 Selects Status Register Bit 15

3.4.2.3 DLA (Shift Memory Double Left Arithmetic).

Operand: <r>,[*][@]<n>[,<xr>]

Overflow: Yes
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12–15	16–31
110010	x	*	0	xr	r	n

The concatenated contents of the effective address and the effective address plus one are shifted left r places and stored in the effective address and the effective address plus one. If r = 0, 16 places are shifted. If the sign bit is changed during the shifting, the overflow indicator is set. Zeros fill vacated bit positions.

NOTE

One to 16 place shifts may be specified by this instruction.

This is an optional instruction.

3.4.2.4 DRA (Shift Memory Double Right Arithmetic).

Operand: <r>,[*] [@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12–15	16–31
110101	x	*	0	xr	r	n

The concatenated contents of the effective address and the effective address plus one are shifted r places and stored in the effective address and the effective address plus one. The sign bit is propagated during the shift. If r = 0, 16 places are shifted.

NOTE

One to 16 place shifts may be specified by this instruction.

This is an optional instruction.

3.4.2.5 DRL (Shift Memory Double Right Logical).

Operand: <r>[*] [@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12–15	16–31
110110	x	*	0	xr	r	n

The concatenated contents of the effective address and the effective address plus one are shifted right r places and stored in the effective address and the effective address plus one. Zeros fill the vacated bit positions.

NOTE

One to 16 place shifts may by specified by this instruction.

This is an optional instruction.

3.4.2.6 DRR (Double Right Rotate).

Operand:<r>,[*] [@]<n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12–15	16–31
110111	x	*	0	xr	r	n

The concatenated contents of the effective address and the effective address plus one are shifted right r places and stored in the effective address and the effective address plus one. If r = 0 sixteen places are shifted. Bit fifteen of the least significant half replaces bit zero of the most significant half.

NOTE

One to 16 place shifts may be specified by this instruction.

This is an optional instruction.

3.4.2.7 MLA (Shift Memory Left Arithmetic).

Operand: <r>,[*] [@] <n>[,<xr>]

Overflow: Yes
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12–15	16–31
011000	x	*	0	xr	r	n

The effective operand is shifted left r places and stored in the effective address. r = 0 indicates a shift of 16 places.

If the sign bit is changed during the shifting, the overflow indicator is set. Zeros fill vacated bit positions.

3.4.2.8 MRA (Shift Memory Right Arithmetic).

Operand:<r>,[*] [@] <n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12–15	16–31
011001	x	*	0	xr	r	n

The effective operand is shifted right r places and stored in the effective address. The sign bit is propagated during the shift.

Sixteen places are shifted if r = 0.

3.4.2.9 MRR (Memory Rotate Right).

Operand: <r>,[*] [@] <n>[,<xr>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6	7	8	9–11	12–15	16–31
011010	x	*	0	xr	r	n

The effective operand is shifted right r places. The bits shifted out of bit position 15 are placed in bit position 0. The result is stored in the effective address. If a shift count of zero is specified, 16 places are shifted.

3.4.3 **FORMAT I-C.** Format I-C instructions include combinations of status block storage, mode transfers, and branches. The Load Status Block and the No Operation instructions are also part of this set. Table 3-4 contains a list of the instructions. A typical source statement has the format:

[<label>] <oper> [*][@]<address>[,<xr>][,<rq>] [<comments>] [<seq>]

<rq> is the relative address control bit. If the rq bit in the machine instruction is evaluated as logic 1, general register 5 biases the address. The machine instruction format is:

0–5	6	7	8	9–11	12	13–15	16–31
op	x	*	op	xr	rq	op	n

Table 3-4. Format I-C Instructions

Mnemonic	Instruction Name	Hexadecimal Operation Code
B	Unconditional Branch	7082 0000
*B	Unconditional Branch Indirect	7002 0000
LDS	Load Status Block	7C00 0000
NOP	No Operation	7007 0000
SS	Store Status Block	7886 0000
SSB	Store Status Block and Branch	7882 0000
SXBS	Store Status Block, Transfer and Branch in Supervisor Mode	7880 0000
SXBW	Store Status Block, Transfer and Branch in Worker Mode	7881 0000
SXS	Store Status Block and Transfer to Supervisor Mode	7884 0000
SXW	Store Status Block and Transfer to Worker Mode	7885 0000
XS	Transfer to Supervisor Mode	7004 0000
XSB	Transfer to Supervisor Mode and Branch	7080 0000
*XSB	Transfer to Supervisor Mode and Branch Indirect	7000 0000
XW	Transfer to Worker Mode	7005 0000
XWB	Transfer to Worker Mode and Branch	7081 0000
*XWB	Transfer to Worker Mode and Branch Indirect	7001 0000

Bit 12 is the rq field. The OP field is extended to include bits 8 and 13 through 15 in addition to bits 0 through 5. As an example of a specific instruction, consider:

SXBS *127

The operation code for SXBS is 7880 0000_{16}. Bits 0 through 5 contain the binary digits corresponding to the first hexadecimal digit and the first two bits of the second digit, or 011110_2. Bit 8 is equal to 1. Bits 13 through 15 contain the three least significant bits of the fourth hexadecimal digit, in this case three binary zeros.

The x, ia and xr fields are equal to 0, 1 and 0 respectively. The n field contains 127, which in binary is 1111111_2. The n field is right-justified with leading zeros. Therefore, the machine instruction in object format is:

0–5	6	7	8	9–11	12	13–15	16–31
0111 10	0	1	1	000	0	000	0000 0000 0111 1111

($7980007F_{16}$)

The following paragraphs contain the descriptions of the format I-C instructions and coding information. The instruction descriptions reflect the order in which events occur. For instructions which store the current mode location counter (PC or EC), the location counter will already be updated by two. For instructions which change modes, the previous mode location counter will have been updated by two.

3.4.3.1 B (Unconditional Branch).

Operand: [*] [@] <n>[,<xr>] [,<rq>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: No

0–5	6	7	8	9–11	12	13	14	15	16–31
011100	x	*	1	xr	rq	0	1	0	n

The effective address is loaded into either the PC or the EC depending on the execution mode. If rq = 1, the effective address plus the contents of the execution mode register 5 is loaded into either the PC or EC.

3.4.3.2 *B (Unconditional Branch Indirect).

Operand: [*] [@] <n>[,<xr>] [,<rq>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: No

0–5	6	7	8	9–11	12	13	14	15	16–31
011100	x	*	0	xr	rq	0	1	0	n

The effective operand is loaded into either the PC or EC depending on the execution mode. If rq = 1, the effective operand plus the contents of execution mode register 5 is loaded into the PC or the EC.

3.4.3.3 LDS (Load Status Block).

Operand: [*] [@] <n>[,<xr>] [,<rq>]

Overflow: Yes
Carry: Yes
Mode Switching: Yes
Comparison Indication: Yes

0–5	6	7	8	9–11	12	13	14	15	16–31
011111	x	*	0	xr	rq	0	0	0	n

The effective operand is placed in the PC or EC, based on the execution mode. Then the contents of the effective address plus one are placed in the status register. The operation is performed with the PC if execution is in the supervisor mode and with the EC if execution is in worker mode. If rq = 1, the contents of register 5 are added to the effective operand before it is placed in the PC or the EC.

PROGRAMMING NOTE

Care must be taken when changing modes with the LDS instruction since the active mode location counter (PC or EC) is loaded before the new status is loaded.

3.4.3.4 NOP (No Operation).

Operand: Not used

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: No

0–5	6–11	12	13	14	15	16–31
011100	not used	0	1	1	1	not used

The PC or the EC, depending on the mode of execution, is incremented by two.

3.4.3.5 SS (Store Status Block).

Operand: [*] [@] <n>[,<xr>] [,<rq>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: No

0–5	6	7	8	9–11	12	13	14	15	16–31
011110	x	*	1	xr	rq	1	1	0	n

The PC or the EC (depending upon the execution mode) is stored at the effective address. The status register is stored at the next address.

If rq = 1, the contents of register 5 are subtracted from the PC or EC before it is stored.

3.4.3.6 SSB (Store Status Block and Branch).

Operand: [*] [@] <n>[,<xr>] [,<rq>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: No

0–5	6	7	8	9–11	12	13	14	15	16–31
011110	x	*	1	xr	rq	0	1	0	n

The PC or EC (depending upon the execution mode) is stored at the effective address. The status register is stored at the next address. The PC or EC is loaded with the contents of the effective address plus 2. If rq = 1, the contents of register 5 are subtracted from the PC or EC before it is stored and the contents of register 5 are added to the contents of the effective address plus 2 before it is placed in the PC or EC.

3.4.3.7 SXBS (Store Status Block, Transfer And Branch In Supervisor).

Operand: [*] [@] <n>[,<xr>] [,<rq>]

Overflow: No
Carry: No
Mode Switching: Yes
Comparison Indication: No

0–5	6	7	8	9–11	12	13	14	15	16–31
011110	x	*	1	xr	rq	0	0	0	n

The PC is stored at the effective address and the status register is stored at the next address. A transfer to supervisor mode is forced and the PC is loaded with the contents of the effective address plus 2. If rq = 1, the contents of supervisor mode register 5 are subtracted from the PC before it is stored. The contents of the same register are added to the contents of the effective address plus 2 before it is loaded into the PC.

3.4.3.8 SXBW (Store Status Block, Transfer And Branch In Worker Mode).

Operand: [*] [@] <n>[,<xr>] [,<rq>]

Overflow: No
Carry: No
Mode Switching: Yes
Comparison Indication: No

0–5	6	7	8	9–11	12	13	14	15	16–31
011110	x	*	1	xr	rq	0	0	1	n

The EC is stored at the effective address and the status register is stored at the next address. A transfer to worker mode is forced and the EC is loaded with the contents of the effective address plus 2. If rq = 1, the contents of worker mode register 5 are subtracted from the EC before it is stored and the contents of the same register are added to the contents of the effective address plus 2 before it is loaded into the EC.

3.4.3.9 SXS (Store Status Block, Transfer to Supervisor Mode).

Operand: [*] [@] <n>[,<xr>] [,<rq>]

Overflow: No
Carry: No
Mode Switching: Yes
Comparison Indication: No

0–5	6	7	8	9–11	12	13	14	15	16–31
011110	x	*	1	xr	rq	1	0	0	n

The PC is stored at the effective address. The status register is stored at the next address and a transfer to supervisor mode is forced. If rq = 1, the contents of supervisor register 5 are subtracted from the PC before it is stored.

3.4.3.10 SXW (Store Status Block, Transfer to Worker Mode).

Operand: [*] [@] <n>[,<xr>] [,<rq>]

Overflow: No
Carry: No
Mode Switching: Yes
Comparison Indication: No

0–5	6	7	8	9–11	12	13	14	15	16–31
011110	x	*	1	xr	rq	1	0	1	n

The EC is stored at the effective address. The status register is stored at the next address and a transfer to worker mode is forced. If rq = 1, the contents of worker mode register 5 are subtracted from the EC before it is stored.

3.4.3.11 XS (Transfer To Supervisor Mode).

Operand: Not used

Overflow: No
Carry: No
Mode Switching: Yes
Comparison Indication: No

0–5	6–12	13	14	15	16–31
011100	not used	1	0	0	not used

A transfer to supervisor mode is forced.

3.4.3.12 XSB (Transfer To Supervisor Mode And Branch).

Operand: [*] [@] <n>[,<xr>] [,<rq>]

Overflow: No
Carry: No
Mode Switching: Yes
Comparison Indication: No

0–5	6	7	8	9–11	12	13	14	15	16–31
011100	x	*	1	xr	rq	0	0	0	n

A transfer to supervisor mode is forced and the PC is loaded with the effective address. If rq = 1, the contents of supervisor mode reigster 5 are added to the effective address before it is placed in the PC.

3.4.3.13 *XSB (Transfer To Supervisor Mode And Branch Indirect).

Operand: [*] [@] <n>[,<xr>] [,<rq>]

Overflow: No
Carry: No
Mode Switching: Yes
Comparison Indication: No

0–5	6	7	8	9–11	12	13	14	15	16–31
011100	x	*	0	xr	rq	0	0	0	n

A transfer to supervisor mode is forced, and the PC is loaded with the effective operand. If rq = 1, the contents of supervisor mode register 5 are added to the effective operand before it is placed in the PC.

3.4.3.14 XW (Transfer To Worker Mode).

Operand: Not used

Overflow: No
Carry: No
Mode Switching: Yes
Comparison Indication: No

0–5	6–12	13	14	15	16–31
011100	not used	1	0	1	not used

A transfer to worker mode is forced.

3.4.3.15 XWB (Transfer To Worker Mode and Branch).

Operand: [*] [@] <n>[,<xr>] [,<rq>]

Overflow: No
Carry: No
Mode Switching: Yes
Comparison Indication: No

0–5	6	7	8	9–11	12	13	14	15	16–31
011100	x	*	1	xr	rq	0	0	1	n

A transfer to worker mode is forced, and the EC is loaded with the effective address. If rq = 1, the contents of worker mode register 5 are added to the effective address before it is placed in the EC.

3.4.3.16 *XWB (Transfer To Worker Mode And Branch Indirect).
Operand: [*][@]<n>[,<xr>][,<rq>]

Overflow: No
Carry: No
Mode Switching: Yes
Comparison Indication: No

0–5	6	7	8	9–11	12	13	14	15	16–31
011100	\	*	0	xr	rq	0	0	1	n

A transfer to worker mode is forced. The EC is loaded with the effective operand. If rq = 1, the contents of worker mode register 5 are added to the effective operand before it is placed in the EC.

3.4.4 FORMAT I-D. Format I-D consists of only one instruction, the ARB (Add to Register and Branch on No Sign Change) instruction, with a typical source statement of:

[<label>] ARB <addnd>,[@]<address>,<xr>,[<rq>] [<comments>] [<seq>]

The addend, <addnd>, is contained in the r field of the machine instruction. The n field contains the address. The rq field is bit 8 in this format. The first six bits of the ARB operation code, which is $0C00\ 0000_{16}$, are in bit positions 0 through 5 of the instruction. Bits 6 and 7 are unused. The machine instruction format is:

0–5	6–7	8	9–11	12–15	16–31
0000 11	not used	rq	xr	r	n

3.4.4.1 ARB (Add To Register And Branch).
Operand:<r>,[@]<n>,<xr>[,<rq>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

RELATIVE MODE SET.

0–5	6–7	8	9–11	12–15	16–31
000011	not used	rq	xr	r	n

The 4-bit, two's-complement value in the r field is added to the index register specified in the xr field. The result is placed in the index register. If the sign of the index register changes, the next instruction in sequence is executed. If the sign does not change, the next instruction is executed at the address specified by n.

If rq = 1, the effective address is the value from the n field plus the contents of execution mode register 5.

PROGRAMMING NOTE

Occasionally stand-alone programs can use the ARB instruction to decrement a counter for timing purposes. The instruction execution time is 4.000 microseconds only when the RQ feature is invoked. Therefore the following provides a 15-millisecond delay:

```
LA      2,-3750     Counter
ARB     1,@$,2,1    Delay here
```

(Register 5 is assumed to be set to the beginning of the PSEG in which the code is contained.)

3.4.5 **FORMAT I-E.** Format I-E is the format for the Activate Direct Memory Access Channel (ADAC) instruction. A typical source statement is:

[<label>] ADAC <devadd>,<listad>,[<q>] [<comments>] [<seq>]

<devadd> is the device address on the Direct Memory Access Channel, and <listad> is the input/output command list address. The a field of the machine instruction contains <devadd> and the n field contains <listad>. q, which corresponds to <q>, is an optional field that may contain additional device address data. The machine instruction format is:

0 1 2 3 4 5	6 7	8 9 10 11 12	13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0010 01	not used	q	a	n
		OPTIONAL DEV. ADDR DATA.	DEVICE ADDR.	COMMAND LIST ADDRESS

The first six bits of the machine instruction correspond to the operation code for the ADAC instruction, $2400\ 0000_{16}$.

3.4.5.1 **ADAC (Activate Direct Access Channel).**

Operand:<a>, <n>[,<q>]

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: No

0 1 2 3 4 5	6 7	8 9 10 11 12	13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
001001	not used	q	a	n

A command is generated to a device or controller on the direct memory access channel. The a field contains a device or controller address. The n field contains the memory address of an initialization list. The q field contains information related to a specific device or controller.

3.4.6 **FORMAT I-F.** The Store Panel Switches (STPS) instruction is the only Format I-F instruction. A typical source statement for STPS is:

[<label>] STPS [*][@]<n>[,<xr>] [<comments>] [<seq>]

The front panel data switch setting is stored in the effective address. Bits 12 through 15 of the machine instruction are unused. Bit 8 is a logical 1. The machine instruction format is:

0–5	6	7	8	9–11	12–15	16–31
1110 01	x	*	1	xr	unused	n

The operation code for STPS is E480 0000_{16}.

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

3.5 FORMAT GROUP II–MEMORY BASE RELATIVE INSTRUCTIONS

Format Group II instructions have two operands. This format has three subsets and are described in the following paragraphs. The general format for this group is:

0–5	6–15	16–18	19–21	22–31
op	m	mb	nb	n

The two address fields are designated m and n. The contents of the fields are:

op Basic operation code of the instruction

m m address field

mb Base register to be used with the m address field

nb Base register to be used with the n address field

n n address field

In each case, the basic operand format is:

(<disp>,<gr>)

which is evaluated as the displacement (<disp>) plus the contents of the general register (<gr>). If <disp> is relocatable, the assembler automatically uses its segment relative address. The segment relative address is also used if <disp> is a relocatable external reference. The user may specify the operand in either of these two ways:

(<disp>,<gr>) (Explicit)

<disp> (Implicit)

If the implicit form is chosen, it must be a defined location in a procedure segment or data segment within the program. SAL assembles the implicit form as (<disp>,4) or (<disp>,5) depending on whether <disp> is defined in a data or procedure segment, where register 4 is the data base register, and register 5 is the procedure base register.

The user should make sure that the appropriate base register or registers being used contain the assumed values.

3.5.1 FORMAT II-A. Format II-A is used for instructions that move or compare words within memory. The Format II-A instructions are listed in table 3-5.

Table 3-5. Format II-A Instructions

Mnemonic	Instruction Name	Hexadecimal Operation Code
CM	Compare Memory to Memory	1000 0000
CML	Compare Memory to Limits in Memory	1800 0000
MOV	Move Memory Word	1400 0000

The typical source statement has two forms, explicit and implicit, and the user may choose either one:

[<label>] <oper> [@] (<displ>,<gr1>), [@] (<disp2>,<gr2>) [<comments>] [<seq>] (Explicit)

[<label>] <oper> [@] <disp1>, [@] <disp2> [<comments>] [<seq>] (Implicit)

In the implicit form, no external references may appear. Both operands need not be expressed in the same form, as shown in these examples:

```
MOV    (disp1,4), disp2
CM     (disp1,3), (disp2,7)
CML    disp1,disp2
```

The machine instruction format is identical to that shown for format group II. The m field contains either <displ> or the relative address of <from>, and the n field either <disp2> or the relative address of <to>, depending on the source statement option chosen. mb and nb are explicitly defined base registers when they contain <gr1> and <gr2>, but are determined by the segment classes (data or procedure segment) in which <from> and <to> are defined when the implicit form of the source statement is used.

The following paragraphs contain the descriptions of the format II-A instructions and coding information.

3.5.1.1 CM (Compare Memory With Memory)

Operand:

[@](<m>,<mb>),[@](<n>,<nb>)
[@]<this>,[@]<with>

Option 1, Explicit base register definition.
Option 2, Base register determined by segment class in which symbol is defined.

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: No

0–5	6–15	16–18	19–21	22–31
000100	m	m_b	n_b	n

The contents of the memory location addressed by m plus the contents of the register specified by m_b are compared arithmetically with the contents of the memory location addressed by n plus the contents of the register specified by n_b.

If the first operand is less than the second, the next instruction in sequence is executed.

If the first operand is greater than the second, one instruction is skipped.

Two instructions are skipped if the operands are equal.

3.5.1.2 CML (Compare Memory With High and Low Limits in Memory)

Operand:

[@](<m>,<mb>),[@](<n>,<nb>)
[@]<namem>,[@]<limits>

Option 1, Explicit base register definition.
Option 2, Base register determined by segment class in which symbol is defined.

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: No

0–5	6–15	16–18	19–21	22–31
000110	m	m_b	n_b	n

The first operand, address by m plus the contents of the register specified by m_b, is compared with a lower and upper limit. The lower limit is addressed by n plus the contents of the register specified by n_b. The upper limit must occupy the next memory location after the lower limit.

If the first operand is arithmetically less than the lower limit, the next instruction in sequence is executed.

If the first operand is arithmetically greater than the upper limit, one instruction is skipped.

If the first operand is within the limits or equal to a limit, two instructions are skipped.

3.5.1.3 MOV (Move Memory to Memory).

Operand:

[@](<m>,<mb>),[@](<n>,<nb>) Option 1, Explicit base register definition.

[@]<namem>,[@]<namen> Option 2, Base register determined by segment class in which symbol is defined.

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6–15	16–18	19–21	22–31
000101	m	m_b	n_b	n

The contents of the register specified by m_b are added to m to obtain the effective address of operand 1. Likewise the contents of the register specified by n_b are added to n to obtain the effective address of operand 2.

Operand 2 is replaced by operand 1.

Execution in the supervisor mode uses supervisor mode registers for m_b and n_b; worker mode uses worker mode registers.

3.5.2 FORMAT II-B. Format II-B instruction is the Branch Relative and Link (BRRL). The typical source statement for this instruction has two formats:

[<label>] BRRL <link>, [@](<disp>,<gr>) [<comments>] [<seq>](Explicit)

[<label>] BRRL <link>, [@]<there> (Implicit)

In the implicit form, external references can not appear in the branch address.

The operation code for the BRRL instruction is 2800 0000_{16}. The first six bits of this code are 001010_2, the contents of the OP field in the machine instruction format. The r field contains <link>, the nb field <gr> if the first source statement option is used, and the N field either <disp> or the relative address of <there>. If the implicit form is used, the operand base in the nb field is determined by the segment class in which the label is defined. The machine instruction format has the format:

0–5	6–15	16–18	19–21	22–31
0010 10	unused	r	nb	n

BRRL (Branch Relative To Register and Link to Subroutine).
Operand: <r>,[@](<n>,<nb>)
<r>,[@]<namen>

Option 1, Explicit base register definition.
Option 2, Base register determined by segment class in which symbol is defined.

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: No

0–5	6–15	16–18	19–21	22–31
001010	unused	r	n_b	n

The contents of the register specified by the n_b field are subtracted from the PC or EC, depending on the mode of execution, and the result is stored in the register specified by the r field. The contents of the register specified by the n_b field are added to the value of the n field. This result is placed in the PC or the EC depending on the execution mode.

3.5.3 **FORMAT II-C.** The format II-C instructions are AMI (Add to Memory Immediate) CMI (Compare Memory Immediate) and can have either of the following two formats:

[<label>] AMI [@](<disp>,<gr>),<value> [<comments>] [<seq>] (Explicit)

[<label>] AMI [@] <locat>,<value> [<comments>] [<seq>] (Implicit)

If the implicit form is chosen, no external references would be used in the memory address. In the machine instruction format, the m field contains <disp> or the relative address of <locat> and the 13-bit signed value field contains <value>. mb contains <gr>, the explicitly defined base register, if the first source statement option is used. If the second option is used, the base register is determined by the segment class in which the symbol <locat> is defined. The machine instruction format is:

0–5	6–15	16–18	19–31
op	m	mb	13-bit value

The Format II-C instructions are described in the following paragraphs.

3.5.3.1 **AMI (Add to Memory Immediate).**
Mnemonic: AMI
Operand: [@](<m>,<mb>),<value>
[@]<namem>,<value>

Option 1, Explicit base register definition.
Option 2. Base register determined by segment class in which symbol is defined.

Overflow: Yes
Carry: Yes
Mode Switching: No
Comparison Indication: Yes

0–5	6–15	16–18	19–31
001000	m	m_b	13-bit value

The 13-bit two's-complement value is added to the memory location addressed by m plus the contents of the register specified by m_b. The result is placed in the same memory location.

The sign of the 13-bit value is extended before addition is performed.

3.5.3.2 CMI (Compare Memory Immediate).

Operand: [@](<m>,<mb>),<value> Option 1, Explicit base register definition

[@]<namem>,<value> Option 2, Base register determined by segment class in which symbol is defined.

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: No

0–5	6–15	16–18	19–31
000111	m	m_b	13-bit signed value

The contents of the memory location addressed by m plus the contents of the register specified by m_b are compared arithmetically to the 13-bit two's-complement value in the instruction. The sign bit of the 13-bit value is extended to bit zero before the comparison is made.

If the memory operand is less than the immediate value the next instruction in sequence is executed.

One instruction is skipped if the memory operand is greater than the immediate value.

Two instructions are skipped if the memory operand is equal to the immediate value.

3.6 FORMAT GROUP III–FLAG AND CRU DATA MANIPULATION INSTRUCTIONS

Format Group III instructions allow flag or CRU bit or CRU field manipulation. The general format for this instruction group is.

0–5	6–15	16–19	20	21	22–31
op	m	b	v_1	am	n

The contents of the fields are:

- op Basic operation code of the instruction
- m m address field
- b Flag address within a memory word or the number of bits in a communication register
- vl Immediate value in flag and bit instructions
- am Bit that specifies whether alternate mode registers are used
- n n address field

Register 6, the flag base register, must point to a flag work area for flag instructions, and register 7, the CRU base register, must contain the CRU base address.

3.6.1 **FORMAT III-A.** Format III-A is used with two software flag instructions: Switch Mode if Flag Not Equal (XFNE, operation code 8000 0000_{16}) and Set Flag (SETF, operation code 8800 0000_{16}). The typical source statement can have one of two formats:

[<label>] <oper> [#]<flagn>,<f>[<comments>] [<seq>] (Implicit)

[<label>] <oper> [#](<word>,<bit>),<f>[<comments>] [<seq>] (Explicit)

The two source statement options represent different ways of defining the flag word and bit addresses. To specify the flag bit, the implicit form uses a software flag name that has been defined by the FLAG assembly directive in a flag segment. This flag name identifies both the word and bit addresses of the flag. In the explicit form, the flag word and the bit addresses are each defined separately. They may be non-relocatable symbols or constants that specifically identify the word and bit addresses by number.

In the implicit example, <flagn> represents the flag name. This name causes the correct entries to be placed in the M and B fields of the machine instruction. In the second example, the M field contains <word> and the B field contains <bit>. The value bit (V1) is used to make a comparison with the flag bit in memory. This bit corresponds to <f> in the source statement operand list. The machine instruction format is:

0–5	6–15	16–19	20	21	22–31
op	m	b	V_1	am	unused

The following paragraphs contain the descriptions of the format III-A instructions and coding information.

3.6.1.1 SETF (Set Flag)

Operand: [#](<m>,<b>),<v1> Explicit Definition
[#]<namef>,<v1> Definition by Name

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

0–5	6–15	16–19	20	21	22–31
100010	m	b	V_1	#	unused

The contents of the software Flag Base Register (6) are added to m to obtain the effective address. Bit b of the effective operand is set equal to v1.

If the # attribute is used, bit 21 is logic 1 and the alternate mode Software Flag Base Register (6) is used to calculate the effective address. If bit 21 is logic 0 the execution mode register is used.

3.6.1.2 XFNE (Switch Modes If Flag Not Equal).

Operand: [#](<m>,<b>),<v1> Explicit Definition
[#]<namef>,<v1> Definition by Name

Overflow: No
Carry: No
Mode Switching: Yes
Comparison Indication: No

0–5	6–15	16–19	20	21	22–31
100000	m	b	v_1	#	unused

The value v1 is compared with bit b of the memory location addressed by m plus the contents of the Software Flag Base Register (6). If the comparison fails a mode change is forced.

If the # attribute is used, bit 21 is logic 1 and the alternate mode Software Flag Base Register (6) is used to calculate the effective address. If bit 21 is logic 0, the execution mode register is used.

When XFNE causes a mode change the EC or PC that addressed the instruction is not changed.

3.6.2 FORMAT III-B. The Branch If Flag Not Equal (BFNE) instruction is the Format III-B instruction and a typical source statement can be either of two formats:

[<label>] BFNE [#]<flagn>,<f>,<there> [<comments>] [<seq>] [Implicit]

[<label>] BFNE [#](<word>,<bit>),<f>,<there> [<comments>] [<seq>] (Explicit)

The relative flag word address (<word>) appears in the m field. The bit address within the word appears in the b field. <flagn> in the first source statement option is a symbol that causes the flag word and bit addresses to be placed in the m and b field respectively. The n field contains the relative branch address (<there> in the source statement). If the branch is taken, it uses register 5 as a base register. The first source statement option uses the flag name designated in the flag segment of the assembly and the second option explicitly defines the flag bit address. The symbol <word> in the sublist of the explicit form must be nonrelocatable. Constants can also be used in place of symbols in this sublist. The value bit, v1, is used to make a comparison with the flag bit in memory. v1 corresponds to <f> in the source statement operand list.

The BFNE operation code is $8400\ 0000_{16}$. The first six bits of this code, 100001_2, appear in bit positions 0 through 5 of the machine instruction and have the format.

0–5	6–15	16–19	20	21	22–31
1000 01	m	b	v_1	am	n

The following paragraph contains the description of the format III-B instruction and coding information.

3.6.2.1 BFNE (Branch If Flag Is Not Equal).

Operand: [#]（<m>,<b>),<v1>,<n> Explicit Definition
[#]<namef>,<v1>,<namen>Definition by Name

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: No

0–5	6–15	16–19	20	21	22–31
100001	m	b	v_1	#	n

The value v1 is compared with bit b of the memory location addressed by m plus the contents of the Software Flag Base Register (6). If the comparison fails, the PC or EC, depending on execution mode, is loaded with n plus the contents of the Procedure Base Register.

If the # attribute is used, bit 21 is logic 1 and the alternate mode Software Flag Base Register (6) is used to calculate the effective address. If bit 21 is logic 0, the execution mode register is used.

3.6.3 FORMAT III-C.

There are two Format III-C instructions: Set CRU Output Bit (SETB, operation code 3400 0000_{16}) and Switch Mode On Bit Not Equal (XBNE, operation code 3800 0000_{16}). The two forms for the typical source statement are:

[<label>] <oper> [#]<bitout>,<b> [<comments>] [<seq>] (Implicit)

[<label>] <oper> [#]<bitin>,<b> [<comments>] [<seq>] (Explicit)

The m field of the machine instruction contains the relative CRU bit address (<bitout> and <bitin> in the source statement). The value bit, v1 (which corresponds to <b> in the source statement operand list) is used to set the addressed CRU bit or make a comparison with it. <bitout> must have been defined in a CRU segment by a CON directive as specific CRU lines. <bitin> is a constant between 0 and 1023. Bits 16 through 19 are not used. The machine instruction format is:

0–5	6–15	16–19	20	21	22–31
op	m	unused	v_1	am	unused

The following paragraphs contain the descriptions of the format III-C instructions and their coding information.

3.6.3.1 SETB (Set CRU Output Bit)

Operand: [#]<m>,<v1> Explicit Definition
[#]<namem>,<v1> Definition by Name

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: No

0–5	6–15	16–19	20	21	22–31
001101	m	unused	v_1	#	unused

The value in the v1 field is used to set the CRU output bit addressed by m plus the contents of the CRU Base Register (7).

The execution mode base register is used unless the # attribute has been invoked: then, bit 21 is logic 1, and the alternate mode base register is used to calculate the CRU bit address.

3.6.3.2 XBNE (Switch Modes If Bit Not Equal).

Operand: [#]<m>,<v1> Explicit Definition
[#]<namem>,<v1> Definition by Name

Overflow: No
Carry: No
Mode Switching: Yes
Comparison Indication: No

Bits 0–5	Bits 6–15	Bits 16–19	Bit 20	Bit 21	Bits 22–31
001110	m	unused	v_1	#	unused

The value v1 is compared with the CRU input line address by m plus the contents of the CRU Base Register (7). If the comparison fails a mode change is forced.

The execution mode base register is used unless the # attribute has been invoked; then, bit 21 is logic 1, and the alternate mode CRU Base Register is used to calculate the CRU address.

When XBNE causes a mode change, the PC or EC that addressed the instruction is not changed.

3.6.4 FORMAT III-D. There is only one Format III-D instruction the Branch On Bit Not Equal (BBNE) instruction. A typical source statement using BBNE is:

[<label>] BBNE [#]<bitin>,<b>,<there> [<comments>] [<seq>]

The operation code for BBNE is 3000 0000_{16} , and the first six bits of the code, 001100_2, constitute the OP field in the object format. The m field <bitin> in the source statement example) plus the contents of register 7 (or alternate mode register 7 if alternate mode is specified) is the CRU bit address. v1 is the immediate value operand corresponding to <b> in the source statement operand list. n is the procedure relative branch address. Bits 16 through 19 are not used. The machine instruction format is:

Bits 0–5	Bits 6–15	Bits 16–19	Bit 20	Bit 21	Bits 22–31
0011 00	m	unused	v_1	am	n

The following paragraph contains the description of the format III-D instruction and its coding information.

3.6.4.1 BBNE (Branch If Bit Not Equal).

Operand: [#]<m>,<v1>,<n> Explicit Definition
[#]<namem>,<v1>, <namen> Definition by Name

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: No

Bits 0–5	Bits 6–15	Bits 16–19	Bit 20	Bit 21	Bits 22–31
001100	m	unused	v_1	#	n

The value v1 is compared with the CRU input line addressed by m plus the contents of the CRU Base Register (7). If the comparison fails, the PC or EC is loaded with n plus the contents of the Procedure Base Register (5).

The execution mode base registers are used unless the # attribute has been invoked; then, bit 21 is a 1, and the alternate mode CRU Base Register (2) is used in the CRU address calculation. The branch address is always calculated using the current mode Procedure Base Register (5).

