

88A00508A-E

GA-16/110/220 system reference manual

GENERAL AUTOMATION

1055 South East Street
Anaheim, California 92805
(714) 778-4800
TWX 910-591-1695

© 1979, General Automation, Inc.

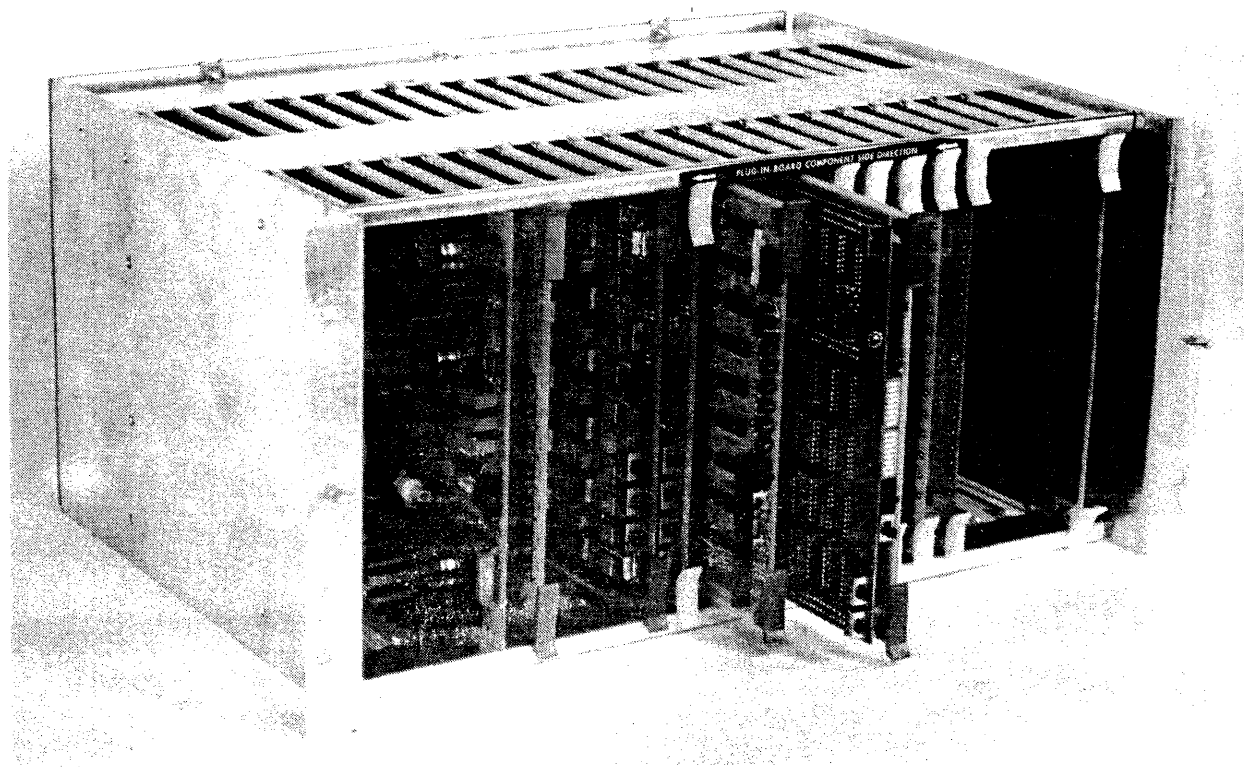
RECORD OF REVISIONS

Title: GA-16/110/220 System Reference Manual

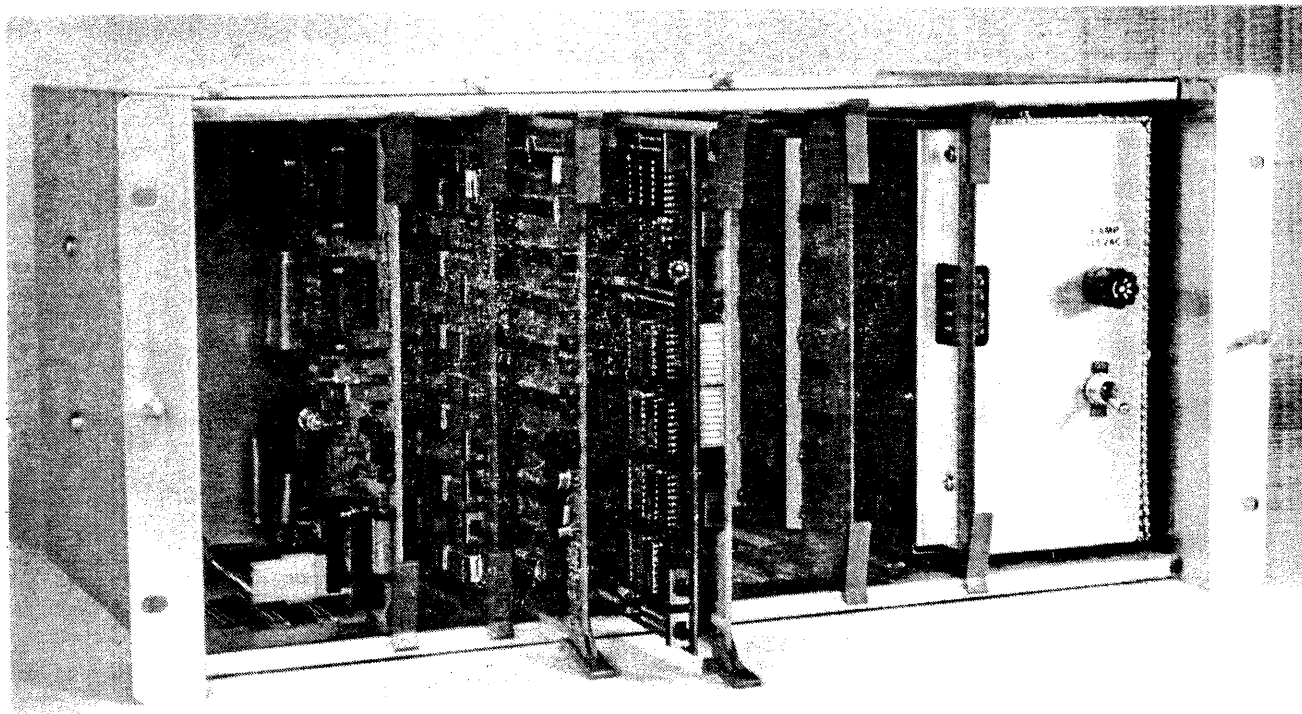
Document Number: 88A00508A

DATE	ISSUE
July 76	Preliminary Issue
Dec. 76	Revision B Released issue revised to include GA-16/110 computer information.
June 77	Revision C
Sept 77	Change 1 Pages Affected are: 1-11 2-14 2-15 4-38 4-43 4-85 6-45 A-5 A-7
Jan 78	Revision D Incorporates Change Package C1 and change on page: 4-134
Mar 79	Revision E

88A00508A-E



GA-16/220 IN JUMBO CHASSIS



GA-16/220 IN COMPACT CHASSIS

FOREWORD

This GA-16/110/220 System Reference manual describes the configuration, operation, programming, and input/output interfaces of both the GA-16/110 and GA-16/220 computers. Both computers are similar in their basic design, configurations, and programming and are described in one manual. The manual is intended for use as follows:

- For the original equipment manufacturer who will incorporate a GA-16/110 or a GA-16/220 into his product, particular emphasis is placed upon flexibility of configurations and I/O controller design considerations.
- For a system programmer who will be designing, maintaining, or adapting software for use on a GA-16/110 or GA-16/220, all references to instructions are in terms of both machine code and GA's CAP-16 Assembler.
- For general reference to operating procedures.

The following manuals are recommended for GA-16/110/220 hardware maintenance and supplemental software information:

- GA-16/110/220 Hardware Maintenance Manual (88A00509A)
- System Generation & Operator Interface for FSOS or a GA-16/220/330 (94A01563A)
- Disk Drum-Based Operating System, DBOS-16 (88A00115A)
- Real-Time Executive System, RTX-16 (88A00114A)
- Real-Time Operating System, RTOS-16 (88A00230A)
- How to Use Your GA-16/220 (88A00525A)
- Series-16 CAP-16 Assembler (88A00481A)
- Series-16 Input/Output System (IOS) (88A00471A)
- Series-16 I/O Controller Manuals (as required for peripherals used)
- GA-16/220/330 Stand-Alone Utilities (94A01531A)
- Execution of T&V Programs on the GA-16 Series Computers (94A01519A)

The basic unit for both computers is the GA-16/110 CPU module, which provides a microprogrammed CPU to carry out the GA-16/SPC-16 series instruction set. The GA-16/110 provides controls and indicators, memory bus, programmed I/O bus, vectored priority interrupts, cold start, power fail/auto restart, and operations monitor; alarm. An optional piggyback memory module plugs into the GA-16/110 module to provide 2K words of random access memory (RAM) (or combination RAM and erasable programmable read-only-memory (EPROM). An optional 64-word ROM may be installed on the piggyback memory to provide initial program load (IPL) capability from a peripheral device. A complete GA-16/110 computer (with memory) can therefore, be provided on a single printed circuit module for insertion into a users device.

A GA-16/220 computer consists of the GA-16/110 module (CPU-1) plus a second module (CPU-2 module) which adds additional controls including data entry switches that may be sensed by a users program, and a vectored console interrupt button. The CPU-2 module also provides a 1 millisecond real-time clock (RTC), memory mode change under

program control, and control instructions for I/O reset and single-step execution. The CPU-2 module enlarges the I/O capabilities to include a built-in serial I/O controller (with vectored interrupt) for TTY or CRT.

An optional system console interface module (SCI) may be plugged into the CPU-2 module to provide an interactive operator program via TTY or CRT (console ROM). An optional IPL ROM may be installed on the SCI to provide program load capabilities from a variety of peripheral devices.

Both the GA-16/110 and GA-16/220 computers may be installed in either a compact or jumbo enclosure which provides printed circuit card slots and connectors for the GA-16/110 CPU (CPU-1) and the 220 module (CPU-2). In addition, the enclosure also provides slots and connectors for the installation of 4K or 8K dynamic RAM memory modules, a memory parity and write protect (MPP) module, and a backup memory power supply. I/O controller slots are also provided together with rear interface connectors for peripheral devices and provision for the connection of an I/O expansion chassis. The compact enclosure provides a compartment for a main power supply to run the CPU and I/O controllers, while the jumbo enclosure requires an external power supply unit.

Specific differences between the two computers are defined in the text, and the term "GA-16/110/220" is generally used where characteristics are shared.

CAUTION

Several standard GA controllers have had to be updated for compatibility with both the SPC-16 and GA-16 series processors. The controllers affected are listed below:

<u>Model No.</u>	<u>Name</u>
1615-xxxx	CIT-16
3346-3347-xxxx	Disk Controller
1615-xxxx0208	MHSDC
1615-xxxx0230	ABTU
1615-0240	APU
1561-xxxx	Asynchronous Communication Controller
1581-xxxx	Asynchronous Communication Controller
1574-xxxx	DMA Asynchronous Communication Controller
1530-xxxx	DMA MUX
1579-xxxx	SDLC
1582-xxxx	Console Controller
1571-xxxx	Paddle Board (Loop Back)

Controllers constructed prior to May 1, 1976, not reflect the proper changes. Any of these controllers, not incorporating the necessary modifications, are not compatible with the GA-16/440, 16/330, 16/220, and 16/110 processors.

Controllers constructed after May 1, 1976, are fully compatible with all GA-16 series and SPC-16 series processors.

88A00508A-E

CAUTION

Before applying power to either a GA-16/110 or a GA-16/220 configured in a compact or jumbo enclosure, verify that plug-in modules are inserted correctly (component side to the left). . If a module is inserted upside down, severe damage to the equipment will occur. The user should also verify that the memory service module (if used) and the serial I/O paddleboard (on a GA-16/220) are inserted in the correct rear connectors. A first-time user of a GA-16/220 should read "How to Use Your GA-16/220" (88A00525A) and run the CPU T&V Program.

TABLE OF CONTENTS

Section	Title	Page
1	INTRODUCTION.	1-1
1.1	SYSTEM ARCHITECTURE.	1-6
1.2	STANDARD FEATURES OF THE GA-16/110 AND GA-16/220 SYSTEMS . . .	1-7
1.2.1	MICROCONSOLE AND OTHER CONTROLS AND INDICATORS.	1-7
1.2.1.1	Microconsole for GA-16/110 System.	1-7
1.2.1.2	Microconsole for GA-16/220 System.	1-8
1.2.1.3	Additional Controls and Indicators for GA-16/110 and GA-16/220.	1-9
1.2.2	MEMORY.	1-10
1.2.2.1	Auxiliary RAMs and ROMs.	1-10
1.2.2.2	EPROM.	1-10
1.2.3	ADDRESSING MODES.	1-11
1.2.4	GENERAL-PURPOSE REGISTERS	1-11
1.2.5	INPUT/OUTPUT SYSTEM	1-12
1.2.6	INTERRUPT SYSTEM.	1-13
1.2.7	SPECIAL FEATURES.	1-14
1.3	OPTIONAL FEATURES OF THE GA-16/110 AND GA-16/220 SYSTEMS . . .	1-15
1.3.1	MEMORY PARITY PROTECT (MPP)	1-15
1.3.1.1	Memory Parity.	1-15
1.3.1.2	Memory Write Protect	1-16
1.3.1.3	Program Timeout.	1-16
1.3.1.4	Error Corrections.	1-16
1.3.2	SYSTEM CONSOLE INTERFACE (GA-16/220 ONLY)	1-17
1.4	PERIPHERALS AND CONTROLLERS.	1-18
1.5	SOFTWARE	1-19
2	SYSTEM ORGANIZATION	2-1
2.1	ELEMENTS OF THE CPU	2-1
2.2	WORD FORMAT	2-1
2.3	DATA FORMAT	2-4
2.3.1	ARITHMETIC QUANTITY (SIGN BIT)	2-4
2.3.2	LOGICAL WORDS (NO SIGN BIT)	2-5
2.3.3	ADDRESSES AND MEMORY MODE	2-7
2.3.4	BYTE DATA	2-7
2.3.5	BIT DATA	2-8
2.4	ELEMENTS OF THE CENTRAL PROCESSING UNIT (CPU)	2-8
2.4.1	THE ARITHMETIC/LOGICAL AND CONTROL SECTION	2-8
2.4.2	SEMICONDUCTOR MEMORY	2-10
2.4.3	SYSTEM CONSOLE INTERFACE (SCI) (GA-16/220 ONLY)	2-11
2.4.4	MEMORY PARITY PROTECT (MPP)	2-11
2.4.5	INPUT/OUTPUT (I/O) CONTROL	2-11

Section	Title	Page
2.5	MAJOR REGISTERS.	2-12
2.5.1	GENERAL-PURPOSE REGISTERS	2-12
2.5.2	ISE AND INTERRUPTS.	2-14
2.5.3	STATUS REGISTER (REGISTER S).	2-14
2.5.4	PROGRAM COUNTER (REGISTER P).	2-17
2.5.5	REGISTER I (NON-PROGRAMMABLE)	2-18
2.5.6	TA TRI-STATE LATCH.	2-18
2.5.7	REGISTER W (NON-PROGRAMMABLE)	2-18
2.6	HARDWARE INTERRUPT (NI AND IN)	2-19
2.6.1	IN INTERRUPTS	2-22
2.6.2	NI INTERRUPTS	2-23
2.7	ERROR CORRECTION OPTION.	2-24
2.7.1	OPERATIONAL MODES	2-24
	2.7.1.1 Input Status Mode.	2-25
2.7.2	MODE SELECTION.	2-26
2.7.3	ERROR CORRECTION INTERRUPTS	2-26
	2.7.3.1 Correctable Error Interrupt.	2-26
	2.7.3.2 Non-Correctable Error Interrupt.	2-26
2.7.4	MPP STATUS WORD	2-26
	2.7.4.1 ECC Board Status Indicators.	2-26
3	GA-16/220 CONFIGURATION AND OPERATION	3-1
3.1	TYPICAL GA-16/110/220 CONFIGURATIONS	3-1
3.1.1	SELF-CONTAINED GA-16/110 SYSTEM	3-2
3.1.2	SELF-CONTAINED GA-16/220 SYSTEM	3-3
3.1.3	LARGE GA-16/110 SYSTEM WITH MEMORY PARITY PROTECT OPTION	3-3
3.1.4	LARGE GA-16/220 SYSTEM WITH MEMORY PARITY PROTECT OPTION	3-4
3.2	MODULE DESCRIPTION	3-11
3.2.1	POWER SUPPLY	3-11
3.2.2	CPU-2 MODULE	3-11
3.2.3	SYSTEM CONSOLE INTERFACE (SCI) MODULE (GA-16/220)	3-11
3.2.4	MULTI-DEVICE INITIAL PROGRAM LOAD READ-ONLY-MEMORY (IPL ROM) FOR GA-16/220	3-12
3.2.5	CPU-1 MODULE	3-12
3.2.6	MEMORY PARITY PROTECT (MPP) MODULE	3-12
3.2.7	MEMORY MODULES	3-13
	3.2.7.1 General Description of Memory Modules	3-13
	3.2.7.2 Setting Memory Addresses	3-14
3.2.8	BACKUP POWER SUPPLY (BATTERY BACKUP)	3-18
3.3	CONTROLS AND INDICATORS	3-18
3.4	START-UP AND PROGRAM LOAD	3-25
3.4.1	START-UP OF GA-16/220 GENERAL-PURPOSE SYSTEM	3-27
	3.4.1.1 Preconditions for System Start-Up	3-27
	3.4.1.2 Verify Memories Are Powered	3-27
	3.4.1.3 Main Power Application Procedure	3-28

Section	Title	Page
3.4.2	START-UP OF GA-16/110 OR GA-16/220 DEDICATED SYSTEM WITHOUT SCI	3-28
3.4.2.1	Preconditions for System Start-Up	3-28
3.4.2.2	Verify Memories are Powered	3-29
3.4.2.3	Main Power Turn-On	3-29
3.5	IPL ROMS	3-30
3.5.1	PGS AND MINI-PGS FORMAT	3-30
3.5.2	SINGLE-DEVICE ROM ON GA-16/110 AND DEDICATED GA-16/220	3-31
3.5.3	GA-16/220 MULTI-DEVICE INITIAL PROGRAM LOAD ROM ON SCI	3-32
3.5.3.1	Teletype (TTY)	3-32
3.5.3.2	High-Speed Paper Tape Reader (PTR)	3-33
3.5.3.3	Card Reader (CDR)	3-33
3.5.3.4	Disk (DKx)	3-33
3.6	SCI CONSOLE ROM (GA-16/220).	3-33
3.6.1	CONSOLE ROM COMMANDS	3-34
3.6.2	COMMAND COMPONENTS.	3-35
3.6.2.1	Address or Value Constant.	3-35
3.6.2.2	Command Mnemonics.	3-35
3.6.2.3	Command Modifier	3-36
3.6.3	COMMAND DESCRIPTIONS.	3-36
3.6.3.1	Memory Display and Alter	3-37
3.6.3.2	Execute Program.	3-37
3.6.3.3	Punch Memory In Binary	3-38
3.6.3.4	Load Memory In Binary.	3-39
3.6.3.5	Memory Dump.	3-39
3.6.3.6	Register Display and Alter	3-40
3.6.3.7	Executing a Single Instruction	3-41
3.6.3.8	Set Trap	3-42
3.6.3.9	Set Relative Addressing Bias	3-43
3.6.3.10	Fill Memory.	3-43
3.6.3.11	I/O Reset.	3-44
3.7	CONSOLE ROM SYSTEM INTERFACES (GA-16/220).	3-44
3.7.1	USE OF INTERRUPT VECTORS.	3-44
3.7.2	USE OF THE BREAK INTERRUPT FEATURES	3-44
3.7.3	USE OF CONSOLE ROM CLOSED SUBROUTINES	3-45
3.8	BUS-16 STAND-ALONE UTILITY	3-47
3.9	TEST AND VERIFY (T&V) PROGRAMS	3-48
3.9.1	CPU T&V	3-48
3.9.2	MPP T&V	3-48
3.9.3	PERIPHERAL PRODUCT T&Vs	3-48

Section	Title	Page
4	INSTRUCTION REPERTOIRE	4-1
4.1	INSTRUCTION GROUPS	4-1
	4.1.1 CONVENTIONS	4-5
	4.1.2 ADDRESSING PARAMETERS	4-6
	4.1.3 REGISTER IDENTIFIER CODES	4-8
4.2	INSTRUCTION FORMAT	4-8
4.3	ADDRESSING MODES	4-9
	4.3.1 EFFECTIVE ADDRESS GENERATION - STAGE 1 (ABSOLUTE, BASE-RELATIVE, PROGRAM-RELATIVE, OR LITERAL)	4-9
	4.3.1.1 Absolute	4-11
	4.3.1.2 Base-Relative	4-11
	4.3.1.3 Program-Relative	4-12
	4.3.1.4 Literal	4-13
	4.3.2 EFFECTIVE ADDRESS GENERATION - STAGE 2 (DIRECT OR INDIRECT)	4-14
	4.3.2.1 Direct	4-14
	4.3.2.2 Indirect	4-15
	4.3.3 EFFECTIVE ADDRESS GENERATION - STAGE 3 (INDEXED OR NON-INDEXED)	4-16
	4.3.3.1 Non-Indexed	4-16
	4.3.3.2 Indexed	4-17
	4.3.4 EFFECTIVE ADDRESS WRAPAROUND	4-19
	4.3.5 EFFECTIVE ADDRESS BEYOND MEMORY	4-23
4.4	MEMORY REFERENCE INSTRUCTIONS	4-24
	4.4.1 JUMP UNCONDITIONALLY (JMP)	4-27
	4.4.2 JUMP TO SUBROUTINE (JSR)	4-28
	4.4.3 LOAD REGISTER A (LDA)	4-30
	4.4.4 STORE REGISTER A (STA)	4-31
4.5	MEMORY REFERENCE WITH INDEXING INSTRUCTIONS	4-32
	4.5.1 COMPARE MEMORY WITH REGISTER (CMR)	4-37
	4.5.2 DECREMENT MEMORY (DECM)	4-37
	4.5.3 INCREMENT MEMORY (INCM)	4-38
	4.5.4 LOAD ALL REGISTERS AND STATUS (LARS)	4-38
	4.5.5 LOAD BYTE (LDBY)	4-39
	4.5.6 LOAD REGISTER (LDR)	4-40
	4.5.7 RESET BIT (RBIT)	4-42
	4.5.8 STORE ALL REGISTERS AND STATUS (SARS)	4-43
	4.5.9 SET BIT (SBIT)	4-44
	4.5.10 STORE BYTE (STBY)	4-45
	4.5.11 STORE REGISTER (STR)	4-46
	4.5.12 TEST BIT (TBIT)	4-47
	4.5.13 PROGRAMMING EXAMPLES	4-48

Section	Title	Page
4.6	CONDITIONAL JUMP INSTRUCTIONS (SKIP)	4-51
4.6.1	SKIP IF MINUS (SKM)	4-54
4.6.2	SKIP IF NOT ZERO (SKN)	4-54
4.6.3	SKIP IF OVERFLOW FALSE (SKOF)	4-54
4.6.4	SKIP IF OVERFLOW TRUE (SKOT)	4-55
4.6.5	SKIP IF PLUS (SKP)	4-55
4.6.6	SKIP IF LINK RESET (SKR)	4-55
4.6.7	SKIP IS LINK SET (SKS)	4-56
4.6.8	SKIP IF ZERO (SKZ)	4-56
4.6.9	PROGRAMMING EXAMPLES	4-57
4.7	REGISTER OPERATE AND REGISTER OPERATE COMPARE INSTRUCTIONS . .	4-57
4.7.1	ADD REGISTERS (ADD)	4-59
4.7.2	ADD COMPARE REGISTERS (ADDC)	4-59
4.7.3	AND REGISTERS (AND)	4-60
4.7.4	AND COMPARE REGISTERS (ANDC)	4-60
4.7.5	OR REGISTERS (OR)	4-61
4.7.6	OR COMPARE REGISTERS (ORC)	4-61
4.7.7	TRANSFER REGISTER (RTR)	4-62
4.7.8	SUBTRACT REGISTERS (SUB)	4-62
4.7.9	SUBTRACT COMPARE REGISTERS (SUBC)	4-63
4.7.10	EXCLUSIVE-OR REGISTERS (XOR)	4-63
4.7.11	EXCLUSIVE-OR COMPARE REGISTERS (XORC)	4-64
4.7.12	PROGRAMMING EXAMPLES	4-64
4.8	REGISTER OPERATE LITERAL AND REGISTER OPERATE LITERAL COMPARE INSTRUCTIONS	4-65
4.8.1	ADD VALUE TO REGISTER (ADDV)	4-67
4.8.2	ADD COMPARE VALUE WITH REGISTER (ADDVC)	4-67
4.8.3	AND VALUE WITH REGISTER (ANDV)	4-68
4.8.4	AND COMPARE VALUE WITH REGISTER (ANDVC)	4-68
4.8.5	LOAD VALUE TO REGISTER (LDV)	4-69
4.8.6	OR VALUE WITH REGISTER (ORV)	4-69
4.8.7	OR COMPARE VALUE WITH REGISTER (ORVC)	4-70
4.8.8	SUBTRACT VALUE FROM REGISTER (SUBV)	4-70
4.8.9	SUBTRACT COMPARE VALUE WITH REGISTER (SUBVC)	4-70
4.8.10	EXCLUSIVE-OR VALUE WITH REGISTER (XORV)	4-71
4.8.11	EXCLUSIVE-OR COMPARE VALUE WITH REGISTER (XORVC)	4-71
4.8.12	PROGRAMMING EXAMPLES	4-71
4.9	SUBROUTINE RETURN VIA INDIRECT VECTOR (RTNIV)	4-74
4.10	REGISTER CHANGE INSTRUCTIONS	4-76
4.10.1	ADD SHIFT COUNTER TO REGISTER (ADDS)	4-77
4.10.2	COMPLEMENT REGISTER (CMPL)	4-78
4.10.3	DECREMENT REGISTER (DECR)	4-78
4.10.4	DISPLAY REGISTER CONTENTS (DSPL)	4-79
4.10.5	EXCHANGE BYTES (EXBY)	4-79

Section	Title	Page
	4.10.6 EXIT FROM SUBROUTINE (EXIT)	4-80
	4.10.7 INCREMENT REGISTER (INCR)	4-80
	4.10.8 RCSW (INVALID INSTRUCTION)	4-81
	4.10.9 RESTORE INTERRUPT SYSTEM ENABLE (RISE)	4-82
	4.10.10 ADD LINK TO REGISTER (RLK)	4-83
	4.10.11 RETURN FROM SUBROUTINE (RTRN)	4-84
	4.10.12 TRANSFER REGISTER TO STATUS (TRS)	4-85
	4.10.13 TRANSFER STATUS TO REGISTER (TSR)	4-85
	4.10.14 EXECUTE REGISTER CONTENTS (XEC)	4-86
	4.10.15 ZERO REGISTER (ZERO)	4-86
	4.10.16 ZERO LEFT BYTE (ZLBY)	4-87
	4.10.17 ZERO RIGHT BYTE (ZRBY)	4-87
	4.10.18 PROGRAMMING EXAMPLES	4-88
4.11	SHIFT LEFT INSTRUCTIONS	4-90
	4.11.1 SHIFT LEFT CIRCULAR (SLC)	4-91
	4.11.2 SHIFT LEFT CIRCULAR THROUGH LINK (SLCL)	4-91
	4.11.3 SHIFT LEFT INSERT ONE (SLIO)	4-91
	4.11.4 SHIFT LEFT INSERT ZERO (SLIZ)	4-91
4.12	SHIFT RIGHT INSTRUCTIONS	4-92
	4.12.1 SHIFT RIGHT ARITHMETIC (SRA)	4-93
	4.12.2 SHIFT RIGHT CIRCULAR (SRC)	4-93
	4.12.3 SHIFT RIGHT CIRCULAR THROUGH LINK (SRCL)	4-94
	4.12.4 SHIFT RIGHT LOGICAL AND COUNT (SRLC)	4-94
	4.12.5 PROGRAMMING EXAMPLE	4-95
4.13	CONTROL INSTRUCTIONS	4-95
	4.13.1 SET BACKGROUND MODE (BMS)	4-97
	4.13.2 SET FOREGROUND MODE (FMS)	4-97
	4.13.3 ENABLE INTERRUPTS (INE)	4-98
	4.13.4 INHIBIT INTERRUPTS (INH)	4-98
	4.13.5 RESET LINK (LKR)	4-98
	4.13.6 SET LINK (LKS)	4-99
	4.13.7 PULSE OPERATIONS MONITOR ALARM (PMA)	4-99
	4.13.8 GENERATE SYNC PULSE (SYNC)	4-99
	4.13.9 PROGRAM SEQUENCE INTERRUPT (TRAP)/RESERVED OP CODES	4-100
	4.13.10 WAIT (WAIT)	4-101
4.14	PROGRAMMED INPUT/OUTPUT INSTRUCTIONS	4-102
	4.14.1 OUTPUT CONTROL FUNCTION (CTRL)	4-104
	4.14.2 DATA TRANSFER IN TO MEMORY (DTIM)	4-104
	4.14.3 DATA TRANSFER IN TO REGISTER (DTIR)	4-105
	4.14.4 DATA TRANSFER OUT FROM MEMORY (DTOM)	4-105
	4.14.5 DATA TRANSFER OUT FROM REGISTER (DTOR)	4-106
	4.14.6 TEST DEVICE (TEST)	4-106

Section	Title	Page
4.15	SPECIAL FEATURES	4-107
4.15.1	READ CONSOLE SWITCHES (GA-16/220)	4-108
4.15.1.1	Read Console Switches Into Memory (RCSM)	4-109
4.15.1.2	Read Console Switches Into Register (RCSR)	4-109
4.15.2	INTERNAL MASK WORDS (GA-16/220)	4-109
4.15.2.1	Console Interrupt (CNSL INT)	4-111
4.15.2.2	Real-Time Clock Interrupt (RTC)	4-111
4.15.2.3	Teletype Interrupt	4-112
4.15.2.4	Memory Mode Change Mask	4-112
4.15.3	SERIAL I/O CONTROLLER	4-112
4.15.4	OPERATIONS MONITOR ALARM (OMA)	4-119
4.15.5	NI POWER-FAIL INTERRUPT	4-119
4.15.6	AUTOMATIC RESTART INTERRUPT	4-120
4.15.7	GA-16/110 MEMORY MODE SET	4-120
4.16	HARDWARE MULTIPLY AND DIVIDE	4-120
4.16.1	HARDWARE DIVIDE (DIV)	4-121
4.16.2	HARDWARE MULTIPLY (MPY)	4-127
4.17	SPECIAL INSTRUCTIONS	4-133
4.17.1	I/O RESET (IORST)	4-133
4.17.2	ENABLE SINGLE STEP INTERRUPT (SSTEP)	4-134
4.18	SAMPLE GA-16/220 PROGRAM	4-134
5	MEMORY PARITY PROTECT (MPP)	5-1
5.1	MEMORY PARITY	5-1
5.2	WRITE PROTECT FEATURES	5-2
5.3	PROGRAM TIMEOUT (PTO)	5-4
5.4	MPP CONTROLS AND INDICATORS	5-4
5.5	NON-INHIBITABLE MPP INTERRUPT	5-4
5.6	MPP COMMANDS	5-5
5.6.1	OUTPUT WORDS	5-5
5.6.1.1	Output Map Address or Map Data	5-5
5.6.1.2	Output Command Word (Resets Status)	5-6
5.6.2	INPUT WORDS	5-8
5.6.2.1	Input Status Word	5-8
5.6.2.2	Input Memory Address Word	5-10
5.6.2.3	Input Memory Data Word	5-10
5.6.2.4	Input Protect Map Data	5-11
5.7	MPP OPERATION	5-12
5.8	INSTRUCTION AND DATA SUMMARY	5-12

Section	Title	Page
6	INPUT/OUTPUT OPERATIONS	6-1
6.1	PIO OPERATIONS	6-1
6.2	INTERRUPT-DRIVEN PROGRAMMED I/O	6-8
6.3	GA-16/220 DIRECT MEMORY ACCESS VIA DATA CHANNEL I/O (DCIO)	6-9
6.4	INTERFACING TO THE DATA CHANNEL BUS, MHSDC	6-10
6.5	GA-16/220 DIRECT MEMORY ACCESS (DMA) DIRECT I/O	6-13
6.6	DELETED	6-13
6.7	I/O MEMORY	6-14
6.8	I/O HARDWARE INTERFACE	6-44
	6.8.1 CONTROLLER DESIGN CONSIDERATIONS	6-44
	6.8.2 PHYSICAL DESCRIPTION OF I/O SYSTEM	6-47
	6.8.3 I/O BUS DESCRIPTION	6-47
	6.8.4 I/O BUS DRIVERS/RECEIVERS	6-56
	6.8.5 I/O CONNECTOR PIN ASSIGNMENTS	6-57
6.9	I/O BUS TIMING AND INTERFACING	6-60
	6.9.1 TEST AND CONTROL LOGIC TIMING	6-60
	6.9.2 DATA OUT LOGIC AND TIMING	6-64
	6.9.3 DATA IN LOGIC AND TIMING	6-64
	6.9.4 INTERRUPT LOGIC AND TIMING	6-69
	6.9.4.1 Request Phase	6-69
	6.9.4.2 Priority Determination Phase	6-69
	6.9.4.3 Acknowledge Phase	6-71
	6.9.5 GA-16/220 DMA TRANSFERS	6-72
6.10	INTERFACE TO MHSDC	6-75
	6.10.1 GA-16/220 DATA CHANNEL BUS DESCRIPTION (MHSDC)	6-75
	6.10.2 DATA CHANNEL TIMING AND INTERFACING	6-77
	6.10.3 DATA CHANNEL INITIALIZATION SEQUENCE	6-79
	6.10.4 DATA CHANNEL INPUT BLOCK TRANSFER OPERATIONS	6-81
	6.10.5 DATA CHANNEL OUTPUT BLOCK TRANSFER OPERATIONS	6-85
	6.10.6 DATA CHANNEL/CONTROLLER CHAINING SEQUENCE	6-85
	6.10.7 LATENCY TIME	6-90
6.11	INTERFACE EXAMPLES	6-90
	6.11.1 EXAMPLE CONTROLLERS 1 AND 2	6-90
	6.11.2 EXAMPLE CONTROLLERS 3 AND 4	6-94
	6.11.3 EXAMPLE CONTROLLER 3	6-95
	6.11.4 EXAMPLE CONTROLLER 4	6-97
7	INSTALLATION PROCEDURES	7-1
7.1	GENERAL	7-1
7.2	PHYSICAL DESCRIPTION	7-1
	7.2.1 DESCRIPTION OF THE EXTERNAL I/O ENCLOSURE	7-3
7.3	SITE REQUIREMENTS	7-10
	7.3.1 ENVIRONMENTAL CONDITIONS	7-10
	7.3.2 POWER REQUIREMENTS	7-10

LIST OF ILLUSTRATIONS

Number	Title	Page
2-1	GA-16/110 Components.	2-2
2-2	GA-16/220 Components.	2-3
2-3	Internal Organization of the CPU.	2-9
2-4	IN Interrupt Flow	2-23
2-5	MPP Status Word	2-27
3-1	GA-16/110 System in Compact Chassis (No MPP).	3-5
3-2	GA-16/220 System in Compact Chassis (No MPP)	3-6
3-3	GA-16/110 System in Jumbo Chassis (With MPP)	3-8
3-4	GA-16/220 System in Jumbo Chassis (With MPP)	3-9
3-5	Memory Address Map	3-14
3-6	Dedicated Memory Locations	3-16
3-7	System Console Interface and IPL ROM Memory Locations	3-17
3-8	General System Start-Up Flow Diagram	3-26
4-1	Address Stages for Instructions that Reference Memory	4-10
4-2	Effective Address Generation, Memory Reference Instructions	4-26
4-3	Effective Address Generation, Memory Reference with Indexing Instructions	4-36
4-4	Effective Address Generation, Conditional Jump Instructions (SKIP).	4-53
4-5	Effective Address Generation, Literal Addressing Mode	4-66
4-6	Internal Mask Words (GA-16/220 Only)	4-110
4-7	TTY Exercise Program	4-117
4-8	Sample GA-16/220 Program	4-135
5-1	GA-16/220 Memory Parity Protect Module	5-3
5-2	MPP Protect Maps	5-7
5-3	Illustration of Memory Protection	5-9
5-4	Output Data for MPP	5-14
5-5	Input Data from MPP	5-16
5-6	MPP Demonstration Program	5-17
6-1	Typical Programmed I/O Sequence	6-7
6-2	Example Block Transfer Memory Map (GA-16/220)	6-12
6-3	Input Data From HSPTR	6-15
6-4	Output Data to HSPTP	6-16
6-5	Object Word Format to Floppy Disk Controller	6-18
6-6	Status Word Format for Floppy Disk Controller	6-20
6-7	Read Card Input Format (via DCIO)	6-22
6-8	Object Word to Line Printer	6-24
6-9	Data Words to Line Printer (via DCIO)	6-25
6-10	Object Word Format to 3341/3343 Disk Controller	6-27
6-11	Status Word for the 3341/3343 Disk Controller	6-29
6-12	Object Words to 3345 Disk Controller	6-31
6-13	Status Words From 3345 Disk Controller	6-34
6-14	Status Word From 3346/3347 Disk Storage System Controller	6-35
6-15	Object Word for 3346/3347 Disk Storage System Controller	6-36
6-16	Object Word to Models 3331, 3332, and 3333 Magnetic Tape Unit Controller	6-37
6-17	Status Word for Models 3331, 3332, and 3333 Magnetic Tape Controller	6-38

LIST OF ILLUSTRATIONS (Continued)