3.6.5 FORMAT III-E. There is only one Format III-E instruction, the Test Input Bit and Switch Mode or Set Output Bit (TSBX) instruction, with a typical source statement:

<label> TSBX [#]<bitin>,<b>,<bitout>,<ob> [<comments>] [<seq>]

The labels <bitin> and <bitout> must have been defined as CRU lines in a CRU symbolic address segment or must be constants between 0 and 1023. <bitin> plus the contents of register 7 (or alternate mode register 7 if alternate mode is specified) is the CRU bit address under test, <b>, in the source statement, is the input bit value to be tested, and corresponds to v1 in the machine instruction format. <ob> is the output bit value to be set, and <bitout> plus the contents of register 7 (or alternate mode register 7 if alternate mode is specified) is the output bit address. The m and n fields contain <bitin> and <bitout> respectively. The machine instruction format is:

0–5	6–15	16–18	19	20	21	22–31
0011 11	m	unused	ob	v_1	am	n

The contents of the OP field, 001111_2, corresponds to the first six bits of the TSBX operation code, $3C00\ 0000_{16}$.

The following paragraph contains the description of the format III-D instruction and its coding information.

3.6.5.1 TSBX (Test Input Bit and Set Output Bit or Switch Modes).

Operand:		
[#]<m>,<v1>,<n>,<b>	Explicit Definition	Overflow: No
[#]<namem>,<v1>, <namen>,<b>	Definition by Name	Carry: No
		Mode Switching: Yes
		Comparison Indication: No

0–5	6–15	16–18	19	20	21	22–31
001111	m	unused	b	v_1	#	n

The CRU input line addressed by m plus the contents of the CRU Base Register (7) is compared to v1. If the test fails, a mode change is forced; otherwise, the value of bit b is output to the CRU output line addressed by n plus the contents of the CRU Base Register (7).

The execution mode base register is used unless the # attribute has been invoked; then, bit 21 is logic 1 and the alternate mode CRU Base Register (7) is used to calculate the CRU addresses. Also, when bit 21 is logic 1, mode switching is inhibited.

When TSBX causes a mode change the PC or EC which addressed the instruction is not changed.

3.6.6 FORMAT III-F. The format III-F instructions are two instructions that transfer data between the CRU register and memory. They are Load Communication Register (LDCR, operation code 0800 0000_{16}) and Store Communication Register (STCR, operation code 2C00 0000_{16}). (See paragraph 6-6 for illustration of data transfer from CPU to CRU.) The typical source statement can have either of the two formats:

[<label>] <oper> [#](<line>,<f1>),<memory>[<comments>] [<seq>] (Explicit)

[<label>] <oper> [#]<crfld>,<memory>[<comments>] [<seq>] (Implicit)

Symbols used to implicitly reference CRU registers must be defined using the CON directive. The m field in the machine instruction format contains the CRU starting line address. The b field contains the number of bits in a field except that a 16-bit field is indicated by 0. In the first source statement option, <line> plus the contents of register 7 (or alternate mode register 7 if alternate mode is specified) is the CRU starting line address and <f1> is the number of bits in the field. <crfld> is a symbol that causes the correct values of the starting line address and the number of field bits to be placed in the m and b fields if the implicit form is chosen. n is the relative address of the data (<memory> in the source statement). The memory location is found by adding n with the contents of general register 4. The machine instruction format is:

0–5	6–15	16–19	20	21	22–31
op	m	b	unused	am	n

The following paragraphs contain the descriptions of the Format III-F instructions and their coding information.

3.6.6.1 LDCR (Load Communication Register).

Operand: [#](<m>,<b>),<n> Explicit Definition
[#]<namem>,<namem> Definition by Name

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: No

0–5	6–15	16–19	20	21	22–31
000010	m	b	unused	#	n

The right-justified bit field in the memory location addressed by n plus the contents of the Data Base Register (4) is output to consecutive CRU output lines starting at CRU address m plus the contents of the CRU Base Register (7). The CRU bit addressed by m plus the contents of the CRU Base Register is loaded with the least significant bit of the memory word. The contents of the b field defines the width of the communication register and controls the number of bits sent to the CRU.

b	=	0001	register width = 1
b	=	0010	register width = 2
		.	.
		.	.
		.	.
b	=	1111	register width = 15
b	=	0000	register width = 16

If the # attribute is used, bit 21 is a 1, and the alternate mode registers are used to calculate the effective address and the CRU address. See section 6.6 for an example of how data flows between memory and CRU.

3.6.6.2 STCR (Store Communications Register).

Operand: [#](<m>,<b>),<n> — Explicit Definition
[#]<namem>, <namen> — Definition by Name

Overflow: No
Carry: No
Mode Switching: No
Comparison Indication: Yes

Bits 0–5	Bits 6–15	Bits 16–19	Bit 20	Bit 21	Bits 22–31
001011	m	b	unused	#	n

Sequential CRU input lines are read and stored as right-justified bit fields in memory location n plus the contents of the Data Base Register (4). The CRU input line addressed by m plus the contents of the CRU Base Register (7) is stored as the least significant bit of the field. The b field of the instruction word defines the number of bits read and stored in memory. (See LDCR.) The memory word bit positions to the left of the data stored are forced to agree with the most significant input bit position.

If the # attribute is used, bit 21 is logic 1, and the alternate mode registers are used to calculate the effective address and the CRU address. See section 6.6 for an example of how data flows between memory and CRU.

SECTION IV

LANGUAGE REQUIREMENTS

4.1 SOURCE STATEMENT FORMAT

Assembly language source program statements can contain assembler directives, machine instructions, user-defined operation codes, blank records, or comments. A statement other than a comment or blank statement can contain up to four fields: a label field, an operation field, an operand field, and a comment field. These fields are separated by one or more blanks and, except for the comment field, cannot contain embedded blanks.

Examples of source statement formats are shown in figure 4-1.

The first 60 characters that are read from a source record constitute a symbolic input line. If the !*X (extend printer width) option is used, the input record size is 20 characters less than output printing width. (Options are described in Section VII). This option can decrease input record size to as few as 40 characters or increase input record size to as many as 116 characters. In any case, up to 60 characters are always scanned even if only 40 are printed.

Comment records consist of a single field with an asterisk (*) in the first character position. Each remaining character can be any ASCII character, including a blank. Comment statements appear in the source listing but do not affect the assembly.

Blank records consist of an input record with the first 30 columns containing blanks. The remaining columns are processed as a comment. Blank statements do not affect the assembly.

Null statements are ignored by SAL. A null statement is a zero-length record. If input statements are being entered on a data terminal keyboard, for example, a null statement may be generated by typing only a carriage return (a record-delimiting character).

4.1.1 CHARACTER SET. The SAL assembler recognizes these ASCII characters:

- Capital letters of the alphabet
- Arabic numerals
- Space character
- 23 punctuation marks, signs and symbolic characters

Appendix A contains a list of the characters, their ASCII codes and their Hollerith codes.

4.1.2 LABEL FIELD. Labels, also called names, are used to symbolically reference instructions, values or data. The label field starts in the first character position and extends to the first blank. A label can contain up to six alphanumeric characters, however, the first character must be alphabetic. A label is optional for machine instructions and some assembler directives. If the label is omitted, a blank must appear in the first character position.

TI 960/980 ASSEMBLY CODING FORM

```
LABEL   OPER  OPERAND      COMMENTS
1     6 8  11 13  17  21 25 26  30   35   40   45   50   55   60
* CONVENTIONAL SOURCE STATEMENT FORMAT

START   LA    3,X'25'      LOAD GR 3

        A     5,3          ADD GR 5

        XS                 RETURN TO CALLING PROGRAM

* PACKED SOURCE STATEMENT FORMAT

START LA 3,X'25' LOAD GR 3

 A 5,3 ADD GR 5

 XS RETURN TO CALLING PROGRAM
```

PROGRAM	PROGRAMMED BY	CHARGE	PAGE	OF

(A)128935

Figure 4-1. Source Statement Formats

4.1.3 OPERATION FIELD. The operation field follows the blank that terminates the label field, or starts with the first non-blank character if there is no label field. It contains an assembler directive (Section V) or a machine instruction (Section IV) that defines the operation. It is terminated by one or more blanks. The first character of the operation field must occur at or before character position 19 (but not before position 2). The maximum length of the operation field is four characters.

4.1.4 OPERAND FIELD. The operand field consists of a list of expressions or sublists, separated by commas. It starts after the blank or blanks that terminate the operation field, and at or before character position 21 (but not before position 4). It is followed by one or more blanks, and can not extend past character position 60 of the source record. The operand field can contain one or more expressions, terms or constants, depending on the requirements for the operation field.

An example of an operand field follows:

```
EXPRS1,EXPRS2,(EXPRS3,EXPRS4)
```

In this example, (EXPRS3,EXPRS4) is a sublist. Parentheses are required to delimit a sublist.

4.1.5 COMMENT FIELD. A comment can be written on any line. The comment field starts after the blank or blanks that terminate the operand field. For statements having no operand field, the comment must begin after column 22. The end of the source record or column 60 terminates the comment field. The Extend Printer Width option, !*X (described in Section V), can be used to extend column termination.

The field can contain any ASCII character, including blanks. The contents of the field appear in the source portion of the assembly listing, but do not affect the assembly process.

4.2 EXPRESSIONS

Expressions are used in the operand fields of assembler directives and machine instructions.

4.2.1 DEFINITION. An expression is a constant or symbol, or a series of constants and/or symbols separated by arithmetic operators. In other words, an expression is a chain of terms (paragraph 4.5) and operations. The first constant or symbol of an expression can be preceded by a plus sign (urary plus) or a minus sign (unary minus). A unary sign identifies a number as positive or negative and is not an operator. The expression cannot contain embedded blanks. Only one symbol in an expression can be subsequently defined in the program, but that symbol must not be part of an operand in a multiplication or division operation within the expression. An expression that contains a relocatable symbol or constant immediately following a multiplication or division operator is an illegal expression. Also, when the result of evaluating an expression up to a multiplication or division operator is relocatable, the expression is illegal. An expression in which N_+ minus N_- is not equal to zero or one, where N_+ is the number of relocatable symbols or constants added to the expression and N_- is the number of relocatable symbols or constants subtracted from the expression, is an illegal expression.

An externally referenced symbol appearing in an expression is invalid in format type II and III instructions. If used, it results in an expression error. The use of a referenced symbol in an expression is valid in format type I instructions only when the referenced symbol appears first in the expression.

The following are examples of valid expressions:

BLUE+1
GREEN-4
2*16+RED
BLUE+@GREEN

The following are examples of invalid expressions (assuming all symbols are relocatable):

BLUE+GREEN+4	Invalid because N_+ is not equal to N_- or $N_- + 1$.
4+BLUE/6	Invalid because of the division of a relocatable expression.
4*GREEN	Invalid because a relocatable term follows a multiplication operator.

4.2.2 **ARITHMETIC OPERATORS AND ORDER OF EVALUATION.** The following arithmetic operators can appear in expressions:

- + for addition
- – for subtraction
- * for multiplication
- / for division

In evaluating an expression, the assembler first negates any constant or symbol preceded by a unary minus, then performs the arithmetic operations from left to right. The assembler does not assign precedence to the operations other than unary plus or minus.

For example, the expression 5+6*2 would be evaluated as (5+6)*2=22, not as 5+(6*2)=17. The expression 5+1/2 would be evaluated as 3 rather than 5.

4.3 CONSTANTS

Constants are used in expressions, and can be one of three types: decimal integer constants, hexadecimal integer constants and character constants.

4.3.1 **DECIMAL INTEGER CONSTANTS.** A decimal integer constant is written as a numeric integer with decimal digits. When a decimal integer constant represents data, its range of values is from -32,768 to +32,767.

The following are valid decimal constants:

2000

-32767

15

4.3.2 **HEXADECIMAL INTEGER CONSTANTS.** A hexadecimal integer constant is written as a number with up to four hexadecimal digits enclosed in single quotation marks and preceded by the letter X. The hexadecimal digits are the decimal numerals 0 through 9 and the letters A through F.

The following are valid hexadecimal constants:

X'87'

X'C'

X'46BF'

A hexadecimal number of up to eight digits may be written as a hexadecimal integer string (paragraph 4.6).

4.3.3 CHARACTER CONSTANTS. A character constant is written as a string of one or two characters enclosed within single quotation marks and preceded by the letter C. If a single quotation mark is required within a character constant, it must be represented by two single quotation marks. Eight-bit ASCII codes represent the characters internally, with the first bit in the code for each character equal to zero. A character constant consisting only of two single quotation marks is invalid. If a single character constant is used, the ASCII code is in the most significant eight bits, and a blank (X'20') is placed in the least significant eight bits.

The following are valid character constants:

Constant	Value (ASCII)
C'AB'	4142_{16}
C'C'	4320_{16}
C'N'	$4E20_{16}$
C'''D'	2744_{16}

4.4 SYMBOLS

Symbols are used in the label field and the operand field. The first character of a symbol must be alphabetic, and the others can be any alphanumeric character. None of these characters can be a blank. A symbol cannot consist of more than six characters. A symbol is valid only during the assembly in which it is defined.

To access a symbol external to the assembly, the symbol must be referenced externally. To make a symbol accessible by another assembly, the symbol must be defined externally. (Refer to the descriptions of the DEF and REF directives in Section VII.) A symbol is automatically defined externally unless it is defined in the procedure segment. External references are resolved at link edit time.

A symbol used in the label field is associated with a specific location in the program and must not be used as a label in another statement. The mnemonic operation codes and the names of assembler directives are valid user-defined symbols when placed in the label field. A symbol in the label field can be equated to a valid expression in the operand field by use of the EQU directive.

Any symbol used in the operand field must appear elsewhere in the label field of a statement or in the operand field of a REF directive, with one exception. The exception is the dollar sign character ($) used in expressions to represent the current location within the program.

The following are examples of valid symbols:

START

A1

OPER

$

The following are examples of invalid symbols:

$$

1A

OPERATION

>842

4.5 TERMS

Terms are used in the operand fields of some machine instructions and assembler directives. A term is a decimal or hexadecimal constant, a character string of one or two characters, or a symbol.

The following are examples of valid terms:

12	(decimal constant)
X'C'	(hexadecimal constant)
WR2	(symbol)
C'A'	(character constant)

4.6 HEXADECIMAL INTEGER STRINGS

A hexadecimal integer string is written as a sequence of up to eight hexadecimal digits enclosed in single quotation marks and preceded by the letter X. Up to four hexadecimal digits represent one 16-bit word of data; otherwise, two words of data are represented. All values are right-justified with leading zeros.

The following are valid hexadecimal integer strings:

String	Number of Words	Memory Image	
X'12FA'	One	12FA	
X'2'	One	0002	
X'00002'	Two	0000	0002
X'456ABC'	Two	0045	6ABC

4.7 CHARACTER STRINGS

A character string is written as a sequence of characters enclosed in single quotation marks and preceded by the letter C. To represent a single quotation mark in the character string within the delimiting marks, two consecutive single quotation marks are used. The characters are represented internally as 8-bit ASCII characters.

The following are valid character strings:

C'SAMPLE PROGRAM'

C'PLAN "C" (This string produces the ASCII code for PLAN 'C')

C'OPERATOR MESSAGE * PRESS START SWITCH'

4.8 RELOCATABILITY

SAL produces relocatable object code. This object code can be placed in any available memory locations. Relocatable address information must be incremented by the program's loading address (load bias) at load time so that it can be executed in the specific memory locations in which the program is placed. This is the function of the system's relocating loader. Relocatability allows one program to occupy one of many possible locations by merely changing the load bias.

4.8.1 RELOCATABILITY OF TERMS IN SOURCE STATEMENTS. A term in a source statement is either a constant or a symbol. The relocatability of expressions and terms within the expressions are explained in the following paragraphs. An expression is a chain of terms and arithmetic operators (+,–,*,/).

The relocatability of an expression is a function of the relocatability of the symbols that make up the expression. If N_+ is the number of relocatable symbols added to the expression, and N_- is the number of relocatable symbols subtracted from the expression, then an expression is relocatable when it contains one or more relocatable symbols and

$$N_+ = N_- + 1$$

All valid expressions that do not meet these criteria are absolute.

An expression is invalid if it contains (1) multiplication or division of a relocatable expression or (2) multiplication or division of an expression by a relocatable term. Operations are executed from left to right. The following are examples of valid, invalid, relocatable and non-relocatable (absolute) expressions:

$+5	Valid; relocatable
256-@$	Valid; non-relocatable
LEA-6	Valid; relocatable if LEA is relocatable
LEA/6	Invalid if LEA is relocatable; otherwise, non-relocatable
A-B	Invalid if B is relocatable, but A is not; relocatable if A is relocatable, but B is not; absolute if A and B are both absolute or both relocatable.
6*LEA	Invalid if LEA is relocatable; otherwise, non-relocatable

Any symbol that appears in the label field of a source statement other than a FLAG, CON or EQU directive is relocatable. The symbol in the label field of an EQU directive is relocatable if the expression in the operand field is relocatable.

SECTION V

ASSEMBLER DIRECTIVES

5.1 DIRECTIVES THAT IDENTIFY PROGRAM SEGMENTS

Under SAL, the programmer can construct his programs as stand-alone units or collections of one or more modules of basic segment types. The four segment types are:

- Procedure segment–normally the main body of the program. It contains computer instructions and is the action portion of a program.
- Data segment–used to provide storage, I/O buffers, and constants for use by procedure segments.
- Flag segment–allows the programmer to address memory symbolically, bit-by-bit.
- Communication Register Unit (CRU) symbolic address segment–simplifies assignment and use of symbolic addresses for references to bit signal lines in the CRU, both by register field and by individual bit.

The assembler directives that identify each of these segment types are described in paragraphs 5.1.1 through 5.1.4.

A segment identifier directive is typically used:

```
LABEL  PSEG
```

PSEG is the procedure segment identifier directive. The LABEL entry is passed to the link editor as an external definition and is sent to the link editor in the segment object identification record. A label is required for all four segment identifier directives.

In order to identify symbols within a segment as belonging to that particular segment class, the assembler sets an identification bit and constructs a table of the symbols and segments.

The automatic use of specific base registers in format group III machine instructions (refer to Section IV) makes segmentation convenient and efficient. For example, when referring to a software flag in the flag segment (FSEG) with a software flag instruction, the value of the symbol representing the software flag is automatically added with the contents of the Software Flag base register (register 6) to calculate the effective address during execution of the instruction. This becomes the bit address of the software flag. So, as the assembler is building an instruction that uses relative addressing, the displacement of the symbol relative to the origin of the segment in which it was defined is placed in the instruction rather than the program relative address of the symbol (which is the same as the program relative value if it occurs in the first segment of the program). Format group II instructions allow base registers to be specified in the instruction. For format group I and II instructions, the segment relative value of the symbol can be specified with the use of the relative attribute, the "at" symbol (@). Use of the "at" symbol in format group III instructions causes a syntax error. The relative attribute may also be used with the DATA assembler directive, as in this example:

```
DATA @SYMBOL
```

This directive causes a data word to be initialized with the value of the displacement of SYMBOL relative to the origin of the segment in which it is defined. If the relative attribute is not used in the example, the data word is initialized with the relocatable address of SYMBOL rather than its segment relative value. The following example further illustrates use of the "at" symbol:

```
LA   1,@TEMP,4
```

If the symbol TEMP is defined in a data segment, the "at" symbol effectively converts format group I instructions to the base-displacement addressing mode used in format group II instructions.

Tasks in a segmented program might be handled in the following manner. Three independent process tasks are executed simultaneously under program and monitor control. Each process task is uniquely assigned to one data segment. The addresses of the task process information inputs and outputs are defined by the CRU symbolic address segment, used by all three tasks. (See sample program no. 3 in Appendix F.)

5.1.1 **PROCEDURE SEGMENT (PSEG).** The PSEG assembler directive identifies the procedure segment. PSEG does not require a comment field entry, but requires a label field entry (automatically defined externally). The operand field is omitted, and characters that appear after the operation field are handled as a comment. Any directive can be used within the procedure segment type except FLAG and CON.

5.1.2 **DATA SEGMENT (DSEG).** The DSEG assembler directive identifies the data segment. Proper use of a DATA statement within this segment allows storage to be reserved and initialized. DSEG does not require a comment field entry, but requires a label field entry. The operand field is omitted, and characters that appear after the operation field are handled as a comment. The DSEG label and all symbols defined within this segment are passed to the link editor as external definitions. Any directive can be used within the data segment type except FLAG, CON or DEF.

5.1.3 **FLAG SEGMENT (FSEG).** The FSEG assembler directive identifies the flag segment. This segment does not ordinarily reserve storage, but allows symbolic addresses to be assigned to a particular flag or memory bit. The programmer has the option of reserving storage by including a RES or DATA directive. FSEG does not require a comment field entry, but requires a label field entry. The operand field is omitted, and characters that appear after the operation field are handled as a comment. The label and all symbols defined within this segment are passed to the link editor as external definitions. These directives can be used within the flag segment type: END; FLAG; EQU; RES; DATA; PAGE; TITL; LIS; UNL.

5.1.4 **CRU SYMBOLIC ADDRESS SEGMENT (BSEG).** The BSEG assembler directive identifies the Communication Register Unit (CRU) symbolic address segment. This segment does not reserve storage, but allows symbolic addresses to be assigned to a particular CRU bit or group of bits. BSEG requires an operand (which is an absolute bit address) and a label. The BSEG operand value becomes the CRU base address for all symbolic CRU addresses defined within the segment. This base address is subtracted from all CRU addresses (by SAL) supplied by CON directives. The BSEG label and all symbols defined within this segment are passed to the link editor as external definitions. These directives can be used within the CRU symbolic address segment: END; CON; EQU; PAGE; TITL; LIS; UNL.

5.2 DIRECTIVES THAT CONTROL REGISTERS AND PROGRAM SEGMENTS.

The following paragraphs discuss the Alternate Mode Registers and the Segment Termination assembler directives.

5.2.1 ALTERNATE MODE REGISTERS (MODE). The general form of the MODE directive is:

```
MODE
```

This statement notifies SAL to permit reference to alternate mode (i.e., inactive mode) registers using the alternate mode attribute, indicated by a number symbol (#). Note that any use of the number symbol, when SAL is not provided with a currently active MODE directive, causes an assembly error. Label and operand field entries are ignored. The MODE directive is terminated by an END directive.

5.2.2 SEGMENT TERMINATION (END). The general format of the END directive is:

```
END [OPERAND]
```

This directive terminates a segment and revokes an active MODE statement. A non-blank non-external entry in the operand field is passed to the loader identified as a transfer vector. When more than one END directive is found only the last one is used. Entries in the label field are ignored. The system bootstrap or Programming Support Monitor (PSM) alternate loader exectues an LDS (Load Status Block) instruction on the memory location specified by the END directive operand. Note that an END directive causes a page eject. The following two examples illustrate the use of the END directive.

Example 1:

```
P1          PSEG
            MODE
 PROC       LA  7,0
            L    #3,NUMBER
             .
             .
             .
             .
 START      DATA      PROC, X'8000'
            END       START
```

Example 2:

```
P2          PSEG
             .
             .
             .
            END
FS1         FSEG
             .
             .
             .
```

In the first example, the label START is the address of a status block that is loaded to start the program. The START label in the operand of the END directive tells the loader where to transfer control. The MODE directive enables the use of alternate mode register 3. The END directive terminates the PSEG and the MODE directives.

In the second example, the END directive terminates the procedure segment. A comment or a PAGE, TITL, or segment identifier directive must immediately follow END, except when it is used to mark the conclusion of a program. In that case, END must be followed by an end-of-file record (/*). Anything else after an END directive does not reserve space. The operand of an END directive must be a relocatable value and cannot be an external reference. (Refer to *Model 960 Computer Programming Support Monitor,* manual no. 955380-9701, for more detail about END vectors.)

5.3 DIRECTIVES THAT GENERATE LINKAGE DATA

The following paragraphs discuss the Define Entry Point Symbols (DEF) and the Identify External References (REF) assembler directives. The program linking assembler directives DEF and REF allow independently assembled programs to be symbolically linked into one larger executable program. Symbolic linkages between programs are created by means of symbols defined in one program and used as operands in another program. Such symbols are termed linkage symbols. A linkage symbol is called a defined entry point symbol in the program in which it is defined; it is called a referenced external symbol in the program in which it is used as an operand.

5.3.1 DEFINE ENTRY POINT SYMBOLS (DEF). The general form of the DEF directive is:

DEF OPERAND$_1$,OPERAND$_2$,. . . ,OPERAND$_n$

Every linkage symbol must be properly identified in the source program. A linkage symbol used as an entry point must be identified in the defining program by the DEF directive. DEF is used in PSEG only. DEF directive statements can be placed anywhere in the program as long as they are within the program segments in which their use is allowed. Not more than 256 linkage symbols can be used.

The symbols (separated by commas) in the operand field must be defined elsewhere in the program and can be used as an entry point by other programs. A symbol that is used as an operand in a DEF directive and is not defined in the program is flagged in the listing as an error.

In the following sequence, SQRT is identified as an entry point symbol:

```
PROGA      RES      2
           DEF      SQRT
            .
            .
            .
SQRT       ST       0,SAVE
```

5.3.2 IDENTIFY EXTERNAL REFERENCES (REF). The general form of the REF directive is:

REF OPERAND$_1$,OPERAND$_2$, . . . ,OPERAND$_n$

This statement identifies symbols appearing in the operand list as external references. These externally referenced symbols are passed to the link editor with appropriate data for processing.

REF can be used only in procedure and data program segments.

The external symbols (separated by commas) in the first operand field must be defined in another program and identified in that program as an entry point symbol. For example, if MTPLY is an entry point symbol in another program, the using program identifies it as an external symbol.

```
REF   MTPLY
```

To link to a program named SINE, the following coding could be used.

```
PROGA     RES     2
          REF     SINE
           .
           .
ADSINE    SSB     SINE
```

The following two examples show a method of gaining access to the value of a label in another program segment.

Example 1:

```
MAIN      PSEG
          REF     SINE
           .
           .
           .
ADSINE    BL      1,SINE
          END
```

Example 2:

```
SUB       PSEG
          RES     10
          DEF     SINE
SINE      ST      1,SAVE
           .
           .
           .
          L       1,SAVE
          B       2,1 RETURN
          END
```

Line 2 of the first example indicates that the label SINE appears in a different segment which is to be assembled separately. Line 3, in the second example, declares the label SINE to be externally defined so that the first example segment can have access to the value of the label. SINE, when the two segments are linked together.

5.4 DIRECTIVES THAT ASSIGN NAMES, VALUES AND LABELS

The following paragraphs discuss the Name Flag Bit Address, Name CRU Bit Address, Assign Value to Symbol, and Format a Source Language Extension assembler directives.

5.4.1 NAME FLAG BIT ADDRESS (FLAG). The general form of the FLAG directive is:

FLAG OPERAND$_1$,OPERAND$_2$, . . . ,OPERAND$_n$

This directive allows naming the flag bit addresses in a flag segment (FSEG).

Flag addresses are assigned sequentially to the previous symbols appearing in the operand list. The flag base register must be set to the address of the first word to be used for flags. The Current Flag Counter is started at bit 0 and is maintained modulo 16.

Modulo 16 means with respect to a modulus of 16. A modulus is an integer (x, for example) whose relationship to two other integers (y and z, for example) is such that y-z divided by x is a whole number. In other words, if counting modulo 16, count from 0 to 15, then go back to 0. The number 0 follows 15 in endless sequence. For example, the number 7 is equal to 23 modulo 16 and is also equal to 55 modulo 16.

Each time the counter passes through zero, the Flag Word Address Counter is incremented by one, thus addressing the next word. For example:

```
XLABEL FSEG
       FLAG F1, F2, F3, 1, F4, F5,3, F6
       FLAG   2, F7, F8, 3, F9
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
WORD 1	F1	F2	F3		F4	F5				F6			F7	F8		

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
WORD 2		F9														

Terms encountered in the operand list are added to the current location counter to advance the flag address accordingly. A constant or a symbol representing a constant appearing in the operand list specifies the number of flags to be skipped.

The FLAG directive can be used only in the flag segment. FLAG directives do not allocate memory and are used as a convenient method for generating flag addresses.

5.4.2 NAME CRU BIT ADDRESS (CON). The general form of the CON directive is:

LABEL CON OPERAND,MODIFIER

This directive is used in the CRU symbolic address segment to name the address of a CRU bit or the address of a series of CRU bits. The modifier is optional and has a default value of 1. The following example illustrates the use of CON:

```
COMM     BSEG    0
TAPE1    CON     18
TAPE2    CON     19,3
TAPE3    CON     20
TREG     CON     18
         END
```

Line 2 of the example assigns the name TAPE1 to bit location 18_{10} of the CRU symbolic address program segment. Line 5 assigns a different name, TREG, to the same location. Line 3 assigns a name to bits 19, 20, and 21. These three bits may be referenced as a communication register. Line 4 assigns a label to bit 20. This bit is contained within the TAPE2 register, but may also be addressed as TAPE3. A sequence of bits defined as a communication register with the CON directive (TAPE2 in the examples) may be addressed as a register by the LDCR and STCR instructions.

A second example gives the identical bit addresses as the first:

```
COMM     BSEG    400
TAPE1    CON     418
TAPE2    CON     419,3
TAPE3    CON     420
TREG     CON     418
         END
```

The bit addresses are the same because the difference between the BSEG operand and the CON operand determines the CRU bit address. This displacement value is limited by the number of bits in the instruction address field to a minimum of 0 and a maximum of 1023. Addressing the CRU requires the use of base register 7.

Note that the CON directive does not initialize any bits. CON is used only in a CRU symbolic address segment (BSEG).

5.4.3 **ASSIGN VALUE TO SYMBOL (EQU).** The general form of the EQU directive is:

```
LABEL   EQU   OPERAND
```

The EQU assignment directive assigns the value and attributes of the expression in the operand field to the symbol appearing in the label field. Among other uses, this statement can be used to assign a name to a register.

The expression in the operand field can be relocatable or absolute, and the symbol is defined accordingly. Any symbols in the expression must not be external.

The expression may contain symbols whose values are determined later in the assembly. Examples of valid and invalid uses of forward referenced symbols in EQU statements follow:

```
A   EQU   B+2
      .
      .
      .
B   EQU   3
```

A and B are labels of legal EQU statements and the above results in the values 5 and 3 being assigned to A and B respectively.

```
C    EQU    D
     .
     .
     .
D    EQU    E+2
     .
     .
     .
E    EQU    1
```

In the above example, D is assigned the value 3 and E the value 1. However, C is the label of an invalid EQU statement and will result in an undefined symbol error.

```
F    EQU    4
     .
     .
     .
G    EQU    H+3
     .
     .
     .
H    EQU    F
```

The above 3 EQU statements are valid. The values 4, 7, and 4 are assigned to the symbols F, G and H respectively. The label of a forward referenced EQU may not be externally defined. This usage generates an ILLEGAL DEF error.

The EQU directive is the usual way of equating symbols to register numbers, input/output unit numbers, immediate data, actual addresses, and other arbitrary values. The following example shows a series of EQU directives:

```
REGX      EQU    2       REGISTER X (RFGX) IS GIVEN THE
                         VALUE 2.
IO125     EQU    125     INPUT/OUTPUT DATA
TEST      EQU    X'3F'   IMMEDIATE DATA. TEST IS GIVEN THE
                         VALUE OF 3F (HEX).
TIMER     EQU    80      ACTUAL ADDRESS. TIMER IS EQUATED
                         TO DECIMAL 80.
LP1       EQU    $       LP1 IS ASSIGNED THE PRESENT LOCA-
                         TION COUNTER VALUE.
```

To reduce programming time, symbols can be equaled to frequently used compound expressions and then symbols used as operands in place of the expressions. The following example illustrates this use of EQU:

```
FIELD   EQU   ALPHA-BETA+GAMMA
```

FIELD is defined as ALPHA-BETA+GAMMA and can be used in place of it. Note, however, that ALPHA, BETA, and GAMMA must not be external references.

5.4.4 FORMAT A SOURCE LANGUAGE EXTENSION (FRM). The general form of the FRM directive is:

LABL FRM $OPERAND_1,OPERAND_2,\ldots,OPERAND_n$

FRM assigns the label as an operation code. The label symbol cannot exceed four characters in length. The expression values in the operand field are positive integers and their sum must be 16 or 32. When the label is used as an operation code, the n fields are evaluated, truncated without reference to sign to the length specified by the corresponding $OPERAND_n$ and placed beginning in the same relative bit location as the FRM directive list. New instructions and data structures can be designed using the FRM directive. An FRM defined operation mnemonic overrides any SAL machine instruction or directive mnemonic. The label assigned to an FRM directive may not be an existing directive mnemonic or instruction mnemonic.

An instruction defined by a format statement must not precede the FRM statement. If it does, a pass 1 error occurs although the statement is correctly assembled.

Field widths in bits are listed in the operand field, as shown in these examples:

```
XLAB FRM 4,2,10,5,5,6          Format declaration
WORD XLAB 4,1,103,26,9,15      Format reference
```

In memory, the above example reference would appear in two words as:

0–3	4–5	6–15	16–20	21–25	26–31
0100	01	0001100111	11010	01001	001111

MEMORY

The option is available to include a two-entry, parenthetic sublist in the FRM operand, such as:

```
XLAB FRM 6,3,7(X'FF',X'0')
     XLAB 15,4,23
```

Should this option be exercised, a logical AND is performed between the first sublist entry and the final version of the formatted word (or double word) and logical OR performed between the second sublist entry and the formatted word (or double word). When the sublist is used, the number of binary bits (in the example, 16) required to represent each number of the sublist must not exceed the number of bits specified by the field width operands. Should the sublist number require fewer than the specified number of bits, the number is right-justified in a field with leading zeros. In memory, the formatted word above would appear as:

0			15
0000	0000	0001	0111

MEMORY

The number of bits specified in the operand list must equal either 16 or 32. Furthermore, when the 32-bit format is used, a field cannot be defined that extends across the internal word boundary. An entry in the label field is required. After XLAB has been defined, it is referenced by entering its label symbol in the operation code field. For example:

```
XLAB FRM 4,4,8
ALPHA XLAB 12,6,21
BRAVO XLAB 13,5,20
TEST FRM 8,8,16(X'5FFEFFFF',X'09F')
BAKER TEST 22,44,66
CAT TEST -20,3+18,32760
```

5.5 DIRECTIVES THAT RESERVE OR PLACE DATA IN MEMORY

The following paragraphs discuss the Reserve Memory and Place Data in Memory assembler directives.

5.5.1 RESERVE MEMORY (RES). The general form of the RES directive is:

```
LABEL  RES  OPERAND
```

This statement is used to reserve word locations in memory. The term in the operand field entry is added algebraically to the contents of the current location counter. An entry in the label field is optional. An example of RES:

```
XLABEL  RES  10
```

In this example, the operand 10 specifies that ten consecutive words will be reserved in memory. The label XLABEL is the address of the first word reserved. Subsequent words in this reserved area can be addressed using XLABEL and indexed by the proper integer (1 to 9). Use of a relocatable expression, external reference, or forward-defined symbol (which means that the symbol has not been defined previous to this statement) in the operand field is an error.

5.5.2 PLACE DATA IN MEMORY (DATA). The general form of the DATA directive is:

LABEL DATA $OPERAND_1, OPERAND_2, \ldots, OPERAND_n$

This directive is used to place specific values in memory. Values specified in the operand list are entered in adjacent memory locations. Three types of data are permitted: hexadecimal integer strings, ASCII character strings, and valid expressions. A decimal integer list might appear as:

```
DATA  529,-3,65
```

A hexadecimal list might appear as:

```
DATA  X'ABC0',X'A',X'3F10',X'12345'
```

Note that for hexadecimal numbers, each entry is preceded by X and is enclosed by apostrophes. Decimal numbers cannot require more than 16 binary bits for internal representation and hexadecimal numbers more than 32. If a number (such as X'A') requires fewer than 16 bits, the number is right-justified internally and the remainder of the field is filled with leading zeros. In both decimal and hexadecimal, the number's value (V) must fall within the range $-2^{15} \leqslant V \leqslant 2^{15}-1$. For two-word hexadecimal entries, the range must be $-2^{31} \leqslant V \leqslant 2^{31}-1$.

An example of an ASCII list follows:

```
DATA  C'TEXAS INSTRUMENTS',C'SAMPLE'
```

Note that the data is enclosed in apostrophes and is preceded by C.

The first constant, TEXAS INSTRUMENTS, requires nine storage words since one memory word can store two ASCII characters. There are 17 characters in the constant including the space between words. SAL left-justifies ASCII characters and fills the right half of the last word with a blank if an odd number of characters is specified. Two consecutive apostrophes must be used to represent an apostrophe character within an ASCII list. The DATA directive can be used in the procedure, data, and flag segments.

The following two constructions are allowed in SAL, where MDAT is a relocatable address and @MDAT is a segment relative address.