Number	Title	Page
6-18	Object Word Formats to HPT Controller	6-39
6-19	Status Word Format for HPT Controller	6-40
6-20	Mask Word for Internal Interrupts	6-41
6-21	Mask Word for Setting Memory Mode	6-41
6-22	TTY Input or Output Data Format	6-42
6-23	I/O Interfaces	6-45
6-24	Typical I/O System	6-48
6-25	Flowchart of I/O Instructions	6-61
6-26	Timing for I/O TEST and CONTROL Instructions	6-62
6-27	Logic for I/O TEST and CTRL Instructions	6-63
6-28	Timing and Logic for Data Output Function	6-65
6-29	Timing and Logic for Data Input Function	6-67
6-30	Logic and Timing for Program Interrupt Sequence	6-70
6-31	Timing and Logic for DMA Data Output	6-73
6-32	Timing for Consecutive DMA Data Output Transfers	6-74
6-33	Timing for Consecutive DMA Data Input Transfers	6-74
6-34	Timing and Logic for DMA Input	6-76
6-35	Timing for Data Channel Initialization Sequence	6-80
6-36	Data Channel Transfer (Input)	6-82
6-37	Timing for Consecutive Data Channel Transfers (Input)	6-84
6-38	Timing and Logic for Data Channel Data Transfer (Output)	6-86
6-39	Timing for Consecutive Data Channel Data Transfer (Output)	6-87
6-40	Data Channel/Controller Chaining Sequence	6-88
6-41	Standard Controller to Data Channel Interface Featuring Logic to Accommodate All Cases	6-89
6-42	Example Controller 1	6-92
6-43	Example Controller 2	6-93
6-44	Example Controller 3	6-96
6-45	Example Controller 4	6-98
7-1	GA-16/220 CPU Configuration	7-2
7-2	Installation Drawing I/O Power Supply Model 1615	7-4
7-3	Installation Drawing, I/O Enclosure Model 1615-0001	7-5
7-4	Circuit Board Layout	7-6
7-5	Pin Relationship Between Controller Card and Paddle Board	7-7
7-6	Pin Relationship Between CIT-16 Board and External I/O Bus Cable	7-8
7-7	I/O Signal Connections, External I/O Enclosure	7-9
D-1	Binary Paper Tape Format	D-1
D-2	PGS Paper Tape Format	D-1
D-3	Binary Card Format	D-2
D-4	PGS Card Format	D-2
F-1	GA-16/110 System in Compact Chassis (No MPP)	F-2
F-2	GA-16/220 System in Compact Chassis (No MPP)	F-3
F-3	GA-16/110 System in Jumbo Chassis (With MPP)	F-5
F-4	GA-16/220 System in Jumbo Chassis (With MPP)	F-6
F-5	Typical I/O System Interrupt Chain for Early Compact Jumbo Chassis	F-8
G-1	Redesigned GA-16/110/220 Processor Boards	G-2

LIST OF TABLES

Number	Title	Page
1-1	GA-16/110/220 Computer Specifications	1-1
2-1	Register Contents	2-12
2-2	Operation Affecting ISE F/F or ISE Status Indicator	2-15
2-3	Interrupt Summary	2-20
2-4	Dedicated Memory Map.	2-21
2-5	Mode Selection Bits	2-27
3-1	Connector Assignments for Compact Chassis Computer System	3-7
3-2	Connector Assignments for Jumbo Chassis Computer System	3-10
3-3	GA-16/110/220 Controls and Indicators	3-19
3-4	IPL Selector Switch on SCI Module (GA-16/220)	3-25
3-5	Console ROM Closed Subroutines	3-46
4-1	GA-16/110/220 Instruction Repertoire	4-2
4-2	EA Coding, Memory Reference Instructions	4-25
4-3	Coding of Effective Address — Memory Reference with Indexing Instructions	4-35
4-4	Dedicated Memory for NI Interrupts	4-74
4-5	Effects of Memory Mode Change	4-112
4-6	TTY Modes	4-113
4-7	Instruction Summary for Serial I/O Controller	4-117
5-1	PTO Time Period Jumpers	5-4
5-2	Instruction Summary for MPP Module	5-13
6-1	Standard Device Select Codes and Interrupt Assignments	6-2
6-2	I/O Signal Summary	6-5
6-3	Data Channel Assignments for Standard Peripheral Units	6-11
6-4	Instruction Summary for High-Speed Paper Tape Reader (HSPTR) Controller	6-15
6-5	Instruction Summary for High-Speed Paper Tape Punch (HSPTP) Controller	6-16
6-6	Instruction Summary for Model 3349 for Floppy Disk Controller	6-17
6-7	Instruction Summary for the Model 3356 Controller (Card Reader System)	6-22
6-8	Character Codes for Card Reader	6-23
6-9	Instruction Summary for the Model 3356 Controller (Line Printer Section)	6-24
6-10	Paper Feed Codes for Printer	6-25
6-11	Character Codes for the Line Printer	6-26
6-12	Instruction Summary for the 3341/3343 Disk Storage System	6-27
6-13	Instruction Summary for the 3345 Disk Controller	6-30
6-14	Instruction Summary for the 3346/3347 Disk Storage System Controller	6-35
6-15	Instruction Summary for Models 3331, 3332, 3333 Magnetic Tape Unit Controller	6-37
6-16	Instruction Summary for Model 3342 Head-Per-Track Controller	6-39
6-17	Instruction Summary for 16/220 Internal Functions and Console	6-41
6-18	Instruction Summary for Serial I/O Controller (GA-16/220) or External Model 1582 Controller (GA-16/110)	6-42

LIST OF TABLES (Continued)

Number	Title	Page
6-19	Detail Description of ASR 33 TTY Functions.	6-43
6-20	I/O Bus Descriptions.	6-51
6-21	I/O Cable and CIT-16 Connector Pin Assignments.	6-58
6-22	I/O Signal Pin Assignments.	6-59
6-23	Data Channel Bus Signals.	6-78
6-24	Interface Signals for Example Controllers 1 and 2	6-91
6-25	Interface Signals for Example Controllers 3 and 4	6-94
A-1	General Instructions.	A-2
A-2	Summary of General Instructions	A-4
A-3	Memory Reference Instructions EA Calculations	A-9
A-4	Memory Reference with Indexing, EA Calculations	A-10
A-5	Standard I/O Data & Instructions.	A-14
B-1	ASCII Character Code Summary.	B-2
F-1	Connector Assignments for Compact Chassis Computer System	F-4
F-2	Connector Assignments for Jumbo Chassis Computer System	F-7

LIST OF APPENDICES

Appendix	Title	Page
A	GA-16/110/220 INSTRUCTION SUMMARY	A-1
B	ASCII CHARACTER CODE.	B-1
C	CONVERSION TABLES	C-1
D	BINARY, PGS AND DATA FORMAT	D-1
E	SUMMARY OF SYSTEM CONSOLE ROM COMMANDS.	E-1
F	EARLY GA-16/110 AND GA-16/220 CONFIGURATIONS.	F-1
G	REDESIGNED GA-16/110/220 PROCESSOR BOARDS	G-1

introduction 1

The GA-16/110 and GA-16/220 computer systems are based upon a fast and powerful microprocessor. This microprocessor supports a large instruction set and many programming features that are not available in most microprocessors. The GA-16/110 is especially designed for dedicated applications where high performance is required to function as remote controllers and concentrators in a large network system. The GA-16/110 is a total microcomputer with CPU, memory, program loader, and I/O on a single small plug-in card for OEM designers to install in their products. The GA-16/110 CPU may also be installed in a GA Compact or Jumbo enclosure, together with additional memory, to provide a computer system for rack mounting.

The GA-16/110 computer may be converted to a general-purpose GA-16/220 computer by adding one additional module which adds the capabilities identified in Table 1-1 as GA-16/220.

Applications for a GA-16/220 include computing systems operating under a GA-16 series operating system for batch processing, program development, and process and machine control, with operator interaction provided via a teletype or CRT unit.

Table 1-1 lists the specifications of the GA-16/110 and GA-16/220 computer system.

Table 1-1. GA-16/110/220 Computer Specifications

Architecture	Microprogrammed 16-bit General-Purpose Computer Parallel Binary Two's Complement Arithmetic Single- and Double-Word Instructions Programmed and Direct Memory Access Input/Output																
Technology	Processor — nMOS LSI and Tri-State (7400, 7400H and Schottky Logic) Circuitry and Unique Four-Layer Printed Circuit Board Packaging																
Memory	Semiconductor RAM, PROM, and EPROM																
Registers	<table> <tbody> <tr> <td>Programmable Registers</td> <td>16</td> </tr> <tr> <td>Accumulators</td> <td>6</td> </tr> <tr> <td>Index/Accumulators</td> <td>6</td> </tr> <tr> <td>Base Register</td> <td>2</td> </tr> <tr> <td>Subroutine Linkage</td> <td>2</td> </tr> <tr> <td>Program Counter</td> <td>1</td> </tr> <tr> <td>Instruction Register</td> <td>1</td> </tr> <tr> <td>Status Register</td> <td>1</td> </tr> </tbody> </table>	Programmable Registers	16	Accumulators	6	Index/Accumulators	6	Base Register	2	Subroutine Linkage	2	Program Counter	1	Instruction Register	1	Status Register	1
Programmable Registers	16																
Accumulators	6																
Index/Accumulators	6																
Base Register	2																
Subroutine Linkage	2																
Program Counter	1																
Instruction Register	1																
Status Register	1																

Table 1-1. GA-16/110/220 Computer Specifications (Continued)

Registers (Cont)	Status Indicators Zero Plus Link Overflow Foreground/Background 32K/64K Memory Mode Interrupt System Enable (ISE) Save Shift Count	7 1
Standard Processor	Real-Time Clock (RTC) Operations Monitor Alarm (OMA) Power Fail/Auto Restart (PF/AR) Cold Start 64K Direct Addressing Memory Parity Protect (MPP) (Optional) Interrupt Program Timeout (Optional)	1 ms 150 ms - 300 ms
Arithmetic and Logic	Parallel Binary, Two's Complement, Fixed-Point, Bit, Byte, and Word ADD, SUB, COMP, INC, DEC, AND, OR, XOR, SET BIT, RESET BIT, TEST BIT Hardware Multiply and Divide (Unsigned) Signed Multiply/Divide (Optional)	
Instructions (Fourteen Classes)	Memory Reference Memory Referenced Indexed Conditional Jump (Skip) on Eight (8) Conditions Register Operate and Register Operate and Compare Register Operate Literal and Register Operate Literal and Compare Subroutine Return Via Indirect Vector Register Change Shift Left Shift Right Control Input/Output (Addressing to 64 Device Select Codes) Read Console Switches (Additional Input/Output on GA-16/220) Multiply/Divide (Unsigned) Special (Enable Single Step Interrupt, I/O Reset) (Additional Input/Output on GA-16/220) (All instructions listed by class and alphabetically in Section 4.)	

Table 1-1. GA-16/110/220 Computer Specifications (Continued)

Addressing (Eleven Modes)	Direct Direct, Indexed Indirect Indirect, Indexed Program Relative Program Relative, Indirect Base Relative Base Relative, Indexed Base Relative, Indirect Base Relative, Indirect, Indexed Literal
Programmed I/O (PIO)	16-Bit Parallel Transfers Vectored Priority Interrupts 120K Word Transfer Rate Data Transfer To/From Memory To/From Any One of the Sixteen (16) Programmable Registers
Serial I/O (GA-16/220)	Integral Universal Asynchronous Receiver Transmitter (UART) Serial I/O Controller for Console TTY or CRT Data Rates 110 or 9600 Baud (Early Model CPU2 Board) Sixteen rates, 0 thru 9600 Baud (CPU2 Board 31D02519A or 31D02574A)
Direct Memory Access (DMA) I/O (GA-16/220)	16-Bit Parallel Transfers Multi-Channel Operation with Vectored Priority Sequencing 1.25M Word-Per-Second Transfer Rate Interleaved with CPU
Interrupts	Non-Inhibitable (NI) Interrupts Power Fail Auto-Restart Memory Parity, Write Protect, and Program Timeout (Options) TRAP Instruction Single Step/Break (GA-16/220)

88A00508A-E

Table 1-1. GA-16/110/220 Computer Specifications (Continued)

Interrupts (Cont)	Internal Inhibitible (IN) Interrupts (GA-16/220) Real-Time Clock Console TTY Console Interrupt External Inhibitible (IN) Interrupts 64 Priority Interrupt Levels																																				
Dimensions	GA-16/110 CPU Module or Additional 220 Module <table> <tr> <td>Height</td> <td>7.250 in.</td> <td>(18.415 cm)</td> </tr> <tr> <td>Width</td> <td>0.625 in.</td> <td>(1.588 cm)</td> </tr> <tr> <td>Width w/Piggyback RAM (or SCI)</td> <td>1.063 in.</td> <td>(2.700 cm)</td> </tr> <tr> <td>Depth</td> <td>11.000 in.</td> <td>(27.940 cm)</td> </tr> </table> <p>Compact GA-16/110/220 System with Internal Power Supply or Jumbo GA-16/110/220 System (Requires Separate Power Supply)</p> <table> <tr> <td>Height</td> <td>8.750 in.</td> <td>(18.415 cm)</td> </tr> <tr> <td>Width</td> <td>19.000 in.</td> <td>(48.260 cm)</td> </tr> <tr> <td>Depth (w/o Cables)</td> <td>21.150 in.</td> <td>(53.721 cm)</td> </tr> <tr> <td>Depth (w/Cables)</td> <td>22.250 in.</td> <td>(56.515 cm)</td> </tr> </table> <p>Separate Power Supply for Jumbo GA-16/110/220 System</p> <table> <tr> <td>Height</td> <td>10.750 in.</td> <td>(27.305 cm)</td> </tr> <tr> <td>Width</td> <td>19.000 in.</td> <td>(48.260 cm)</td> </tr> <tr> <td>Depth (w/o Cables)</td> <td>5.750 in.</td> <td>(14.605 cm)</td> </tr> <tr> <td>Depth (w/Cables)</td> <td>7.750 in.</td> <td>(19.685 cm)</td> </tr> </table>	Height	7.250 in.	(18.415 cm)	Width	0.625 in.	(1.588 cm)	Width w/Piggyback RAM (or SCI)	1.063 in.	(2.700 cm)	Depth	11.000 in.	(27.940 cm)	Height	8.750 in.	(18.415 cm)	Width	19.000 in.	(48.260 cm)	Depth (w/o Cables)	21.150 in.	(53.721 cm)	Depth (w/Cables)	22.250 in.	(56.515 cm)	Height	10.750 in.	(27.305 cm)	Width	19.000 in.	(48.260 cm)	Depth (w/o Cables)	5.750 in.	(14.605 cm)	Depth (w/Cables)	7.750 in.	(19.685 cm)
Height	7.250 in.	(18.415 cm)																																			
Width	0.625 in.	(1.588 cm)																																			
Width w/Piggyback RAM (or SCI)	1.063 in.	(2.700 cm)																																			
Depth	11.000 in.	(27.940 cm)																																			
Height	8.750 in.	(18.415 cm)																																			
Width	19.000 in.	(48.260 cm)																																			
Depth (w/o Cables)	21.150 in.	(53.721 cm)																																			
Depth (w/Cables)	22.250 in.	(56.515 cm)																																			
Height	10.750 in.	(27.305 cm)																																			
Width	19.000 in.	(48.260 cm)																																			
Depth (w/o Cables)	5.750 in.	(14.605 cm)																																			
Depth (w/Cables)	7.750 in.	(19.685 cm)																																			
Temperature	Operable at 0°C to 50°C (System Cooling Fan used)																																				
Humidity	Up to 90 Percent Relative (Non-condensing)																																				
Input/Output Interface Signals	Low Level - 0.0 to 0.4 volt High Level- 2.0 to 5.0 volts I/O Drivers - 74365 Tri-State TTL Drivers Saturation voltage 0.4 volt Capable of sinking 40 ma to ground																																				
I/O Cable Termination	Twisted Pairs - 100Ω characteristic impedance Returns grounded at both ends																																				

Table 1-1. GA-16/110/220 Computer Specifications (Continued)

Power	GA-16/110 Module Only
	+5V@2.7A, +15V@0.092A, -15V@0.012A
	Optional Piggyback RAM Memory for GA-16/110/220
	w/IPL +5V@1.620A, w/o IPL +5V@1.236A
	GA-16/220 Module
	+5V@2.58A, +15V@0.000A, -15V@0.012A
	8K by 16K Memory Module
	+5V@1.5A, -5V@0.004A, +12V@0.400A
	MPP
	+5V@2.25A
	Optional System Console Interface (SCI)
	w/IPL +5V@1.2A w/o IPL +5V@0.64A
	GA-16/110 or GA-16/220 Power Supplies
	115 Volts AC, Single-Phase ± 10 Percent, 47 to 63 Hz (Internal or External Power Supply)
	220 Volts AC, Single-Phase ± 10 Percent, 47 to 63 Hz (External Power Supply Only)
	230 Watts Maximum (Compact)
	500 Watts Maximum (Jumbo)

88A00508A-E

The GA-16/110/220 Systems have 16 general-purpose registers, 12 (GA-16/110) or 14 (GA-16/220) powerful instructions classes, 11 addressing modes, and active priority interrupt system. The systems offer the user the ability to:

- Directly address any word, byte or bit in 64K of memory.
- Relative address, allowing programs to be self-relocating (i.e., they will run without modification anywhere in memory).
- Base Relative address, allowing the independent location of data and instructions; temporary storage required by subroutines can be allocated dynamically, thus, minimizing memory requirements by allowing subroutines to share portions of memory.
- Reduce the overhead incurred when servicing a program interrupt from an external controller; control may be transferred directly to the routine responsible for servicing the interrupting controller, since each controller has its own interrupt level.
- Locate data and instructions independently and to execute the contents of any register as an instruction, allowing any program to be written as "pure procedure" and, consequently, to be implemented in Read-Only-Memory,

1.1 SYSTEM ARCHITECTURE

The central part of both the GA-16/110 and the GA-16/220 is a powerful microprogrammed processor. This processor provides a larger, more sophisticated instruction set than any other computer in its class. Microprogramming allows software compatibility with the extensive SPC-16 library of programs, while offering a large number of new features and instructions. Significant enhancements to the instruction repertoire for the GA-16/110 include:

- New Left Shift Instructions
- Subroutine Return Instruction for Non-Inhibitable Interrupts
- Program TRAP Instruction

The GA-16/220 adds the following instructions:

- Instruction Permitting I/O Reset
- Instruction Permitting Single-Step Operation

1.2 STANDARD FEATURES OF THE GA-16/110 AND GA-16/220 SYSTEMS

Standard Features of the GA-16/110 system include:

- Microconsole
- 16 General-Purpose Registers
- Fast Context Switching with Foreground/Background Registers
- Unsigned Multiply/Divide
- Direct Memory Addressability to 64K
- Power Fail/Auto Restart
- Operations Monitor Alarm
- Priority Interrupt System
- Programmed Input/Output System
- Interrupt for Real-Time Clock (External Pulses and Enable Required)

A GA-16/220 system provides the following additional standard features:

- Programmable Memory Mode Change
- Pulses and Enable Signal for Real-Time Clock Interrupt- 1 ms
- Console Interrupt
- Serial I/O Controller for TTY, CRT or RS-232C Interface
- Direct Memory Access I/O (Using Multiple High-Speed Data Channel (MHSDC) or Direct DMA)
- Data Entry Switches for use with RCSR and RCSM Instructions

1.2.1 MICROCONSOLE AND OTHER CONTROLS AND INDICATORS

1.2.1.1 Microconsole for GA-16/110 System

The microconsole on a GA-16/110 consists of switches and indicators mounted on the single plug-in CPU module. These switches and indicators permit an operator to carry out basic control operations. Section 3 provides a complete description of all switches and indicators.

The following switches (or jumpers) are included on the GA-16/110 module:

- 32K or 64K Addressing Mode Selection (via jumper)
- Reset Button (RESET)
- Internal or backup memory power (+5VB)
- Jumper to disable internal 20 MHz crystal oscillator for test purposes

Indicators include the following:

- Interrupt Acknowledge (IACK)
- Interrupt System Enabled (ISE)
- Stall Indicator (OMA)
- Foreground Register Usage Indicator (FGND)
- Run Indicator (RUN)
- Wait Indicator (WAIT)

1.2.1.2 Microconsole for the GA-16/220 System

The microconsole on a GA-16/220 consists of switches and indicators mounted on the two plug-in CPU modules and, when provided, on the system console interface (SCI) module. These switches and indicators permit an operator to monitor CPU status and carry out basic control operations. Section 3 provides a complete description of all switches and indicators.

The switches are identified as being on the CPU-1 module (110), the CPU-2 module (220) and the System Console Interface (SCI) which mounts onto the CPU-2 module:

- 32K or Programmed Addressing Mode Selector (220)
- Console Interrupt Button (CI) (220)
- Data Entry Switches (bits 15 to 0) (220)
- Teletype or CRT Baud Rate Selector (BAUD) (220) depends on CPU-2 Board; may be 100 or 9600 or 16 rates 0 through 9600 baud.
- TTY Break Enable Switch (BKDS) on SCI (220-SCI)
- TTY Break Mode Selector (BKINT) on SCI (220-SCI)
- IPL Button (IPL) on SCI (220-SCI)
- IPL Device Selector (IPL SLT) on SCI (220-SCI)

- Reset Enable Selector (RESET) on SCI (220-SCI)
- CPU and I/O Reset Button (RESET) (110)
- Internal or backup memory (+5VB) (110)

Indicators include the following:

- Direct Memory Access Acknowledge (DMA ACK) (220)
- Interrupt Acknowledge (IACK) (110)
- Interrupt System Enabled (ISE) (110)
- Stall Indicator (OMA) (110)
- Foreground Register Usage Indicator (FGND) (110)
- Run Indicator (RUN) (110)
- Wait Indicator (WAIT) (110)

1.2.1.3 Additional Controls and Indicators for GA-16/110 and GA-16/220

There are switches on each memory module for determining addressing (up to 64K) and (in the case of 18-bit modules) a parity error override switch and error indicators.

When installed, the MPP module contains a switch for determining whether a stall or an interrupt will occur when a parity error is detected and indicators to aid in diagnosing whether errors are due to a program protect violation, a DMT port 1 (GA-16/110) or DMA (GA-16/220) protect violation, a parity error, or a combination of errors. Indicators also display the parity bits for upper and lower bytes of words in which parity error was detected.

The battery backup power supply, when installed, has an indicator to show when power is being supplied to preserve memory contents and has a manual cutoff button to disconnect battery power when cutoff of AC power is deliberate (as in a total system shutdown). AC power application is independently applied to CPU main power supply and auxiliary (battery backup) power supply.

1.2.2 MEMORY

The principle memory building block for both the GA-16/110 and GA-16/220 systems is a plug-in dynamic RAM module of 8K words in both 16-bit and 18-bit configurations. The 18-bit memories provide parity generation and error detection. A memory module may have 4, 8, 16, 32, or 64K capacity.

The 8K modules can be set to beginning addresses on any 8K boundary up to 56K, 4K modules can be set on 4K boundaries up to 60K, and 16K and 32K modules can similarly be set on boundaries up to 64K.

1.2.2.1 Auxiliary RAMs and ROMs

A special-purpose 1K control and memory module is available as an option for a GA-16/220. It is called the System Console Interface (SCI). This module, in addition to microconsole controls, has 512 words of ROM containing interactive console TTY routines. It also has 256 words of RAM for working storage of data. A 256-word IPL ROM provides bootstrap loading capabilities for TTY, paper tape reader, card reader, floppy disk, moving arm disk, and head-per-track disk (or drum). Use of the SCI is described in Section 3.5

A 2K static RAM is optional for both GA-16/110 and GA-16/220. It mounts on the CPU-1 module (and is referred to as the piggyback RAM memory) and therefore, does not require a memory slot. It may be set for beginning address of 0 or 8K, depending on the application. In addition to 2K RAM, the piggyback memory may also include a single device 64-word IPL ROM which can be specified to load from a TTY or high-speed paper tape reader; however, when the SCI is installed (on a GA-16/220), the multi-device IPL ROM on the SCI takes precedence and the 64-word IPL ROM should not be installed.

1.2.2.2 EPROM

Another piggyback memory is also available for the CPU-1 board; it has 3K words of EPROM and 1K words of RAM. The 1K work RAM resides below the 3K EPROM. The EPROM (after ultraviolet erasure) may be reprogrammed via a 4K programming module that plugs into the memory bus. The programming module may be set to occupy any 4K boundary up to 60K. The EPROM programmer (01P01786A) and an application manual (82S00761A) for the programmer are available from General Automation.

1.2.3 ADDRESSING MODES

Eleven addressing modes are available with instructions that address memory. Instructions may specify an absolute, program-relative (relative to the location of the instruction itself), or base-relative address, which may be either a direct or an indirect address. In addition, other instructions may specify one of three index registers to be used as a word byte or bit index to the referenced location.

1.2.4 GENERAL-PURPOSE REGISTERS

The GA-16/110/220 computer is organized around 16 general-purpose registers. These registers are divided into two banks of eight registers, foreground and background, for fast context switching. Data placed in foreground registers in foreground mode may be saved by changing to background mode to use background registers and vice-versa. All general-purpose registers in a bank may be used as accumulators for arithmetic and logical operations, as program loop counters, or as a data input/output buffers.

In addition to the functional capabilities common to all general-purpose registers, certain of them have special properties. In each bank of eight registers, one register (D) is used in preindexed (base-relative) addressing. When an instruction specifies base-relative addressing, the address field of the instruction is added to the contents of this register in determining the memory location to be referenced by the instruction.

Three registers in each bank (X,Y, and Z) are designated postindex registers. When one of these registers is specified in an instruction as an index register, the specified register's contents are added to the computer address to determine the word or byte in memory to be referenced.

One register in each bank (E) serves as a subroutine return register; it holds the return address after a Jump-to-Subroutine (JSR) instruction is executed or when a program interrupt occurs. Registers A, B and C are used only as general-purpose registers.

1.2.5 INPUT/OUTPUT SYSTEM

The input/output system for the GA-16/110 and GA-16/220 is designed for maximum efficiency and flexibility. Data moves between the computer and peripheral devices either under program control, or automatically, using the direct memory access (DMA), (GA-16/220) features.

A program communicates with peripheral equipment via programmed input/output (PIO) instructions. Four classes of PIO instructions provide data input, data output, control, and test functions. A PIO instruction addresses any one of up to 64 device select codes. A serial I/O controller and an internal interrupt mask register are integral features of the CPU-2 module, leaving 62 device codes for external I/O controllers.

An active priority interrupt system minimizes interrupt response time. For each of the possible controllers there is a dedicated memory location which may contain that controller's interrupt service routine address. When a processor acknowledges an interrupt, control is transferred to the address stored in the interrupting device's dedicated memory location; by virtue of arriving at the addressing routine, the interrupting controller has been identified.

On a GA-16/220, the direct memory access feature allows automatic, high-speed transfer of blocks of data between memory and peripherals. A DMA transfer operation is initiated by a program and runs to completion under the supervision of the peripheral controllers and Multiple High-Speed Data Channel Controller module (MHSDC), (for some DMA controllers MHSDC is not required). Each time the controller is ready to send or receive a data item, it acquires the memory bus for a DMA transfer. Usually this proceeds in parallel with CPU operations without stealing cycles from the CPU. This is referred to as interleaved DMA.

1.2.6 INTERRUPT SYSTEM

The GA-16/220 priority interrupt system consists of two classes of interrupts, Inhibitible (IN) and Non-Inhibitible (NI). Refer to the following list of interrupts (those which apply only to the GA-16/220 are identified by (220)):

- Non-Inhibitible (NI) Interrupts (highest priority, in order listed)
 - Power Fail
 - Auto-Restart
 - Memory Parity and Write Protect Error
 - TRAP Instruction (or reserved opcode)
 - TTY Break/Single Step Instruction (220)
 - Inhibitible (IN) Interrupts (lower priority, in order listed)
 - Real-Time Clock
 - TTY Not Busy (220)
 - Console Interrupt (220)
 - Peripheral Device Controllers (External interrupt, priority determined by controller location)
- } Controller by Internal Mask Word (220)

The NI interrupts are non-inhibitible by the Interrupt System Enable flip-flop (ISE F/F). These interrupts handle the abnormal operations caused by malfunctions of the system and, therefore, require special hardware manipulation of the current status of the computer. During an NI interrupt the hardware stores the P counter and ISE status in memory so the interrupt processing program can store the interrupted program after the malfunction has been rectified.

The inhibitible interrupts (IN) are generally used for I/O control of external devices. This group consists of a number of individually vectored interrupts, each with its own priority level determined by physical location.

The ISE flip-flop controls the IN interrupts as a group; certain individual interrupts are controlled by a mask. (The internal interrupts, for example, are enabled by setting bits in an internal mask word. External I/O controllers may have provision for enabling or disabling interrupt capability by setting a mask in the controller.) The instructions associated with these interrupts are described in Section 4.15.2.

1.2.7 SPECIAL FEATURES

The following special features are provided for the GA-16/110 or GA-16/220 computer; these features insure reliable operation even in remote or unattended environments. (Those features which apply only to the GA-16/220 are prefaced by (220)):

- (220) The Real-Time Clock (RTC) provides a periodic interrupt that occurs every 1.0 millisecond when the interrupt level from the RTC is enabled. (1.0 millisecond pulses and enable signal originate in CPU-2). In addition to providing a multiprogramming capability, the interrupt may be used to protect against unusual conditions by periodically returning control to a system monitor program. The interrupt from the RTC may be enabled or disabled under program control.
- (110) The Real-Time Clock (RTC) may be provided if the user provides clock pulses and enable signals from an external source
- The Operations Monitor Alarm (OMA) protects the system against abnormal operation and provides a signal to warn of the abnormality. Once activated, the OMA brings instruction execution to an orderly halt and changes the system safe signal (SFEC) to the unsafe condition. This signal can be used to control an audiovisual alarm or automatic switchover. The Pulse Operations Monitor Alarm (PMA) instruction is provided to both arm the OMA and reset it during normal operation. The instruction starts the alarm timer when it is executed for the first time and resets the timer each time it is executed thereafter. Failure to execute the instruction within 150 to 300 milliseconds after the previous execution activates the alarm. When the alarm becomes activated, the computer automatically switches from the run mode to the idle mode and the safe signal (SFEC) is removed from the I/O Bus.
- The OMA may be cleared by auto-restart or by manually pressing the RESET button on the microconsole.
- The Power Fail Interrupt issues a warning to the running program if the unregulated DC voltage drops below a predetermined limit (approximately 105 vac input for a fully loaded system). The warning is in the form of a program interrupt that transfers control, via an indirect memory location, to a power fail service routine. The service routine can then save register contents and bring the system to an orderly shutdown before DC power drops below a critical level.
- At the same time the Power Fail interrupt is required, the Power Fail timer is initialized to time out in 90-150 μ s. After 90-150 μ s, the Power Fail times out and the machine switches to the idle mode; the SFEC signal remains in the safe condition. The regulated direct current voltages are guaranteed to be good during the timeout period.
- The Auto-Restart Interrupt causes an orderly start-up with a special restart interrupt when power is restored after a power failure. When the unregulated DC voltage goes above a predetermined value, the Auto-Restart circuit requests an interrupt provided system is equipped with a battery backup power supply. The service routine that is activated as a result of this interrupt may restore registers and status and return control to the program. Thus, computers placed at remote locations do not require operator intervention for restart after a power failure. The entire system is initialized prior to the Auto-Restart Interrupt request.

1.3 OPTIONAL FEATURES OF THE GA-16/110 AND GA-16/220 SYSTEMS

1.3.1 MEMORY PARITY PROTECT (MPP)

There is one optional module that provides additional memory integrity and management functions. This is the Memory Parity and Write Protection option (MPP). The MPP option provides response to a parity error indication from an 18-bit memory module, Write Protection, and program timeout. A detailed description of MPP is contained in Section 5.

1.3.1.1 Memory Parity

This feature (in conjunction with the use of 18-bit memory modules) provides a memory error detection and recovery system. Response to a parity error is switch-selectable to force either an interrupt or stall condition. If the stall mode is selected and a parity error occurs, the computer will stop and there will be a visual indication of the error condition.

If the interrupt mode is selected and a parity error occurs (odd parity is normal), then a non-inhabitable (NI) interrupt is forced and an interrupt service routine is executed. The parity hardware provides the following diagnostic information for this recovery service routine.

- Address of memory location where the parity error occurred
- Content of the memory location where the parity error occurred
- The value of the parity bits for upper and lower bytes of the memory word in which parity error was detected
- Whether or not the word was to be used by the processor or DMA

The above diagnostic information should be used to determine what further action need be taken. This further action is highly application-dependent.

A special instruction is provided that will force a parity error for hardware maintenance.

1.3.1.2 Memory Write Protect

This feature protects memory locations (in 1K segments) from being written into by user programs or DMA devices. This is especially valuable in a multiprogramming system. The write protect feature can be selected to protect against both program and DMA (or DMT port 1) attempts to write into protected areas of memory. If a program or DMA tries to write in a protected memory area, it will provide a visual indication and cause an (NI) interrupt that calls a service routine. The write protect hardware provides the following features which may be used by the service routine:

- Address of word being written into
- Address of DMT port 1 (GA-16/110), DMA (GA-16/220), or instruction trying to write
- Violation caused by CPU or DMA
- Independent enabling of features by programmed I/O instructions
- Violation signaled by non-inhibitible (NI) interrupt
- Separate CPU (program) and DMA (or DMT port 1) protect maps
- CPU and DMA write protect individually enabled

1.3.1.3 Program Timeout

The program timeout feature provides a means of setting a limit on the time that any program may inhibit the interrupt system or execute non-interruptable instructions (primarily jump-to-subroutine (JSR)). The time limit is adjustable by jumpers on the MPP board and timeout is enabled by a program I/O instruction.

1.3.1.4 Error Correction

The error correction option provides single-bit error correction and multiple-bit error detection for a 32K or a 64K memory board. The error correction (ECC) option consists of a single printed circuit module which plugs into the 32/64K memory board.

1.3.2 SYSTEM CONSOLE INTERFACE (GA-16/220 ONLY)

An optional System Console Interface (SCI) module is available for the GA-16/220. It is mounted in piggyback fashion on the CPU-2 module. This option provides an operator with interactive console capability via a teletype or CRT.

Firmware is provided in the 512-word ROM and 256-word RAM to perform the following:

- Display and change registers
- Display and change memory locations
- Display a prescribed block of memory
- Store a specified data pattern in a block of memory
- Load and punch binary tapes via teletype reader/punch
- Single step through program instructions
- Set four traps
- Reset I/O
- Go (enter user's program at selected locations)

The firmware also contains subroutines which may be called from a user's program. These subroutines are described in Section 3.7.

The SCI module may also contain a 256-word IPL ROM which permits an operator to load operating system and user software from the following peripheral devices which may be ordered with the GA-16/220 system:

- Teletype
- High-Speed Paper Tape Reader
- Card Reader
- Model 3347 Disk
- Model 3346 Removable Disk
- Model 3349 Floppy Disk
- Model 3346 Fixed Disk
- Model 3342 Disk
- Model 3341 Disk

An IPL select switch and an IPL button are provided on the SCI module microconsole to select device and initiate IPL. IPL may also be initiated remotely via a hardware interface or by auto-restart with cold start line grounded.

1.4 PERIPHERALS AND CONTROLLERS

Standard controllers manufactured and supplied by General Automation interface the GA-16/110 or the GA-16/220 to a full range of peripherals as well as to standard serial and parallel communications lines and digital and analog data acquisition. Among the controllers and peripherals offered are:

- Multiple High-Speed Data Channel Controller (DMA, 220)
- Teletype or CRT Display Terminals Controller (Built-in on a 220, External on a 110)
- High-Speed Paper Tape Reader/Punch
- Card Reader/Punch (DMA, 220)
- Line Printer (DMA, 220) (PIO, 110)
- Disk Storage Drive (DMA, 220)
- Drum storage Drive (DMA 220)
- Magnetic Tape Drive (DMA, 220)
- Floppy Disk or Cassette Drive (DMA, 220)
- Synchronous Data Link Controller (SLDC) (DMA, 220; does not require MHSDC) (PIO, 110)
- Synchronous Communications Controller
- Asynchronous Communications Controller
- Industrial Process I/O Controllers, include:
 - AC Power Switching Controller
 - Analog Signal Multiplexer/Analog-to-Digital Converter
 - Digital-to-Analog Converter
 - Digital Input Controllers
 - Digital Output Controllers
 - Variable Threshold Digital Input Controller

All programmed and DMA I/O controllers are housed in either the chassis containing the CPU or one or more external I/O enclosures.

1.5 SOFTWARE

A GA-16/110, being a dedicated machine, will normally use programs developed on a general-purpose machine, such as a GA-16/220, 330, 440 or SPC-16. The programs are usually punched on a paper tape for loading into the GA-16/110, or programs may be loaded from a host computer via a communications data link.

All GA series (and earlier SPC-16 series) computer systems are backed by a variety of system software packages. Programming languages include FORTRAN IV, Commercial FORTRAN, COBOL, BASIC, and CAP-16 macro assembler.

Operating systems which are available for data processing and control applications include:

- Free-Standing Operating System - FSOS-16
- Disk-Based Operating System - DBOS-16
- Real-Time Executive - RTX-16
- Real-Time Operating System - RTOS-16

system organization **2**

This section describes the organization of the GA-16/110 and the GA-16/220: (1) elements of the Central Processing Unit (CPU), (2) instruction and data word formats, and (3) description of the interrupt system.

The system organization is described in detail to provide the reader with background to evaluate information contained in the following sections; a clear understanding of the information provided in Section 2 is an essential prerequisite to the comprehension of subsequent sections.

Figure 2-1 gives a high-level overview of the GA-16/110 system's components. Figure 2-2 gives a high-level overview of the GA-16/220 system's components.

2.1 ELEMENTS OF THE CPU

The instruction repertoire includes 91 basic instructions, each of which identifies one of the elementary operations that can be performed by the computer. Any location in memory may contain either a word of data or an instruction word. The internal representation of data and instructions is described in Section 2.2

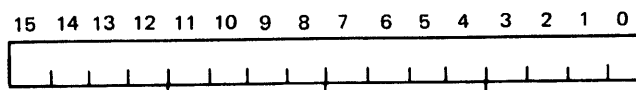
There are eleven ways, or "addressing modes", by which an instruction can refer to a memory location; e.g., values in registers or other locations can be added to the address bits of an instruction. However, when the final address is formed, it is called the Effective Address (EA). The EA is a full, 16-bit, absolute address. Addressing modes are discussed in Section 4 of this manual.