```
DATA  MDAT
DATA  @MDAT
```

5.6 DIRECTIVES THAT CONTROL ASSEMBLER OUTPUT

The following paragraphs discuss the Page Eject and Program Identification assembler directives.

5.6.1 PAGE EJECT (PAGE). The general form of the PAGE directive is:

```
PAGE
```

This directive causes the listing output device to be advanced to the top-of-form position. The directive itself, PAGE, is printed before the page is ejected.

5.6.2 PROGRAM IDENTIFICATION (TITL). The general form of the TITL directive is:

```
TITL  OPERAND
```

This directive is used to specify the ASCII characters to be used in program identification. These characters are printed on the first line of each page of the list generated by the assembler. The heading occurs at assembly time, and not at run time. The following example illustrates how TITL is used:

```
TITL TI 960 MONITOR SYSTEM
```

TITL causes the operand to be stored in a title buffer; the title is printed at the top of a new page or after a page eject. If another TITL directive with a different title as its operand is encountered, the new title is printed at the top of subsequent pages. However, the first title is used in the symbol and segment table listing.

5.6.3 UNLIST DIRECTIVE (UNL). This directive disc continues a list output in progress on LUNO 6 until a LIST directive is encountered. Lines with errors are listed when a UNL is in effect. This directive does not override any input option.

5.6.4 LIST DIRECTIVE (LIS). This directive cancels a previous UNLIST directive and resumes the interrupted list output on LUNO 6. This directive does not override any input option.

SECTION VI

PROGRAMMING TECHNIQUES

6.1 GENERAL

This section is intended primarily for the user new to 960 assembly language programming. The purpose of this section is to show some commonly used SAL techniques. Consider the following introductory remarks.

- Annotate programs thoroughly. Use more comments with assembly language than when working with a higher-level language. A line of comments per instruction should be considered minimum and sometimes even this is not sufficient.
- Use all available formatting aids, e.g., the TITL option and the fact that the SAL assemblers pass blank cards can be used to delineate logical groupings.
- Do not use free-form coding. By starting operations and operands in the same field each time on every card, program listings are easier to debug and more readable.

6.2 SAVING REGISTERS

Since the 960 instruction set does not have multiple load and save instructions, saving registers upon entry to a routine and the restoration at the end of the routine the iteration has to be done explicitly.

Example:

```
SUBR    ST      0,SAV0+1
        ST      1,SAV1+1
        ST      2,SAV2+1
        .
        .
        .
        ST      7,SAV7+1
        (body of routine)
        . . .
        (return sequence)

SAV0    LA      0,$-$          Store area in second
SAV1    LA      1,$-$          word of instruction
SAV2    LA      2,$-$
        .
        .
        .
SAV7    LA      7,$-$
```

NOTE

$-$ assembles as a zero and is conventionally used to designate a location to be modified.

6.3 MOVE OPERATIONS

The 960 instruction set includes a memory-to-memory move instruction. This provides the user with a faster, easier method of moving data from one location in memory to another. Instead of using load and store instructions thus:

```
L       1,HERE
ST      1,THERE
```

one instruction is all that is needed:

```
MOV     HERE,THERE
```

NOTE

HERE and THERE must be within 1024 words of the beginning of the segment in which they are each defined (however, they can be separate segments) and the appropriate base registers must point to the top of the respective segments.

6.4 ZEROING MEMORY

Memory words can be cleared by shifting left to zero all bits, to save a line of source code and a word of storage:

```
        MLA     0,WORD          16 bit shift
```

instead of:

```
        MOV     ZERO,WORD
        .
        .
        .
ZERO    DATA    0
```

However, it should be noted that, for time-critical applications, the latter is faster by approximately 3 microseconds.

6.5 SHIFTING DATA

The 960 shift instructions all shift memory, not registers. However, to shift worker register 0 left 4 bits, the following example could be used.

Example:

```
        MLA     4,WRO
        .
        .
        .
WRO     EQU     X'88'
```

instead of

```
        ST      0,HERE
        MLA     4,HERE
        L       0,HERE
```

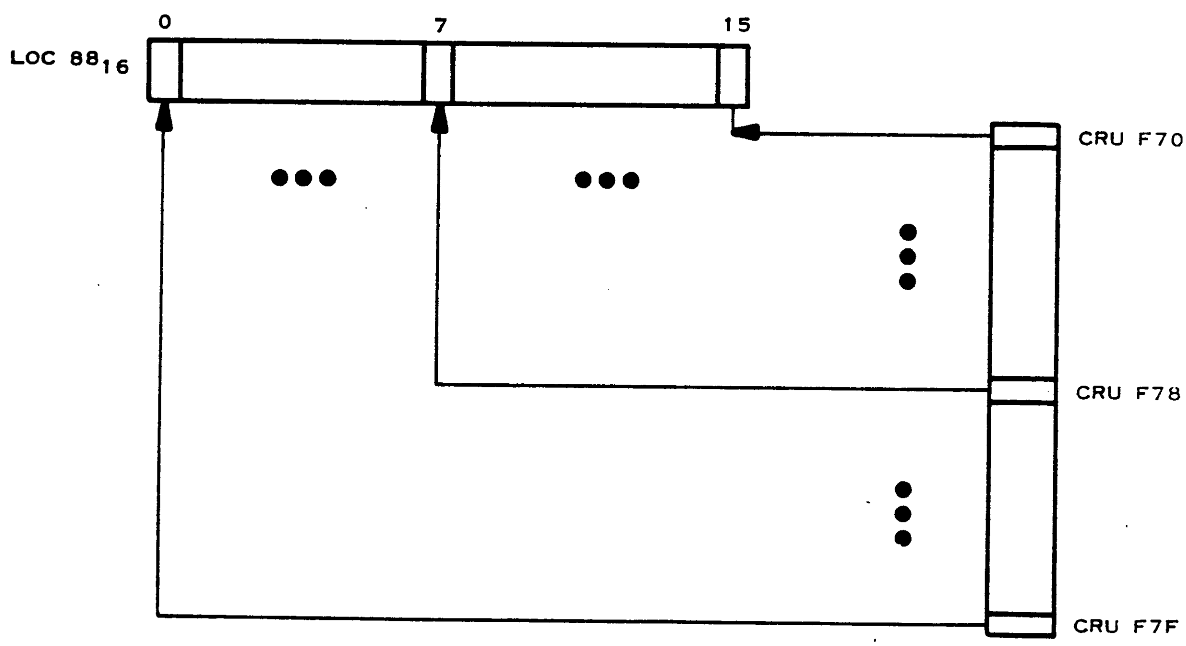
6.6 CRU LOAD AND STORE EXAMPLE

Although the CRU instructions are described in section IV, the function is sufficiently peculiar to warrant illustration here.

Consider the example:

LA	7,X'F70'	F70 is address of module on CRU
LA	4,X'88'	data to be stored in worker register 0
STCR	(0,0),0	

The STCR instruction causes one word of data to be read from CRU lines 'F70' to 'F7F' to the (scratchpad) word in memory location 88_{16}. However, if line 'F70' is a one, and 'F71' to 'F7F' are zeros, the scratch pad contains the bit pattern '0001' and if CRU line 'F7F' is a one and 'F70' to 'F7E' are zeros then the scratch pad word contains '8000'. This happens because the CRU is physically organized from right to left, unlike main memory. The LDCR instruction is the inverse of the STCR Instruction. In the example below, data flows in the opposite direction. Memory bit 15 transfers to CRU bit F70, and memory bit 0 transfers to CRU bit F7F.

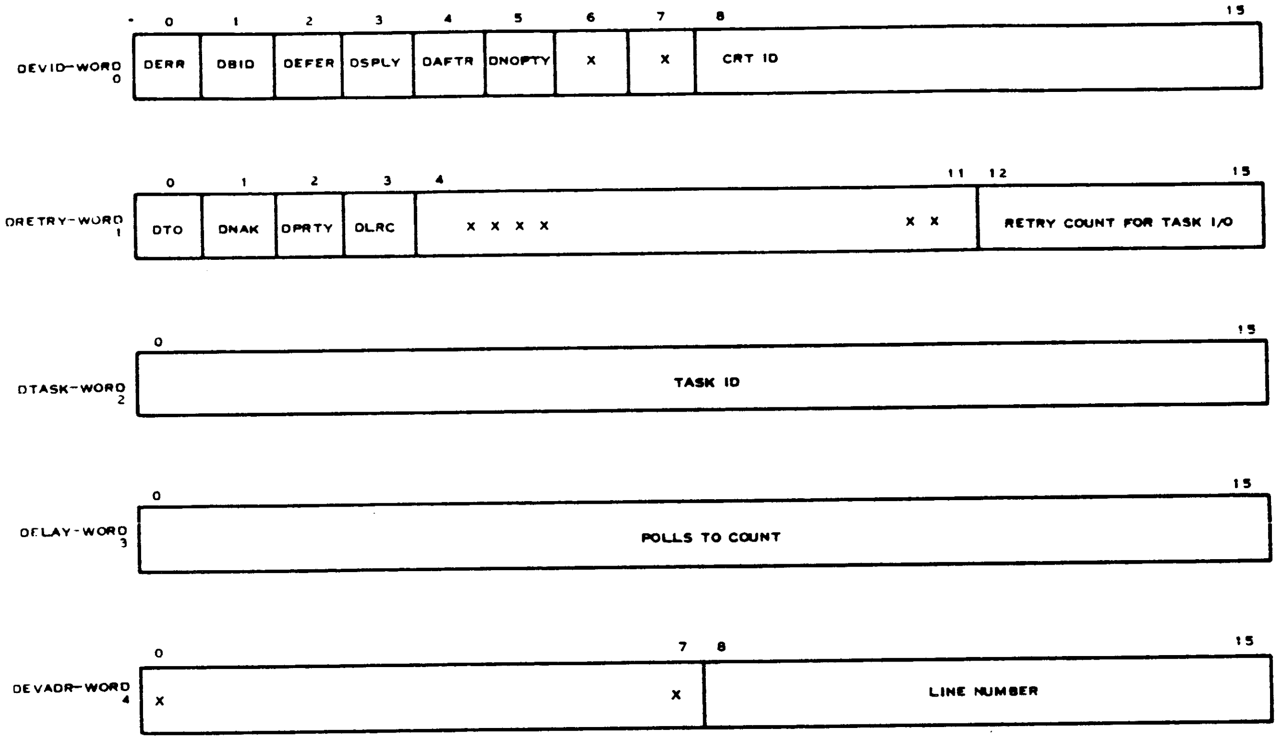


(A)131861A

6.7 LABELING CONTROL BLOCKS

The FLAG directive is used to label a template for a control block and can only be used in an FSEG (Flag Segment). Very often it is desired to intersperse bits (flags), groups of bits (unlabeled), and words in a block of data (control block). This is illustrated in the following example by mixing FLAG and EQU directives in an FSEG.

Consider the following data configuration:



Sample Control Block Layout

```
01DC     DCBSEG FSEG DCB (DEVICE CONTROL BLOCK)
0000     DEVID  EQU  0
0000            FLAG DERR          CRT I/O ERROR
0001            FLAG DBID          TASK BID
0002            FLAG DEFER         TASK DEFERS EXECUTION
0003            FLAG DSPLY         TASK REQUIRES OUTPUT
0004            FLAG DAFTR         POST TASK AFTER OUTPUT
0005            FLAG DNOPTY        DO NOT GENERATE PARITY OR LRC
         *      BITS 8-15          CRT ID
0006            FLAG 10
0001     DRETRY EQU  1
         *      BITS 0-3           I/O ERROR FLAGS
0010            FLAG DTO           DEVICE TIME OUT
0011            FLAG DNAK          OUTPUT ERROR
0012            FLAG DPRTY         INPUT PARITY ERROR
0013            FLAG DLRC          INPUT LRC ERROR
         *      BITS 12-15         RETRY COUNT FOR TASK I/O
0014            FLAG 12
0002     DTASK  EQU  2             TASK ID
0003     DELAY  EQU  3             POLLS TO COUNT BEFORE EXECUTING
         *                         DEFERRED TASK
0004     DEVADR EQU  4             DEVICE SYMBOLIC ADDRESS IN ASCII
         *      BITS 0-7
         *      BITS 8-15          LINE NO. (A-I)
         *      BITS 16-31         DEVICE ID  (00-15)
0006     DCBWK  EQU  6             10 WORD WORK AREA
0010     DCBLN  EQU  16
01DC            END
```

In this example the FSEG is used to create a template for a 5-word block of data. The single bits can be referenced directly in Flag instructions. The entire words can be referred to. The groups of bits can only be referred to as words with the unused bits masked off.

As an example of the use of this template in referring to entire words, consider a 5-word block in memory labeled DCB. It is desired to load word 3 (DELAY) into register 1:

```
LA      2,DCB          Set base register 2 to beginning of block
L       1,DELAY,2      Load register 1
```

Other words can be referred to in the same manner. It should be noted that an EQU directive in an FSEG is the same as in any other segment. In this case the EQUs are in the FSEG for clarity in defining the template.

6.8 COMPARISON CODE

It should be noted that for comparison tests (2-way) against an immediate value the structure of the CRA instruction is more useful than the CMI instruction.

Example:

```
        L       1,VALUE
        CRA     1,X'20'
        BC      EQ,BLANK
        CRA     1,X'41'
        BC      EQ,LETTER
```

Instead of:

```
        CMI     VALUE,X'20'
        B       NEXT
        B       NEXT
        B       BLANK
NEXT    CMI     VALUE,X'41'
        B       AGAIN
        B       AGAIN
        B       LETTER
AGAIN   EQU     $
```

6.9 "MVC" LOOP AND GENERAL ITERATIONS

Long strings of data (ASCII characters) can be moved from location to location within memory through the use of the following type of code.

Example:

```
        LA      1,HERE         Load pointing registers
        LA      2,THERE
        LA      3,-100         Move 200 characters
MOVE    MOV     (0,1),(0,2)
        AA      1,1            Augment pointer
        AA      2,1
        ARB     1,MOVE,3       Step and test index register
```

Very obviously, other types of iterated loops could be implemented similarly through the use of the ARB instruction.

6.10 SAL I/O OUTPUT

I/O operations are typically operating System dependent. Since this manual is designed to be used with 960 systems under the operating control of PSM, PAM, or PAM/D, it is not the place for much discussion of I/O. However, a fairly typical example of console output (under PSM, PAM or PAM/D) is included in the following example.

Example:

```
DATASP      DSEG
             .
             .
             .
MSAG        DATA      C'MESSAGE'
MSAGWC      EQU       $-MSAG

MSGPRB      DATA      DATASP              PAM EXAMPLE
            DATA      MSAG
            DATA      0
            DATA      MSAGWC
            DATA      X'00'
             .
             .
             .
            END

PROC        PSEG
             .
             .
             .
            LA        3,@MSGPRB
            SXBS      *SVC
             .
             .
             .
SVC         EQU       X'7F'

            END
```

6.11 PRE-INDEXING AND POST-INDEXING DESCRIPTION

The *B instruction (*BC and *BL also) always uses the pre-indexing mode, regardless of the contents of status register bit 6. This is helpful in programs where the post-indexing mode is generally desired (status register bit 6 set), but the pre-indexing mode is wanted for some branch instructions.

The example below contains a table addressed by POINTR and contains a data word followed by a branch vector. The instructions given compare each value in the table (beginning with the end of the table and working up) with the contents of VALUE. If the words compare, a branch is taken to the address given in the word following the value. The comparison instruction uses the indexing mode defined by the status register, but the branch will always be pre-indexed. For the example the status register is assumed to contain 8200_{16} (post-indexing and worker mode).

```
EQ          EQU     11
            LA      1,END-2-POINTR       Compute table size
            L       2,VALUE              Value to be compared
COMP        CRL     2,*POINTR,1          Compare with next value
            *BC     EQ,POINTR+1,1        Branch to address given
            ARB     -2,COMP,1            Decrement index and retry
            .                            value not found
            .
            .
POINTR      DATA    VALUE1,ENTRY1        For post-indexing, the
            DATA    VALUE2,ENTRY2        contents of POINTR,
            DATA    VALUE3,ENTRY3        VALUE1 is used as the
                                         address by which register
                                         1 indexes to get the actual
                                         value for comparison.
            .                            Branch table
            .
            .
            DATA    VALUEn,ENTRYn
END         EQU     $
            .
            .
            .
ENTRY1      EQU     $                    ROUTINE #1 entry pnt.
            .
            .
            .
            .
```

6.12 COMMON SUBROUTINES

The following subroutines use the same set of general registers that the calling subroutine uses. The BL instruction branches to a common subroutine and stores the address of the BL instruction in the linking register designated by the r field of the machine instruction. The subroutine can return control to the calling routine following the BL instruction by executing the statement:

B 2,R

where the r register contains the address of the BL instructions upon the time of entry to this subroutine. The SSB instruction branches to a common subroutine and stores the address of SSB instruction +2 at the effective address, stores the status at effective address +1, and branches to the address specified by the effective address +2. One advantage of the SSB over the BL instruction is that registers are not affected. The return call to the calling routine at the instruction after SSB is an LDS instruction. A disadvantage, however, is that reentrant code can not contain an SSB instruction. A common subroutine can use other branch instructions as appropriate to transfer control to other points in the calling routine or in other subroutines.

The following examples assume that program and subroutines are assembled separately and link edited. When assembly at the same time the REF statements are not required. An example of the BL instruction is:

Calling Program

```
P               PSEG
                REF       SQRT
R3              EQU       3
                 .
                 .
                 .
                ST        0,NUMBER
                BL        R3,SQRT
NUMBER          DATA      0
*SUBROUTINE RETURNS HERE
                 .
                 .
                 .
                END
```

Subroutine

```
SQRT            PSEG
R3              EQU       3
                ST        R3,SAVE3
                 .
                 .
                 .
RETURN          LA        R3,SAVE3
SAVE3           EQU       $-1
                B         2,3 RETURN
                END
```

An example of the SSB instruction is:

Calling Program

```
P               PSEG
                REF       SQRT
                 .
                 .
                 .
                SSB       SQRT
*SUBROUTINE RETURNS HERE
                 .
                 .
                 .
                END
```

Subroutine

```
SQRT      PSEG
          DATA      0,0,$+1
          .
          .
          .
          LDS       SQRT            RETURN
          END
```

6.13 PROGRAM MODULES

Since the assembler includes directives that generate the information required to link program modules, it is not necessary to assemble an entire program during the same assembly. The link editor links modules into one large program module satisfying all external symbol references if the conditions described in the following paragraphs are met. These paragraphs define the linking information that must be included in all program modules.

A program module can contain one or more program segments. A long program can be divided into separately assembled modules to avoid a long assembly or to reduce the size of the symbol table.

6.13.1 EXTERNAL REFERENCE DIRECTIVE. Each symbol defined in some other program module (in a separate assembly) must be placed in the operand field of an REF directive in the program module that requires the symbol.

6.13.2 EXTERNAL DEFINITION DIRECTIVE. Each symbol defined in a program module and required by another program module must be placed in the operand field of a DEF directive. The label of a PSEG directive and all symbols of data, flag, or CRU symbolic address segment are automatically defined as external. Therefore, placing any of these symbols in a DEF symbol string is an invalid procedure and causes an error condition.

6.13.3 LINKING PROGRAM MODULES. When program segments are linked, the link editor builds a list of symbols from the externally defined symbols (either by the DEF directive or automatically) during pass 1. Each program segment bias (the starting address of the segment relative to the start of the program) is incremented by the segment lengths of the previous segments. Each relocatable external symbol definition is incremented by its defining segment bias. During pass 2, the reference data (a symbol string in a REF directive) is resolved and new text data is generated. The output is then one larger loadable, linkable (only if linked with LNK960 and specifying certain symbols to remain external), and executable program module.

SECTION VII

SAL INPUTS AND OUTPUTS

7.1 GENERAL

This section discusses the assembler input data required and output data produced. Included is an explanation of the formats for the source listing, user's data input, and object output. Also included are a list of error codes and a list of input options.

7.1.1 SOURCE LISTING FORMAT. The SAL assembler prints a source listing that shows the symbol table, source statements and the resulting object code. Appendix F includes such a listing example. The symbol table headings are:

FLAG	The symbol is defined as a Flag
REF	"REFed" in program
REL	Relocatable
DEF	Define in program (not necessarily DEFed)
EXT	Externally defined (REFed or DEFed)
MUL	Multiply defined
ILL	Illegal
USED	Referenced in program

Each page of the source listing can have a title line at the top of the page (when supplied by TITL directive). Any title supplied by a TITL directive is printed in this line, and a page number is printed to the right of the title area. The printer skips a line below the title line and prints a line for each source statement listed. The line for each source statement contains a source record number, a location counter value, object code assembled, and the source statement image.

7.1.1.1 Listing Fields. The listing line for a machine instruction source statement is shown in the following example:

```
0018 0156 14039416 MOV DATA1,DATA2
```

The source record number, 0018 in the example, is the first field in a source listing line and is a 4-digit decimal number. Source records are numbered in the order in which they are entered.

The next field on a line of the listing contains the hexadecimal location counter value, 0156 in the example. Not all directives affect the location counter, and those that do not affect the location counter leave this field blank. The EQU directive places the value corresponding to the label in the location code field.

The third field contains the hexadecimal representation of the object code generated by the assembler, 14039416 in this example. All machine instructions and the DATA directive use this field for object code.

The fourth field contains the first 60 characters of source statement as supplied to the assembler. The number of source statement characters read and printed may be altered by the Extend Printing Width option (X); see the list of options in paragraph 7.1.2.2. Spacing in this field is determined by the spacing in the source statement. The four fields of source statements are aligned in the listing only when they are aligned in the same character positions in the source statements.

The object code corresponds to the operands in the order in which they appear in the source statement. The DATA directive prints a line of code for each entry that takes one or two words of data. For example:

```
0005    0010    0003        DATA A,1,X'138F',C'ABCDE
        0011    0001
        0012    138F
        0013    41424344
        0015    4520
```

The first line in the example contains the source record number (0005 decimal), location counter (0010 hexadecimal), value of the symbol A, and the first 60 characters of the input record image. The second line contains the second entry (operand) value (0001) and the third contains the third entry value (138F hexadecimal). The fourth entry takes up more than two words of data; therefore, more than one data line is printed. The hexadecimal value (41424344) of the fourth line is the hexadecimal ASCII code object value of the characters ABCD. The next line is the hexadecimal ASCII code value of the character E and a blank fill (4520).

7.1.1.2 **Model 33 ASR Teletypewriter.** When object code is punched on the 33 ASR teletypewriter, the object code is printed as it is punched. Since the listing is being printed on the same device, lines of object code are printed between the lines of the source listing. Also, the 33 ASR prints 72-character columns to a line although SAL assumes 80. To avoid the inconvenience caused by these incompatibilities, the following actions are suggested:

1. Always generate the source tape starting with an !*Xƀ0072 record. This changes the printed line width to 72 columns and the input record size to 52 characters.

2. Assign logical unit number (LUNO) 4 to a keyboard/printer device, not to the dummy. Options can be easily changed.

3. Type !*RX for the option input at the start of execution of pass 1. This allows re-execution of pass 2. During execution of pass 1, the symbol and segment table are printed with the object listing interspersed between the symbol and segment tables. ID and linkage data is punched during pass 1; it is not repunched during each execution of pass 2. This data is not used by the loader, but is used by the linking editor.

4. Type !*L for the option input at the start of pass 2. A listing is printed along with any source errors but object data is not punched. The assembler then asks if re-execution of pass 2 is desired. The user requests re-execution if an output of object data is desired.

5. Type !*P for the option input to the second execution of pass 2. SAL then punches out object without an assembly listing. Pass 2 object is executable but not linkable. Re-execution of pass 2 may continue for as many copies of listings or text data as desired.

7.1.1.3 **Error Messages.** During pass 1, the assembler prints an error message on a separate line of the listing when it detects an error. The errrr message is printed out on the listing device. The total number of pass 1 errors is printed at the end of pass 1. When the assembler is accepting data, it prints messages such as:

SEGMENT TABLE EXCEEDED BY AAAAAA	(AAAAAA is a symbol)
*** ILLEGAL FRM *** BBBB	(BBBB is a mnemonic)

Table 7-1 contains a list of the pass 1 error messages.

During pass 2, the assembler prints a 1-character error code on the source line of the listing in front of the location counter number when it detects an error. An example of this type of error code is:

```
0005  S0010  14000000  MOV  (A,B,C),D
```

The letter S immediately in front of the number 0010 is the error code. The total number of errors from pass 1 and pass 2 is printed upon termination of pass 2. (Refer to the !*S option in paragraph 7.1.2.2 for information about suppressing the listing of error summaries.) Because some errors are pass 1, some are pass 2, and some are both pass 1 and pass 2, two errors per statement may be printed on the assembly listing. The pass 2 error codes and messages are listed in tables 7-2 and 7-3. The total error summary printed at the end of the assembly is the sum of both pass 1 and pass 2 errors.

An End-of-File object record is punched at the end of assembly regardless of Error Override option (EO), unless List Only option is specified.

7.1.2 **INPUT FORMAT.** The input format for source statements is constrained to four fields. A description of each of these fields and the limitations on the character column positions they can occupy is found in Section III.

The width of the listing is constrained to the printer width specified by an option or the 80-column default. Because the assembler uses the first 20 columns of a printout line for the record number, error code, assembly program counter number, and object, the remainder of the columns are used for the input record image. The input record size is therefore limited to the printer width minus 20 characters (or 60 characters as the default width). Up to sixty characters of input record are scanned by the assembler.

7.1.2.1 **Logical Unit Numbers.** The specific peripherals to be used for input and output are assigned by means of logical unit numbers (LUNOs). The LUNOs are assigned as follows:

LUNO	Description
4	Secondary option input
5	Source Input
6	Listing Output
7	Object Output
10	Sequential scratch file; dummy

Table 7-1. Pass 1 Error Messages

Messages	Explanation
SYMBOL TABLE EXCEEDED BY AAAAAA	Symbol table entry caused overflow; assembly aborts. AAAAAA is the symbol that caused the overflow.
SEGMENT TABLE EXCEEDED BY AAAAAA	Segment table entry caused overflow; assembly aborts.
** ILLEGAL INSTRUCTION ** BBBB	Assembler did not recognize mnemonic. BBBB is the mnemonic.
MISSING END	Assembler recognized a new segment, and the previous segment is missing an end card.
*** ILLEGAL FRM *** BBBB	Format was not 16 or 32 bits in width, or a label was used that is already a standard SAL label.
INSTRUCTION BETWEEN SEGMENTS	Instruction was not within a segment. The assembler aborts after detecting 25 consecutive instructions between segments.
MULTIPLE SYMBOL AAAAAA	Symbol was defined more than once.
ILLEGAL LABEL AAAAAA	Label had invalid syntax.
ILLEGAL DEF AAAAAA	Forward referenced label used in DEF directive.
READ ERROR TYPE R TO RETRY	An input read error occurred. Ready the last record read for another input if possible and type the letter R on the keyboard to retry. Any other legal character response terminates the assembler. May also occur in pass 2.
XXXX PASS ONE ERRORS	A total of XXXX (decimal) errors were found during pass 1. This message is printed upon termination of pass 1.
ASSEMBLER ABORTED – INVALID OPTION	The assembler received an invalid option. May also occur in pass 2.
ASSEMBLER ABORTED – NOT IN BACKGROUND	The assembler was not loaded into the background partition of PAM/D memory.
ASSEMBLER ABORTED	The assembler encountered an undefined LUNO, a memory parity error or an illegal instruction.

LUNO 10 is typically assigned to a dummy device under the Programming Support Monitor (PSM) or the Process Automation Monitor (PAM) and to a rewindable device under the Process Automation Monitor/Disc (PAM/D), unless the CI option (paragraph 7.1.2.2) is used. PSM, PAM and PAM/D are the monitors (i.e., executive systems) under which the assembler runs.

7.1.2.2 Options. The SAL assembler accepts certain options specified from the option input device (LUNO 4) and/or within the source input (LUNO 5). The format of an option record is:

```
!*ab
```

where the combination ab can be one of the options shown in table 7-4, which lists the options and their functions. An invalid option causes the assembler to abort.

If an option record is not input, the options LP are set by default. An option record is recognized anywhere in the input file by pass 1 and/or pass 2.

Table 7-2. Pass 2 Error Codes

Error Code	Type of Error
A	Address error – wrong symbol type used in instruction
D	Symbols defined more than once
E	Expression error
L	Illegal label
M	Illegal attempt to specify alternate mode registers
N	Illegal mnemonic
O	Undefined operand
P	Illegal procedure
S	Syntax error
T	Truncation error – value calculated was truncated to fit in operand field or overflow occurred
U	Undefined symbol
W	Warning

Table 7-3. Pass 2 Error Messages

Message	Explanation
READ ERROR TYPE R TO RETRY	An input read error occurred. Ready the last record read for another input if it is not running under PAM/D or the CI option, and type the letter R on the keyboard to retry. Any other legal character response terminates the assembler.
UNDEF/FWDREF SYM AAAAAA	A symbol was not defined in the assembly or a symbol used as an operand of an assembler directive (except EQU) was encountered before the symbol was defined.
XXX ERRORS : LENGTH = YYYY	A total of XXXX (decimal) errors from pass 1 and pass 2 were found. YYYY (hexadecimal) is the length of the created object module. This message is printed upon termination of pass 2.

The secondary option input device (LUNO 4) allows the user to enter an option at the start of pass 1 and an option at the start of pass 2 without having to modify the input source file (LUNO 5). If LUNO 4 is assigned to a keyboard/printer device, the following message is printed at the start of pass 1 and 2:

```
OPTION?
```

Type one option record followed by a carriage return. The above message follows every option record input. After all desired options have been input, type a carriage return in response to the message. If LUNO 4 option input is not desired, type a carriage return in response to the first "OPTION?".

Table 7-4. Assembler Input Options

Option ab	Pass 1 Function	Pass 2 Function
L	List symbol and segment table only.	List line number, error code, APC, object, and card image only.[1,2]
P	Punch the ID and LD records only.	Punch text, end-of-record, end-of-file records only.[2]
B	No printed or punched output.	List line number, error code, APC, and object only. Does not print card image.[1,3]
LP	List symbol and segment table, and punch ID and LD records.[3]	List line number, error code, APC, object, and card image and punch text records.[1,2,4]
BP	List symbol and segment table, and punch ID and LD records.[3]	List line number, error code, APC, object, and punch text, end-of-record, end-of-file records.[1]
D	List symbol and segment table, and punch ID and LD records.	List line number, error code, APC, object, and card image, and punch text, end-of-module, end-of-files. Suppress text records of all data segments.[1,5,8]
EP	List pass 1 errors and punch the ID and LD records.	List line number, error code, APC, object and card image where errors have occurred, and punch pass 1 object records.[1]
E	List pass 1 errors only.	List line numbers, error code, APC, object and card image where error has occurred.[1]
NM	Old bit addressing. (M field in instructions must be divided by 16).	Old bit addressing.
S	(Not applicable)	Suppress listing of total error summary.[6]
XbDDDD	Extend printing width on listing device.	Extend printing width on listing device. Outside the range $60 \leq DDDD \leq 136$, this option is ignored.
EA	Abort after listing symbol and segment tables if errors.	Abort on the first occurrence of an error.
EO	Do not stop punching ID and LD records on assembly error.[7]	Do not stop punching text on an assembly error.
RX	(Not applicable)	Allows reexecution of pass 2. The option of the previous pass is a default .[2]
T	Suppress listing of symbol and segment tables.	(Not applicable)

Table 7-4. Assembler Input Options (Continued)

Option ab	Pass 1 Function	Pass 2 Function
CI	Changes input LUNO of pass 2. If PSM or PAM, LUNO 5 source input is copied to device assigned to LUNO 10. (see table 7-5).	If PAM/D, the input is rewindable from LUNO 5 (a cassette is not considered a rewindable device and must be readied by user). If PSM or PAM, LUNO 10 is input (see table 7-5).
SP	Process the data, flag, and CRU symbolic address segments only.	(Same as pass 1 function).[5]
LL	Print undefined forward references.	(Not applicable).[9]

NOTES

1 APC means the memory address relative to the start of assembly, i.e., Assembly Program Counter.

2 L, P and LP are the only interchangeable options between passes.

3 B is used if the input is assigned to a teletypewriter keyboard and the source is read from paper tape by turning the reader on. In this case, the teletypewriter prints as it reads.

4 If no options are specified on the option record or no option record is found, the option LP sets.

5 This option is desirable for assembling re-entrant procedures and tasks.

6 The total error summary is always printed unless !*S is used.

7 Punching of text always terminates on the first occurrence of an error unless !*EO is used.

8 Must be before first DSEG.

9 The error count will always indicate zero errors. This does not mean that no errors were detected. The error count is used by the compiler as a flag to ignore the object output, so the user should verify all undefined forward references.

If LUNO 4 is assigned to other than a keyboard/printer device, the option input from LUNO 4 is terminated by an /* (End-of-File) record.

If the assembler is running under PSM or PAM, LUNO 4 is assigned to keyboard, and the !*CI option (Change Input option) is not specified, then the following message is printed at the start of pass 2 telling the user to ready the input file for pass 2:

```
READY INPUT
```

While the assembler is printing the symbol table, segment table and error summary, the user can ready the input file for pass 2 and, if LUNO 4 is assigned to the dummy, the assembler starts pass 2 without pausing to ask for an option or to tell the user to ready the input file.

Table 7-5. Assembler Input Options, Pass 1 or Pass 2

	Standard		CI Option	
	PSM/PAM	PAM/D	PSM/PAM	PAM/D
Pass 1	Input on LUNO 5	Input on LUNO 5	Input on LUNO 5	Input on LUNO 5
		Output on LUNO 10	Output on LUNO 10	Output on LUNO 10
Pass 2		Rewind LUNO 10	Rewind LUNO 10	Rewind LUNO 5
	Input on LUNO 5	Input on LUNO 10	Input on LUNO 10	Input on LUNO 5

7.2 LOADING AND EXECUTING SALM AND SALD

The assembler runs under one of three monitors, or executive systems. The monitors are PSM, PAM and PAM/D. SALD runs under PAM/D control in a 960 series computer with a memory size of at least 16K. It is the overlay version of the assembler and requires about 9.5K of background to run (assuming 100 symbols and 8 segments). SALM operates under PSM, PAM, or PAM/D control. It is the non-overlay version of the assembler and requires about 12K of memory, including a symbol table, to run. Under PAM/D, SAL must be a background task since it uses memory between itself and the end of the background for the symbol table area. Relative location 18 (12_{16}) under PSM or PAM contains the symbol table size (number of words). SAL uses a predetermined symbol table size of 558 words, for 100 symbols and 8 segments. SAL may be memory patched (relative location 12_{16}) after loading to alter the symbol table area to the desired size. If the relative location 12_{16} is changed to zero, all memory between SALM and the monitor is used for the symbol and segment table areas. The symbol table starts at the lower address memory and the segment table at higher address memory. The two tables work toward each other. The area needed for the symbol and segment table is zeroed at the start of execution. The symbol table size must be at least 5 words per symbol and 7 words per segment plus 2.

The peripheral devices to be used for inputs and outputs are selected by assigning logical unit numbers (LUNO's, described in paragraph 7.1.2.1).

At the start of execution, if insufficient memory is available for a symbol and segment table entry, SAL prints:

```
UNABLE TO ALLOCATE SUFFICIENT TABLE AREA
ASSEMBLY CANCELLED
```

The first input/output call of pass 1 and 2 (after LUNO 10 is rewound) is to LUNO 4. This message is printed or displayed:

```
OPTION?
```

If a single 33 ASR teletypewriter is used for source input, listing output and object output, then pass 2 must be executed twice using the RX option to obtain both the object and the listing. Selecting the LP option (Default option) results in spurious object data on the listing device. Normally the L option is selected first so that errors can be corrected and the program reassembled before any text object is punched using the P option. (Refer to paragraph 7.1.1.2 for a discussion of special considerations in the use of the 33 ASR teletypewriter.)

The user responds by supplying the code for the option desired (paragraph 7.1.2.2). Under PAM/D, all other inputs are from LUNO 5 for pass 1 and its image from the sequential scratch file, LUNO 10 (recorded during the execution of pass 1) for pass 2. Under PSM or PAM, all other inputs are from LUNO 5 for both pass 1 and pass 2. The !*CI option can be used to alter these inputs. Because the assembler ignores zero-length records, LUNO 4 may be assigned to a dummy device. In that way, assemblies are done without pausing for option inputs.

SAL object output is terminated upon the first occurrence of an assembly error (unless otherwise specified by means of the EO option). Under PAM/D the Skip On Condition Set monitor flag is set when an error occurs.

The following six conditions cause the assembler to abort.

1. A symbol or segment table entry caused an overflow.

2. When listing output is assigned to a data terminal, an ESC entered during table dumps terminates pass 1 and initiates pass 2. Entered during pass 2, the ESC terminates the Assembler.

3. An invalid option was input.

4. Twenty-five consecutive instructions between segments were input.

5. The assembler was not loaded in background (PAM/D).

6. An undefined LUNO, memory parity error, or internal interrupt occurred during the execution of the assembler.

In the first two cases, the total error summary is printed (if the option allows it) and the assembly terminates.

7.2.1 **LOADING UNDER PSM.** This paragraph gives the procedure for loading the non-overlay version of SAL (SALM) under PSM control. It is assumed that PSM has already been loaded and that the user has read the *Model 960 Computer Programming Support Monitor*, manual number 955380-9701.

1. Define all LUNOs necessary for SALM by using WDFIO (type D on console if WDFIO is monitor resident (it is in PSMALL)). Be sure that the LUNO for the alternate loader (LUNO 1) is assigned to the object input device if the alternate loader is monitor resident (in PSMALL it is).

2. Ready SALM object in the object input device.

3. Type the letter L on the console keyboard. PSM will now load SALM.

4. If a symbol table size other than 100 symbols and 8 segments is desired, specify the symbol table size by patching to the desired size memory location 16 (10_{16}) of SALM for versions preceding and including V4L2 and relative location 18 (12_{16}) for versions after V4L2.

5. Ready the source input in the input device for pass 1.

6. Start execution by typing an X on the console keyboard.

The procedure for executing SALM is given in paragraph 7.2.4.

An example follows:

D

OP?	DFIO	0001	0001	LUNO 1 to cassette (PSM alternate loader)
OP?	DFIO	0004	0000	Option input to console
OP?	DFIO	0005	0001	Source input to cassette
OP?	DFIO	0006	0000	Listing to console
OP?	DFIO	0007	0001	Object to cassette
OP?	DFIO	0010	.0002	Scratch file to dummy
OP?	^ -or ↑			Exit WDFIO
LX				Load and execute SALM

7.2.2 LOADING UNDER PAM. This paragraph describes the job control records needed to load SALM beginning at memory location 0100_{16} under PAM control. This job stream allows sequential loading and execution of SALM. The loading procedure for the non-overlay version of SAL is:

1. Load SALM into memory.

2. If memory between SALM and PAM for the symbol table is desired, patch the relative memory location as follows:

   For SALM version V4L2 or earlier, place a zero in memory location 16 (10_{16}).

3. Install and enable SALM as follows:

   For SALM versions later than V4L2, place a zero in memory location 18 (12_{16}).

4. Ready all input/output devices.

5. Execute SALM. The execution procedure is given in paragraph 7.2.4.

As an example, assume that LUNO's are assigned, with LUNO 10 assigned to the dummy, and that the !*CI option is not being used. The job control statements are:

```
$$LDTS**0100**0009        Load Task 09, SALM, at location 100
/*
[SALM object]
/*
$$PACH**
0112:b0000
/*
$$INST**0009**00E0        Install task 09 with priority EO
$$ABLE**0009              Able task 09
$$EXCT**0009              Execute Task 09
```

In the example, 0112:b0000 is a patch to tell SALM to use the memory between SALM end and the beginning of PAM for the symbol and segment table area.

7.2.3 LOADING UNDER PAM/D
Memory for both the symbol table and the segment table is allocated from the background memory remaining after the assembler is loaded in memory.

Pass 2 of SALD is filed as record one of overlay file 16_{16}. It is read immediately after pass 1 has completed execution.

The loading procedure for the overlay version of SAL is:

1. Load SALD pass 1 as a disc-resident task with a priority that makes it a background task.
2. Load SALD pass 2 as overlay file 16, record 1.
3. Make sure all the logical unit numbers are assigned to the appropriate devices, all appropriate devices readied, and assign background. The assembler is now ready for execution.

The procedure for executing SALD is given in paragraph 7.2.4.

The following example assumes that all LUNO's are assigned and background released:

$$LDDT**0009**00E0	Load Task 9, SALD, with E0 priority
/*	
[SALD pass 1 object]	
/*	
$$ABLE**0009	Able Task 9
$$LDOV**0016**0001	Load overlay file 16 record 01
/*	
[SALDOV pass 2 object]	
/*	
$$DFBG**3000**0050**00D0	Define background as 3000 words or 12K
$$EXCT**0009	Execute task 9, SALD
$$RLBG**	

The loading procedure for the non-overlay version of SAL is:

1. Load SALM as a disc-resident background task.
2. Assign logical unit numbers to devices, ready devices, and define background. SALM is now ready for execution.

The procedure for executing SALM is given in paragraph 7.2.4.

An example follows:

$$LDDT**0009**00E0	Load Task 9, SALM, with E0 priority
/*	
[SALM object]	
/*	
$$ABLE**0009	Able Task 9
$$DFIO**0004**0002	Assign option to dummy.
$$DFIO**0005**0001	Assign input to cassettes.
$$DFIO**0006**0011	Assign listing to DMAC line printer.
$$DFIO**0007**0001	Assign object output to cassette.

$$DFSF**0010**3000**3FFF**0001	Assign scratch to disc
$$DFBF**3100**0050**00D0	Define background.
$$EXCT**0009	Execute Task 9, SALM
$$RLBG**	Release background.

7.2.4 EXECUTING UNDER PSM, PAM AND PAM/D

Ready the source input before execution.

The procedure for executing both SALM and SALD is:

1. If LUNO 4 was assigned to a keyboard/printer terminal, SAL will print:

   OPTION?

   Respond by typing in one of the options listed in table 7-4 or by typing a carriage return.

2. The assembler starts pass 1 by reading from the input device. The symbol and segment tables and the pass 1 error summary are printed, and the ID and LD records are created. During the printing of the symbol table, the input can be readied for pass 2 (PSM/PAM without CI option).

3. At the start of pass 2 and with LUNO 4 assigned to a keyboard/printer, the assembler prints:

   READY INPUT (if CI option was not specified or not running under PAM/D)

   OPTION?

   NOTE

   When running under PSM or PAM and inputting source from cassette, paper tape or magnetic tape, assign LUNO 4 to a keyboard/printer device. Failure to make this assignment allows SAL to begin processing pass 2 before the operator can manually rewind the source media.

   Prepare the source input for pass 2 and enter options or a carriage return.

4. SAL continues with pass 2 execution by reading in the source and giving a listing, text object data and total error summary (if options allow).

5. If reexecution of pass 2 was specified by an option, steps 3 and 4 will be repeated.

7.3 ASSEMBLER RESTRICTIONS

Several restrictions affect the use of the SAL assembler:

1. Under PAM/D, the symbol and segment tables are built in a memory available in background. To be able to run an assembly of 100 symbols and 8 segments, a background of 2700_{16} for the overlay version of 3100_{16} for the nonoverlay version is necessary. For calculating a larger symbol table, a symbol table entry takes 5 words and a segment table entry takes 7 words.

2. The FORM table is fixed at 180 words. An FRM entry uses (W - 1)/4 + 8 words, where W is the number of field widths. For example.

   N FRM 1,b,8,1

   takes 8 words of memory, and

   NN FRM 1,1,3,8,3

   takes 9 words of memory.

3. There are restrictions on the source input format and the listing format. These are described in Section VII and paragraph 7.1.3.2.

   The input format is constrained to:

	Character Length		**Starting Column Number**	
Field Name	**Max**	**Min**	**Max**	**Min**
LABEL	6	1	1	1
OPERATION	4	1	19	2
OPERAND	56	1	21	4
COMMENT	-	0	-	5

   A blank must separate a label from the operation, the operation from the operand, and the operand from the comment. The maximums for comment field are not specified above because it is dynamic with the printer width.

   The listing width is constrained to the printer width specified by an option or 80 columns by default. Since the assembler uses the first 20 columns of printout for record number, error code, APC (Assembly Program Counter) and object, the rest of the columns are used for the input record image. Therefore, the input record size is limited to printer width size minus 20 characters or 60 characters as default. The input record size or 60 characters, whichever is less, is scanned by the assembler during the processing of each input record.

4. Following are the restrictions on the directives:

   a. PSEG, DSEG, FSEG, BSEG, EQU, CON and FRM must have a label.

   b. A FLAG directive is used within an FSEG only.

   c. REF, MODE and FRM are used within a PSEG or DSEG only.

   d. DEF is used within a PSEG only.

   e. DATA and RES are used within an FSEG, PSEG or DSEG only.

   f. A CON directive is used within a BSEG only.

7.4 OBJECT OUTPUT FORMAT. SAL outputs object records in the order shown in figure 7-1. A brief description of each record follows.

- The pass 1 records created are the identification record and the linkage data records for each program segment in succession, beginning with the first segment that appears and ending with the last segment that appears. The program segment's linkage data consists of external references and external definitions.

- The text records for all segments, an end record, and an end-of-file record are output in that order during pass 2.

7.4.1 OUTPUT RECORDS. Binary output records on paper tape are separated by an x-off (special control character) and four null characters. Successive text records are punched as required until the end of the segment is reached. A segment end record follows the last text record. It can obtain a transfer location (end vector) when it is specified on the last end record in the assembly.

All data on the object paper tape is punched two frames per 16-bit word. Figure 7-2 shows a punched binary paper tape and the memory contents corresponding to the punched hexadecimal data.

On cassette binary output records, each 16-bit data word is encoded as three 7-bit ASCII characters in a packed format. Bit 7 of each character is always set to logic 1 to provide better discrimination between data characters and the x-off (DC3) character (hexadecimal character code 13). Bit 6 in the first character of a 3-character coded object word is also always a logic 1 to aid in differentiation of object and source records. Bit 5 of the first character is the odd parity for all three 7-bit characters. The first four most significant bits of the object word fill bits 4 to 1 in the first character. The next six significant bits are packed in the second character. The six least significant bits are in the third character.

A coded object data word is shown in figure 7-3 as a sequence of three characters.

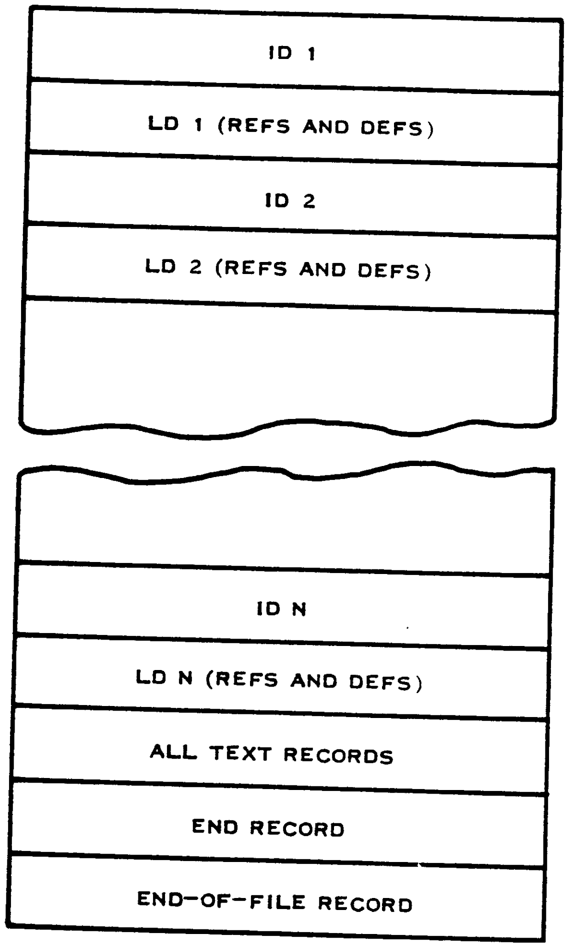
Both the cassette boot loaders and device service routine check for parity errors upon input.

Binary (object) records on cassette are ended by an x-off and at least one rub-out (DEL) character (hexadecimal character code 7F). The last record in an object file is followed by up to 85 rub-out characters to ensure purging the hardware cassette record buffer. Any number of rub-out characters are considered as an acceptable file separator or leader. Binary data is always recorded and played back on or from cassette with the Model 733 ASR keyboard/printer off.

On cards each column represents one 8-bit character that is decoded according to the list in table 7-6. For example, the binary card punch pattern for the character represented by CA_{16} is 12-11-8-2.

All binary formats have several common features. The binary record is indicated by a 17_{16} code in the first character. A 2-bit record indicator code is defined for the four different types of binary records in the following table:

Binary Code	Record Type
00	Identification (ID) record
11	Linkage Data (LD) or External Symbol record
10	Text record
01	End record



ID N – IDENTIFICATION RECORD FOR NTH PROGRAM SEGMENT
LD N – LINKAGE DATA RECORD FOR NTH PROGRAM SEGMENT
REFS – EXTERNAL REFERENCES
DEFS – EXTERNAL DEFINITIONS

(A)128936

NOTE: ID AND LD RECORDS ARE CREATED DURING PASS 1. TEXT, END-OF-FILE AND END RECORDS ARE CREATED DURING PASS 2.

Figure 7-1. SAL Object File Format

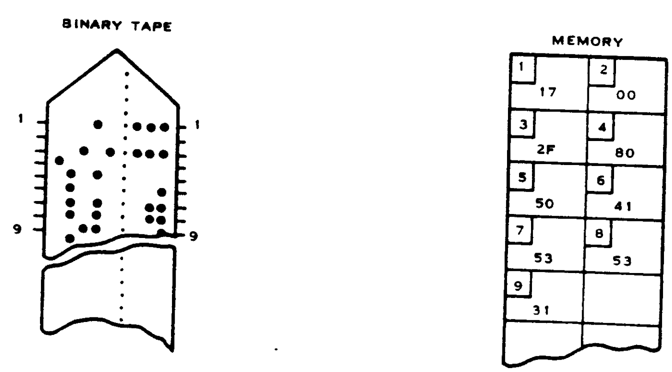
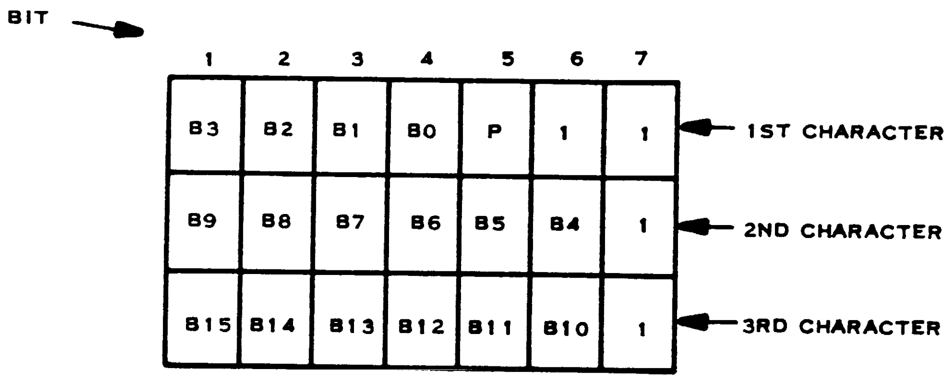


Figure 7-2. Hexadecimal Data Punched on Object Paper Tape



NOTE: P IS THE ODD-PARITY BIT FOR THE 21-BIT SEQUENCE. B0 IS THE FIRST (MOST SIGNIFICANT) BIT OF THE OBJECT DATA WORD.

(A)128938A

Figure 7-3. Coded Object Data Word

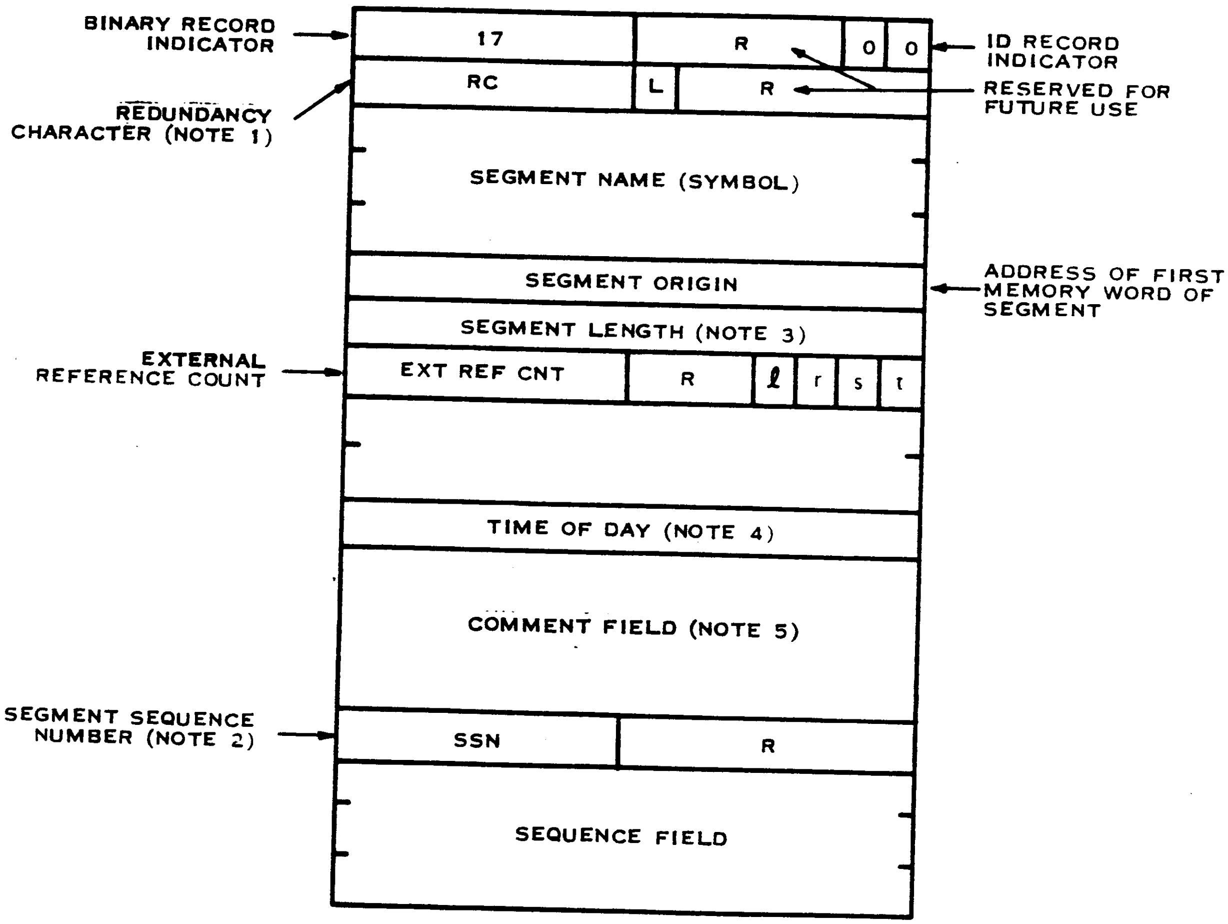
Table 7-6. Binary Internal Code to Binary Card Code Conversion

Most Significant Digit	Rows 12-11-0-9	Least Significant Digit	Rows 1 through 8
0	Blank	0	Blank
1	9	1	1
2	0	2	2
3	0-9	3	3
4	11	4	4
5	11-9	5	5
6	11-0	6	6
7	11-0-9	7	7
8	12	8	8
9	12-9	9	8-1
A	12-0	A	8-2
B	12-0-9	B	8-3
C	12-11	C	8-4
D	12-11-9	D	8-5
E	12-11-0	E	8-6
F	12-11-0-9	F	8-7

The redundancy character is the sum modulo 256 of all bits equal to one contained within the record excluding the redundancy character itself. The segment sequence number is increased by one for every new segment defined within any assembly containing multiple segments.

7.4.2 **OBJECT RECORD FORMATS.** Several of the records in the SAL object file have specific formats. The contents of these object records are included in this discussion: program or program segment identification record; linkage data (external symbol) record; text record; assembly end record; and end-of-file record.

1. The program or program segment identification (ID) record format is shown in figure 7-4. If the symbol in the Segment Name field has less than six characters, it is left-justified with trailing blanks. The two fields labeled R are reserved for future use.



NOTES:

1. THE REDUNCANCY CHARACTER IS THE SUM OF ALL BITS EQUAL TO ONE CONTAINED WITHIN THE RECORD EXCLUDING THE REDUNDANCY CHARACTER ITSELF.
2. THE SEGMENT SEQUENCE NUMBER IS INCREASED BY ONE FOR EVERY NEW SEGMENT DEFINED WITHIN ANY ASSEMBLY CONTAINING MULTIPLE SEGMENTS.
3. THE SEGMENT LENGTH IS THE VALUE OF THE SEGMENT RELATIVE PROGRAM LOCATION COUNTER WHEN THE SEGMENT IS TERMINATED.
4. THE NINE-WORD TIME OF DAY IS IN THE FOLLOWING ASCII-CODED BINARY FORMAT.

   HR:MN:MNTHDY,bYEARbb

   WHERE HR IS THE HOUR, MN IS THE MINUTE, MNTH IS THE MONTH, AND DY IS THE DAY OF THE MONTH. THE TIME OF DAY IS PROCESSED UNDER PAM/D ONLY.
5. A 34 CHARACTER COMMENT IS INSERTED IN WORDS 19-35 OF THE ID RECORD BY LIB960

(A)128939B

Figure 7-4. Program and Program Segment Identification Record Format

If the L bit equals 0, the program or program segment is processed by SAL; if equal to 1, the program or program segment is processed by LRL960 or LNK960. The four bits labeled ℓ, r, s and t have the following significance:

ℓ=0 Program segment has been linked.

ℓ=1 Linking is required. The text contains unsatisfied references.

r=0 Program segment is absolute.

r=1 Program segment is relocatable.

st=00 Procedure segment

st=01 Data segment

st=10 Flag segment

st=11 CRU symbolic address segment

2. Figure 7-5 shows the format for the linkage data (LD) record, also called the external symbol record. Fields labeled R are reserved. The first three bits of the SYMBOL 1 field have the following significance:

fb=0 Neither flag nor bit address.

fb=1 Either flag or bit address. The fb bit is set if the symbol is defined in a FLAG directive operand or is a CON directive statement label.

xr=0 The symbol is an external definition accompanied by a value.

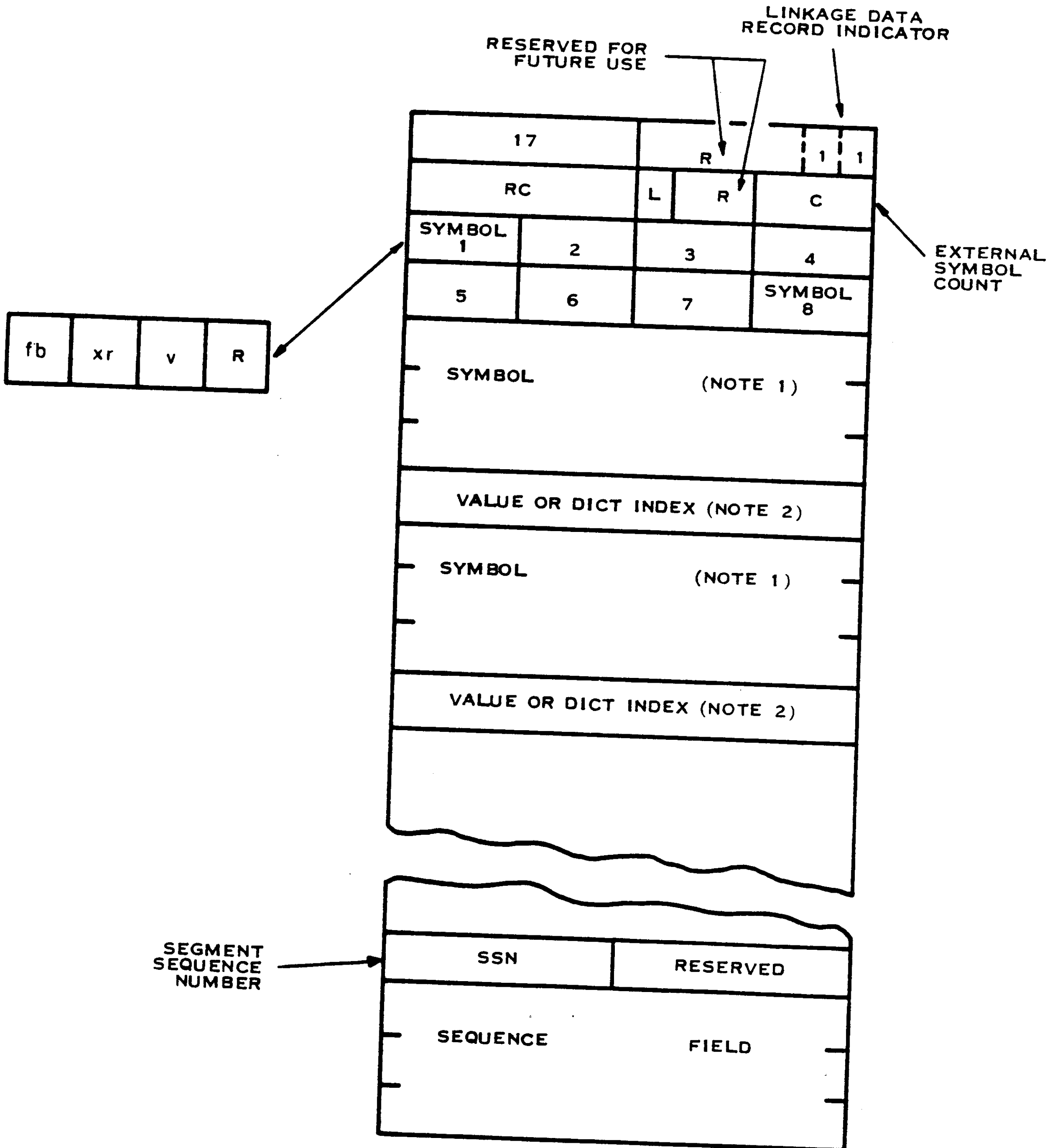
xr=1 The symbol is an external reference accompanied by a dictionary index.

v=0 The symbol is assigned a relocatable value.

v=1 The symbol is assigned a self-defining or relative value. The symbol is not relocatable.

3. The text record format is illustrated in figure 7-6. A bit in the second word of this record indicates whether the program is absolute (r=0) or relocatable (r=1). The R field is reserved.

The relocation map is contained in two words, RM 1 and RM 2. These words consist of bits (TW1, TW2, etc.) that specify the relocatability of the corresponding text words. Each map bit is set to 0 if the corresponding text word is not relocatable and 1 if the word is relocatable.

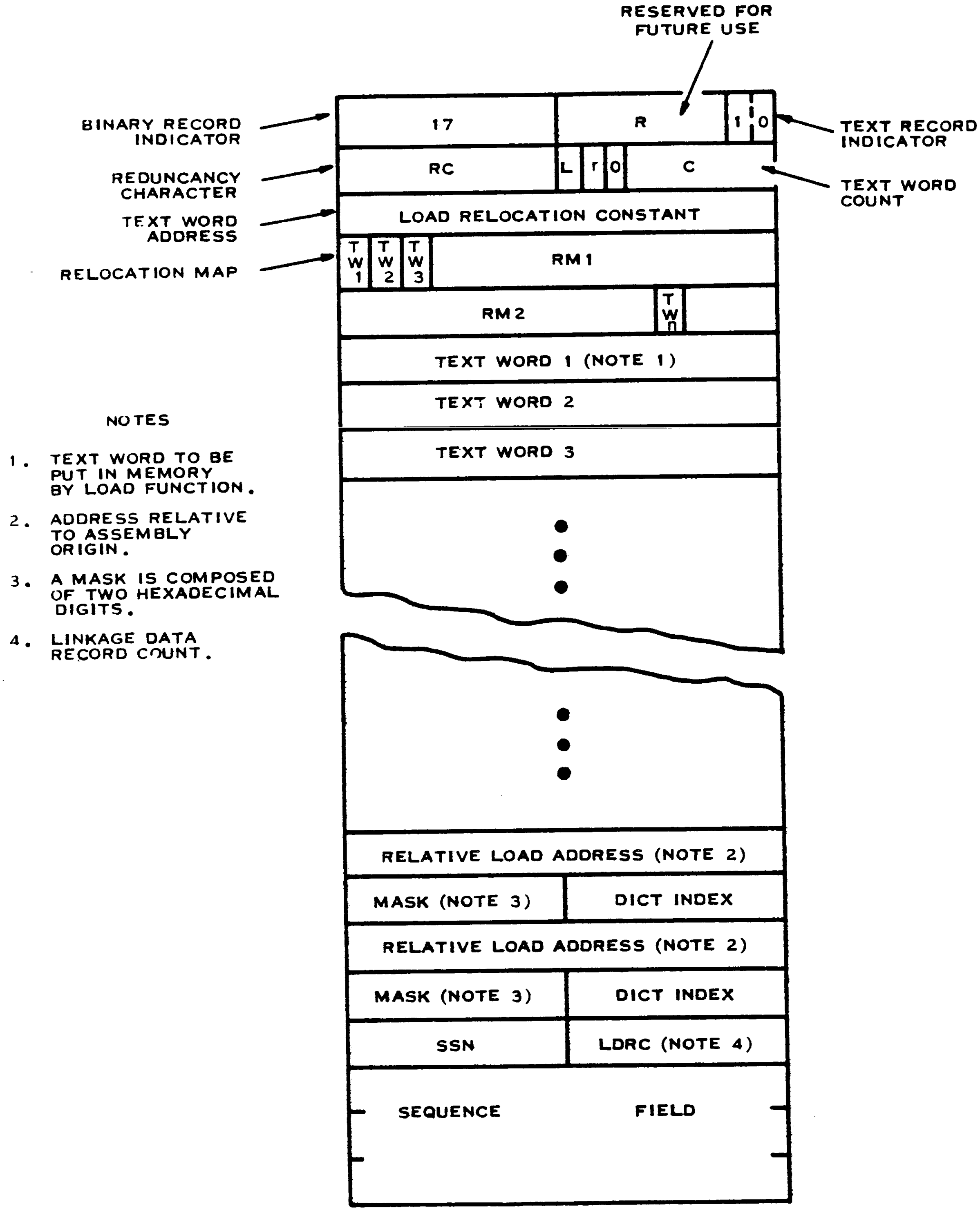


NOTES

1. A SYMBOL OF LESS THAN SIX CHARACTERS IS FILLED WITH TRAILING BLANKS.
2. "VALUE" IS THE VALUE ASSIGNED TO AN EXTERNALLY DEFINED SYMBOL. THE "DICTIONARY INDEX" IS AN INTEGER IN THE RANGE 1-255 THAT IS ASSIGNED TO AN EXTERNAL REFERENCE IN THE SEQUENCE IN WHICH IT IS DECLARED IN THE PROGRAM.

(A)128940A

Figure 7-5. Linkage Data Record Format



(A)128941

Figure 7-6. Text Record Format

Three fields are used for specifying the linkage information for an external symbol reference. These fields are Relative Load Address, Mask, and Dictionary Index. The Mask field consists of two hexadecimal digits, designated M(1) and M(2). M(1) is the starting bit position of a field. M(2) is the width of the field, with 0 used to represent a width of 16 bits. For example, if the Mask field contains A4, the field starts at bit 10 ($A_{16}=10_{10}$) and is 4 bits wide. Certain Mask field values are used to identify special cases, as follows:

Mask Field Value	Meaning
FB	Flag reference (4-bit field)
FC	Relative address required in format group I and II instructions (13-bit field)
FD	CRU register definition (10-bit field)
FE	CRU bit reference (4-bit field)
FF	Flag reference (10-bit field)

4. The assembly end record format is shown in figure 7-7. The fields labeled R are reserved. The B bit indicates that a branch vector does not follow if it is equal to 0, and that a branch vector does follow if it is equal to 1.

Figure 7-8 is a diagram of the end-of-file record format.

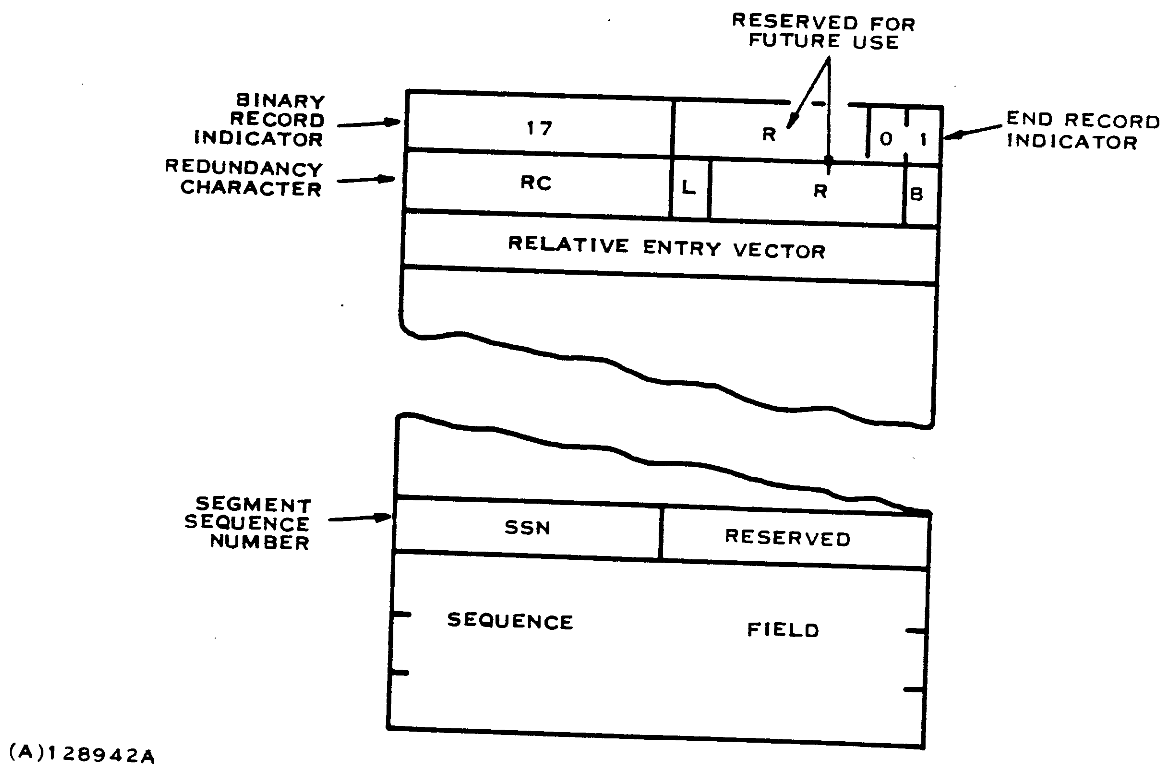
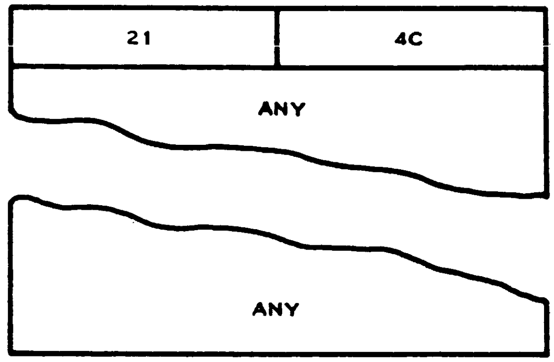


Figure 7-7. Assembly End Record Format



(A)128943

Figure 7-8. End-of-File Record Format

APPENDIX A

SAL CHARACTER SET

APPENDIX A

SAL CHARACTER SET

The ASCII characters are listed in table A-1. The table includes the ASCII code for each character, represented as a hexadecimal value and as a decimal value. The table also shows the corresponding Hollerith code.

Table A-1. Character Set

USASCII Hexadecimal Value	Decimal Value	Function	Hollerith Code
00	0	Null	12-0-1-8-9
01	1	Start Heading	12-1-9
02	2	Start Text	12-2-9
03	3	End Text	12-3-9
04	4	End Transmission	7-9
05	5	Enquiry	0-5-8-9
06	6	Acknowledge	0-6-8-9
07	7	Bell	0-7-8-9
08	8	Backspace	11-6-9
09	9	Horizontal Tab	12-5-9
0A	10	Line Feed	0-5-9
0B	11	Vertical Tab	12-3-8-9
0C	12	Form Feed	12-4-8-9
0D	13	Carriage Return	12-5-8-9
0E	14	Shift Out	12-6-8-9
0F	15	Shift In	12-7-8-9
10	16	Data Link Escape	12-11-1-8-9
11	17	Device Control 1	11-1-9
12	18	Device Control 2	11-2-9
13	19	Device Control 3	11-3-9
14	20	Device Control 4	4-8-9
15	21	Negative Acknowledge	5-8-9
16	22	Synchronous Idle	2-9
17	23	End Transmission Block	0-6-9
18	24	Cancel	11-8-9
19	25	End Medium	11-1-8-9
1A	26	Substitute	7-8-9
1B	27	Escape	0-7-9
1C	28	File Separator	11-4-8-9
1D	29	Group Separator	11-5-8-9
1E	30	Record Separator	11-6-8-9
1F	31	Unit Separator	11-7-8-9
20	32	Space	Blank
21	33	!	11-8-2 (or 12-8-7)[1]
22	34	"	8-7
23	35	#	8-3
24	36	$	11-8-3
25	37	%	0-8-4
26	38	&	12
27	39	'	8-5
28	40	(	12-8-5
29	41	)	11-8-5
2A	42	*	11-8-4
2B	43	+	12-8-6
2C	44	,	0-8-3
2D	45	-	11
2E	46	.	12-8-3
2F	47	/	0-1

Table A-1. Character Set (Continued)

USASCII Hexadecimal Value	Decimal Value	Function	Hollerith Code
30	48	0	0
31	49	1	1
32	50	2	2
33	51	3	3
34	52	4	4
35	53	5	5
36	54	6	6
37	55	7	7
38	56	8	8
39	57	9	9
3A	58	:	8-2
3B	59	;	11-8-6
3C	60	<	12-8-4
3D	61	=	8-6
3E	62	>	0-8-6
3F	63	?	0-8-7
40	64	@	8-4
41	65	A	12-1
42	66	B	12-2
43	67	C	12-3
44	68	D	12-4
45	69	E	12-5
46	70	F	12-6
47	71	G	12-7
48	72	H	12-8
49	73	I	12-9
4A	74	J	11-1
4B	75	K	11-2
4C	76	L	11-3
4D	77	M	11-4
4E	78	N	11-5
4F	79	O	11-6
50	80	P	11-7
51	81	Q	11-8
52	82	R	11-9
53	83	S	0-2
54	84	T	0-3
55	85	U	0-4
56	86	V	0-5
57	87	W	0-6
58	88	X	0-7
59	89	Y	0-8
5A	90	Z	0-9
5B	91	[	12-2-8
5C	92	\	0-8-2
5D	93	]	-1
5E	94	^	11-7-8
5F	95	_	0-5-8

Table A-1. Character Set (Continued)

USASCII Hexadecimal Value	Decimal Value	Function	Hollerith Code
60	96	`	
7B	123	{	
7C	124	¦	
7D	125	}	
7E	126	~	
7F	127	DEL	

1. During card input, the computer interprets Hollerith codes 11-8-2 and 12-8-7 as an ASCII code of 21_{16} to produce an exclamation point input (!). Card input for a close-bracket character ( ]) is not possible.

APPENDIX B

GENERAL TABLES

Table B-1. Hexadecimal Arithmetic

ADDITION TABLE

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
2	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11
3	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12
4	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
5	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14
6	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15
7	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16
8	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17
9	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18
A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19
B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A
C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

MULTIPLICATION TABLE

1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
3	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
4	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

Table B-2. Table of Powers of 16_{10}

16^n	n	16^{-n}				
1	0	0.10000	00000	00000	00000	x 10
16	1	0.62500	00000	00000	00000	x 10^{-1}
256	2	0.39062	50000	00000	00000	x 10^{-2}
4 096	3	0.24414	06250	00000	00000	x 10^{-3}
65 536	4	0.15258	78906	25000	00000	x 10^{-4}
1 048 576	5	0.95367	43164	06250	00000	x 10^{-6}
16 777 216	6	0.59604	64477	53906	25000	x 10^{-7}
268 435 456	7	0.37252	90298	46191	40625	x 10^{-8}
4 294 967 296	8	0.23283	06436	53869	62891	x 10^{-9}
68 719 476 736	9	0.14551	91522	83668	51807	x 10^{-10}
1 099 511 627 776	10	0.90949	47017	72928	23792	x 10^{-12}
17 592 186 044 416	11	0.56843	41886	08080	14870	x 10^{-13}
281 474 976 510 656	12	0.35527	13678	80050	09294	x 10^{-14}
4 503 599 627 370 496	13	0.22204	46049	25031	30808	x 10^{-15}
72 057 594 037 927 936	14	0.13877	78780	78144	56755	x 10^{-16}
1 152 921 504 606 846 976	15	0.86736	17379	88403	54721	x 10^{-18}

Table B-3. Table of Powers of 10_{16}

10^n	n	10^{-n}				
1	0	1.0000	0000	0000	0000	
A	1	0.1999	9999	9999	999A	
64	2	0.28F5	C28F	5C28	F5C3	x 16^{-1}
3E8	3	0.4189	374B	C6A7	EF9E	x 16^{-2}
2710	4	0.68DB	8BAC	710C	B296	x 16^{-3}
1 86A0	5	0.A7C5	AC47	1B47	8423	x 16^{-4}
F 4240	6	0.10C6	F7A0	B5ED	8D37	x 16^{-4}
98 9680	7	0.1AD7	F29A	BCAF	4858	x 16^{-5}
5F5 E100	8	0.2AF3	1DC4	6118	73BF	x 16^{-6}
3B9A CA00	9	0.44B8	2FA0	9B5A	52CC	x 16^{-7}
2 540B E400	10	0.6DF3	7F67	5EF6	EADF	x 16^{-8}
17 4876 E800	11	0.AFEB	FF0B	CB24	AAFF	x 16^{-9}
E8 D4A5 1000	12	0.1197	9981	2DEA	1119	x 16^{-9}
918 4E72 A000	13	0.1C25	C268	4976	81C2	x 16^{-10}
5AF3 107A 4000	14	0.2D09	370D	4257	3604	x 16^{-11}
3 8D7E A4C6 8000	15	0.480E	BE7B	9D58	566D	x 16^{-12}
23 86F2 6FC1 0000	16	0.734A	CA5F	6226	F0AE	x 16^{-13}
163 4578 5D8A 0000	17	0.B877	AA32	36A4	B449	x 16^{-14}
DE0 B6B3 A764 0000	18	0.1272	5DD1	D243	ABA1	x 16^{-14}
8AC7 2304 89E8 0000	19	0.1D83	C94F	B6D2	AC35	x 16^{-15}

Table B-4. Table of Powers of Two

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

Table B-5. Hexadecimal–Decimal Integer Conversion Table

The table appearing on the following pages provides a means for direct conversion of decimal integers in the range of 0 to 4095 and for hexadecimal integers in the range of 0 to FFF.

To convert numbers above those ranges, add table values to the figures below:

Hexadecimal	Decimal	Hexadecimal	Decimal
01 000	4 096	20 000	131 072
02 000	8 192	30 000	196 608
03 000	12 288	40 000	262 144
04 000	16 384	50 000	327 680
05 000	20 480	60 000	393 216
06 000	24 576	70 000	458 752
07 000	28 672	80 000	524 288
08 000	32 768	90 000	589 824
09 000	36 864	A0 000	655 360
0A 000	40 960	B0 000	720 896
0B 000	45 056	C0 000	786 432
0C 000	49 152	D0 000	851 968
0D 000	53 248	E0 000	917 504
0E 000	57 344	F0 000	983 040
0F 000	61 440	100 000	1 048 576
10 000	65 536	200 000	2 097 152
11 000	69 632	300 000	3 145 728
12 000	73 728	400 000	4 194 304
13 000	77 824	500 000	5 242 880
14 000	81 920	600 000	6 291 456
15 000	86 016	700 000	7 340 032
16 000	90 112	800 000	8 388 608
17 000	94 208	900 000	9 437 184
18 000	98 304	A00 000	10 485 760
19 000	102 400	B00 000	11 534 336
1A 000	106 496	C00 000	12 582 912
1B 000	110 592	D00 000	13 631 488
1C 000	114 688	E00 000	14 680 064
1D 000	118 784	F00 000	15 728 640
1E 000	122 880	1 000 000	16 777 216
1F 000	126 976	2 000 000	33 554 432

Table B-5. Hexadecimal–Decimal Integer Conversion Table (Cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
010	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
020	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
030	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
040	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
050	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
060	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
070	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
080	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
090	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A0	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B0	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C0	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D0	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E0	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F0	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
100	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
110	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
120	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
130	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
140	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
150	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
160	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
170	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
180	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
190	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A0	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B0	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C0	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D0	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E0	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F0	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
200	0512	0513	0514	0515	0516	0517	0518	0519	0529	0521	0522	0523	0524	0525	0526	0527
210	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
220	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
230	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
240	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
250	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
260	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
270	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
280	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
290	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A0	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B0	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C0	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D0	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E0	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F0	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767

Table B-5. Hexadecimal–Decimal Integer Conversion Table (Cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
300	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
310	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
320	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
330	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
340	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
350	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
360	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
370	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
380	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
390	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A0	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B0	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C0	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D0	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E0	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F0	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023
400	1024	1025	0126	0127	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
410	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
420	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
430	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
440	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
450	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
460	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
470	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
480	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
490	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A0	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B0	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C0	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D0	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E0	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F0	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
500	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1291	1293	1294	1295
510	1296	1297	1298	1299	1399	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
520	1312	1313	1314	1315	1316	1317	1318	1319	1329	1321	1322	1323	1324	1325	1326	1327
530	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
540	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1367	1358	1359
550	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
560	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
570	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
580	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1429	1421	1422	1423
590	1324	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A0	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
3B0	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C0	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D0	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E0	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F0	1520	1521	1522	1523	1524	1515	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535

Table B-5. Hexadecimal–Decimal Integer Conversion Table (Cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
600	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
610	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
620	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
630	1584	1585	1586	1587	1588	1589	1590	1591	1592	1592	1594	1595	1596	1597	1598	1599
640	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
650	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
660	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
670	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
680	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
690	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A0	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B0	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	17231	1724	1725	1726	1727
6C0	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D0	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E0	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F0	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791
700	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	8102	1803	1804	1805	1806	1807
710	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
720	1824	1825	1826	1827	1818	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
730	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
740	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
750	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
760	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1909	1902	1903
770	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
780	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
790	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A0	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B0	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C0	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D0	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E0	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F0	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
800	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
810	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
820	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
830	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
840	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
850	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
860	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
870	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
880	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
890	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A0	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B0	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C0	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D0	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E0	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F0	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303

Table B-5. Hexadecimal–Decimal Integer Conversion Table (Cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
900	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
910	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
920	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
930	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
940	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
950	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	3496	2397	2398	2399
960	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
970	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
980	2432	2433	2434	24351	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
990	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A0	2464	2465	2466	2467	2468	2469	2479	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B0	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C0	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D0	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E0	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F0	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559
A00	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A10	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A20	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A30	2608	2609	2610	2611	2612	2613	2614	2615	2626	2617	2618	2619	2620	2621	2622	2623
A40	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A50	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A60	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A70	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A80	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A90	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA0	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
Ab0	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC0	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD0	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE0	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF0	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B00	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B10	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B20	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B30	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B40	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B50	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B60	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B70	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B80	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B90	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA0	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB0	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC0	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD0	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE0	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF0	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071

Table B-5. Hexadecimal–Decimal Integer Conversion Table (Cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C00	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C10	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C20	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C30	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C40	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C50	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C60	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C70	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C80	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C90	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA0	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB0	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC0	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD0	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE0	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF0	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
D00	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D10	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D20	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D30	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D40	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D50	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D60	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D70	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D80	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D90	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA0	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB0	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC0	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD0	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE0	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF0	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E00	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E10	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E20	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E30	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E40	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E50	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E60	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E70	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E80	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E90	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA0	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB0	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775

Table B-5. Hexadecimal–Decimal Integer Conversion Table (Cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
EC0	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED0	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE0	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF0	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F00	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F10	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F20	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F30	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F40	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F50	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F60	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F70	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F80	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F90	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA0	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB0	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC0	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD0	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE0	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF0	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

Table B-6. Hexadecimal–Decimal Fraction Conversion Table

Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.40 00 00 00	.25000 00000	.80 00 00 00	.50000 00000	.C0 00 00 00	.75000 00000
.01 00 00 00	.00390 62500	.41 00 00 00	.25390 62500	.81 00 00 00	.50390 62500	.C1 00 00 00	.75390 62500
.02 00 00 00	.00781 25000	.42 00 00 00	.25781 25000	.82 00 00 00	.50781 25000	.C2 00 00 00	.75781 25000
.03 00 00 00	.01171 87500	.43 00 00 00	.26171 87500	.83 00 00 00	.51171 87500	.C3 00 00 00	.76171 87500
.04 00 00 00	.01562 50000	.44 00 00 00	.26562 50000	.84 00 00 00	.51562 50000	.C4 00 00 00	.76562 50000
.05 00 00 00	.01953 12500	.45 00 00 00	.26953 12500	.85 00 00 00	.51953 12500	.C5 00 00 00	.76953 12500
.06 00 00 00	.02343 75000	.46 00 00 00	.27343 75000	.86 00 00 00	.52343 75000	.C6 00 00 00	.77343 75000
.07 00 00 00	.02734 37500	.47 00 00 00	.27734 37500	.87 00 00 00	.52734 37500	.C7 00 00 00	.77734 37500
.08 00 00 00	.03125 00000	.48 00 00 00	.28125 00000	.88 00 00 00	.53125 00000	.C8 00 00 00	.78125 00000
.09 00 00 00	.03515 62500	.49 00 00 00	.28515 62500	.89 00 00 00	.53515 62500	.C9 00 00 00	.78515 62500
.0A 00 00 00	.03906 25000	.4A 00 00 00	.28906 25000	.8A 00 00 00	.53906 25000	.CA 00 00 00	.78906 25000
.0B 00 00 00	.04296 87500	.4B 00 00 00	.29296 87500	.8B 00 00 00	.54296 87500	.CB 00 00 00	.79296 87500
.0C 00 00 00	.04687 50000	.4C 00 00 00	.29687 50000	.8C 00 00 00	.54687 50000	.CC 00 00 00	.79687 50000
.0D 00 00 00	.05078 12500	.4D 00 00 00	.30078 12500	.8D 00 00 00	.55078 12500	.CD 00 00 00	.80078 12500
.0E 00 00 00	.05468 75000	.4E 00 00 00	.30468 75000	.8E 00 00 00	.55468 75000	.CE 00 00 00	.80468 75000
.0F 00 00 00	.05859 37500	.4F 00 00 00	.30859 37500	.8F 00 00 00	.55859 37500	.CF 00 00 00	.80859 37500
.10 00 00 00	.06250 00000	.50 00 00 00	.31250 00000	.90 00 00 00	.56250 00000	.D0 00 00 00	.81250 00000
.11 00 00 00	.06640 62500	.51 00 00 00	.31640 62500	.91 00 00 00	.56640 62500	.D1 00 00 00	.81640 62500
.12 00 00 00	.07031 25000	.52 00 00 00	.32031 25000	.92 00 00 00	.57031 25000	.D2 00 00 00	.82031 25000
.13 00 00 00	.07421 87500	.53 00 00 00	.32421 87500	.93 00 00 00	.57421 87500	.D3 00 00 00	.82421 87500
.14 00 00 00	.07812 50000	.54 00 00 00	.32812 50000	.94 00 00 00	.57812 50000	.D4 00 00 00	.82812 50000
.15 00 00 00	.08203 12500	.55 00 00 00	.33203 12500	.95 00 00 00	.58203 12500	.D5 00 00 00	.83203 12500
.16 00 00 00	.08593 75000	.56 00 00 00	.33593 75000	.96 00 00 00	.58593 75000	.D6 00 00 00	.83593 75000
.17 00 00 00	.08984 37500	.57 00 00 00	.33984 37500	.97 00 00 00	.58984 37500	.D7 00 00 00	.83984 37500
.18 00 00 00	.09375 00000	.58 00 00 00	.34375 00000	.98 00 00 00	.59375 00000	.D8 00 00 00	.84375 00000
.19 00 00 00	.09765 62500	.59 00 00 00	.34765 62500	.99 00 00 00	.59765 62500	.D9 00 00 00	.84765 62500
.1A 00 00 00	.10156 25000	.5A 00 00 00	.35156 25000	.9A 00 00 00	.60156 25000	.DA 00 00 00	.85156 25000
.1B 00 00 00	.10546 87500	.5B 00 00 00	.35546 87500	.9B 00 00 00	.60546 87500	.DB 00 00 00	.85546 87500
.1C 00 00 00	.10937 50000	.5C 00 00 00	.35937 50000	.9C 00 00 00	.60937 50000	.DC 00 00 00	.85937 50000
.1D 00 00 00	.11328 12500	.5D 00 00 00	.36328 12500	.9D 00 00 00	.61328 12500	.DD 00 00 00	.86328 12500
.1E 00 00 00	.11718 75000	.5E 00 00 00	.36718 75000	.9E 00 00 00	.61718 75000	.DE 00 00 00	.86718 75000
.1F 00 00 00	.12109 37500	.5F 00 00 00	.37109 37500	.9F 00 00 00	.62109 37500	.DF 00 00 00	.87109 37500
.20 00 00 00	.12500 00000	.60 00 00 00	.37500 00000	.A0 00 00 00	.62500 00000	.E0 00 00 00	.87500 00000
.21 00 00 00	.12890 62500	.61 00 00 00	.37890 62500	.A1 00 00 00	.62890 62500	.E1 00 00 00	.87890 62500
.22 00 00 00	.13281 25000	.62 00 00 00	.38281 25000	.A2 00 00 00	.63281 25000	.E2 00 00 00	.88281 25000
.23 00 00 00	.13671 87500	.63 00 00 00	.38671 87500	.A3 00 00 00	.63671 87500	.E3 00 00 00	.88671 87500
.24 00 00 00	.14062 50000	.64 00 00 00	.39062 50000	.A4 00 00 00	.64062 50000	.E4 00 00 00	.89062 50000