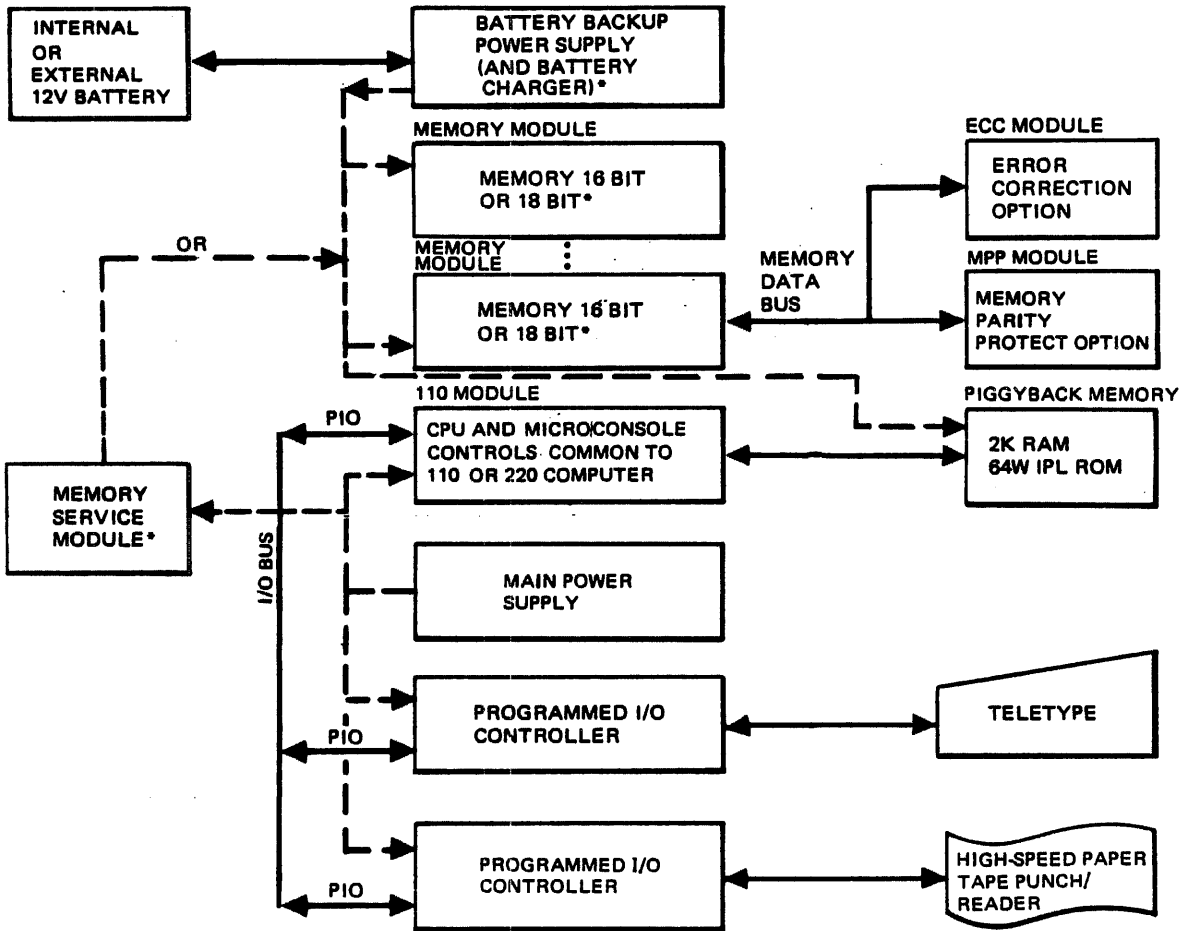
The Input/Output (I/O) system allows the CPU and peripheral equipment to communicate. The general organization of the Input/Output system is described in Section 6.

2.2 WORD FORMAT

The architecture of the GA-16/110/220 processor is based on a 16-bit binary word. Instructions and data are transferred between the CPU and memory in units of the 16-bit word. Arithmetic and logical operations are performed in parallel on 16-bit data words.

Bit positions within a word are numbered sequentially from right to left; the lowest bit is numbered zero and the highest order bit is numbered 15.



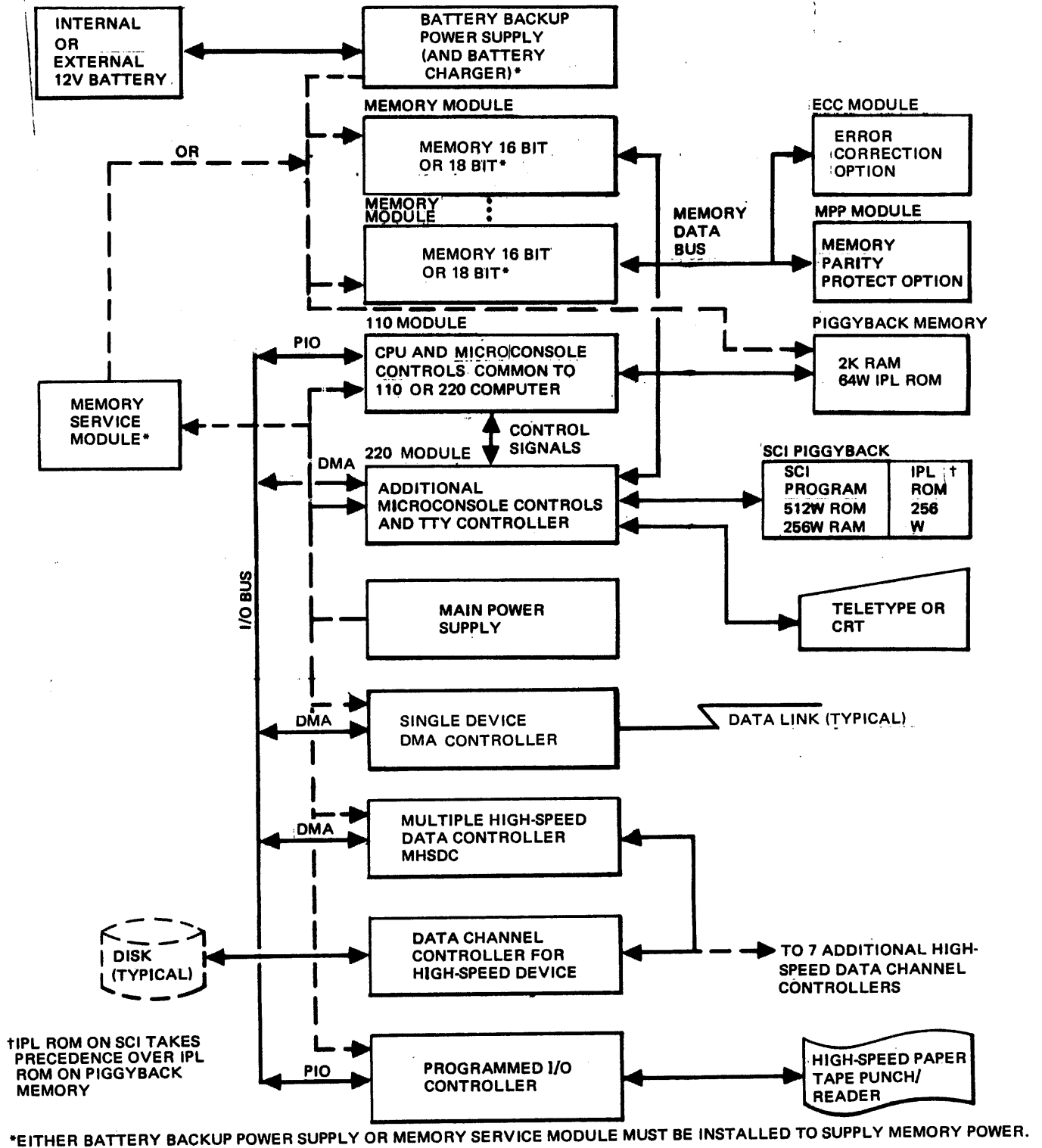


508-2.1.

*EITHER BATTERY BACKUP POWER SUPPLY OR MEMORY SERVICE MODULE MUST BE INSTALLED TO PROVIDE MEMORY POWER.

Figure 2-1. GA-16/110 Components

88A00508A-E



508-2-2

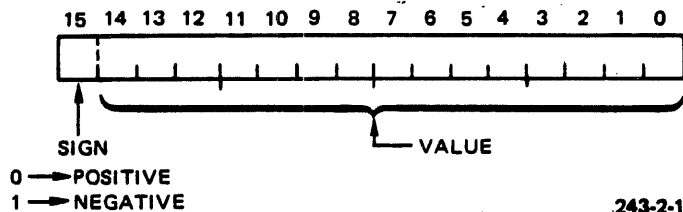
Figure 2-2. GA-16/220 Components

2.3 DATA FORMAT

The interpretation of data is largely a function of the program that operates on that data. A data word may be interpreted as an arithmetic quantity, a logical word, an address, two 8-bit bytes, or 16 bits (each of the 16 having a significant independence of other bits).

2.3.1 ARITHMETIC QUANTITY (SIGN BIT)

The GA-16/110/220 hardware performs arithmetic operations in parallel on 16-bit quantities using two's complement, binary arithmetic. In a word containing a signed integer, the high-order bit represents the sign.



A positive number is presented as its binary equivalent in the value portion of a word.

The representation for the negative of a number is obtained by taking its two's complement with the sign bit included in the operation. A two's complement operation may be performed by taking the one's complement (or logical complement of the number) and adding one to the result, ignoring any carry out of bit 15.

A 16-bit word can represent any negative number in the range of $-32,768_{10}$ (8000_{16}) to -1 ($FFFF_{16}$) and positive numbers from $+1$ (0001_{16}) through $+32,767_{10}$ ($7FFF_{16}$). Zero is considered a positive number since its sign bit is reset.

NOTE

Rather than representing hexadecimal numbers as $nnnn_{16}$, these numbers are also represented as $X'nnnn'$ (i.e., $X'7FFF'$, $X'21'$, $X'3F'$, etc.) unless otherwise noted. The $X'nnnn'$ notation corresponds to the convention used to specify hexadecimal numbers when writing CAP-16 assembler statements.

2.3.2 LOGICAL WORDS (NO SIGN BIT)

The GA-16/110/220 hardware performs logical operations in parallel on 16-bit words. The four logical operations are: (1) logical complement, (2) logical AND, (3) logical OR, and (4) logical Exclusive-OR (XOR). The logical complement is performed on a single word. The other logical operations are performed on corresponding bits of two words.

The logical complement (or one's complement) of a word is obtained by changing each 1 bit to 0 and each 0 bit to 1.

Example:

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

The complement of:

is:

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

The logical AND of two words is the result obtained by the logical AND of each bit in the first word with the corresponding bit in the second word. The logical AND of two bits results in a 1 only if both bits are 1. The AND operation (indicated by \wedge) can be represented in a truth table as follows:

A_i	\wedge	B_i	$= R_i$	NOTE: The subscript i denotes bit position in a word.
0	0	0	0	
0	1	0	0	
1	0	0	0	
1	1	1	1	

Thus

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

AND

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

results in:

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

88A00508A-E

The logical OR of two words is the result obtained by the logical OR of each bit in the first word with the corresponding bit in the second word. The logical OR of two bits results in a 1 if either bit is 1. The OR operation (indicated by v) can be represented in a truth table as follows:

$$A_1 \vee B_1 = R_1$$

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

Thus

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

OR

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

results in:

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

The logical XOR of two words is the result obtained by the logical XOR of each bit in the first word with the corresponding bit in the second word. The logical XOR of two bits results in 1, only if the two bits are different. The XOR operation (indicated by Ψ) can be represented in a truth table as follows:

$$A_1 \Psi B_1 = R_1$$

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

Thus

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

XOR

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

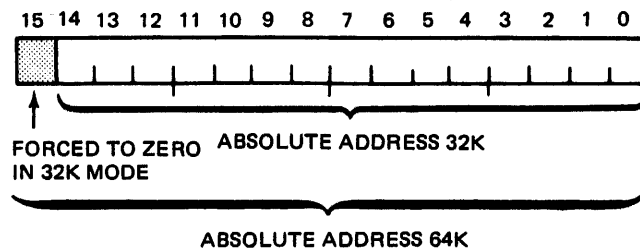
results in:

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

2.3.3 ADDRESSES AND MEMORY MODE

A memory location is identified by a number called its address. The GA-16/110/220 memory is directly addressable to a capacity of 64K (65,535₁₀ words). The highest memory address which can be referenced is selectable. The terms 32K or 64K memory mode refer to the capability of selecting whether the processor will use 16 or 15 bits to define the addresses of data or instructions in memory. The GA-16/110 may be run in the 32K mode by wiring the +64KM line to ground. The GA-16/220 may be run in the 32K mode by setting MODE switch or by executing an I/O instruction for memory mode change. The 32K mode allows software, developed for the General Automation SPC-16 series computers, to also run on the GA-16/220. For further details on memory mode change, refer to Section 4.15.2.4.

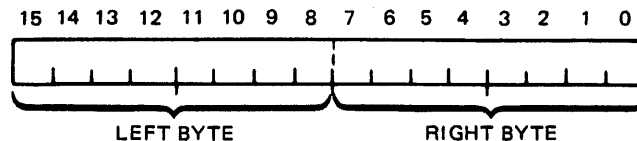
In the 64K mode, memory addresses may range from 0000 to FFFF₁₆; in the 32K mode, the upper address limit is 7FFF₁₆. When a memory location or hardware register contains an address, its format is as shown below:



When the address field of an instruction and the contents of a hardware register are used in determining an address (e.g., indexed addressing), the address calculation is performed using 16-bit, signed numbers (negative numbers are in two's complement form).

2.3.4 BYTE DATA

A word may be considered as two bytes, designated the "left byte" and the "right byte".

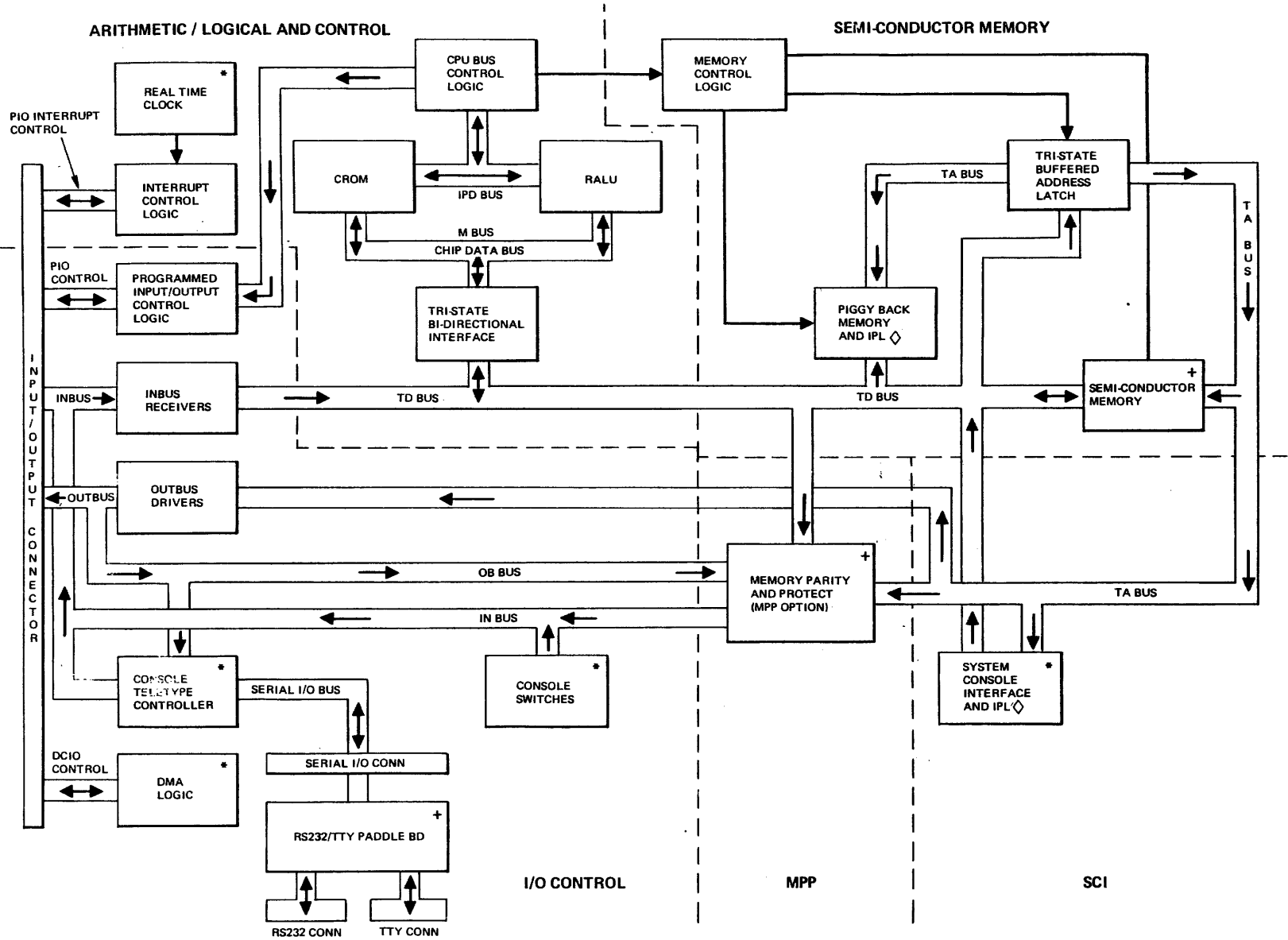


466-2.2.

An 8-bit byte is a convenient unit for representing some forms of data. For example, any of the 128 ASCII characters (see Appendix B for ASCII character codes) can be represented in a single byte. (Only the 7 lowest order bits of a byte are significant for ASCII character data; the high-order bit always contains 1.)

Instructions are provided to load or store any byte in memory, to zero the left or right byte of a register, and to exchange the two bytes in a register.

Figure 2-3. Internal Organization of the CPU



- Control Read-Only-Memory (CROM) — The CROM contains the microcoded programs used to control all instruction decode and gating internal to the CPU. The CROM receives the instruction, from memory, via the Memory Bus (M-Bus) and issues gating commands, based on the decode of the instruction in the Instruction register I. These commands are gated over the IPD-Bus and M-Bus. Register I in the GA-16/110/220 is internal to the CROM.
- The Register and Arithmetic/Logical Unit (RALU) — The RALU, in addition to being the computational/logical unit of the GA-16/110/220, also contains all high-speed registers, other than the instruction register I, which is internal to the CROM. These registers are the program counter (P), the working register (W), the status register (S), and the 16 general-purpose registers (foreground A, X, Y, Z, B, C, D, E, and background A', X', Y', Z', B', C', D', and E').
- Tri-State Bidirectional Interface — Data from the tri-state data bus (TD-Bus), that is to be written into memory, will be transferred through this interface. Similarly, data read from memory (excepting parity bits on 18-bit memories) is transferred in the opposite direction to the CROM/RALU. Specific memory addressing is accomplished via memory control logic and the tri-state address bus (TA-Bus) described in Section 3.

2.4.2 SEMICONDUCTOR MEMORY

All memory on a GA-16/110/220 is a semiconductor memory. Both dynamic memory (requiring refresh cycles) and static memory (requiring no refresh) are used for random-access read/write memory (RAM). Read-Only-Memory (ROM) is implemented by programmable (initially) read-only-memory chips (PROM). User-programmable read-only-memory is available with ultraviolet-erasable electrically-programmable memory chips (EPROM). The following components comprise the semiconductor memory and control:

- Memory Control Logic — This logic aids in selection of the correct memory module to be written or read, gating when an address is available on the TA-Bus, and determination of whether memory being accessed is external, piggyback, IPL, or system console interface.
- Tri-State Buffered Latch — This logic outputs the address of the memory word to be accessed to the appropriate semiconductor memory unit. The address is also gated to the MPP option (if installed) for checking whether address is write protected.
- Piggyback Memory — This memory is an option which is installed on the CPU-1 module and is normally either static RAM or a combination of EPROM and static RAM. The piggyback memory also may contain an IPL ROM (more detail in Section 3.7).
- Semiconductor Memory — This memory is formed from 4096-word (4K), 8192-word (8K), 16384-word (16K), 32768-word (32K), or 65536-word (64K) dynamic RAM memory boards. Two word lengths are available in each size; a 16-bit word and an 18-bit word. The extra two bits are used to write odd parity for the upper and lower bytes. When a read occurs, the parity bits are checked and even parity causes an error condition to be indicated. It further indicates that action may be controlled via the MPP option, if installed, or by the memory module itself (more detail in Section 3.2.7). An error correction module may be used with the 32/64K boards; it increases the effective word length to 22 bits (refer to Section 2.7).
- System Console Interface — Described in Section 2.4.3 and Section 3.

2.4.3 SYSTEM CONSOLE INTERFACE (SCI) (GA-16/220 ONLY)

The System Console Interface contains a 256-word random-access-memory (RAM) and a 512-word read-only-memory (ROM) to provide an interactive console teletype interface utility program. Optionally, the SCI may contain a 256-word IPL ROM. The SCI also contains several control switches that are described in the next section (reference Table 3-3).

2.4.4 MEMORY PARITY PROTECT (MPP)

The optional Memory Parity Protect module receives memory data to verify memory parity. For write protection, the MPP contains an internal matrix, set by program instructions, to protect 1K segments, of memory from access; both program and DMA accesses may be independently inhibited. Through this protect feature, the MPP receives the address of the memory location being accessed from the Memory Address latch for verification that the specified location is not a protected location.

2.4.5 INPUT/OUTPUT (I/O) CONTROL

The Input/Output Control section of the GA-16/110/220 contains all circuitry necessary for gating and control of data input and output from peripheral devices.

- The Programmed Input/Output Control logic oversees and controls all I/O transfers that are non-DMA in nature. This would include all program issuances of DTOR/M, DTIR/M, Control and Test instructions.
- The In-Bus Receivers accept data from the MPP, Console TTY Controller (220 only), or I/O Bus. The data from the In-Bus Receivers are passed to the correct destination via the TD-Bus.
- The Out-Bus Drivers/Out-Bus Latches — Output information from the TD-Bus is gated into the Out-Bus latches from which it may be transferred to the MPP module, console TTY controller (220 only), or to the Out-Bus drivers. Output from the Out-Bus drivers is destined for the I/O Bus.
- The Console Data Switches (220 only) are the 16 miniature rocker-type switches on the microconsole used for manual entry of data via the In-Bus.
- The Console Teletype Controller (220 only) is a Universal Asynchronous Receiver/Transmitter (UART) logic chip. The UART serial I/O controller is a standard, internal, controller that performs serial-to-parallel and parallel-to-serial conversion of teletype or CRT output and input data. All communication with the teletype or CRT is accomplished through the Console Teletype Controller over the serial I/O Bus via the RS232/Current Loop Adapter Card (TTY Paddleboard).
- The DMA logic (220 only) controls all Direct Memory Access input/output over the I/O Bus to/from memory.

2.5 MAJOR REGISTERS

To clarify the functions of the internal registers of the GA-16/110/220, further descriptions are presented in Sections 2.5.1 through 2.5.8.

2.5.1 GENERAL-PURPOSE REGISTERS

Sixteen general-purpose programmable 16-bit hardware registers are standard on the GA-16/110/220. These registers, located in the RALU chip, are functionally divided into two sets, eight foreground registers and eight background registers.

Table 2-1 shows the function of these registers.

Table 2-1. Register Contents

Foreground or Background Register	Description	Function	CPU Logic	Register Select
A or A'	General-Purpose ¹	Accumulator	000	Register A
X or X'	Index/General-Purpose	Used for postindexed addressing	001	Register X
Y or Y'	Index/General-Purpose	As post index-register	010	Register Y
Z or Z'	Index/General-Purpose	As post index-register	011	Register Z
B or B'	General Purpose ¹	Accumulator	101	Register B
C or C'	General-Purpose ¹	Accumulator	101	Register C
D or D'	Base-Relative Addressing/General-Purpose	As preindex register, contains the base address for base-relative addressing	110	Register D
E or E'	Subroutine Linking/General-Purpose	During subroutine or interrupt execution, contains return address and interrupt system enable control bit	111	Register E

¹ For hardware multiply (MPY) and divide (DIV) instructions, register A, B, and C have the following usage:

<u>A</u>	<u>B</u>	<u>C</u>	
Multiplicand	-	Multiplier	} MPY
-	Product (MSBs)	Product (LSBs)	
Divisor	Dividend	-	} DIV
-	Quotient	Remainder	

The general-purpose registers are implemented as a fast-access, "scratch pad" memory. A four-bit code is used to select the register being referenced for reading or writing. The high-order bit of the select code comes from the Foreground (F) indicator; the three low-order bits are supplied by the CPU logic as required for the execution of a particular instruction.

F INDICATOR

1 → Foreground Registers (A - E)

0 → Background Registers (A' - E')

By using the Foreground/Background instruction, either one of two sets of identical register groups may be used ($A \leftrightarrow E$ and $A' \leftrightarrow E'$). Only one set of registers is active at a given time. Any reference to a register, say Register A, refers to the active one of the pair, either A' if in background mode or A if in foreground mode. However, the inactive registers retain their content while they are inactive.

Subsequent register descriptions apply equally to foreground or background registers. All discussions of register operations imply the active set.

Instructions are provided to perform the following operations on any of the eight general-purpose registers.

- Load register from memory (word or byte).
- Store register in memory (word or byte).
- Arithmetic and logical operations between any two registers: ADD, SUBtract, OR, AND, exclusive OR (XOR) and transfer contents of one to another.
- Arithmetic and logical operations between any register and memory (literal addressing).
- Compare any register with memory.
- Increment, decrement, complement, or clear any register.
- Shift bits in any register, right or left.
- Exchange the bytes in any register.
- Execute the contents of any register as an instruction.
- Input and output of data to any peripheral may be performed with any register.

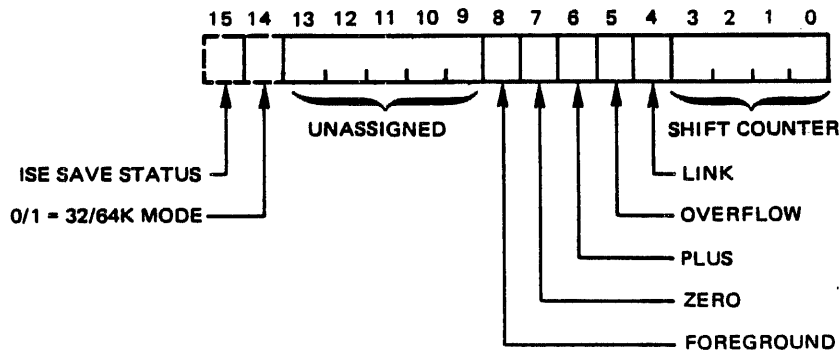
2.5.2 ISE AND INTERRUPTS

The ISE flip-flop controls the inhibitible (IN) interrupts. When it is set (=1), the IN interrupts are enabled. When reset (=0), the IN interrupts are disabled. The ISE flip-flop may be set/reset under program control when INE, INH, RTRN, RISE, RTNIV and JSR instructions are executed. The status of the ISE flip-flop is maintained in bit 15 of the program counter (register P) in 32K mode and in bit 15 in the status register (S) in 64K mode. After executing an instruction which resets the ISE, the status bit may not reflect the actual ISE state, since its purpose is to preserve the contents before instruction was executed so that state may be reestablished.

In addition to the ISE flip-flop, an individual mask bit may be associated with each device which can generate an IN interrupt. Both the mask bit and the ISE flip-flop must be set for an interrupt to occur. Further details on interrupts are contained in Section 2.6.

2.5.3 STATUS REGISTER (REGISTER S)

The status register (S) contains ISE save status, the 32K/64K memory mode indicator, the Shift Counter (4-bits) and the Link, Overflow, Plus, Zero and Foreground/Background indicators.



508-24

Instructions TSR and TRS are provided to transfer the contents of register S to any general-purpose register and vice versa. Its contents are also loaded from or stored into memory with the execution of a LARS (load all registers and status) or SARS (store all registers and status) instruction.

An indicator in register S is "set" if it contains a one, and is "reset" if it contains a zero. The bits in the status register are described as follows:

- The ISE indicator (bit 15), also referred to as S_{15} , relates to the status of the IN class of interrupts.
- The operations which set the ISE flip-flop (ISE) or the ISE save indicator (S_{15}) are tabulated in Table 2-2. It is the ISE flip-flop which determines whether the interrupt system is enabled, not the ISE indicator; therefore, S_{15} does not necessarily show the current interrupt status. This is a design feature which allows a program to reset ISE to a previous status by using the stored, former status.

88A00508A-E

Table 2-2. Operation Affecting ISE F/F or ISE Status Indicator

Operation	ISE F/F	ISE Save Indicator S ₁₅
IN or NI interrupt	Reset (=0)	= ISE F/F before reset
INE instruction	Set (=1)	No change
INH instruction	Reset (=0)	No change
JSR instruction	Reset (=0)	= ISE F/F before reset
LARS instruction	No change	= bit 15 of eighth word loaded
RTNIV instruction	= bit 15 of specified address+1	No change
RTRN instruction	= bit 15 of specified register (32K mode) = S ₁₅ (64K mode)	No change
TRS instruction	No change	= bit 15 of specified register

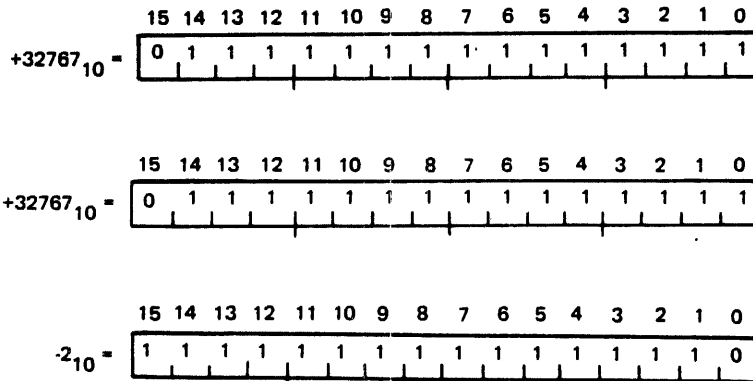
NOTE

When operating in the 32K mode, the status of ISE is also stored in bit 15 of the E register when appropriate instructions are executed.

- The 32K/64K memory mode indicator (bit 14) identifies the maximum address available to the CPU. In the 32K mode, 15 bits of register P are used for addressing and the most significant bit contains ISE. In the 64K mode, the full 16 bits of register P are used for addressing and ISE save status is contained in register S. The 32K mode allows software compatibility with the SPC-16/40 series computers. The memory mode indicator is unaffected by LARS and TRS. On a GA-16/110, the memory mode is hard-wired. On a GA-16/220, the memory mode is switch-selectable for 32K or can be changed by outputting the appropriate mask word (see Section 4.15.2) or by interrupts. Either an IN or an NI interrupt will place the CPU in the 64K mode while CPU reset or power up places the CPU in the 32K mode.
- Bits 13-9 of the status register are unassigned. These bits can be changed with LARS or TRS instructions, and are transferred with SARS or TSR instructions.

88A00508A-E

- The Foreground indicator and control (bit 8) contains 1 if foreground registers are active and 0 if the background registers are active. Instructions are provided to set or reset this bit (FMS or BMS).
- The Zero indicator (bit 7) is set if an arithmetic, logical or transfer operation results in zero (i.e., all bits of the result are 0). Otherwise, it is reset indicating that the result is not zero. Instructions are provided to test this indicator.
- The Plus indicator (bit 6) is set to indicate that the result of the last arithmetic transfer or logical operation is positive (i.e., bit 15 is 0). If the result is negative, the Plus indicator is reset. Instructions are provided to test the Plus indicator.
- The Overflow indicator (bit 5) indicates an arithmetic overflow for an add or subtract instruction. The Overflow indicator is set (=1):
 - If the sum of two numbers of like sign results in a different sign, or
 - If the difference of two numbers of unlike sign results in a sign different from that of the destination register (minuend).
- Overflow conditions occur when the result of an arithmetic operation is not in the allowed range (i.e., not between $-32,768$ and $+32,767$). For example, if $32,767$ is added to $32,767$:



486-2-5

- The Overflow indicator is not necessarily an error indicator. If a program is dealing with only unsigned integers (i.e., using all 16 bits to contain the value, allowing numbers in the range 0 to 65535_{10}), the destination register's contents may be read as $+65534_{10}$ (the correct sum) instead of -2_{10} . However, the overflow indicator would be set by this operation since 16 bits cannot correctly represent the signed result in two's complement form. Instructions are provided to test the overflow indicator for 1 or 0.

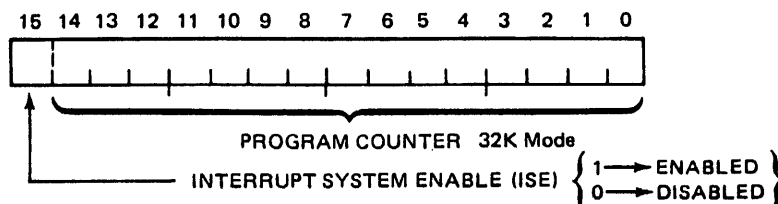
- The Link indicator (bit 4) is affected by two types of operations: arithmetic and shift.
 - Arithmetic — The Link is set (=1) if there is a carry, or reset (=0) if there is no carry, from bit 15 of a previous arithmetic operation. The link is particularly useful in multiple-precision operations where more than one 16-bit word is used to represent a quantity; an instruction is provided to add the contents of the Link to any general-purpose register. Thus, a carry out of bit 15 from one register containing the least significant half of a number may be added to another register containing the most significant half of the number, allowing the contents of the two registers to be treated as a single number.
 - Shift — During the execution of a shift instruction, the Link indicator contains the last value shifted out of bit 0 or bit 15 of the selected register.
- Instructions are provided for setting, resetting, and testing the Link indicator.
- The Shift Counter (bits 3 to 0) is used in the execution of the Shift Right, Multiply, and Divide instructions. A shift right instruction specifies a general-purpose register and the number of shifts to be performed. Each time the selected register is shifted, the Shift Counter is incremented and compared with the corresponding bits in register I (which contains the number of shifts specified by the instruction) to determine whether or not another shift cycle is required. At the completion of the operation, the Shift Counter will contain the number of shifts minus one that were actually performed. The value of the Shift Counter may be read by the ADDS instruction. Shift Left instructions are single bit, and do not affect the shift counter.

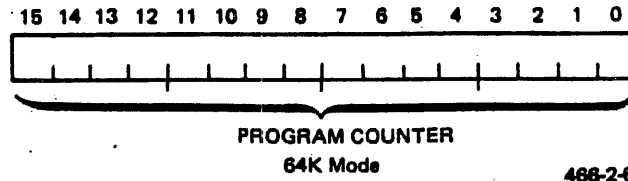
2.5.4 PROGRAM COUNTER (REGISTER P)

The 16-bit register P is used as the program sequence register and is also called the "Program Counter".

In the 32K mode, the high-order bit of register P will reflect the Interrupt System Enable (ISE) flip/flop. (However, this bit is not included in address calculations.) In the 64K mode, all 16 bits are used for addressing.

When discussing program sequencing, the symbol "P" refers to the contents of bits 14 to 0 in the 32K mode, and bits 15 to 0 in the 64K mode.





The Program Counter normally contains the address of the next instruction to be executed. Its contents are transferred to the TA Tri-State Latch following the execution of the current instruction, to read the next instruction from memory; then its contents are incremented (its contents are incremented again if the instruction is a double-word instruction). Instructions that modify instruction sequencing (e.g., JMP) do so by altering the contents of register P. Register P can be changed by "jump", "conditional jump", (skip), "jump-to-subroutine" and "return" instructions.

2.5.5 REGISTER I (NON-PROGRAMMABLE)

Register I (16 bits) is the instruction register. When an instruction is first accessed from memory it is transferred to I. The instruction decode logic operates with the bit pattern in register I to determine the sequence of events necessary to execute the instruction. If an instruction is two words long, register I holds only the first word. As soon as register P is incremented (or loaded manually) to point to the next instruction, register I will be loaded (i.e., register I contains the next instruction to be executed).

2.5.6 TA TRI-STATE LATCH

The Tri-State Latch contains the absolute address of the memory location to or from which an instruction as address or data is to be read/stored.

2.5.7 REGISTER W (NON-PROGRAMMABLE)

Working register (W) usually holds the effective address (EA) for address calculations. In effective address (EA) calculation, register W is utilized as follows:

- In base-relative address, register W is truncated from bit 10 through 15.
- In program-relative address, register W will have 9 extended through 15 for sign purposes.
- In skip instructions, register W bit 8 will be extended through 15 as a sign representation.
- In a single word memory reference with indexing instructions, register W will truncate above bit 4.

2.5.8 CONSOLE SWITCHES

The console switches (available only on a GA-16/220 computer) are part of the CPU-2 module microconsole. The operator sets the console switches (refer to Section 3, Table 3-3, items 5 and 6). The RCSR and RCSM instructions are used to check the conditions of the switches.

2.6 HARDWARE INTERRUPT (NI AND IN)

A program interrupt interrupts the program stream and forces a new sequence of commands to be executed before returning to the interrupted program. An interrupt signal accomplishes this by forcing a hardware jump-to-subroutine (JSR) indirect through a memory location dedicated for that purpose called the interrupt vector. The interrupt vector must contain the address of the interrupt subroutine. See Tables 2-3 and 2-4. It also stores the location plus one (P+1) of where it was interrupted and the old status of ISE (Interrupt System Enable). The dedicated memory location's address is generated on the interrupting controller or CPU logic and is passed to the CPU interrupt logic on the In-Bus during Interrupt Acknowledge (IACK) time. The place where P+1 and ISE is stored is a function of what mode the CPU is in and what type of interrupt occurred. All IN and NI interrupts force the CPU into the 64K memory mode and reset the ISE F/F. ISE being reset inhibits the IN interrupt group. P+1 and ISE storage is as follows:

PROGRAM INTERRUPTS

<u>IN - 32K Mode</u>	<u>IN - 64K Mode</u>	<u>NI - 32K Mode</u>	<u>NI - 64K Mode</u>
P+1→ E ₁₄₋₀	E ₁₅₋₀	(ADRS) ₁₄₋₀	(ADRS) ₁₅₋₀
ISE→ E ₁₅ , S ₁₅	S ₁₅	{ (ADRS) ₁₅ (ADRS+1) ₁₅	(ADRS+1) ₁₅

ADRS is the first of two dedicated memory locations (beginning at X'78') as shown in Tables 2-3 and 2-4.

In the GA-16/220, all (IN and NI) interrupts disable ISE when acknowledged. The burden is on the program to turn ISE on again if other interrupts are to be activated while one interrupt is being serviced.

The basic purpose of interrupts is to free the program from having to periodically test the status of a device to see if a particular event has transpired. Instead of periodically "looking" at a device, a program's attention will automatically be diverted (interrupted) when the event happens. The second function of an interrupt is to do the bookkeeping necessary to accomplish an orderly return to the interrupted program when the interrupt must remember where it came from. Interrupts are serviced by priority. General types of priorities are from highest to lowest, i.e., NI and IN.