.25 00 00 00	.14453 12500	.65 00 00 00	.39453 12500	.A5 00 00 00	.64453 12500	.E5 00 00 00	.89453 12500
.26 00 00 00	.14843 75000	.66 00 00 00	.39843 75000	.A6 00 00 00	.64843 75000	.E6 00 00 00	.89843 75000
.27 00 00 00	.15234 37500	.67 00 00 00	.40234 37500	.A7 00 00 00	.65234 37500	.E7 00 00 00	.90234 37500
.28 00 00 00	.15625 00000	.68 00 00 00	.40625 00000	.A8 00 00 00	.65625 00000	.E8 00 00 00	.90625 00000
.29 00 00 00	.16015 62500	.69 00 00 00	.41015 62500	.A9 00 00 00	.66015 62500	.E9 00 00 00	.91015 62500
.2A 00 00 00	.16406 25000	.6A 00 00 00	.41406 25000	.AA 00 00 00	.66406 25000	.EA 00 00 00	.91406 25000
.2B 00 00 00	.16796 87500	.6B 00 00 00	.41796 87500	.AB 00 00 00	.66796 87500	.EB 00 00 00	.91796 87500
.2C 00 00 00	.17187 50000	.6C 00 00 00	.42187 50000	.AC 00 00 00	.67187 50000	.EC 00 00 00	.92187 50000
.2D 00 00 00	.17578 12500	.6D 00 00 00	.42578 12500	.AD 00 00 00	.67578 12500	.ED 00 00 00	.92578 12500
.2E 00 00 00	.17968 75000	.6E 00 00 00	.42968 75000	.AE 00 00 00	.67968 75000	.EE 00 00 00	.92968 75000
.2F 00 00 00	.18359 37500	.6F 00 00 00	.43359 37500	.AF 00 00 00	.68359 37500	.EF 00 00 00	.93359 37500
.30 00 00 00	.18750 00000	.70 00 00 00	.43750 00000	.B0 00 00 00	.68750 00000	.F0 00 00 00	.93750 00000
.31 00 00 00	.19140 62500	.71 00 00 00	.44140 62500	.B1 00 00 00	.69140 62500	.F1 00 00 00	.94140 62500
.32 00 00 00	.19531 25000	.72 00 00 00	.44531 25000	.B2 00 00 00	.69531 25000	.F2 00 00 00	.94531 25000
.33 00 00 00	.19921 87500	.73 00 00 00	.44921 87500	.B3 00 00 00	.69921 87500	.F3 00 00 00	.94921 87500
.34 00 00 00	.20312 50000	.74 00 00 00	.45312 50000	.B4 00 00 00	.70312 50000	.F4 00 00 00	.95312 50000
.35 00 00 00	.20703 12500	.75 00 00 00	.45703 12500	.B5 00 00 00	.70703 12500	.F5 00 00 00	.95703 12500
.36 00 00 00	.21093 75000	.76 00 00 00	.46093 75000	.B6 00 00 00	.71093 75000	.F6 00 00 00	.96093 75000
.37 00 00 00	.21484 37500	.77 00 00 00	.46484 37500	.B7 00 00 00	.71484 37500	.F7 00 00 00	.96484 37500
.38 00 00 00	.21875 00000	.78 00 00 00	.46875 00000	.B8 00 00 00	.71875 00000	.F8 00 00 00	.96875 00000
.39 00 00 00	.22265 62500	.79 00 00 00	.47265 62500	.B9 00 00 00	.72265 62500	.F9 00 00 00	.97265 62500
.3A 00 00 00	.22656 25000	.7A 00 00 00	.47656 25000	.BA 00 00 00	.72656 25000	.FA 00 00 00	.97656 25000
.3B 00 00 00	.23046 87500	.7B 00 00 00	.48046 87500	.BB 00 00 00	.73046 87500	.FB 00 00 00	.98046 87500
.3C 00 00 00	.23437 50000	.7C 00 00 00	.48437 50000	.BC 00 00 00	.73437 50000	.FC 00 00 00	.98437 50000
.3D 00 00 00	.23828 12500	.7D 00 00 00	.48828 12500	.BD 00 00 00	.73828 12500	.FD 00 00 00	.98828 12500
.3E 00 00 00	.24218 75000	.7E 00 00 00	.49218 75000	.BE 00 00 00	.74218 75000	.FE 00 00 00	.99218 75000
.3F 00 00 00	.24609 37500	.7F 00 00 00	.49609 37500	.BF 00 00 00	.74609 37500	.FF 00 00 00	.99609 37500

Table B-6. Hexadecimal–Decimal Fraction Conversion Table (Cont.)

Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.00 40 00 00	.00097 65625	.00 80 00 00	.00195 31250	.00 C0 00 00	.00292 96875
.00 01 00 00	.00001 52587	.00 41 00 00	.00099 18212	.00 81 00 00	.00196 83837	.00 C1 00 00	.00294 49462
.00 02 00 00	.00003 05175	.00 42 00 00	.00100 70800	.00 82 00 00	.00198 36425	.00 C2 00 00	.00296 02050
.00 03 00 00	.00004 57763	.00 43 00 00	.00102 23388	.00 83 00 00	.00199 89013	.00 C3 00 00	.00297 54638
.00 04 00 00	.00006 10351	.00 44 00 00	.00103 75976	.00 84 00 00	.00201 41601	.00 C4 00 00	.00299 07226
.00 05 00 00	.00007 62939	.00 45 00 00	.00105 28564	.00 85 00 00	.00202 94189	.00 C5 00 00	.00300 59814
.00 06 00 00	.00009 15527	.00 46 00 00	.00106 81152	.00 86 00 00	.00204 46777	.00 C6 00 00	.00302 12402
.00 07 00 00	.00010 68115	.00 47 00 00	.00108 33740	.00 87 00 00	.00205 99365	.00 C7 00 00	.00303 64990
.00 08 00 00	.00012 20703	.00 48 00 00	.00109 86328	.00 88 00 00	.00207 51953	.00 C8 00 00	.00305 17578
.00 09 00 00	.00013 73291	.00 49 00 00	.00111 38916	.00 89 00 00	.00209 04541	.00 C9 00 00	.00306 70166
.00 0A 00 00	.00015 25878	.00 4A 00 00	.00112 91503	.00 8A 00 00	.00210 57128	.00 CA 00 00	.00308 22753
.00 0B 00 00	.00016 78466	.00 4B 00 00	.00114 44091	.00 8B 00 00	.00212 09716	.00 CB 00 00	.00309 75341
.00 0C 00 00	.00018 31054	.00 4C 00 00	.00115 96679	.00 8C 00 00	.00213 62304	.00 CC 00 00	.00311 27929
.00 0D 00 00	.00019 83642	.00 4D 00 00	.00117 49267	.00 8D 00 00	.00215 14892	.00 CD 00 00	.00312 80517
.00 0E 00 00	.00021 36230	.00 4E 00 00	.00119 01855	.00 8E 00 00	.00216 67480	.00 CE 00 00	.00314 33105
.00 0F 00 00	.00022 88818	.00 4F 00 00	.00120 54443	.00 8F 00 00	.00218 20068	.00 CF 00 00	.00315 85693
.00 10 00 00	.00024 41406	.00 50 00 00	.00122 07031	.00 90 00 00	.00219 72656	.00 D0 00 00	.00317 38281
.00 11 00 00	.00025 93994	.00 51 00 00	.00123 59619	.00 91 00 00	.00221 25244	.00 D1 00 00	.00318 90869
.00 12 00 00	.00027 46582	.00 52 00 00	.00125 12207	.00 92 00 00	.00222 77832	.00 D2 00 00	.00320 43457
.00 13 00 00	.00028 99169	.00 53 00 00	.00126 64794	.00 93 00 00	.00224 30419	.00 D3 00 00	.00321 96044
.00 14 00 00	.00030 51757	.00 54 00 00	.00128 17382	.00 94 00 00	.00225 83007	.00 D4 00 00	.00323 48632
.00 15 00 00	.00032 04345	.00 55 00 00	.00129 69970	.00 95 00 00	.00227 35595	.00 D5 00 00	.00325 01220
.00 16 00 00	.00033 56933	.00 56 00 00	.00131 22558	.00 96 00 00	.00228 88183	.00 D6 00 00	.00326 53808
.00 17 00 00	.00035 09521	.00 57 00 00	.00132 75146	.00 97 00 00	.00230 40771	.00 D7 00 00	.00328 06396
.00 18 00 00	.00036 62109	.00 58 00 00	.00134 27734	.00 98 00 00	.00231 93359	.00 D8 00 00	.00329 58984
.00 19 00 00	.00038 14697	.00 59 00 00	.00135 80322	.00 99 00 00	.00233 45947	.00 D9 00 00	.00331 11572
.00 1A 00 00	.00039 67285	.00 5A 00 00	.00137 32910	.00 9A 00 00	.00234 98535	.00 DA 00 00	.00332 64160
.00 1B 00 00	.00041 19873	.00 5B 00 00	.00138 85498	.00 9B 00 00	.00236 51123	.00 DB 00 00	.00334 16748
.00 1C 00 00	.00042 72460	.00 5C 00 00	.00140 38085	.00 9C 00 00	.00238 03710	.00 DC 00 00	.00335 69335
.00 1D 00 00	.00044 25048	.00 5D 00 00	.00141 90673	.00 9D 00 00	.00239 56298	.00 DD 00 00	.00337 21923
.00 1E 00 00	.00045 77636	.00 5E 00 00	.00143 43261	.00 9E 00 00	.00241 08886	.00 DE 00 00	.00338 74511
.00 1F 00 00	.00047 30224	.00 5F 00 00	.00144 95849	.00 9F 00 00	.00242 61474	.00 DF 00 00	.00340 27099
.00 20 00 00	.00048 82812	.00 60 00 00	.00146 48437	.00 A0 00 00	.00244 14062	.00 E0 00 00	.00341 79687
.00 21 00 00	.00050 35400	.00 61 00 00	.00148 01025	.00 A1 00 00	.00245 66650	.00 E1 00 00	.00343 32275
.00 22 00 00	.00051 87988	.00 62 00 00	.00149 53613	.00 A2 00 00	.00247 19238	.00 E2 00 00	.00344 84863
.00 23 00 00	.00053 40576	.00 63 00 00	.00151 06201	.00 A3 00 00	.00248 71826	.00 E3 00 00	.00346 37451
.00 24 00 00	.00054 93164	.00 64 00 00	.00152 58789	.00 A4 00 00	.00250 24414	.00 E4 00 00	.00347 90039
.00 25 00 00	.00056 45751	.00 65 00 00	.00154 11376	.00 A5 00 00	.00251 77001	.00 E5 00 00	.00349 42626
.00 26 00 00	.00057 98339	.00 66 00 00	.00155 63964	.00 A6 00 00	.00253 29589	.00 E6 00 00	.00350 95214
.00 27 00 00	.00059 50927	.00 67 00 00	.00157 16552	.00 A7 00 00	.00254 82177	.00 E7 00 00	.00352 47802
.00 28 00 00	.00061 03515	.00 68 00 00	.00158 69140	.00 A8 00 00	.00256 34765	.00 E8 00 00	.00354 00390
.00 29 00 00	.00062 56103	.00 69 00 00	.00160 21728	.00 A9 00 00	.00257 87353	.00 E9 00 00	.00355 52978
.00 2A 00 00	.00064 08691	.00 6A 00 00	.00161 74316	.00 AA 00 00	.00259 39941	.00 EA 00 00	.00357 05566
.00 2B 00 00	.00065 61279	.00 6B 00 00	.00163 26904	.00 AB 00 00	.00260 92529	.00 EB 00 00	.00358 58154
.00 2C 00 00	.00067 13867	.00 6C 00 00	.00164 79492	.00 AC 00 00	.00262 45117	.00 EC 00 00	.00360 10742
.00 2D 00 00	.00068 66455	.00 6D 00 00	.00166 32080	.00 AD 00 00	.00263 97705	.00 ED 00 00	.00361 63330
.00 2E 00 00	.00070 19042	.00 6E 00 00	.00167 84667	.00 AE 00 00	.00265 50292	.00 EE 00 00	.00363 15917
.00 2F 00 00	.00071 71630	.00 6F 00 00	.00169 37255	.00 AF 00 00	.00267 02880	.00 EF 00 00	.00364 68505
.00 30 00 00	.00073 24218	.00 70 00 00	.00170 89843	.00 B0 00 00	.00268 55468	.00 F0 00 00	.00366 21093
.00 31 00 00	.00074 76806	.00 71 00 00	.00172 42421	.00 B1 00 00	.00270 08056	.00 F1 00 00	.00367 73681
.00 32 00 00	.00076 29394	.00 72 00 00	.00173 95019	.00 B2 00 00	.00271 60644	.00 F2 00 00	.00369 26269
.00 33 00 00	.00077 81982	.00 73 00 00	.00175 47607	.00 B3 00 00	.00273 13232	.00 F3 00 00	.00370 78857
.00 34 00 00	.00079 34570	.00 74 00 00	.00177 00195	.00 B4 00 00	.00274 65820	.00 F4 00 00	.00372 31445
.00 35 00 00	.00080 87158	.00 75 00 00	.00178 52783	.00 B5 00 00	.00276 18408	.00 F5 00 00	.00373 84033
.00 36 00 00	.00082 39746	.00 76 00 00	.00180 05371	.00 B6 00 00	.00277 70996	.00 F6 00 00	.00375 36621
.00 37 00 00	.00083 92333	.00 77 00 00	.00181 57958	.00 B7 00 00	.00279 23583	.00 F7 00 00	.00376 89208
.00 38 00 00	.00085 44921	.00 78 00 00	.00183 10546	.00 B8 00 00	.00280 76171	.00 F8 00 00	.00378 41796
.00 39 00 00	.00086 97509	.00 79 00 00	.00184 63134	.00 B9 00 00	.00282 28759	.00 F9 00 00	.00379 94384
.00 3A 00 00	.00088 50097	.00 7A 00 00	.00186 15722	.00 BA 00 00	.00283 81347	.00 FA 00 00	.00381 46972
.00 3B 00 00	.00090 02685	.00 7B 00 00	.00187 68310	.00 BB 00 00	.00285 33935	.00 FB 00 00	.00382 99560
.00 3C 00 00	.00091 55273	.00 7C 00 00	.00189 20898	.00 BC 00 00	.00286 86523	.00 FC 00 00	.00384 52148
.00 3D 00 00	.00093 07861	.00 7D 00 00	.00190 73486	.00 BD 00 00	.00288 39111	.00 FD 00 00	.00386 04736
.00 3E 00 00	.00094 60449	.00 7E 00 00	.00192 26074	.00 BE 00 00	.00289 91699	.00 FE 00 00	.00387 57324
.00 3F 00 00	.00096 13037	.00 7F 00 00	.00193 78662	.00 BF 00 00	.00291 44287	.00 FF 00 00	.00389 09912

Table B-6. Hexadecimal–Decimal Fraction Conversion Table (Cont.)

Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.00 00 40 00	.00000 38146	.00 00 80 00	.00000 76293	.00 00 C0 00	.00001 14440
.00 00 01 00	.00000 00596	.00 00 41 00	.00000 38743	.00 00 81 00	.00000 76889	.00 00 C1 00	.00001 15036
.00 00 02 00	.00000 01192	.00 00 42 00	.00000 39339	.00 00 82 00	.00000 77486	.00 00 C2 00	.00001 15633
.00 00 03 00	.00000 01788	.00 00 43 00	.00000 39935	.00 00 83 00	.00000 78082	.00 00 C3 00	.00001 16229
.00 00 04 00	.00000 02384	.00 00 44 00	.00000 40531	.00 00 84 00	.00000 78678	.00 00 C4 00	.00001 16825
.00 00 05 00	.00000 02980	.00 00 45 00	.00000 41127	.00 00 85 00	.00000 79274	.00 00 C5 00	.00001 17421
.00 00 06 00	.00000 03576	.00 00 46 00	.00000 41723	.00 00 86 00	.00000 79870	.00 00 C6 00	.00001 18017
.00 00 07 00	.00000 04172	.00 00 47 00	.00000 42319	.00 00 87 00	.00000 80466	.00 00 C7 00	.00001 18613
.00 00 08 00	.00000 04768	.00 00 48 00	.00000 42915	.00 00 88 00	.00000 81062	.00 00 C8 00	.00001 19209
.00 00 09 00	.00000 05364	.00 00 49 00	.00000 43511	.00 00 89 00	.00000 81658	.00 00 C9 00	.00001 19805
.00 00 0A 00	.00000 05960	.00 00 4A 00	.00000 44107	.00 00 8A 00	.00000 82254	.00 00 CA 00	.00001 20401
.00 00 0B 00	.00000 06556	.00 00 4B 00	.00000 44703	.00 00 8B 00	.00000 82850	.00 00 CB 00	.00001 20997
.00 00 0C 00	.00000 07152	.00 00 4C 00	.00000 45299	.00 00 8C 00	.00000 83446	.00 00 CC 00	.00001 21593
.00 00 0D 00	.00000 07748	.00 00 4D 00	.00000 45895	.00 00 8D 00	.00000 84042	.00 00 CD 00	.00001 22189
.00 00 0E 00	.00000 08344	.00 00 4E 00	.00000 46491	.00 00 8E 00	.00000 84638	.00 00 CE 00	.00001 22785
.00 00 0F 00	.00000 08940	.00 00 4F 00	.00000 47087	.00 00 8F 00	.00000 85234	.00 00 CF 00	.00001 23381
.00 00 10 00	.00000 09536	.00 00 50 00	.00000 47683	.00 00 90 00	.00000 85830	.00 00 D0 00	.00001 23977
.00 00 11 00	.00000 10132	.00 00 51 00	.00000 48279	.00 00 91 00	.00000 86426	.00 00 D1 00	.00001 24573
.00 00 12 00	.00000 10728	.00 00 52 00	.00000 48875	.00 00 92 00	.00000 87022	.00 00 D2 00	.00001 25169
.00 00 13 00	.00000 11324	.00 00 53 00	.00000 49471	.00 00 93 00	.00000 87618	.00 00 D3 00	.00001 25765
.00 00 14 00	.00000 11920	.00 00 54 00	.00000 50067	.00 00 94 00	.00000 88214	.00 00 D4 00	.00001 26361
.00 00 15 00	.00000 12516	.00 00 55 00	.00000 50663	.00 00 95 00	.00000 88810	.00 00 D5 00	.00001 26957
.00 00 16 00	.00000 13113	.00 00 56 00	.00000 51259	.00 00 96 00	.00000 89406	.00 00 D6 00	.00001 27553
.00 00 17 00	.00000 13709	.00 00 57 00	.00000 51856	.00 00 97 00	.00000 90003	.00 00 D7 00	.00001 28149
.00 00 18 00	.00000 14305	.00 00 58 00	.00000 52452	.00 00 98 00	.00000 90599	.00 00 D8 00	.00001 28746
.00 00 19 00	.00000 14901	.00 00 59 00	.00000 53048	.00 00 99 00	.00000 91195	.00 00 D9 00	.00001 29342
.00 00 1A 00	.00000 15497	.00 00 5A 00	.00000 53644	.00 00 9A 00	.00000 91791	.00 00 DA 00	.00001 29938
.00 00 1B 00	.00000 16093	.00 00 5B 00	.00000 54240	.00 00 9B 00	.00000 92387	.00 00 DB 00	.00001 30534
.00 00 1C 00	.00000 16689	.00 00 5C 00	.00000 54836	.00 00 9C 00	.00000 92983	.00 00 DC 00	.00001 31130
.00 00 1D 00	.00000 17285	.00 00 5D 00	.00000 55432	.00 00 9D 00	.00000 93579	.00 00 DD 00	.00001 31726
.00 00 1E 00	.00000 17881	.00 00 5E 00	.00000 56028	.00 00 9E 00	.00000 94175	.00 00 DE 00	.00001 32322
.00 00 1F 00	.00000 18477	.00 00 5F 00	.00000 56624	.00 00 9F 00	.00000 94771	.00 00 DF 00	.00001 32918
.00 00 20 00	.00000 19073	.00 00 60 00	.00000 57220	.00 00 A0 00	.00000 95367	.00 00 E0 00	.00001 33514
.00 00 21 00	.00000 19669	.00 00 61 00	.00000 57816	.00 00 A1 00	.00000 95963	.00 00 E1 00	.00001 34110
.00 00 22 00	.00000 20265	.00 00 62 00	.00000 58412	.00 00 A2 00	.00000 96559	.00 00 E2 00	.00001 34706
.00 00 23 00	.00000 20861	.00 00 63 00	.00000 59008	.00 00 A3 00	.00000 97155	.00 00 E3 00	.00001 35302
.00 00 24 00	.00000 21457	.00 00 64 00	.00000 59604	.00 00 A4 00	.00000 97751	.00 00 E4 00	.00001 35898
.00 00 25 00	.00000 22053	.00 00 65 00	.00000 60200	.00 00 A5 00	.00000 98347	.00 00 E5 00	.00001 36494
.00 00 26 00	.00000 22649	.00 00 66 00	.00000 60796	.00 00 A6 00	.00000 98943	.00 00 E6 00	.00001 37090
.00 00 27 00	.00000 23245	.00 00 67 00	.00000 61392	.00 00 A7 00	.00000 99539	.00 00 E7 00	.00001 37686
.00 00 28 00	.00000 23841	.00 00 68 00	.00000 61988	.00 00 A8 00	.00001 00135	.00 00 E8 00	.00001 38282
.00 00 29 00	.00000 24437	.00 00 69 00	.00000 62584	.00 00 A9 00	.00001 00731	.00 00 E9 00	.00001 38878
.00 00 2A 00	.00000 25033	.00 00 6A 00	.00000 63180	.00 00 AA 00	.00001 01327	.00 00 EA 00	.00001 39474
.00 00 2B 00	.00000 25629	.00 00 6B 00	.00000 63776	.00 00 AB 00	.00001 01923	.00 00 EB 00	.00001 40070
.00 00 2C 00	.00000 26226	.00 00 6C 00	.00000 64373	.00 00 AC 00	.00001 02519	.00 00 EC 00	.00001 40666
.00 00 2D 00	.00000 26822	.00 00 6D 00	.00000 64969	.00 00 AD 00	.00001 03116	.00 00 ED 00	.00001 41263
.00 00 2E 00	.00000 27418	.00 00 6E 00	.00000 65565	.00 00 AE 00	.00001 03712	.00 00 EE 00	.00001 41859
.00 00 2F 00	.00000 28014	.00 00 6F 00	.00000 61661	.00 00 AF 00	.00001 04308	.00 00 EF 00	.00001 42455
.00 00 30 00	.00000 28610	.00 00 70 00	.00000 66757	.00 00 B0 00	.00001 04904	.00 00 F0 00	.00001 43051
.00 00 31 00	.00000 29206	.00 00 71 00	.00000 67353	.00 00 B1 00	.00001 05500	.00 00 F1 00	.00001 43647
.00 00 32 00	.00000 29802	.00 00 72 00	.00000 67949	.00 00 B2 00	.00001 06096	.00 00 F2 00	.00001 44243
.00 00 33 00	.00000 30398	.00 00 73 00	.00000 68545	.00 00 B3 00	.00001 06692	.00 00 F3 00	.00001 44839
.00 00 34 00	.00000 30994	.00 00 74 00	.00000 69141	.00 00 B4 00	.00001 07228	.00 00 F4 00	.00001 45435
.00 00 35 00	.00000 31590	.00 00 75 00	.00000 69737	.00 00 B5 00	.00001 07884	.00 00 F5 00	.00001 46031
.00 00 36 00	.00000 32186	.00 00 76 00	.00000 70333	.00 00 B6 00	.00001 08480	.00 00 F6 00	.00001 46627
.00 00 37 00	.00000 32782	.00 00 77 00	.00000 70929	.00 00 B7 00	.00001 09076	.00 00 F7 00	.00001 47223
.00 00 38 00	.00000 33378	.00 00 78 00	.00000 71525	.00 00 B8 00	.00001 09672	.00 00 F8 00	.00001 47819
.00 00 39 00	.00000 33974	.00 00 79 00	.00000 75121	.00 00 B9 00	.00001 10268	.00 00 F9 00	.00001 48415
.00 00 3A 00	.00000 34570	.00 00 7A 00	.00000 72717	.00 00 BA 00	.00001 10864	.00 00 FA 00	.00001 49011
.00 00 3B 00	.00000 35166	.00 00 7B 00	.00000 73313	.00 00 BB 00	.00001 11460	.00 00 FB 00	.00001 49607
.00 00 3C 00	.00000 35762	.00 00 7C 00	.00000 73909	.00 00 BC 00	.00001 12056	.00 00 FC 00	.00001 50203
.00 00 3D 00	.00000 36358	.00 00 7D 00	.00000 74505	.00 00 BD 00	.00001 12652	.00 00 ED 00	.00001 50799
.00 00 3E 00	.00000 36954	.00 00 7E 00	.00000 75101	.00 00 BE 00	.00001 13248	.00 00 FE 00	.00001 51395
.00 00 3F 00	.00000 37550	.00 00 7F 00	.00000 75697	.00 00 BF 00	.00001 13844	.00 00 FF 00	.00001 51991

Table B-6. Hexadecimal–Decimal Fraction Conversion Table (Cont.)

Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.00 00 00 40	.00000 00149	.00 00 00 80	.00000 00298	.00 00 00 C0	.00000 00447
.00 00 00 01	.00000 00002	.00 00 00 41	.00000 00151	.00 00 00 81	.00000 00300	.00 00 00 C1	.00000 00449
.00 00 00 02	.00000 00004	.00 00 00 42	.00000 00153	.00 00 00 82	.00000 00302	.00 00 00 C2	.00000 00451
.00 00 00 03	.00000 00006	.00 00 00 43	.00000 00155	.00 00 00 83	.00000 00305	.00 00 00 C3	.00000 00454
.00 00 00 04	.00000 00009	.00 00 00 44	.00000 00158	.00 00 00 84	.00000 00307	.00 00 00 C4	.00000 00456
.00 00 00 05	.00000 00011	.00 00 00 45	.00000 00160	.00 00 00 85	.00000 00309	.00 00 00 C5	.00000 00458
.00 00 00 06	.00000 00013	.00 00 00 46	.00000 00162	.00 00 00 86	.00000 00311	.00 00 00 C6	.00000 00461
.00 00 00 07	.00000 00016	.00 00 00 47	.00000 00165	.00 00 00 87	.00000 00314	.00 00 00 C7	.00000 00463
.00 00 00 08	.00000 00018	.00 00 00 48	.00000 00167	.00 00 00 88	.00000 00316	.00 00 00 C8	.00000 00465
.00 00 00 09	.00000 00020	.00 00 00 49	.00000 00169	.00 00 00 89	.00000 00318	.00 00 00 C9	.00000 00467
.00 00 00 0A	.00000 00023	.00 00 00 4A	.00000 00172	.00 00 00 8A	.00000 00321	.00 00 00 CA	.00000 00470
.00 00 00 0B	.00000 00025	.00 00 00 4B	.00000 00174	.00 00 00 8B	.00000 00323	.00 00 00 CB	.00000 00472
.00 00 00 0C	.00000 00027	.00 00 00 4C	.00000 00176	.00 00 00 8C	.00000 00325	.00 00 00 CC	.00000 00474
.00 00 00 0D	.00000 00030	.00 00 00 4D	.00000 00179	.00 00 00 8D	.00000 00328	.00 00 00 CD	.00000 00477
.00 00 00 0E	.00000 00032	.00 00 00 4E	.00000 00181	.00 00 00 8E	.00000 00330	.00 00 00 CE	.00000 00479
.00 00 00 0F	.00000 00034	.00 00 00 4F	.00000 00183	.00 00 00 8F	.00000 00332	.00 00 00 CF	.00000 00481
.00 00 00 10	.00000 00037	.00 00 00 50	.00000 00186	.00 00 00 90	.00000 00335	.00 00 00 D0	.00000 00484
.00 00 00 11	.00000 00039	.00 00 00 51	.00000 00188	.00 00 00 91	.00000 00337	.00 00 00 D1	.00000 00486
.00 00 00 12	.00000 00041	.00 00 00 52	.00000 00190	.00 00 00 92	.00000 00339	.00 00 00 D2	.00000 00488
.00 00 00 13	.00000 00044	.00 00 00 53	.00000 00193	.00 00 00 93	.00000 00342	.00 00 00 D3	.00000 00491
.00 00 00 14	.00000 00046	.00 00 00 54	.00000 00195	.00 00 00 94	.00000 00344	.00 00 00 D4	.00000 00493
.00 00 00 15	.00000 00048	.00 00 00 55	.00000 00197	.00 00 00 95	.00000 00346	.00 00 00 D5	.00000 00495
.00 00 00 16	.00000 00051	.00 00 00 56	.00000 00200	.00 00 00 96	.00000 00349	.00 00 00 D6	.00000 00498
.00 00 00 17	.00000 00053	.00 00 00 57	.00000 00202	.00 00 00 97	.00000 00351	.00 00 00 D7	.00000 00500
.00 00 00 18	.00000 00055	.00 00 00 58	.00000 00204	.00 00 00 98	.00000 00353	.00 00 00 D8	.00000 00502
.00 00 00 19	.00000 00058	.00 00 00 59	.00000 00207	.00 00 00 99	.00000 00356	.00 00 00 D9	.00000 00505
.00 00 00 1A	.00000 00060	.00 00 00 5A	.00000 00209	.00 00 00 9A	.00000 00358	.00 00 00 DA	.00000 00507
.00 00 00 1B	.00000 00062	.00 00 00 5B	.00000 00211	.00 00 00 9B	.00000 00360	.00 00 00 DB	.00000 00509
.00 00 00 1C	.00000 00065	.00 00 00 5C	.00000 00214	.00 00 00 9C	.00000 00363	.00 00 00 DC	.00000 00512
.00 00 00 1D	.00000 00067	.00 00 00 5D	.00000 00216	.00 00 00 9D	.00000 00365	.00 00 00 DD	.00000 00514
.00 00 00 1E	.00000 00069	.00 00 00 5E	.00000 00218	.00 00 00 9E	.00000 00367	.00 00 00 DE	.00000 00516
.00 00 00 1F	.00000 00072	.00 00 00 5F	.00000 00221	.00 00 00 9F	.00000 00370	.00 00 00 DF	.00000 00519
.00 00 00 20	.00000 00074	.00 00 00 60	.00000 00223	.00 00 00 A0	.00000 00372	.00 00 00 E0	.00000 00521
.00 00 00 21	.00000 00076	.00 00 00 61	.00000 00225	.00 00 00 A1	.00000 00374	.00 00 00 E1	.00000 00523
.00 00 00 22	.00000 00079	.00 00 00 62	.00000 00228	.00 00 00 A2	.00000 00377	.00 00 00 E2	.00000 00526
.00 00 00 23	.00000 00081	.00 00 00 63	.00000 00230	.00 00 00 A3	.00000 00379	.00 00 00 E3	.00000 00528
.00 00 00 24	.00000 00083	.00 00 00 64	.00000 00232	.00 00 00 A4	.00000 00381	.00 00 00 E4	.00000 00530
.00 00 00 25	.00000 00086	.00 00 00 65	.00000 00235	.00 00 00 A5	.00000 00384	.00 00 00 E5	.00000 00533
.00 00 00 26	.00000 00088	.00 00 00 66	.00000 00237	.00 00 00 A6	.00000 00386	.00 00 00 E6	.00000 00535
.00 00 00 27	.00000 00090	.00 00 00 67	.00000 00239	.00 00 00 A7	.00000 00388	.00 00 00 E7	.00000 00537
.00 00 00 28	.00000 00093	.00 00 00 68	.00000 00242	.00 00 00 A8	.00000 00391	.00 00 00 E8	.00000 00540
.00 00 00 29	.00000 00095	.00 00 00 69	.00000 00244	.00 00 00 A9	.00000 00393	.00 00 00 E9	.00000 00542
.00 00 00 2A	.00000 00097	.00 00 00 6A	.00000 00246	.00 00 00 AA	.00000 00395	.00 00 00 EA	.00000 00544
.00 00 00 2B	.00000 00100	.00 00 00 6B	.00000 00249	.00 00 00 AB	.00000 00398	.00 00 00 EB	.00000 00547
.00 00 00 2C	.00000 00102	.00 00 00 6C	.00000 00251	.00 00 00 AC	.00000 00400	.00 00 00 EC	.00000 00549
.00 00 00 2D	.00000 00104	.00 00 00 6D	.00000 00253	.00 00 00 AD	.00000 00402	.00 00 00 ED	.00000 00551
.00 00 00 2E	.00000 00107	.00 00 00 6E	.00000 00256	.00 00 00 AE	.00000 00405	.00 00 00 EE	.00000 00554
.00 00 00 2F	.00000 00109	.00 00 00 6F	.00000 00258	.00 00 00 AF	.00000 00407	.00 00 00 EF	.00000 00556
.00 00 00 30	.00000 00111	.00 00 00 70	.00000 00260	.00 00 00 B0	.00000 00409	.00 00 00 F0	.00000 00558
.00 00 00 31	.00000 00114	.00 00 00 71	.00000 00263	.00 00 00 B1	.00000 00412	.00 00 00 F1	.00000 00561
.00 00 00 32	.00000 00116	.00 00 00 72	.00000 00265	.00 00 00 B2	.00000 00414	.00 00 00 F2	.00000 00563
.00 00 00 33	.00000 00118	.00 00 00 73	.00000 00267	.00 00 00 B3	.00000 00416	.00 00 00 F3	.00000 00565
.00 00 00 34	.00000 00121	.00 00 00 74	.00000 00270	.00 00 00 B4	.00000 00419	.00 00 00 F4	.00000 00568
.00 00 00 35	.00000 00123	.00 00 00 75	.00000 00272	.00 00 00 B5	.00000 00421	.00 00 00 F5	.00000 00570
.00 00 00 36	.00000 00125	.00 00 00 76	.00000 00274	.00 00 00 B6	.00000 00423	.00 00 00 F6	.00000 00572
.00 00 00 37	.00000 00128	.00 00 00 77	.00000 00277	.00 00 00 B7	.00000 00426	.00 00 00 F7	.00000 00575
.00 00 00 38	.00000 00130	.00 00 00 78	.00000 00279	.00 00 00 B8	.00000 00428	.00 00 00 F8	.00000 00577
.00 00 00 39	.00000 00132	.00 00 00 79	.00000 00281	.00 00 00 B9	.00000 00430	.00 00 00 F9	.00000 00579
.00 00 00 3A	.00000 00135	.00 00 00 7A	.00000 00284	.00 00 00 BA	.00000 00433	.00 00 00 FA	.00000 00582
.00 00 00 3B	.00000 00137	.00 00 00 7B	.00000 00286	.00 00 00 BB	.00000 00435	.00 00 00 FB	.00000 00584
.00 00 00 3C	.00000 00139	.00 00 00 7C	.00000 00288	.00 00 00 BC	.00000 00437	.00 00 00 FC	.00000 00586
.00 00 00 3D	.00000 00142	.00 00 00 7D	.00000 00291	.00 00 00 BD	.00000 00440	.00 00 00 FD	.00000 00589
.00 00 00 3E	.00000 00144	.00 00 00 7E	.00000 00293	.00 00 00 BE	.00000 00442	.00 00 00 FE	.00000 00591
.00 00 00 3F	.00000 00146	.00 00 00 7F	.00000 00295	.00 00 00 BF	.00000 00444	.00 00 00 FF	.00000 00593

Table B-7. Common Mathematical Constants

Constant	Decimal Value			Hexadecimal Value	
π	3.14159	26535	89793	3.243F	6A89
π^{-1}	0.31830	98861	83790	0.517C	C1B7
$\sqrt{\pi}$	1.77245	38509	05516	1.C5BF	891C
$\ln \pi$	1.14472	98858	49400	1.250D	048F
e	2.71828	18284	59045	2.B7E1	5163
e^{-1}	0.36787	94411	71442	0.5E2D	58D9
$\sqrt{e}$	1.64872	12707	00128	1.A612	98E2
$\log_{10} e$	0.43429	44819	03252	0.6F2D	EC55
$\log_2 e$	1.44269	50408	88963	1.7154	7653
γ	0.57721	56649	01533	0.93C4	67E4
$\ln \gamma$	−0.54953	93129	81645	−0.8CAE	9BC1
$\sqrt{2}$	1.41421	35623	73095	1.6A09	E668
$\ln 2$	0.69314	71805	59945	0.B172	17F8
$\log_{10} 2$	0.30102	99956	63981	0.4D10	4D42
$\sqrt{10}$	3.16227	76601	68379	3.298B	075C
$\ln 10$	2.30258	40929	94046	2.4D76	3777

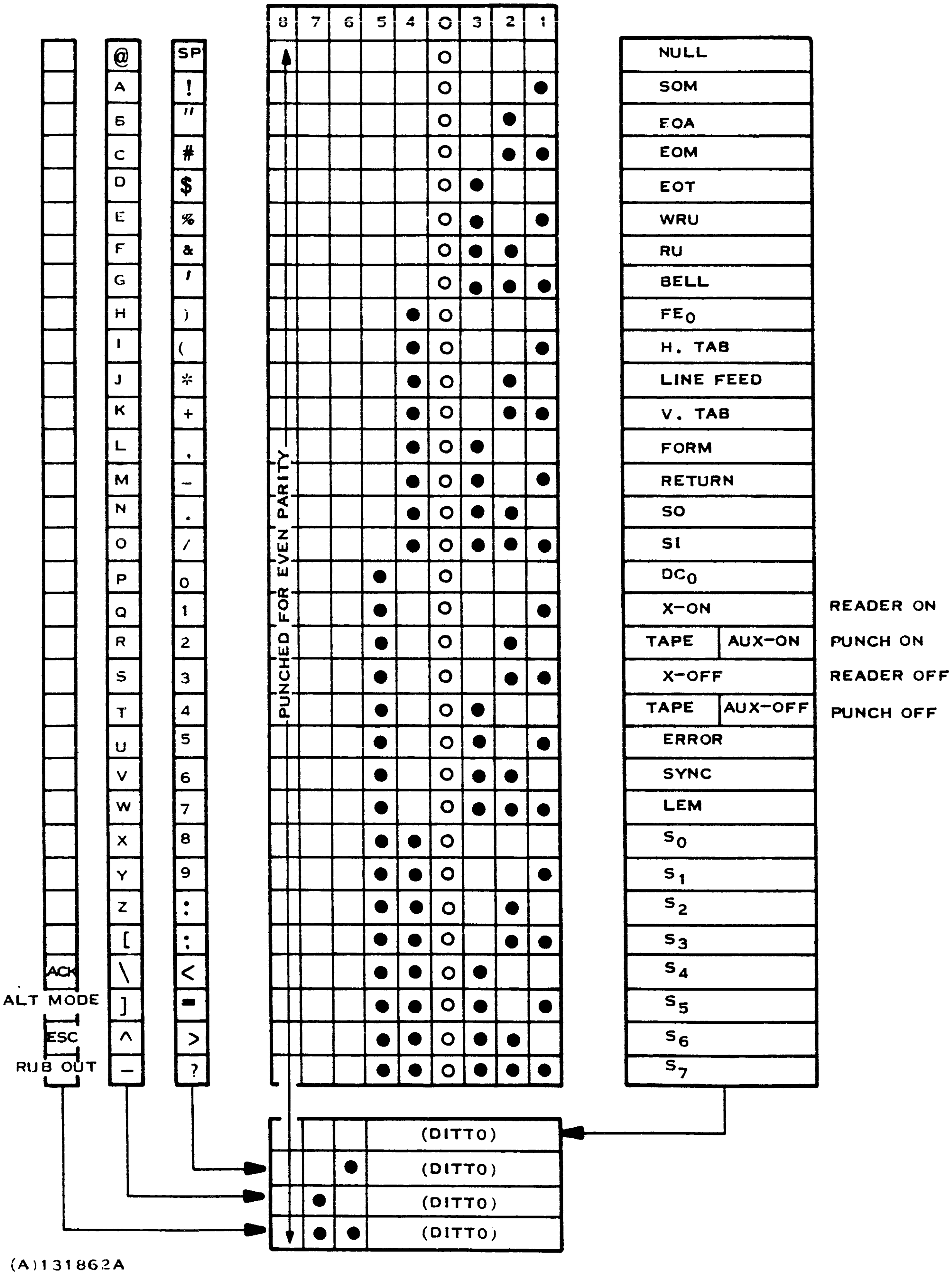


Figure B-1. Character Arrangement (USASCII) for Paper Tape

APPENDIX C

INSTRUCTION TABLES

APPENDIX C

INSTRUCTION TABLES

The source formats and the operation codes for the machine instructions are summarized in the 11 tables in this appendix. Refer to Section III for more detailed descriptions of each of the machine instructions.

Each table lists a group of instructions as follows:

Table No.	Title
C-1	Load and Store Instructions
C-2	Arithmetic Instructions
C-3	Compare Instructions
C-4	Logical Instructions
C-5	Shift Instructions
C-6	Load-Store Status and Store Panel Instructions
C-7	Branch Instructions
C-8	Mode Switching Instructions
C-9	Software Flag Instructions
C-10	CRU Instructions
C-11	Direct Memory Address Instruction

The three symbols that appear in the source statement operands (#,*,@) are addressing mode attributes, described in Section IV. The elements of the source statement operands in these tables have the following meanings:

a	Device or controller address
b	Memory word flag address or number of communication register bits
limits	Symbol name, with implicit base register definition
m	First memory address field in two-address instructions
mb	Base register used with the M address field

n	Memory address field in one-address instructions, or second address field in two-address instructions
namef, namem, namen	Symbol names, with implicit base register definition
nb	Base register used with the N address field
q	Information related to a specific device or controller
r	General register in the register file, or shift count
rq	Bit that specifies adding (or subtracting) the contents of the procedure base register to (or from) the effective operand or a counter
this	Symbol name, with implicit base register definition
vl	Immediate value in flag and bit instructions
value	13-bit signed immediate value
with	Symbol name, with implicit base register definition
xr	Index register number

In the Operand column in each table, the following conventions apply:

- Angle brackets (< >) enclose items supplied by the user
- Brackets ([ ]) enclose optional items

The format number field in the following tables can be used as a key to reference the complete description of the instructions in Section III.

Table C-1. Load and Store Instructions

Instruction Name	Mnemonic	Operand	Operation Code	Format Number	Remarks
Load Ones Tally	LOT	[#]<r>,[*][@]<n>[,<xr>]	5400 0000	I-A	(Note 1)
Load Ones Tally of Address	LOTA	[#]<r>,[*][@]<n>[,<xr>]	5480 0000	I-A	(Note 1)
Load Register	L	[#]<r>,[*][@]<n>[,<xr>]	4400 0000	I-A	(Note 1)
Load Register with Address	LA	[#]<r>,[*][@]<n>[,<xr>]	4480 0000	I-A	(Note 1)
Move Memory Word	MOV	[@](<m>,<mb>),[@](<n>,<nb>) [@]<namem>,[@]<namen>	1400 0000	II-A	(Notes 1, 2)
Store Register	ST	[#]<r>,[*][@]<n>[,<xr>]	4880 0000	I-A	(Note 1)

Note 1: All load and store instructions cause a compare status bit to be set.

Note 2: The source statement operand has two forms, explicit and implicit.

Table C-2. Arithmetic Instructions

Instruction Name	Mnemonic	Operand	Operation Code	Format Number	Remarks
Add Address to Register	AA	[#]<r>,[*][@]<n>[,<xr>]	4C80 0000	I-A	(Note 1)
Add to Memory Immediate	AMI	[@](<m>,<mb>), <value> [@] <namem>, <value>	2000 0000	II-C	(Notes 1, 2)
Add to Register	A	[#]<r>,[*][@]<n>[,<xr>]	4C00 0000	I-A	(Note 1)
Divide	D	[#]<r>,[*][@]<n>[,<xr>]	D000 0000	I-A	(Notes 1, 3)
Divide Immediate	DA	[#]<r>,[*][@]<n>[,<xr>]	D080 0000	I-A	(Notes 1, 3)
Double Add	DAD	[#]<r>,[*][@]<n>[,<xr>]	E800 0000	I-A	(Notes 1, 3)
Double Subtract	DS	[#]<r>,[*][@]<n>[,<xr>]	EC00 0000	I-A	(Notes 1, 3)
Multiply	M	[#]<r>,[*][@]<n>[,<xr>]	CC00 0000	I-A	(Notes 1, 3)
Multiply Immediate	MA	[#]<r>,[*][@]<n>[,<xr>]	CC80 0000	I-A	(Notes 1, 3)
Subtract Address from Register	SA	[#]<r>,[*][@]<n>[,<xr>]	5080 0000	I-A	(Note 1)
Subtract from Register	S	[#]<r>,[*][@]<n>[,<xr>]	5000 0000	I-A	(Note 1)

Note 1: All arithmetic instructions cause a compare status bit to be set.
Note 2: The source statement operand has two forms, explicit and implicit.
Note 3: Optional instruction (Arithmetic Option board)

Table C-3. Compare Instructions

Instruction Name	Mnemonic	Operand	Operation Code	Format Number	Remarks
Compare Memory Immediate	CMI	[@] (<m>,<mb>), <value> [@] <namem>, <value>	1C00 0000	II-C	(Note 1)
Compare Memory to Limits in Memory	CML	[@] (<m>,<mb>), [@] (<n>,<nb>) [@] <namem>, [@] <limits>	1800 0000	II-A	(Note 1)
Compare Memory to Memory	CM	[@] (<m>,<mb>), [@] (<n>,<nb>) [@] <this> , [@] <with>	1000 0000	II-A	(Note 1)
Compare Register with Effective Address	CRA	[#]<r>, [*][@]<n>[,<xr>]	C080 0000	I-A	(Note 2)
Compare Register with Effective Address (Logical)	CRLA	[#]<r>, [*][@]<n>[,<xr>]	C480 0000	I-A	(Note 2)
Compare Register with Memory	CR	[#]<r>, [*][@]<n>[,<xr>]	C000 0000	I-A	(Note 2)
Compare Register with Memory (Logical)	CRL	[#]<r>, [*][@]<n>[,<xr>]	C400 0000	I-A	(Note 2)

Note 1: The source statement operand has two forms, explicit and implicit.
Note 2: This instruction causes a compare status bit to be set.

Table C-4. Logical Instructions

Instruction Name	Mnemonic	Operand	Operation Code	Format Number	Remarks
Exclusive OR	XOR	[#]<r>,[*][@]<n>[,<xr>]	4000 0000	I-A	(Note 1)
Exclusive OR with Address	XORA	[#]<r>,[*][@]<n>[,<xr>]	4080 0000	I-A	(Note 1)
Logical AND	N	[#]<r>,[*][@]<n>[,<xr>]	5800 0000	I-A	(Note 1)
Logical AND with Address	NA	[#]<r>,[*][@]<n>[,<xr>]	5880 0000	I-A	(Note 1)
Logical OR	OR	[#]<r>,[*][@]<n>[,<xr>]	5C00 0000	I-A	(Note 1)
Logical OR with Address	ORA	[#]<r>,[*][@]<n>[,<xr>]	5C80 0000	I-A	(Note 1)

Note 1: All logical instructions cause a compare status bit to be set.

Table C-5. Shift Instructions

Instruction Name	Mnemonic	Operand	Operation Code	Format Number	Remarks
Double Right Rotate	DRR	<r>,[*][@]<n>[,<xr>]	DC00 0000	I-B	(Notes 1, 2)
Double Right Rotate, Count in Register R	DRRX	[#]<r>,[*][@]<n>[,<xr>]	DC80 0000	I-A	(Notes 1, 2)
Rotate Memory Right Logical	MRR	<r>,[*][@]<n>[,<xr>]	6800 0000	I-B	(Note 2)
Rotate Memory Right Logical, Count in Register R	MRRX	[#]<r>,[*][@]<n>[,<xr>]	6880 0000	I-A	(Note 2)
Shift and Add Tally of Leading Zeros	SAT	[#]<r>,[*][@]<n>[,<xr>]	6C00 0000	I-A	(Note 2)
Shift Memory Double Left Arithmetic	DLA	<r>,[*][@]<n>[,<xr>]	C800 0000	I-B	(Notes 1, 2)
Shift Memory Double Left Arithmetic, Count in Register R	DLAX	[#]<r>,[*][@]<n>[,<xr>]	C880 0000	I-A	(Notes 1, 2)
Shift Memory Double Right Arithmetic	DRA	<r>,[*][@]<n>[,<xr>]	D400 0000	I-B	(Notes 1, 2)
Shift Memory Double Right Arithmetic, Count in Register R	DRAX	[#]<r>,[*][@]<n>[,<xr>]	D480 0000	I-A	(Notes 1, 2)
Shift Memory Double Right Logical	DRL	<r>,[*][@]<n>[,<xr>]	D800 0000	I-B	(Notes 1, 2)
Shift Memory Double Right Logical, Count in Register R	DRLX	[#]<r>,[*][@]<n>[,<xr>]	D880 0000	I-A	(Notes 1, 2)
Shift Memory Left Arithmetic	MLA	<r>,[*][@]<n>[,<xr>]	6000 0000	I-B	(Note 2)
Shift Memory Left Arithmetic, Count in Register R	MLAX	[#]<r>,[*][@]<n>[,<xr>]	6080 0000	I-A	(Note 2)
Shift Memory Right Arithmetic	MRA	<r>,[*][@]<n>[,<xr>]	6400 0000	I-B	(Note 2)
Shift Memory Right Arithmetic, Count in Register R	MRAX	[#]<r>,[*][@]<n>[,<xr>]	6480 0000	I-A	(Note 2)

Note 1: Optional instruction (Arithmetic Option board).
Note 2: All shift instructions cause a compare status bit to be set.

Table C-6. Load-Store Status and Store Panel Instructions

Instruction Name	Mnemonic	Operand	Operation Code	Format Number	Remarks
Load Status Block	LDS	[*][@]<n>[,<xr>][,<rq>]	7C00 0000	I-C	(Notes 1, 2)
Store Panel Switches	STPS	[*][@]<n>[,<xr>]	E480 0000	I-F	(Note 2)
Store Status Block	SS	[*][@]<n>[,<xr>][,<rq>]	7886 0000	I-C	(Note 1)
Store Status Block and Branch	SSB	[*][@]<n>[,<xr>][,<rq>]	7882 0000	I-C	(Note 1)
Store Status Block and Transfer to Supervisor Mode	SXS	[*][@]<n>[,<xr>][,<rq>]	7884 0000	I-C	(Note 1)
Store Status Block and Transfer to Worker Mode	SXW	[*][@]<n>[,<xr>][,<rq>]	7885 0000	I-C	(Note 1)
Store Status Block, Transfer and Branch in Supervisor Mode	SXBS	[*][@]<n>[,<xr>][,<rq>]	7880 0000	I-C	(Note 1)
Store Status Block, Transfer and Branch in Worker Mode	SXBW	[*][@]<n>[,<xr>][,<rq>]	7881 0000	I-C	(Note 1)

Note 1: rq without indexing may be specified. Example: LDS [*][@]<n>, ,<rq>
Note 2: This instruction causes a compare status register bit to be set.

Table C-7. Branch Instructions

Instruction Name	Mnemonic	Operand	Operation Code	Format Number	Remarks
Add to Register and Branch on No Sign Change	ARB	<r>,[@]<n>,<xr>[,<rq>]	0C00 0000	I-D	(Note 1)
Branch and Link	BL	[#]<r>,[*][@]<n>[,<xr>]	7480 0000	I-A	
Branch Indirect and Link	*BL	[#]<r>,[*][@]<n>[,<xr>]	7400 0000	I-A	
Branch on Condition	BC	<r>,[*][@]<n>[,<xr>]	E080 0000	I-B	(Note 3)
Branch on Condition Indirect	*BC	<r>,[*][@]<n>[,<xr>]	E000 0000	I-B	(Note 3)
Branch Relative and Link	BRRL	<r>,[@](<n>,<nb>) <r>,[@] <namen>	2800 0000	II-B	(Note 4)
No Operation	NOP		7007 0000	I-C	
Unconditional Branch	B	[*][@]<n>[,<xr>][,<rq>]	7082 0000	I-C	(Note 2)
Unconditional Branch Indirect	*B	[*][@]<n>[,<xr>][,<rq>]	7002 0000	I-C	(Note 2)

Note 1: This instruction causes a compare status register bit to be set.
Note 2: rq without indexing may be specified by a null xr field. Example: B <n>,,<rq>
Note 3: This instruction causes a status register bit to be examined.
Note 4: The source statement operand has two forms, explicit and implicit.

Table C-8. Mode Switching Instructions

Instruction Name	Mnemonic	Operand	Operation Code	Format Number	Remarks
Transfer to Supervisor Mode	XS		7004 0000	I-C	
Transfer to Supervisor Mode and Branch	XSB	[*][@]<n>[,<xr>][,<rq>]	7080 0000	I-C	(Note 1)
Transfer to Supervisor Mode and Branch Indirect	*XSB	[*][@]<n>[,<xr>][,<rq>]	7000 0000	I-C	(Note 1)
Transfer to Worker Mode	XW		7005 0000	I-C	
Transfer to Worker Mode and Branch	XWB	[*][@]<n>[,<xr>][,<rq>]	7081 0000	I-C	(Note 1)
Transfer to Worker Mode and Branch Indirect	*XWB	[*][@]<n>[,<xr>][,<rq>]	7001 0000	I-C	(Note 1)

Note 1: rq without indexing may be specified by a null xr field. Example: XSB <n>,,<rq>

Table C-9. Software Flag Instructions

Instruction Name	Mnemonic	Operand	Operation Code	Format Number	Remarks
Branch If Flag Not Equal	BFNE	[#](<m>,<b>), <vl>,<n> [#]<namef>,<vl>,<namen>	8400 0000	III-B	(Note 1)
Set Flag	SETF	[#](<m>,<b>), <vl> [#]<namef>,<vl>	8800 0000	III-A	(Note 1)
Switch Mode If Flag Not Equal	XFNE	[#](<m>,<b>), <vl> [#]<namef>,<vl>	8000 0000	III-A	(Note 1)

Note 1: The source statement operand has two forms, explicit and implicit.

Table C-10. CRU Instructions

Instruction Name	Mnemonic	Operand	Operation Code	Format Number	Remarks
Branch On Bit Not Equal	BBNE	[#]<m>,<vl>,<n> [#]<namem>,<vl>,<namen>	3000 0000	III-D	(Note 1)
Load Communiction Register	LDCR	[#](<m>,<b>),<n> [#]<namem>,<namen>	0800 0000	III-F	(Note 1)
Set CRU Output Bit	SETB	[#]<m>,<vl> [#]<namem>,<vl>	3400 0000	III-C	(Note 1)
Store Communication Register	STCR	[#](<m>,<b>),<n> [#]<namem>,<namen>	2C00 0000	III-F	(Notes 1, 2)
Switch Mode On Bit Not Equal	XBNE	[#]<m>,<vl> [#]<namem>,<vl>	3800 0000	III-C	(Note 1)
Test Input Bit and Switch Mode or Set Output Bit	TSBX	[#]<m>,<vl>,<n>,<b> [#]<namem>,<vl>,<namen>,<b>	3C00 0000	III-E	(Note 1)

Note 1: The source statement operand has two forms, explicit and implicit.
Note 2: This instruction causes a compare status bit to be set.

Table C-11. Direct Memory Address Instruction

Instruction Name	Mnemonic	Operand	Operation Code	Format Number	Remarks
Activate Direct Memory Access Channel	ADAC	<a>,<n>[,<q>]	2400 0000	I-E	

APPENDIX D

INSTRUCTION EXECUTION TIMING

APPENDIX D

INSTRUCTION EXECUTION TIMING

The instruction execution times contained in the following tables are equally valid for the 960A or the 960B. However, since the 960B memory refresh cycle occurs twice as frequently (750 nanoseconds every 32 microseconds for the 960B as opposed to 750 nanoseconds every 64 microseconds for the 960A) therefore, allowances must be made for this difference in time critical real-time functions.

SEMICONDUCTOR MEMORY TIME IN MICROSECONDS

Standard Instructions	None	Address Modification: Indirect	Indexed	Indirect Indexed	Notes
A	3.583	4.333	4.167	4.917	
AA	2.833	3.583	3.417	4.167	
ADAC	4.000	—	—	—	
ARB	3.167 - 4.000	—	—	—	
B	2.833 - 3.417	3.583 - 4.167	3.417 - 4.000	4.167 - 4.750	
BC	3.00 - 3.25	3.75 - 4.00	3.583 - 3.833	4.333 - 4.583	
BL	2.750	2.500	3.333	4.083	
BRRL	3.333	—	—	—	
CR	3.583	4.333	4.167	4.917	
CRA	2.833	3.583	3.417	4.167	
CRL	3.583	4.333	4.167	4.917	
CRLA	2.833	3.750	3.583	4.333	
L	3.333	4.083	3.917	4.667	
LA	2.583	3.333	3.167	3.917	
LDS	4.07 - 5.83	4.75 - 5.33	4.58 - 5.16	5.33 - 5.91	
LOT	7.750	8.500	8.333	9.083	
LOTA	7.000	7.750	7.583	8.333	
MLA	3.75	4.500	3.333	5.083	+ Shift count/4
MLAX	3.583	4.333	4.167	4.917	+ Shift count/4
MRA	3.75	4.500	3.333	5.083	+ Shift count/4
MRAX	3.583	4.333	4.167	4.917	+ Shift count/4
MRR	3.75	4.500	3.333	5.083	+ Shift count/4
MRRX	3.583	4.333	4.167	4.917	+ Shift count/4
N	3.583	4.333	4.167	4.917	
NA	2.833	3.583	3.417	4.167	
NOP	2.25	—	—	—	
OR	3.583	4.333	4.167	4.917	
ORA	2.833	3.503	3.417	4.167	
S	3.583	4.333	4.167	4.917	
SA	2.833	3.583	3.417	4.167	
SAT	5.33 - 9083	6.083 - 9.833	5.917 - 9.667	6.667 - 10.417	
SS	4.25 - 4.833	6.083 - 9.833	4.83 - 3.417	5.583 - 6.167	
SSB	5.25 - 6.417	6.00 - 7.167	5.833 - 7.000	6.583 - 6.167	
ST	3.583	4.333	4.167	4.917	
STPS	3.00	3.750	3.583	4.333	
SXBS	5.500 - 6.667	6.25 - 7.417	6.083 - 8.750	6.833 - 8.000	
SXBW	5.500 - 6.667	6.25 - 7.417	6.083 - 5.667	6.833 - 8.000	
SXS	4.50 - 5.167	5.25 - 5.833	5.083 - 5.667	5.833 - 6.417	
SXW	4.50 - 5.167	5.250 - 5.833	5.083 - 5.667	5.833 - 6.417	
XOR	3.583	4.333	4.167	4.917	
XORA	2.833	3.583	3.417	4.167	
XS	2.500	—	—	—	
XSB	2.75 - 3.333	3.5 - 4.083	3.333 - 3.917	4.083 - 4.667	
XW	2.50	—	—	—	
XWB	2.75 - 3.333	3.5 - 4.083	3.333 - 3.917	4.083 - 4.667	
*B	3.583 - 4.167	4.333 - 4.917	4.167 - 4.750	4.917 - 5.500	
*BC	3.75 - 4.00	4.500 - 4.750	4.333 - 4.583	5.083 - 5.333	
*BL	3.5	4.25	4.083	4.833	
*XSB	3.500 - 4.083	4.25 - 4.833	4.083 - 4.667	4.833	
*XWB	3.500 - 4.083	4.25 - 4.833	4.083 - 4.667	4.833 - 5.417	

Standard Instructions	Execution Time—Microseconds	Notes
AMI	4.333	
CM	4.917 - 5.417	
CMI	3.833 - 4.333	
CML	5.917 - 6.667	
MOV	4.667	

Standard Instructions	Execution Time—Microseconds	Notes
BBNE	3.083 - 3.167	
BFNE	4.083 - 4.167	
LDCR	4.167	+ Number of bits/4
SETB	2.833	
SETF	4.333	
STCR	7.917	+ 250 nanoseconds for each external bit address
TSBX	3.083 - 3.417	
XBNE	3.083 - 3.33	
XFNE	4.083	

Address Modification

Optional Instructions	None	Indirect	Indexed	Indirect Indexed	Notes
D	10.417 - 10.917	11.167 - 11.667	11.000 - 11.500	11.750 - 12.250	
DA	4.667 - 10.167	10.417 - 10.917	10.250 - 10.750	11.000 - 11.500	
DAD	6.167	6.917	6.750	7.500	
DLA	6.25	7.000	6.833	7.333	+ Shift count/4
DLAX	5.833	6.583	6.417	7.167	+ Shift count/4
DRA	6.25	7.000	6.833	7.333	+ Shift count/4
DRAX	5.833	6.583	6.417	7.167	+ Shift count/4
DRL	6.25	7.000	6.833	7.333	+ Shift count/4
DRLX	5.833	6.583	6.417	7.167	+ Shift count/4
DRR	6.25	7.000	6.833	7.333	+ Shift count/4
DRRX	5.833	6.583	6.417	7.167	+ Shift count/4
DS	6.167	6.917	6.750	7.500	
M	8.583	9.333	9.166	9.916	
MA	7.833	8.583	8.416	9.166	

TYPICAL EXECUTION TIMES WITH OPTIONAL CORE MEMORY TIME IN MICROSECONDS

Address Modification

Standard Instruction	None	Indirect	Indexed	Indirect Indexed	Notes
A	4.33	5.33	4.92	5.92	
B	3.00 - 3.59	4.00 - 4.56	3.58 - 4.17	4.58 - 5.17	
SETB	3.33				
LDCR	4.92				+ Number of bits/4
MOV	5.67				

APPENDIX E

ASSEMBLER DIRECTIVE TABLE

APPENDIX E

ASSEMBLER DIRECTIVE TABLE

The assembler directives for the Symbolic Assembly Language are listed in table E-1. All directives can include a comment field following the operand field. Those directives that do not require an operand field can have a comment field following the operator field. Those directives that have optional operand fields (for example END) can have comment fields which must begin after column 22 if the operand field is not used.

The following symbols and conventions are used in defining the syntax of assembler directives:

- Angle brackets (<>) enclose items supplied by the user
- Brackets ([ ]) enclose optional items
- An ellipsis (. . .) indicates that the preceding item can be repeated.

The following words are used in defining the items used in assembler directives:

- symbol–defined in paragraph 4.4
- label–a symbol used in the label field
- string–a character string defined in paragraph 4.7. of a length defined for each directive
- expr–an expression, defined in paragraph 4.2.1
- const–a constant, defined in paragraph 4.3

Table E-1. Assembler Directives

Directive	Syntax			Note
Procedure Segment	<label>	PSEG		
Data Segment	<label>	DSEG		
Flag Segment	<label>	FSEG		
CRU Symbolic Address Segment	<label>	BSEG	<expr>	
Alternate Mode Registers		MODE		1
Segment Termination		END	[<symbol>]	2
Define Entry Point Symbols		DEF	<symbol> [,<symbol>]. . .	3
Identify External References		REF	<symbol> [,<symbol>]. . .	1
Name Flag Bit Address		FLAG	<symbol> [,<symbol>]. . .	4
Name CRU bit Address	<label>	CON	<expr> [,<expr>]	5
Assign Value to Symbol	<label>	EQU	<expr>	
Format a Source Language Extension	<label>	FRM	<expr> [,<expr>]. . .	6
Reserve Memory	[<label>]	RES	<expr>	7
Place Data in Memory	[<label>]	DATA	<string> [,<string>]. . .	7
Page Eject		PAGE		
Program Identification		TITL	<string>	
Discontinue List Output		UNL		
Resume List Output		LIS		

NOTES

1. Used in procedure and data segments only.
2. The operand of an END directive must be a relocatable value and cannot be an external reference.
3. Used in a procedure segment only.
4. Used in a flag segment only.
5. Used in a CRU symbolic address segment only.
6. The expressions in the operand field must have a sum of 16 or 32.
7. Not used in a CRU symbolic address segment.

APPENDIX F

SAMPLE PROGRAMS