Table 2-3. Interrupt Summary

NI Interrupts	Hexadecimal Vector	Hexadecimal Storage ADRS P+1/ISE ^①
Power Fail - PFD	40	78/79
Auto Restart - RS	41	Register E/S
Memory Parity Protect - MPP		
Parity Error	42	7A/7B
Write Protect Error	42	7A/7B
Program Timeout	42	7A/7B
Program Sequence Interrupt - TRAP	44	7C/7D
*TTY Break/Single Step - CTRL 1,X'3E'	46	7E/7F

① ISE is stored in bit 15 of the specified location

NI Associated Instructions - RTNIV, Section 4.9.

IN Interrupts (Require Mask and ISE on)	Priority	Hexadecimal Vector	Hexadecimal Storage P+1/ISE
***Real-Time Clock - RTC	1	43	E ₁₄₋₀ /E ₁₅ E/S ₁₅
**Teletype Busy - TTY	2	45	↑ ↑
*Console Interrupt - CI	3	47	↓ ↓
<div style="display: flex; align-items: center;"> <div style="border-left: 1px solid black; border-right: 1px solid black; padding: 0 5px;"> (External From I/O Control- lers) </div> <div style="margin: 0 10px;">}</div> <div style="border-left: 1px solid black; border-right: 1px solid black; padding: 0 5px;"> (Priority Determined By Physical Location) </div> <div style="margin: 0 10px;">}</div> <div style="border-left: 1px solid black; border-right: 1px solid black; padding: 0 5px;"> Vector Wired In Controller Logic </div> <div style="margin: 0 10px;">}</div> <div>See Table 6-1</div> </div>			
			E ₁₄₋₀ /E ₁₅ E/S ₁₅
			32K 64K

- * GA-16/220 only
- ** GA-16/220 only; TTY may be provided with external 1582 controller.
- *** 1.0 millisecond pulses and mask provided on GA-16/220
A GA-16/110 requires external pulses and enabling signal.

Mode

IN Associated Instructions - RTRN, RISE, INH, INE, JSR (Affects ISE), DTOR/M X'3E' (Affects mask for RTC, CI and TTY).

Table 2-4. Dedicated Memory Map

Hexadecimal Location	Contents
0 0 8 0	Standard entry point for operating system monitor (E\$MON)
⑥ 0 0 7 F	ISE } *Single Step, Break
⑥ 0 0 7 E	P+1 }
⑤ 0 0 7 D	ISE } TRAP
⑤ 0 0 7 C	P }
③ 0 0 7 B	ISE } Memory Management
③ 0 0 7 A	P+1 }
② 0 0 7 9	ISE } Power Fail
② 0 0 7 8	P+1 }
0 0 7 7	External Interrupts from I/O Controllers
0 0 4 7	*Console Interrupt Vector
⑥ 0 0 4 6	*Single Step, Break
0 0 4 5	**TTY Not Busy Vector
⑤ 0 0 4 4	Op Code TRAP Vector
0 0 4 3	RTC Vector
③ 0 0 4 2	Memory Management Vector
④ 0 0 4 1	Restart Vector
② 0 0 4 0	Power Fail Vector
0 0 3 F	CAR DC 15
0 0 3 E	SCR DC 15
↓	
0 0 2 1	CAR DC 0
0 0 2 0	SCR DC 0
① 0 0 1 F	SFV, } SAB1
0 0 1 E	CSP }
① 0 0 1 D	SFV, } SAB0
0 0 1 C	CSP }
0 0 1 B	
↓	
0 0 1 8	
0 0 1 7	
↓	
0 0 1 0	
0 0 0 F	
↓	
0 0 0 8	
① 0 0 0 7	P+1 - Stack Fault Return
↓	
0 0 0 1	
0 0 0 0	Dynamic Storage Point (DSP), Standard Software

Temporary Storage of Interrupt Return Location and Interrupt System Enable Bit (ISE)

Interrupt Vectors

*High-Speed Data Channel Block Size and Address Initialization

Operating System Use

User Program Use

Operating System Use

NO WRITE PROTECT

*GA-16/220 only
 **GA-16/220 or GA-16/110 with 1582 controller.
 Circled number items are associated functions. Locations are hexadecimal.

2.6.1 IN INTERRUPTS

IN interrupts are controlled as a group by the ISE flip-flop, and individually by interrupt masks. IN interrupts are typically those interrupts generated in a device controller external to the CPU. IN interrupts (Figure 2-4) can also be generated by logic internal to the CPU package from the Real-Time Clock (RTC), Console Interrupt (CI) or the Teletype Controller (TTY). The interrupts inside the CPU chassis have a higher priority than those inside the I/O chassis. The internal interrupts are in the priority order of RTC, TTY, and CI. The priority of the external interrupts are in accordance with the physical location of controllers in the I/O section. The controller closest to the CPU module has the highest priority with priorities descending with distance from the CPU module; (i.e., the module farthest from the CPU module has the lowest priority). Any module will work in any slot of the I/O section so long as there are no empty slots between it and the CPU module (shorting boards are available as desired to fill slots). Thus, priorities are changed easily. Priority as it is used here means which interrupt is acknowledged if two or more become true at the same time. It does not refer to nesting type priority where a higher priority (NI interrupt), will interrupt out a lower priority (IN interrupt) interrupt.

Figure 2-4 shows the flow of a typical IN interrupt operating in a 64K mode. The example assumes an Analog-to-Digital Converter (A/D) in the I/O chassis will generate interrupt vector address X'55' (refer to Note 2 in Figure 2-4), when it had a request on line. The program issues a command in location X'2020' to tell the converter to start. In locations X'2021' and X'2022', it then enables the mask and ISE to allow the request to be acknowledged when the conversion is finished. The program need not monitor the A/D since, when the conversion is finished, it will generate an interrupt. In the example, the interrupt request, signaling the finish of the A/D conversion, occurs during the instruction execution X'2040'. When the CPU acknowledges the interrupt request, the controller passes the interrupt vector address X'55' to the CPU which then forces a hardware JSR. The hardware JSR zeros the ISE, stores the old value of ISE in the Status Register bit 15, and stores the incremented program counter (P+1 = next instruction address) in register E.

NOTE

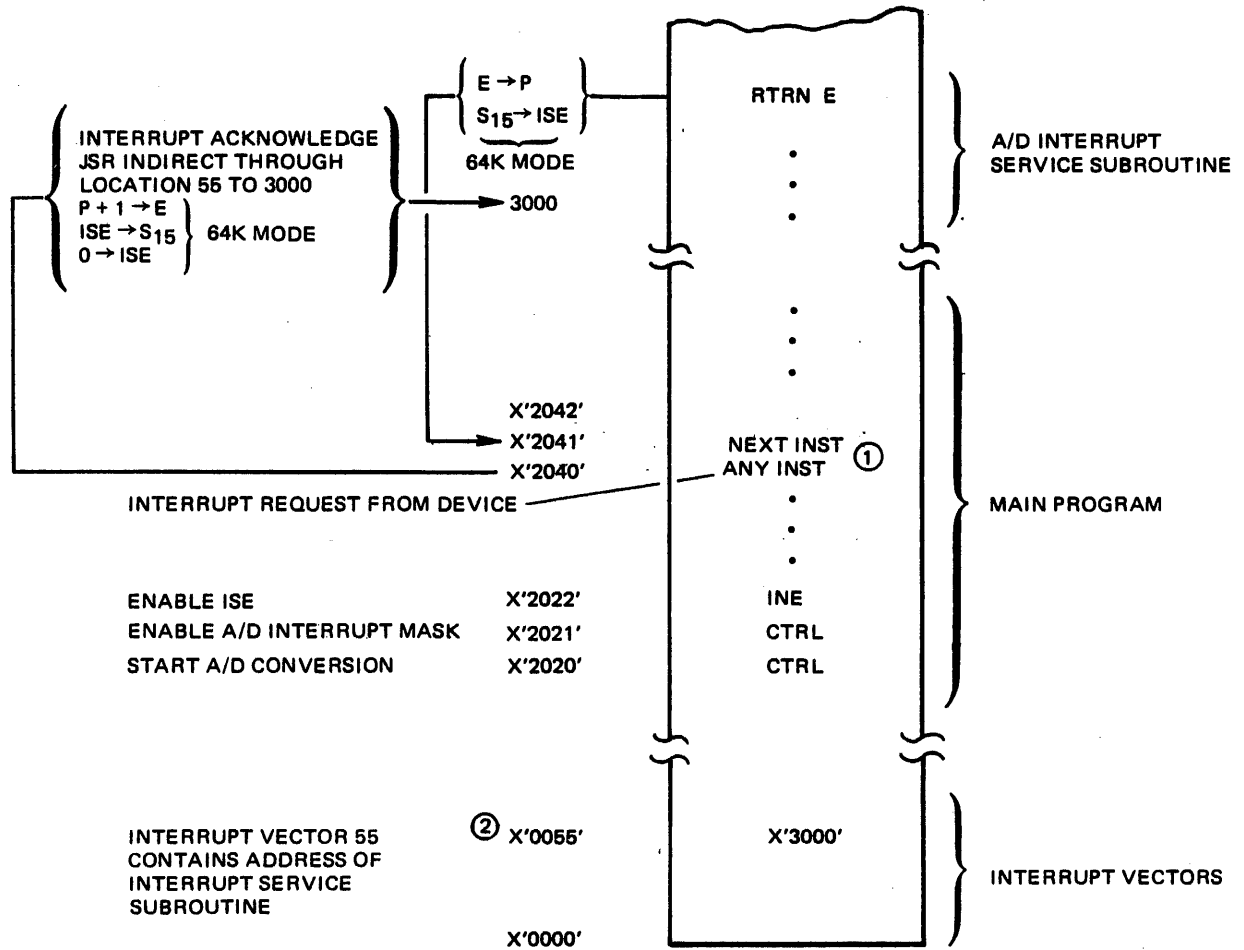
If the CPU were in the 32K mode, the example in Figure 2-4 is correct, except that both P Counter (15 bits in the 32K mode) and ISE are stored in register E. Also, the RTRN operates on just register E and not S₁₅.

The interrupt service routine is then entered at location X'3000'. The subroutine typically operates on the A/D data and asks for another conversion (on a different input line). Regardless of what the subroutine does, it will end with an RTRN; a Return through the E and S registers. This restores ISE and the program counter and execution continues at location X'2041'. Refer to Section 6 for further discussion.

NOTE

If the instruction at X'2040' was such that it turned off ISE, the interrupt would not be acknowledged until ISE was again turned on.

The time required for the CPU to execute the "Forced JSR" for IN interrupts (including the Housekeeping, P-SAVE, etc), is 3.1 microseconds, i.e., the time between exiting the main program and getting to the service routine.



508-2-5

¹Any instruction that is interruptable.
This excludes JSR or any instruction that resets ISE.

²Locations are hexadecimal. Hexadecimal locations are identified by X'nnnn' to correspond with CAP-16 assembler convention for specifying a hexadecimal numbers.

Figure 2-4. IN Interrupt Flow

2.6.2 NI INTERRUPTS

NI interrupts denote an error condition and are unaffected by the ISE state. When NI interrupts occur, the P+1 and ISE status bit are stored in dedicated memory locations as shown in Table 2-3. Unlike IN interrupts, NI interrupts may occur immediately after a JSR instruction because, although it resets the ISE, the status of ISE is ignored. The return from an interrupt routine is accomplished with an RTNIV instruction (Section 4.9). The time required for the CPU to execute the "Forced JSR" for NI interrupts is 5.85 microseconds. In other respects, programming considerations for NI interrupts are the same as for IN interrupts.

2.7 ERROR CORRECTION OPTION

The error correction option provides single-bit error correction and multiple-bit error detection for a 32K or 64K memory board. The error correction option consists of a single printed circuit module which plugs into a 32Kx16-bit or 64Kx16-bit memory board.

To perform error correction and detection, the ECC board generates and stores six check bits as each data word is written into memory. During memory read, a second group of six check bits is generated and exclusive-ORed with the check bits generated during write. If the result is zero, the data word was stored and read correctly. A single-bit error results in a non-zero result with the bit position in error specified by the decode of the result. Multiple-bit errors also generate a non-zero result from the exclusive-OR operation, but the decode does not indicate positions for the erroneous bits.

NOTE

Upon initial power up, all ECC check bits are incorrect. All memory locations must be initialized by a store operation before the error correction and detection is enabled. Failure to initialize the memory will cause false error indications.

2.7.1 OPERATIONAL MODES

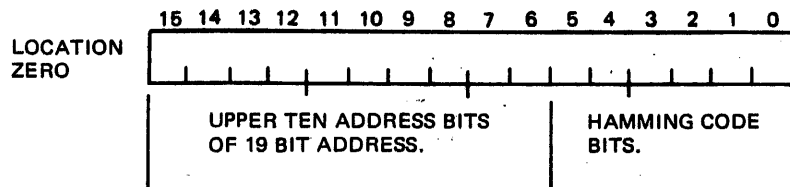
Operational modes for the error correction options are established by the program. The error correction option utilizes the following operational modes:

- Error Correction OFF — Disables the error correction feature only.
- Normal 1 Mode — The ECC module generates an interrupt on both correctable (single-bit) and non-correctable (multiple-bit) errors.
- Normal 2 Mode — The ECC module generates an interrupt on non-correctable errors only while single-bit errors are corrected.
- Write Data Only Mode — Used for test or diagnostic purposes only. New check bits are generated but not stored in memory. The original check bits remain.
- Input Status Mode — Read status word from ECC module if the associated memory board has an error (see Section 2.7.1.1).

2.7.1.1 Input Status Mode

The input status mode allows a status word to be generated by the error correction board for each memory module with an error.

A status priority signal from the CPU is propagated through each error correction board. The first board with an error will trap the status priority signal. Executing a read of memory location zero will cause the error correction board to place the following status word on the data bus:



508-2-6

Following the read command, the status priority signal is sent to the next error correction board having an error. This board also blocks propagation of the priority status signal, and responds to a CPU read of location zero. After transmitting the status priority signal, the CPU continues to read until a status word of all zeros has been received, indicating no more errors exist.

NOTE

Normal Mode Two, single bit errors also cause the ECC to generate a status word which can be examined utilizing the Input Status Mode. However, there is no interrupt for single-bit errors.

2.7.2 MODE SELECTION

Mode selection for the error correction option is accomplished by executing a DTOR to the MPP with the appropriate error correction mode coded into bits 6 through 4 of the output data word. Table 2-5 lists the mode selection bits.

The mode selection may be changed by executing a DTOR, or by system reset. System reset will enable the "Error Correction OFF" mode.

2.7.3 ERROR CORRECTION INTERRUPTS

The following interrupts are generated by the error correction board:

- Correctable Error Interrupt
- Non-Correctable Error Interrupt

2.7.3.1 Correctable Error Interrupt

The Error Correction error signal will cause the MPP to generate a non-inhibitable interrupt.

2.7.3.2 Non-Correctable Error Interrupt

The error correction board will activate the parity error line to signify a non-correctable error. Response to this depends on whether the processor is equipped with MPP. The MPP will generate a non-inhibitable interrupt.

2.7.4 MPP STATUS WORD

The occurrence of both non-correctable and correctable errors may be distinguished by examining the MPP status word. The MPP status word that is set if an error has occurred in the error-correcting memory. The MPP status word is shown in Figure 2-5. The MPP module indicates correctable errors with bit 12; however, non-correctable errors are indicated by both the parity error bit and the Error Correction bit (bit 12) being on (one).

2.7.4.1 ECC Board Status Indicators

Two indicators on the error correction board indicate the error status. For a correctable error, the yellow indicator turns on. For a non-correctable error, the red indicator turns on, and the yellow may also be on.

Table 2-5. Mode Selection Bits

Bits	6	5	4	Mode
	0	0	0	Error Correction Mode Off
	0	0	1	Normal 1 Mode
	0	1	0	Normal 2 Mode
	0	1	1	Input Status Mode
	1	0	0	Write Data Only Mode

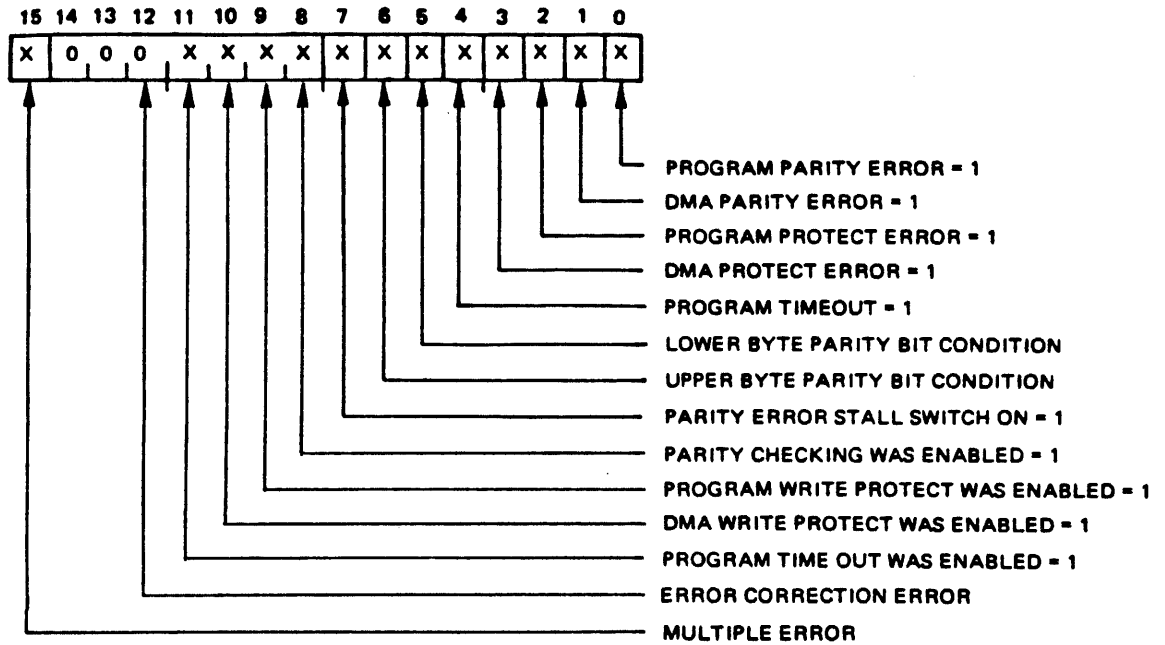


Figure 2-5. MPP Status Word

GA-16/220 configuration and operation 3

The configurations of both the GA-16/110 and the GA-16/220 are flexible due to their modular design. Because of this flexibility, the locations of controls and indicators described in this section may differ in a given application. Typical configurations are described to familiarize the reader with the functions applicable to each of these computers. This section describes typical configurations, modules, functions of controls and indicators, operating procedures, and use of the system console interface (SCI) program (on a GA-16/220).

3.1 TYPICAL GA-16/110/220 CONFIGURATIONS

A complete GA-16/110 computer with 2K of memory, an IPL ROM program, and interfaces to a programmed I/O or DMT I/O controller may consist of a single, printed circuit assembly.

A CPU-2 module (220) may be interconnected to the CPU-1 module (110) to create a GA-16/220 computer. The CPU-2 module adds the DMA port, serial I/O controller, 1 msec RTC, and additional controls and indicators. Both modules may be installed in a chassis using a 140-pin connector (GA part number 41A00105A03) or in a GA compact or jumbo chassis described in the following paragraphs.