```
** ILLEGAL INSTRUCTION ** ADD
MISSING END
```

```
SAL960  V4L2                                                          PAGE  0001
  11:52:08  JULY01, 1974

SYMBOL  TABLE DUMP        0002 ENTRIES

SYMBOL  VALUE  SSN  FLAG REF REL DEF EXT MUL ILL USED
        0000   0001  F    F   T   T   T   F   F   F
PP      0000   0001  F    F   T   T   T   F   F   F
```

```
SAL960  V4L2                                                          PAGE  0002
  11:52:08  JULY01, 1974

SEGMENT TABLE DUMP        0001 ENTRIES

SEGNAM  BIAS LENGTH REFCNT SSN LINK ABS TYPE
        0000  0000   0000   01   F   F   D

  0002 PASS ONE ERRORS
```

```
SAL960  V4L2                                                          PAGE  0003
  11:52:08  JULY01, 1974

0001 L0000                   DSEG                      *LABEL ERROR
0002 P0000                   DEF  PP                   *PROCEDURE ERROR
0003 S0000 14000000 PP       MOV  (1,2,3),4            *SYNTAX ERROR
0004 M0002 44800000          LA   #2,2                 *MODE ERROR
0005 T0004 20004000          AMI  (0,2),X'5F20'        *TRUNCATION ERROR
0006 E0006 44820000          LA   2,C'ABC'             *EXPRESSION ERROR
0007 N0008 00000000          ADD  4,2                  *MNEUMONIC ERROR
0008 W000A 1C005FF0          CMI  (0,2),X'1FF0'        *WARNING - SIGN CHANGE
0009                                                   *MISSING END CARD
  0009  ERRORS : LENGTH = 000C
```

Figure F-2. Sample Program No 1, Assembly Listing

F.3 SAMPLE PROGRAM NUMBER 2 (FIGURES F-3 THROUGH F-5)

This sample program (figures F-3 through F-5) illustrates a data segment, that contains a worker task block, physical record blocks used in communicating to the supervisory program the desired input/output, and working storage. It also illustrates a procedure segment using flag instructions in the explicit form.

This task copies an input data set to an output data set. The input is initially assumed to be ASCII. An input call is made using the INPRB physical record block which relates the input to logical unit number 5. The flag base register (register 6) has been loaded with the address INPRB at the start of execution by the supervisor from the worker task block. Bit one of the fifth word of INPRB is the input/output error flag. If an error occurs on input, this task resets it and prints a message asking to reread the last input. If the response is the character R, input continues; otherwise, an exit occurs. If an error did not occur on input, the record is then written to the output device assigned to logical unit number 10 defined in the OUTPRB physical record block. If an EOF record was read, the input and output modes are switched (if ASCII then binary; if binary then ASCII) and input continues. If an ASCII record is read with the first three characters the letters END, then the task terminates; otherwise it continues.

F.4 SAMPLE PROGRAM NUMBER 3 (FIGURES F-6 THROUGH F-10)

The following sample program is an example of re-entrant programming. Re-entrant programming on the 960 series computers can be accomplished by referencing all external-to-procedure addresses by base relative addressing. For example:

```
LA    3,@DEG,4
```

references relative address DEG incremented by data base register 4. The driving program or procedure (in this example) is attached to and drives two separate tasks at the same time under a multitask environment of PAM or PAM/D. Three separate assemblies are done; one assembly to generate the procedure and two assemblies to generate the two tasks that are later attached to the procedure through job control. Assume that the two tasks have a task ID and priority (rank) of 80 for task1 and 90 for task2.

The first assembly uses the !*D option that generates all linkage data and only the text data of the procedure segment. The object module generated is the procedure (driver) that is, in turn, attached to the two tasks. The two tasks are identical in that locations having the same displacement relative to the segment base serve identical logic purposes.

The tasks read ASCII coded input records from the physical device assigned to them and write the same records to the output device assigned to them. The second and third assemblies are the same source file as the first assembly, but they use only the data section of the assembly. The only difference between the two tasks is the I/O LUNOs defined in the EQU statements. The first task starts the procedure reading a record from its input logical unit then tests for an error condition and proceedes to write this record out to the output logical unit. When it is waiting for the output unit to finish, the procedure is re-entered and the second task is started. This process continues until all records are read and written by both tasks. Figure F-6 is the flowchart of the program. Figure F-7 is the procedure assembly listing. Figure F-8 is the task one assembly listing. Figure F-9 is the task two assembly listing and figure F-10 is the method used to install the tasks under the operating system.

```
COPY    DSEG COPY ASCII AND BINARY FILES
        DATA COPYP,X'8000',0,0,0,0,0
        DATA COPY,COPYP,INPRB,0,X'8000',COPYP,0,0,0
INPRB   DATA COPY,BUFF,80,0,X'405'
OUTPRB  DATA COPY,BUFF,0,80,X'10'
BUFF    RES  40
ERRMSG  DATA COPY,ERR,0,ERR1-ERR*2,0
ERR     DATA X'0D0A',C'INPUT RECORD ERROR, TYPE R TO RETRY'
ERR1    EQU  $
INCHAR  DATA COPY,CHAR,1,0,X'400'
CHAR    DATA 0
        END
COPYP   PSEG
        LA   3,@INPRB
        SXBS *127
        BFNE (4,1),1,CHKEOF    BRANCH IF NO INPUT ERROR
        SETF (4,1),0           RESET ERROR FLAG
        LA   3,@ERRMSG         PRINT ERROR MSG
        SXBS *127
        LA   3,INCHAR          SEE IF TO RETRY
        SXBS *127
        L    A,CHAR
        NA   A,X'FF00'
        SA   A,X'5200'
        ARB  -1,EXIT,A         WAS R TYPED? NO,EXIT
        B    COPYP             YES,RETRY
CHKEOF  LA   3,@OUTPRB         OUTPUT RECORD
        SXBS *127
        BFNE (4,2),1,CHKEND    BRANCH IF NOT EOF
        BFNE (4,6),1,ASCII     BRANCH IF INPUT WAS ASCII
        SETF (4,6),0           SET INPUT TO ASCII
        SETF (9,6),0           SET OUTPUT TO ASCII
        B    COPYP             GO BACK TO INPUT
ASCII   SETF (4,6),1           SET INPUT TO BINARY
        SETF (9,6),1           SET OUTPUT TO BINARY
        B    COPYP             CONTINUE
CHKEND  BFNE (4,6),0,COPYP     CONTINUE IF BINARY
        L    A,BUFF
        XORA A,X'454E'         IS FIRST WORD = EN
        ARB  -1,COPYP,A        IF NOT, CONTINUE
        L    A,BUFF+1
        NA   A,X'FF00'
        XORA A,X'4400'         IS THIRD CHAR =  D
        ARB  -1,COPYP,A        IF NOT, CONTINUE
EXIT    LA   3,X'1800'         EXIT
        SXBS *127
A       EQU  0
        END
/*
```

Figure F-3. Sample Program No. 2, Input Source File

PAGE 0001

SAL960 V4L2
11.54.24 JULY01, 1974

SYMBOL TABLE DUMP 0015 ENTRIES

SYMBOL	VALUE	SSN	FLAG	REF	REL	DEF	EXT	MUL	ILL	USED
A	0000	0002	F	F	F	T	F	F	F	T
ASCII	0088	0002	F	F	T	T	F	F	F	T
BUFF	001A	0001	F	F	T	T	T	F	F	T
CHAR	005F	0001	F	F	T	T	T	F	F	T
CHKEND	008E	0002	F	F	T	T	F	F	F	T
CHKEOF	007A	0002	F	F	T	T	F	F	F	T
COPY	0000	0001	F	F	T	T	T	F	F	T
COPYP	0060	0002	F	F	T	T	T	F	F	T
ERR	0047	0001	F	F	T	T	T	F	F	T
ERR1	005A	0001	F	F	T	T	T	F	F	T
ERRMSG	0042	0001	F	F	T	T	T	F	F	T
EXIT	009E	0002	F	F	T	T	F	F	F	T
INCHAR	005A	0001	F	F	T	T	T	F	F	T
INPRB	0010	0001	F	F	T	T	T	F	F	T
OUTPRB	0015	0001	F	F	T	T	T	F	F	T

PAGE 0002

SAL960 V4L2
11.54.24 JULY01, 1974

SEGMENT TABLE DUMP 0002 ENTRIES

SEGNAM	BIAS	LENGTH	REFCNT	SSN	LINK	ABS	TYPE
COPY	0000	0060	0000	01	T	F	D
COPYP	0060	0042	0000	02	F	F	P

0000 PASS ONE ERRORS

Figure F-4. Sample Program No. 2, Assembly Listing (Sheet 1 of 3)

```
SAL960  V4L2
  11:54:24  JULY01, 1974                                                PAGE  0003

0001  0000           COPY   DSEG COPY ASCII AND BINARY FILES
0002  0000 0060             DATA COPYP, X'8000', 0, 0, 0, 0, 0
      0001 8000
      0002 0000
      0003 0000
      0004 0000
      0005 0000
      0006 0000
0003  0007 0000             DATA COPY, COPYP, INPRB, 0, X'8000', COPYP.'', 0, 0
      0008 0060
      0009 0010
      000A 0000
      000B 8000
      000C 0060
      000D 0000
      000E 0000
      000F 0000
0004  0010 0000      INPRB  DATA COPY, BUFF, 80, 0, X'405'
      0011 001A
      0012 0050
      0013 0000
      0014 0405
0005  0015 0000      OUTPRB DATA COPY, BUFF, 0, 80, X'10'
      0016 001A
      0017 0000
      0018 0050
      0019 0010
0006  001A           BUFF   RES  40
0007  0042 0000      ERRMSG DATA COPY, ERR, 0, ERR1-ERR*2, 0
      0043 0047
      0044 0000
      0045 0026
      0046 0000
0008  0047 0D0A      ERR    DATA X'0D0A', C'INPUT RECORD ERROR, TYPE R TO RETRY'
      0048 494E5055
      004A 54205245
      004C 434F5244
      004E 20455252
      0050 4F522C20
      0052 54595045
      0054 20522054
      0056 4F205245
      0058 54525920
0009  005A           ERR1   EQU  $
0010  005A 0000      INCHAR DATA COPY, CHAR, 1, 0, X'400'
      005B 005F
      005C 0001
      005D 0000
      005E 0400
0011  005F 0000      CHAR   DATA 0
```

Figure F-4. Sample Program No. 2, Assembly Listing (Sheet 2 of 3)

```
SAL960  V4L2                                                PAGE  0004
  11:54:24   JULY01, 1974

0012  0060                    END
```

```
SAL960  V4L2                                                PAGE  0005
  11:54:24   JULY01, 1974

0013  0060           COPYP  PSEG
0014  0060 44830010         LA    3,@INPRB
0015  0062 7980007F         SXBS  *127
0016  0064 8404181A         BFNE  (4,1),1,CHKEOF    BRANCH IF NO INPUT ERROR
0017  0066 88041000         SETF  (4,1),0           RESET ERROR FLAG
0018  0068 44830042         LA    3,@ERRMSG         PRINT ERROR MSG
0019  006A 7980007F         SXBS  *127
0020  006C 4483005A         LA    3,INCHAR          SEE IF TO RETRY
0021  006E 7980007F         SXBS  *127
0022  0070 4400005F         L     A,CHAR
0023  0072 5880FF00         NA    A,X'FF00'
0024  0074 50805200         SA    A,X'5200'
0025  0076 0C0F009E         ARB   -1,EXIT,A         WAS R TYPED? NO,EXIT
0026  0078 70820060         B     COPYP             YES,RETRY
0027  007A 44830015 CHKEOF  LA    3,@OUTPRB        OUTPUT RECORD
0028  007C 7980007F         SXBS  *127
0029  007E 8404282E         BFNE  (4,2),1,CHKEND    BRANCH IF NOT EOF
0030  0080 84046828         BFNE  (4,6),1,ASCII     BRANCH IF INPUT WAS ASCII
0031  0082 88046000         SETF  (4,6),0           SET INPUT TO ASCII
0032  0084 88096000         SETF  (9,6),0           SET OUTPUT TO ASCII
0033  0086 70820060         B     COPYP             GO BACK TO INPUT
0034  0088 88046800 ASCII   SETF  (4,6),1           SET INPUT TO BINARY
0035  008A 88096800         SETF  (9,6),1           SET OUTPUT TO BINARY
0036  008C 70820060         B     COPYP             CONTINUE
0037  008E 84046000 CHKEND  BFNE  (4,6),0,COPYP     CONTINUE IF BINARY
0038  0090 4400001A         L     A,BUFF
0039  0092 4080454E         XORA  A,X'454E'         IS FIRST WORD = EN
0040  0094 0C0F0060         ARB   -1,COPYP,A        IF NOT, CONTINUE
0041  0096 4400001B         L     A,BUFF+1
0042  0098 5880FF00         NA    A,X'FF00'
0043  009A 40804400         XORA  A,X'4400'         IS THIRD CHAR =  D
0044  009C 0C0F0060         ARB   -1,COPYP,A        IF NOT, CONTINUE
0045  009E 44831800 EXIT    LA    3,X'1800'         EXIT
0046  00A0 7980007F         SXBS  *127
0047  0000           A      EQU   0
0048  00A2                  END
  0000  ERRORS   LENGTH = 00A2
```

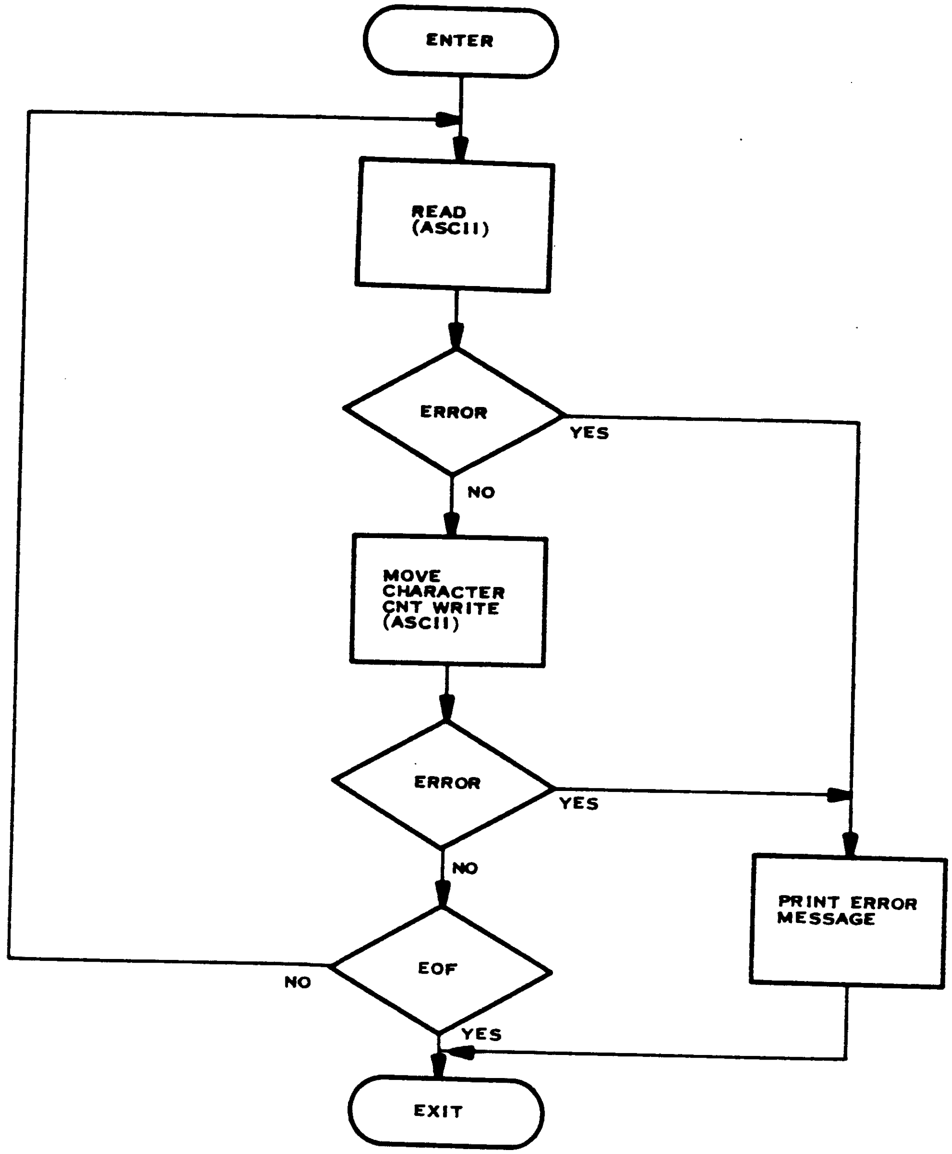
Figure F-4. Sample Program No. 2, Assembly Listing (Sheet 3 of 3)

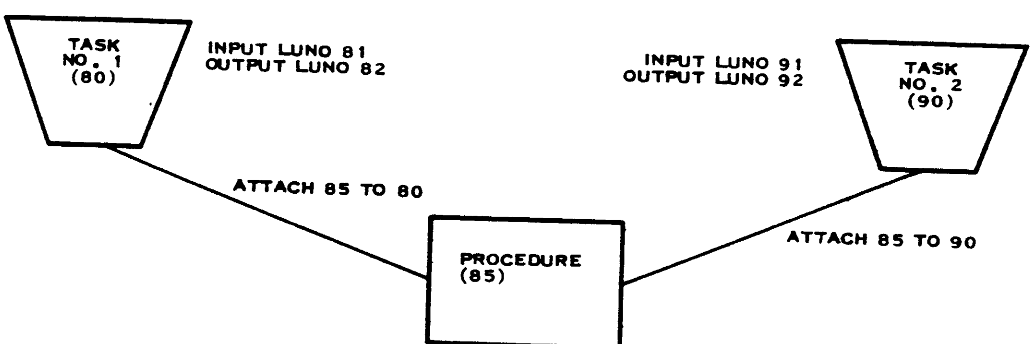
```
OBJECT DUMP
    RECORD NUMBER 0001.

1700  7600  434F  5059  2020  0000  0060  000D
3131  3A35  3131  3A35  343A  4A55  4C59  3031
2C20  3139  3734  2020  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0100  434F  3030  3031

    RECORD NUMBER 0002.

1703  AA08  0000  0000  4255  4646  2020  001A
4348  4152  2020  005F  434F  5059  2020  0000
4552  5220  2020  0047  4552  5231  2020  005A
4552  524D  5347  0042  494E  4348  4152  005A
494E  5052  4220  0010  0100  434F  3030  3032

    RECORD NUMBER 0003.

1703  3001  0000  0000  4F55  5450  5242  0015
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0100  434F  3030  3033

    RECORD NUMBER 0004.

1700  7700  434F  5059  5020  0060  0042  0004
3131  3A35  3131  3A35  343A  4A55  4C59  3031
2C20  3139  3734  2020  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0200  434F  3030  3034

    RECORD NUMBER 0005.

1703  2D01  0000  0000  434F  5059  5020  0060
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0200  434F  3030  3035

    RECORD NUMBER 0006.

1702  3C5A  0000  81C8  C600  0060  8000  0000
0000  0000  0000  0000  0000  0060  0010  0000
8000  0060  0000  0000  0000  0000  001A  0050
0000  0405  0000  001A  0000  0050  0010  0000
0000  0000  0000  0000  0100  434F  3030  3036

    RECORD NUMBER 0007.

1702  9D5E  0042  C000  00C0  0000  0047  0000
0026  0000  0D0A  494E  5055  5420  5245  434F
5244  2045  5252  4F52  2C20  5459  5045  2052
2054  4F20  5245  5452  5920  0000  005F  0001
0000  0400  0000  0000  0100  434F  3030  3037
```

Figure F-5. Sample Program No. 2, Object Output Image (Sheet 1 of 2)

```
     RECORD NUMBER 0008

1702  AC5E  0060  0004  4140  4483  0010  7980
007F  8404  181A  8804  1000  4483  0042  7980
007F  4483  005A  7980  007F  4400  005F  5880
FF00  5080  5200  0C0F  009E  7082  0060  4483
0015  7980  007F  0000  0200  434F  3030  3038

     RECORD NUMBER 0009

1702  905E  007E  0041  1140  8404  282E  8404
6828  8804  6000  8809  6000  7082  0060  8804
6800  8809  6800  7082  0060  8404  6000  4400
001A  4080  454E  0C0F  0060  4400  001B  5880
FF00  4080  4400  0000  0200  434F  3030  3039

     RECORD NUMBER 0010

1702  3B46  009C  4000  0000  0C0F  0060  4483
1800  7980  007F  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0200  434F  3030  3130

OBJECT DUMP
     RECORD NUMBER 0011

1701  1B00  00A2  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0200  434F  3030  3131
```

Figure F-5. Sample Program No. 2, Object Output Image (Sheet 2 of 2)





(A)131864

Figure F-6. ASCII Record Transfer Flowchart

```
SAL960  V4L2  ASCII COPY PROCEDURE                                        PAGE  0001
  11:36:31   AUG.05, 1975

SYMBOL  TABLE DUMP          0012 ENTRIES

SYMBOL   VALUE  SSN  FLAG  REF  REL  DEF  EXT  MUL  ILL  USED
ACOPY    0000   0002  F     F    T    T    T    F    F    F
BUFF     0021   0001  F     F    T    T    T    F    F    T
EOF      0002   0003  T     F    F    T    T    F    F    T
EOP      0016   0002  F     F    T    T    F    F    F    T
ERR      001A   0002  F     F    T    T    F    F    F    T
ERROR    0001   0003  T     F    F    T    T    F    F    T
ERRPRB   001C   0001  F     F    T    T    T    F    F    T
FLAGS    002A   0003  F     F    T    T    T    F    F    F
INLUN    0081   0001  F     F    F    T    T    F    F    T
INPUB    0012   0001  F     F    T    T    T    F    F    T
MESSG    0040   0001  F     F    T    T    T    F    F    T
OUTLUN   0082   0001  F     F    F    T    T    F    F    T
OUTPRB   0017   0001  F     F    T    T    T    F    F    T
READ     0000   0002  F     F    T    T    F    F    F    T
TASK1    0000   0001  F     F    T    T    T    F    F    F
TASKID   0052   0001  F     F    T    T    T    F    F    T
WRITE    000A   0002  F     F    T    T    F    F    F    F
WTB      0000   0001  F     F    T    T    T    F    F    T


SAL960  V4L2  ASCII COPY PROCEDURE                                        PAGE  0002
  11:36:31   AUG.05, 1975

SEGMENT TABLE DUMP          0003 ENTRIES

SEGNAM  BIAS  LENGTH  REFCNT  SSN  LINK  ABS  TYPE
TASK1   0000  000C    0000    01   T     F    D
ACOPY   0000  00[illegible]    0000    02   F     F    P
FLAGS   002A  000C    0000    03   T     F    F

  0000 PASS ONE ERRORS
```

Figure F-7. Procedure Assembly Listing (Sheet 1 of 5)

```
SAL960  V4L2  ASCII COPY PROCEDURE                                        PAGE  0003
  11:36:31  AUG.05, 1975

        0000            TASK1  DSEG
                               TITL   ASCII COPY PROCEDURE
                        *  THIS IS A TASK TO BE ATTACHED TO A PROCEDURE
                        *   AND USED FOR COPYING ASCII RECORDS FROM AN INPUT
                        *   LUNO TO AN OUTPUT LUNO.
                        *
                        * INPUT LUNO:
        0081            INLUN  EQU  X'81'
                        * OUTPUT LUNO:
        0082            OUTLUN EQU  X'82'
                        *
                        *
                        *   WORKER TASK BLOCK:
                        *
       0000 0000        WTB     DATA 0,0,0
       0001 0000
       0002 0000
       0003 0000                DATA 0,0,0,0,WTB          REGISTERS
       0004 0000
       0005 0000
       0006 0000
       0007 0000
       0008 0000                DATA 0,0,0
       0009 0000
       000A 0000
       000B 8000                DATA X'8000',0            ENTRY STATUS + EC
       000C 0000
       000D 0000                DATA 0,0,0,0,0
       000E 0000
       000F 0000
       0010 0000
       0011 0000
                        *
                        *   PHYSICAL RECORD BLOCKS FOR I/O
                        *
       0012 0000        INPRB  DATA WTB,BUFF,80,0
       0013 0021
       0014 0050
       0015 0000
       0016 0481               DATA INLUN+X'0400'         SET FLAGS FOR ASCII READ
       0017 0000        OUTPRB DATA WTB,BUFF,80,0
       0018 0021
       0019 0050
       001A 0000
       001B 0082               DATA OUTLUN                FLAGS SET FOR ASCII WRITE
       001C 0000        ERRPRB DATA WTB,MESSG,33,33
       001D 0049
       001E 0021
       001F 0021
       0020 0000               DATA 0                     ASCII OUT TO CONSOLE
```

Figure F-7. Procedure Assembly Listing (Sheet 2 of 5)

```
SAL990  V4L2  ASCII COPY PROCEDURE                                          PAGE  0004
  11:36:31  AUG.05, 1975

0020                    *
0030                    *    BUFFERS:
0031                    *
0032  0018              BUFF   RES  40
0033  0040 492F4F20 MESSG  DATA C'I/O ERROR - TASK  '
      0044 4552524F
      0048 52202D20
      004C 5441534B
      0050 2020
0034  0052              TASKID RES  2                     4 CHAR FOR TASK ID
0035  0054 20544552        DATA C' TERMINATES'
      0058 4D494E41
      005C 54455320
0036  0000                 END
```

Figure F-7. Procedure Assembly Listing (Sheet 3 of 5)

```
SAL060  V4L2  ASCII COPY PROCEDURE                                    PAGE  0005
  11:36:31  AUG,05, 1975

        0000              ACOPY   PSEG
                          *
                          * OBJECTIVE:       THIS IS A RE-ENTRANT PROCEDURE THAT
                          *                  EXECUTES UNDER PAM OR PAMD AND COPIES
                          *                  VARIABLE LENGTH ASCII RECORDS FROM AN
                          *                  INPUT LUNO TO AN OUTPUT LUNO.
                          * ERRORS:          AN ERROR CONDITION ON INPUT OR OUTPUT
                          *                  PRINTS A MESSAGE ON THE SYSTEM CONSOLE
                          *                  (LUNO 0) AND TERMINATES THE TASK WITH AN
                          *                  END-OF-PROGRAM CALL.
                          * TERMINATION:     THE TASK TERMINATES NORMALLY AFTER AN
                          *                  END-OF-FILE IS COPIED FROM INPUT TO
                          *                  OUTPUT.
                          *  REGISTERS 4 (DATA BASE) AND 5 (PROCEDURE BASE) ARE
                          *   INITIALIZED THROUGH THE WTB WHEN THE APPROPRIATE TASK
                          *   IS ATTACHED AND EXECUTED.
                          *
                          *
        0000 44030012     READ    LA    3,@INPRB
        0002 7900007F             SXRS  *127                      READ A RECORD
                          *
        0004 46C60016             LA    6,@INPRB+4,4              SET FLAG BASE
        0006 8400101A             BFNE  ERROR,0,ERR               BRANCH ON ERROR
                          *
        0008 1415901A             MOV   INPRB+3,OUTPRB+3          MOVE CHAR COUNT
                          *
        000A 44030017     WRITE   LA    3,@OUTPRB
        000C 7900007F             SXRS  *127                      WRITE A RECORD
                          *
        000E 46C60018             LA    6,@OUTPRB+4,4             SET FLAG BASE
        0010 8400101A             BFNE  ERROR,0,ERR               BRANCH ON ERROR
        0012 84002800             BFNE  EOF,1,READ                FALL THRU ON EOF
                          *
        0014 80002800             SETF  EOF,0                     RESET EOF FLAG
        0016 44030400     EOP     LA    3,X'400'
        0018 7900007F             SXRS  *127                      END OF PROGRAM
                          *
                          *
        001A              ERR     EQU   $
        001A 80001000             SETF  ERROR,0                   RESET ERROR FLAG
        001C 4640000F             L     0,15,4                    GET PROC/TASK ID FROM WTB
        001E 580000FF             NA    0,X'FF'                   MASK OFF PROC ID
        0020 44032052             LA    3,X'2000'+@TASKID         CONVERT TO ASCII AND
        0022 7900007F             SXRS  *127                      PUT TASK ID IN MESSAGE
                          *
        0024 4403001C             LA    3,@ERRPRB
        0026 7900007F             SXRS  *127                      PRINT ERROR MESSAGE
        0028 70020016             B     EOP                       GO TERMINATE PROGRAM
        002A                      END
```

Figure F-7. Procedure Assembly Listing (Sheet 4 of 5)

```
SAL960  V4L2  ASCII COPY PROCEDURE                                PAGE  0000
  11:36:31  AUG.05, 1978

0O2A            FLAGS  FSFG
                *
                *   SET UP FLAG DEFINITION FOR PRB WORD-4 FOR
                *   ERROR BIT AND END-OF-FILE BIT.
                *
0000                   FLAG 1,ERROR,EOF
002A                   END
  0000  ERRORS : LENGTH = 002A
```

Figure F-7. Procedure Assembly Listing (Sheet 5 of 5)

```
SAL960  V4L2   TASK #1 - ASCII COPY ROUTINE                                    PAGE  0001
  12:32:10  AUG.05, 1975
```

SYMBOL TABLE DUMP 0010 ENTRIES

SYMBOL	VALUE	SSN	FLAG	REF	REL	DEF	EXT	MUL	ILL	USED
BUFF	0021	0001	F	F	T	T	T	F	F	T
ERRPRA	001C	0001	F	F	T	T	T	F	F	F
INLUN	0081	0001	F	F	F	T	T	F	F	T
INPRB	0012	0001	F	F	T	T	T	F	F	F
MESSG	0040	0001	F	F	T	T	T	F	F	T
OUTLUN	0082	0001	F	F	F	T	T	F	F	T
OUTPRA	0017	0001	F	F	T	T	T	F	F	F
TASK1	0000	0001	F	F	T	T	T	F	F	F
TASKID	0052	0001	F	F	T	T	T	F	F	F
WTB	0000	0001	F	F	T	T	T	F	F	T

```
SAL960  V4L2   TASK #1 - ASCII COPY ROUTINE                                    PAGE  0002
  12:32:10  AUG.05, 1975
```

SEGMENT TABLE DUMP 0001 ENTRIES

SEGNAM	BIAS	LENGTH	REFCNT	SSN	LINK	ABS	TYPE
TASK1	0000	0059	0000	01	T	F	D

0000 PASS ONE ERRORS

Figure F-8. Task One Assembly Listing (Sheet 1 of 3)

```
SAL980 V4L2   TASK #1 - ASCII COPY ROUTINE
  12:32:10  AUG.05, 1975

      0000                   TASK1  DSEG
                                    TITL  TASK #1 - ASCII COPY ROUTINE
                             *  THIS IS A TASK TO BE ATTACHED TO A PROCEDURE
                             *   AND USED FOR COPYING ASCII RECORDS FROM AN INPUT
                             *   LUNO TO AN OUTPUT LUNO.
                             *
                             * INPUT LUNO:
      0081                   INLUN  EQU  X'81'
                             * OUTPUT LUNO:
      0082                   OUTLUN EQU  X'82'
                             *
                             *
                             *   WORKER TASK BLOCK:
                             *
      0000 0000              WTB    DATA 0,0,0
      0001 0000
      0002 0000
      0003 0000                     DATA 0,0,0,0,WTB             REGISTERS
      0004 0000
      0005 0000
      0006 0000
      0007 0000
      0008 0000                     DATA 0,0,0
      0009 0000
      000A 0000
      000B 8000                     DATA X'8000',0               ENTRY STATUS + EC
      000C 0000
      000D 0000                     DATA 0,0,0,0,0
      000E 0000
      000F 0000
      0010 0000
      0011 0000
                             *
                             *   PHYSICAL RECORD BLOCKS FOR I/O
                             *
      0012 0000              INPRB  DATA WTB,BUFF,80,0
      0013 0021
      0014 0050
      0015 0000
                                    DATA INLUN+X'0480'           SET FLAGS FOR ASCII READ
      0016 0481
      0017 0000              OUTPRB DATA WTB,BUFF,80,0
      0018 0021
      0019 0050
      001A 0000
                                    DATA OUTLUN                  FLAGS SET FOR ASCII WRITE
      001B 0082
      001C 0000              ERRPRB DATA WTB,MESSG,33,33
      001D 0049
      001E 0021
      001F 0021                                                  ASCII OUT TO CONSOLE
      0020 0000                     DATA 0
```

Figure F-8. Task One Assembly Listing (Sheet 2 of 3)

```
SAL06A  VAL2   TASK #1 - ASCII COPY ROUTINE                                      PAGE  0004
  12:32:10  AUG.05, 1975

0029                        *
0030                        *    BUFFERS:
0031                        *
0032  0021                  BUFF    RES  40
0033  0049 492F4F20 MESSG   DATA C'I/O ERROR - TASK  '
      004B 4552524F
      004D 52202D20
      004F 5441534B
      0051 2020
0034  0052                  TASKID RES  2                       4 CHAR FOR TASK ID
0035  0054 20544552         DATA C' TERMINATES'
      0056 4D494E41
      0058 54455320
0036  005A                  END
  0000  ERRORS : LENGTH = 005A
```

Figure F-8. Task One Assembly Listing (Sheet 3 of 3)

```
SAL960  V4L2   TASK #2 - ASCII COPY ROUTINE                                   PAGE  0001
  12:38:42  AUG.05, 1975

SYMBOL  TABLE DUMP         0010 ENTRIES

SYMBOL  VALUE   SSN  FLAG REF REL DEF EXT MUL ILL USED
BUFF    0021   0001    F   F   T   T   T   F   F   T
ERRPRB  001C   0001    F   F   T   T   T   F   F   F
INLUN   0091   0001    F   F   F   T   T   F   F   T
INPRB   0012   0001    F   F   T   T   T   F   F   F
MESSG   0040   0001    F   F   T   T   T   F   F   T
OUTLUN  0092   0001    F   F   F   T   T   F   F   T
OUTPRB  0017   0001    F   F   T   T   T   F   F   F
TASK2   0000   0001    F   F   T   T   T   F   F   F
TASKID  0052   0001    F   F   T   T   T   F   F   F
WTB     0000   0001    F   F   T   T   T   F   F   T



SAL960  V4L2   TASK #2 - ASCII COPY ROUTINE                                   PAGE  0002
  12:38:42  AUG.05, 1975

SEGMENT TABLE DUMP         0001 ENTRIES

SEGNAM  BIAS LENGTH REFCNT SSN LINK ABS TYPE
TASK2   0000  0053    0000  01   T   F   D

  0000 PASS ONE ERRORS
```

Figure F-9. Task Two Assembly Listing (Sheet 1 of 3)

```
SAL960   V4L2   TASK #2 - ASCII COPY ROUTINE                              PAGE  0003
  12:38:42  AUG.05, 1975

          0000               TASK2  DSEG
                                    TITL  TASK #2 - ASCII COPY ROUTINE
                             *  THIS IS A TASK TO BE ATTACHED TO A PROCEDURE
                             *   AND USED FOR COPYING ASCII RECORDS FROM AN INPUT
                             *   LUNO TO AN OUTPUT LUNO.
                             *
                             * INPUT LUNO:
          0091               INLUN  EQU  X'91'
                             * OUTPUT LUNO:
          0092               OUTLUN EQU  X'92'
                             *
                             *
                             *   WORKER TASK BLOCK:
                             *
          0000 0000          WTB    DATA 0,0,0
          0001 0000
          0002 0000
          0003 0000                 DATA 0,0,0,0,WTB           REGISTERS
          0004 0000
          0005 0000
          0006 0000
          0007 0000
          0008 0000                 DATA 0,0,0
          0009 0000
          000A 0000
          000B 8000                 DATA X'8000',0             ENTRY STATUS + PC
          000C 0000
          000D 0000                 DATA 0,0,0,0,0
          000E 0000
          000F 0000
          0010 0000
          0011 0000
                             *
                             *   PHYSICAL RECORD BLOCKS FOR I/O
                             *
          0012 0000          INPRB  DATA WTB,BUFF,80,0
          0013 0021
          0014 0050
          0015 0000
          0016 0491                 DATA INLUN+X'0400'         SET FLAGS FOR ASCII READ
          0017 0000          OUTPRB DATA WTB,BUFF,80,0
          0018 0021
          0019 0050
          001A 0000
          001B 0092                 DATA OUTLUN                FLAGS SET FOR ASCII WRITE
          001C 0000          ERRPRB DATA WTB,MESSG,33,33
          001D 0049
          001E 0021
          001F 0021
          0020 0000                 DATA 0                     ASCII OUT TO CONSOLE
```

Figure F-9. Task Two Assembly Listing (Sheet 2 of 3)

```
SAL960  V4L2   TASK #2 - ASCII COPY ROUTINE                          PAGE  0004
  12:38:42  AUG.05, 1975

0029                    *
0030                    *    BUFFERS:
0031                    *
0032  0021              BUFF    RES  40
0033  0049 492F4F20 MESSG   DATA C'I/O ERROR - TASK   '
      004B 4552524F
      004D 52202020
      004F 5441534B
      0051 2020
0034  0052              TASKID RES  2                     4 CHAR FOR TASK ID
0035  0054 20544552         DATA C' TERMINATES'
      0056 4D494E41
      0058 54455320
0036  005A                  END
  0000  ERRORS : LENGTH = 005A
```

Figure F-9. Task Two Assembly Listing (Sheet 3 of 3)

```
$$LDTS◆◆0100◆◆0080      LOAD TASK 80 AT LOCATION 100
/◆
$$INST◆◆0080◆◆0080      INSTALL TASK 80 WITH PRIORITY OF 80
$$ABLE◆◆0080            ABLE TASK 80
$$LDTS◆◆0600◆◆0090      LOAD TASK 90 AT LOCATION 600
/◆
$$INST◆◆0090◆◆0090      INSTALL TASK 90 WITH PRIORITY OF 90
$$ABLE◆◆0090            ABLE TASK 90
$$LDPR◆◆1000◆◆0085      LOAD PROCEDURE 85 AT LOCATION 1000
/◆
$$ATCH◆◆0085◆◆0080      ATTACH PROCEDURE 85 TO TASK 80
$$ATCH◆◆0085◆◆0090      ATTACH PROCEDURE 85 TO TASK 90
```

Figure F-10. Task Installation under PAM

APPENDIX G

INSTRUCTION INDEX

Instruction Index

Mnemonic	Paragraph	Mnemonic	Paragraph
A	3.4.1.1	NA	3.4.1.27
AA	3.4.1.2	NOP	3.4.3.4
ADAC	3.4.5.1	OR	3.4.1.28
AMI	3.5.3.1	ORA	3.4.1.29
ARB	3.4.4.1	S	3.4.1.30
B	3.4.3.1	SA	3.4.1.3.1
BC	3.4.2.1	SAT	3.4.1.32
BBNE	3.6.4.1	SETB	3.6.3.1
BFNE	3.6.2.1	SETF	3.6.1.1
BL	3.4.1.3	SS	3.4.3.5
BRRL	3.5.2	SSB	3.4.3.6
CM	3.5.1.1	ST	3.4.1.33
CMI	3.5.3.2	STCR	3.6.6.2
CML	3.5.1.2	STPS	3.4.6
CR	3.4.1.5	SXBS	3.4.3.7
CRA	3.4.1.6	SXBW	3.4.3.8
CRL	3.4.1.7	SXS	3.4.3.9
CRLA	3.4.1.8	SXW	3.4.3.10
L	3.4.1.17	TSBX	3.6.5.1
LA	3.4.1.18	XBNE	3.6.3.2
LDCR	3.6.6.1	XFNE	3.6.1.2
LDS	3.4.3.3	XOR	3.4.1.34
LOT	3.4.1.19	XORA	3.4.1.35
LOTA	3.4.1.20	XS	3.4.3.11
MLA	3.4.2.7	XSB	3.4.3.12
MLAX	3.4.1.23	XW	3.4.3.14
MOV	3.5.1.3	XWB	3.4.3.15
MRA	3.4.2.8	*B	3.4.3.2
MRAX	3.4.1.24	*BC	3.4.2.2
MRR	3.4.2.9	*BL	3.4.1.4
MRRX	3.4.1.25	*XSB	3.4.3.13
N	3.4.1.26	*XWB	3.4.3.16

Optional Instructions

DAD	3.4.1.11
D	3.4.1.9
DA	3.4.1.10
DLA	3.4.2.3
DLAX	3.4.1.12
DRA	3.4.2.4
DRAX	3.4.1.13
DRL	3.4.2.5
DRLX	3.4.1.14
DRR	3.4.2.6
DRRX	3.4.1.15
M	3.4.1.21
MA	3.4.1.22
DS	3.4.1.16

APPENDIX H

JOB CONTROL STREAM FOR CREATING AN SWJC FILE TO ASSEMBLE A PROGRAM UNDER PAM/D

APPENDIX H

JOB CONTROL STREAM FOR CREATING AN SWJC FILE TO ASSEMBLE A PROGRAM UNDER PAM/D

Using the copy ASCII files task (ACOPYD), an SWJC file of the necessary Job Control statements to assemble a program using SAL/D may be created. An example of this procedure follows:

1.	$$RLIO**0005	
2.	$$DFIO**0005**0005	ACOPYD input LUNO to card reader
3.	$$RLIO**0087	
4.	$$DFSF**0087**4000**407F**0001	Define SWJC file
5.	$$RDFL**0087**0010	ACOPYD output to SWJC file
6.	$$EXCT**000C	Execute copy task
7.	$$SOCS**0000	Set SOCS flag = 0
8.	$$RLIO**0007	
9.	$$RLIO**0010	
10.	$$DFSF**0010**5000**5FFF**0002	Define scratch file
11.	$$DFSF**0007**6000**6FFF**0002	Define object output file
12.	$$RLBG**	Release background
13.	$$DFBG**3000**0020**00D0	Define background
14.	$$ABLE**0009	
15.	$$EXCT**0009	Execute SAL/D assembler
16.	$$RWND**0007	Rewind object file
17.	$$RLBG**	
18.	$$SWJC**0087	Exit SWJC
19.	$$RWND**0002	
20.	/*	
21.	$$RDFL**0010**0087	
22.	$$RWND**0087	Rewind SWJC file

Statements numbered 7 through 19 are the Job Control statements contained in SWJC file 87. The other statements create the SWJC file.

ALPHABETICAL INDEX

ALPHABETICAL INDEX

INTRODUCTION

The following index lists key words and concepts from the subject material of the manual together with the area(s) in the manual that supply major coverage of the listed concept. The numbers along the right side of the listing reference the following manual areas:

- Sections - References to Sections of the manual appear as "Section x" with the symbol x representing any numeric quantity.
- Appendixes - References to Appendixes of the manual appear as "Appendix y" with the symbol y representing any capital letter.
- Paragraphs - References to paragraphs of the manual appear as a series of alphanumeric or numeric characters punctuated with decimal points. Only the first character of the string may be a letter; all subsequent characters are numbers. The first character refers to the section or appendix of the manual in which the paragraph is found.
- Tables - References to tables in the manual are represented by the capital letter T followed immediately by another alphanumeric character (representing the section or appendix of the manual containing the table). The second character is followed by a dash (-) and a number:

  Tx-yy

- Figures - References to figures in the manual are represented by the capital letter F followed immediately by another alphanumeric character (representing the section or appendix of the manual containing the figure). The second character is followed by a dash (-) and a number:

  Fx-yy

- Other entries in the Index - References to other entries in the index are preceded by the word "See" followed by the referenced entry.

USER'S RESPONSE SHEET

Manual Title: Model 960 Computer Assembly Language Programmer's Reference Manual (942779-9701)

Manual Date: 15 May 1979 **Date of This Letter:** ______

User's Name: ______ **Telephone:** ______

Company: ______ **Office/Department:** ______

Street Address: ______

City/State/Zip Code: ______

Please list any discrepancy found in this manual by page, paragraph, figure, or table number in the following space. If there are any other suggestions that you wish to make, feel free to include them. Thank you.

Location in Manual **Comment/Suggestion**

NO POSTAGE NECESSARY IF MAILED IN U.S.A.
FOLD ON TWO LINES (LOCATED ON REVERSE SIDE), TAPE AND MAIL

CUT ALONG LINE

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 7284 DALLAS, TX

POSTAGE WILL BE PAID BY ADDRESSEE

TEXAS INSTRUMENTS INCORPORATED
DIGITAL SYSTEMS GROUP

ATTN: TECHNICAL PUBLICATIONS
P.O. Box 2909 M/S 2146
Austin, Texas 78769

FOLD

TEXAS INSTRUMENTS
INCORPORATED
DIGITAL SYSTEMS GROUP
POST OFFICE BOX 2909 AUSTIN, TEXAS 78769