A self-contained GA-16/110 system, with up to 64K of memory, or a GA-16/220 system, with up to 64K of memory, including a built-in power supply, five I/O controller modules, and provisions for I/O expansion may be installed in a single, relay rack chassis. The chassis which provides this capability, is termed a compact chassis.

A larger GA-16/110 system with 64K of memory or a GA-16/220 system with 64K of memory, eight I/O controllers, and a provision for external I/O expansion may be configured in a chassis termed a jumbo chassis. The main power supply is external to the jumbo chassis.

A battery compartment is provided in both the compact and jumbo chassis. A battery backup power supply may be installed which preserves the contents of the memory if there is a main power failure or when main system power is turned off.

Memory parity protect (MPP) is another option which may be provided by installation of an MPP module instead of one of the memory modules. To achieve parity error detection, memory modules with 18-bit words (16-bit programmable and two bits for parity) are installed.

The error correction option requires a 32Kx16-bit or 64Kx16-bit memory board and the piggyback ECC module.

Figure 3-1 and 3-2 show typical self-contained GA-16/110 and GA-16/220 computer systems in a compact chassis. Table 3-1 identifies the modules which may be installed in the chassis connectors, and identifies interfaces which are located at rear of the chassis. Figures 3-3 and 3-4 and Table 3-2, provide similar information for computer systems installed in a jumbo MIB chassis. The description of these systems is contained in Sections 3.1.1 through 3.1.4.

Modules which are used in GA-16/110/220 are described in Section 3.2 (I/O controllers excepted). Controls and indicators are identified on the figures by circled key-numbers and are described functionally in Section 3.3 (Table 3-3).

NOTE

Key numbers ① and ② do not apply to jumbo chassis since power supply is external. Key numbers ③ through ⑫ do not apply to GA-16/110 in either chassis since the controls and indicators represented are only used on a GA-16/220. Key numbers ⑭ through ⑳ apply to the memory parity protect module which may be used on either computer. The MPP numbers fall out of sequence in Figure 3-3 to illustrate MPP can be located in any connector from 012 through 017.

3.1.1 SELF-CONTAINED GA-16/110 SYSTEM

Figure 3-1 shows a self-contained 110 system (without memory parity protect or error correction option) installed in a compact chassis. As shown, this system has the following features:

1. Built-in power supply.
2. 110 CPU (CPU-1 module) with piggyback 2K RAM and single-device IPL ROM (see note).
3. 12K of additional memory using three 4K memory modules set to occupy address 0 through 8K and 12K through 16K.

NOTE

Total memory capacity is 14K; however, in this configuration (assuming piggyback RAM and IPL begin at 8K) the addresses from the end of the IPL ROM (10K+) through the next 4K boundary (12K) cannot be occupied by memory. A system programmer would avoid writing programs that reference these addresses. For further detail on memory allocation, refer to Section 3.2.7.

4. Battery backup supply.
5. Model 1582 variable asynchronous console controller to provide serial current loop or RS232 (for TTY/CRT).

6. High-speed paper tape reader/punch (HSPTR/P) controller.
7. Space for three additional I/O controller boards.
8. Cable interface driver (CID) permitting use of an external I/O chassis which may contain additional I/O controllers.

3.1.2 SELF-CONTAINED GA-16/220 SYSTEM

Figure 3-2 shows a self-contained 220 system (without memory parity protect or error correction option) installed in a compact chassis. As shown, this system has the following features:

1. Built-in power supply.
2. 220 CPU (CPI-1 and CPU-2 boards) with TTY controller and microconsole.
3. System console interface (SCI) module mounted on 220 board.
4. 16K memory using two 8K memory modules.
5. Battery backup supply.
6. Multiple high-speed data channel (MHSDC) controller.
7. Floppy disk controller (DMA Channel via MHSDC).
8. Space for two additional I/O controller boards.
9. Channel interface driver (CID) permitting use of an external I/O chassis which may contain additional I/O controllers.

3.1.3 LARGE GA-16/110 SYSTEM WITH MEMORY PARITY PROTECT OPTION

Figure 3-3 shows a GA-16/110 system with memory parity protect option. This system is installed in a jumbo chassis. Systems which use a jumbo chassis require an external power supply (not shown) since power requirements are greater than could be accommodated by a built-in supply. As shown, the system has the following features:

1. 110 CPU (CPU-1 module) with piggyback 2K RAM and single-device IPL ROM (see note).
2. 4K of unprotected memory using 4K memory module set to occupy location 12K through 16K (see note).
3. 32K of additional protected memory using four 8K memory modules set to occupy locations 0 through 8K and 16K through 40K (see note). (Error correction requires 32K or 64K memory board.)

NOTE

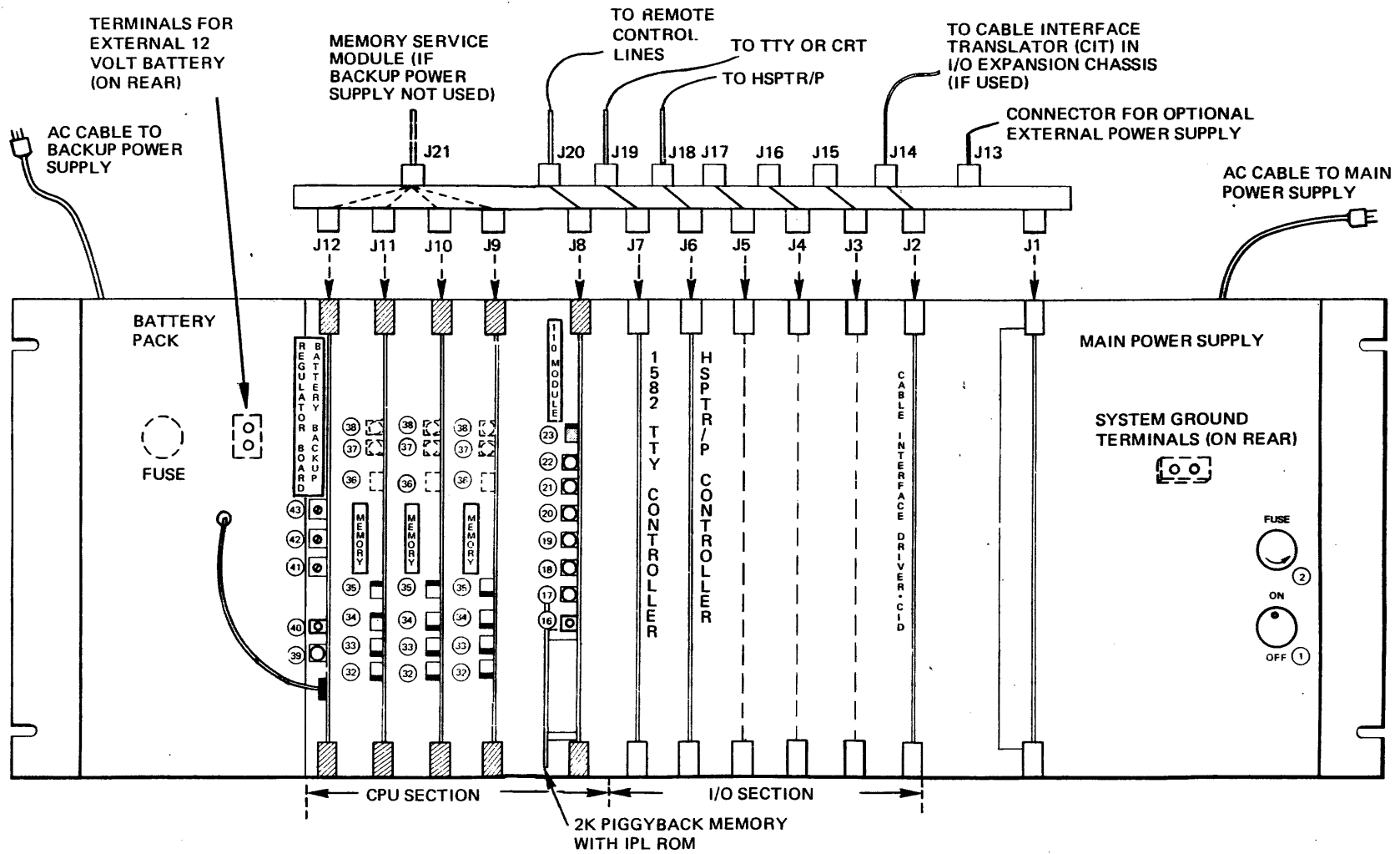
Total memory capacity is 64K; however, in this configuration (assuming piggyback RAM and IPL begin at 8K), the addresses from the end of the IPL ROM (10K+) through the next 4K boundary (12K) cannot be occupied by memory. A system programmer would avoid writing programs that reference these addresses. This configuration also illustrates that the switches on the memory modules, not their locations in the chassis, determine their addressing. For further detail on memory allocation, refer to Section 3.2.7.

4. Memory parity protect module.
5. Battery backup supply.
6. Model 1582 variable asynchronous console controller to provide serial TTY interface.
7. High-speed paper tape reader/punch (HSPTR/P) controller.
8. Space for six additional programmed I/O controllers.
9. Cable interface driver (CID) permitting use of an external I/O chassis which may contain additional I/O controllers.

3.1.4 LARGE GA-16/220 SYSTEM WITH MEMORY PARITY PROTECT OPTION

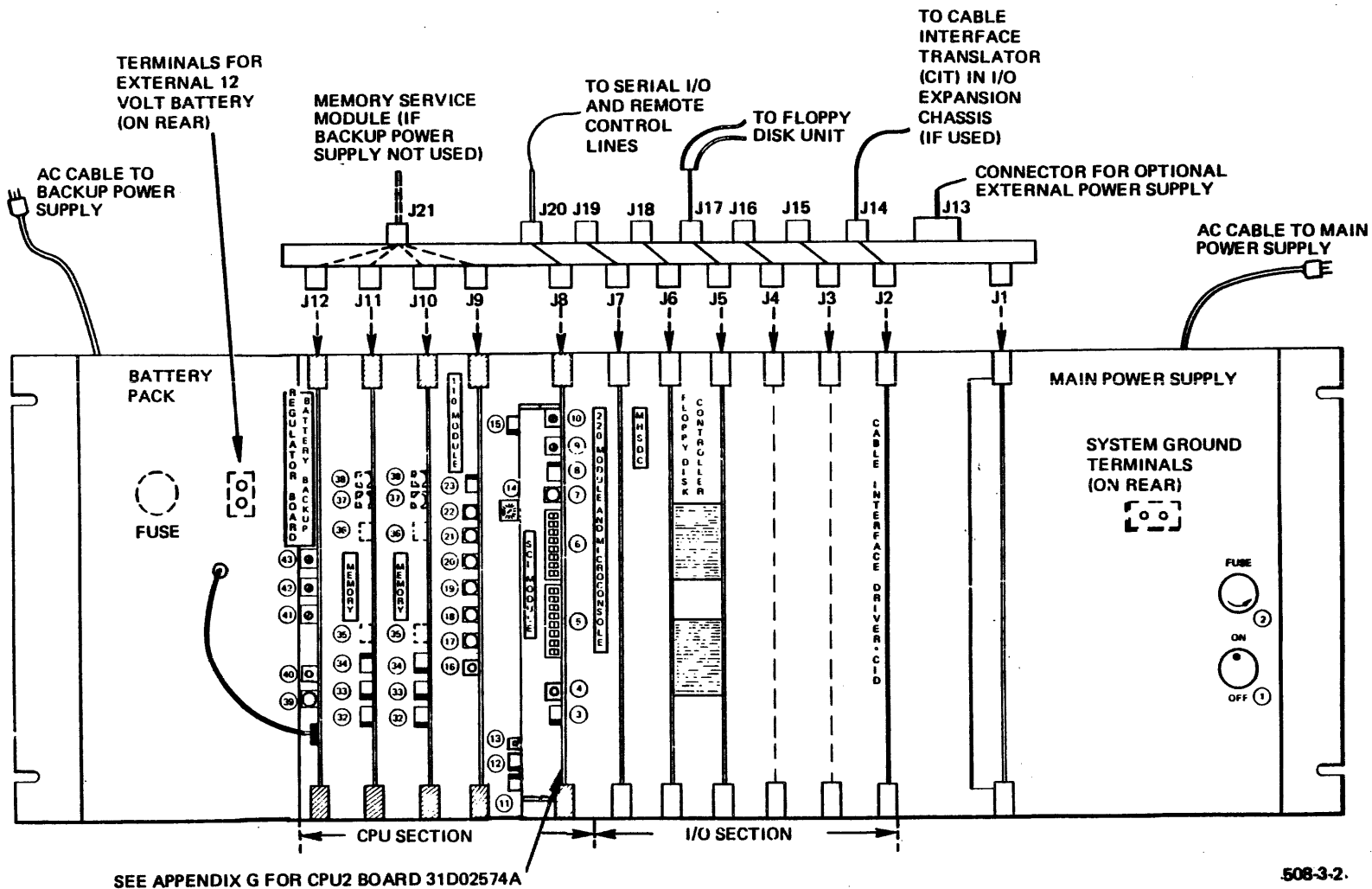
Figure 3-4 shows a GA-16/220 system with memory parity protect option. This system is installed in a jumbo chassis. Systems which use a jumbo chassis require an external power supply (not shown) since power requirements are greater than could be accommodated by a built-in supply. As shown, the system has the following features:

1. 220 CPU (CPU-1 and CPU-2 modules).
2. System console interface (SCI) module mounted on CPU-2 board.
3. Memory parity protect module.
4. 32K memory (using 8K memory modules with 18-bit words for parity detection MPP option).
5. Battery backup supply.
6. Multiple high-speed data channel (MHSDC) controller.
7. Floppy disk controller (DMA Channel via MHSDC).
8. High-speed paper tape reader/printer (HSPTR/P) controller (programmed I/O).
9. Space for four additional I/O controller boards.
10. Cable interface driver (CID) permitting use of an external I/O chassis which may contain additional I/O controllers.



88A00508A-E

Figure 3-1. GA-16/110 System in Compact Chassis (No MPP)

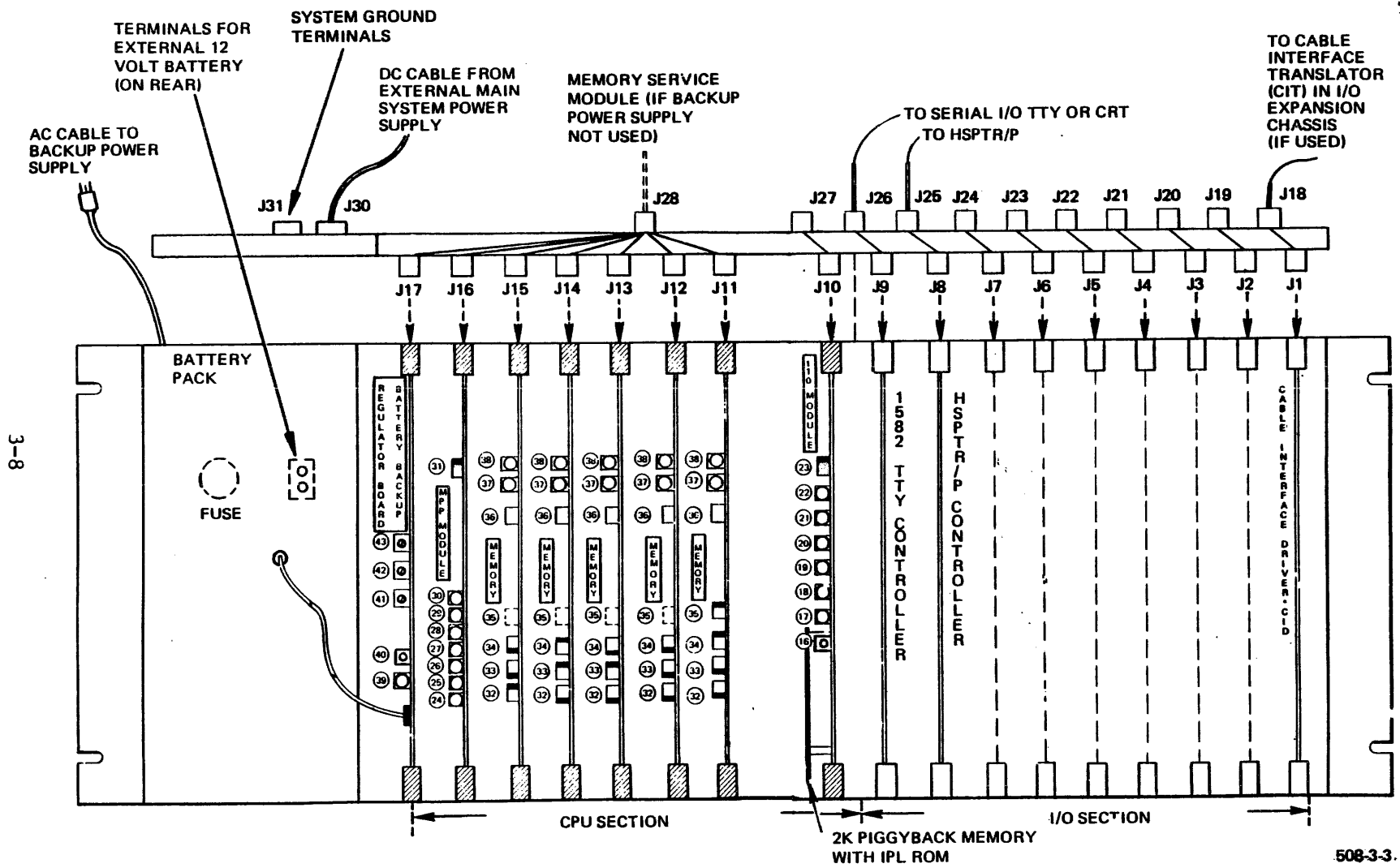


88A00508A-E

Figure 3-2. GA-16/220 System in Compact Chassis (No MPP)

Table 3-1. Connector Assignments for Compact Chassis Computer System

FRONT/MODULE (110)		FRONT/MODULE (220)	
J1	Main power supply	J1	Main power supply
J2	Cable interface driver (CID)	J2	Cable interface driver (CID)
J3	I/O controllers or shorting boards. Shorting boards are used in empty slots between controllers or CID to maintain priority chain continuity.	J3	I/O controllers or shorting boards. Shorting boards are used in empty slots between controllers or CID to maintain interrupt chain continuity.
J4		J4	
J5		J5	
J6		J6	
J7		J7	
J8	CPU-1 module with optional piggyback memory and IPL ROM.	J7	MHSDC controller, if used, should go into J7.
J9	Memory module (can't be serviced properly by MPP).	J8	CPU-2 module with optional System Console Interface.
J10	Memory module, or Memory Parity Protect module. J12 also used for battery backup regulator board.	J9	CPU-1 module with optional piggyback memory. Memory module, or Memory Parity Protect module. J12 also used for battery backup regulator board.
J11		J10	
J12		J11	
-		J12	
-	Backup power supply and battery pack (use regulator board in J12).	-	Backup power supply battery pack (use regulator board in J12).
REAR/INTERFACE (110)		REAR/INTERFACE (220)	
-	AC input connector and system grounding terminals on power supply.	-	AC input connector and system grounding terminals on power supply.
J13	Connector for optional external power supply cable.	J13	Connector for optional external power supply cable.
J14	Cables to I/O expansion chassis (6-feet maximum).	J14	Cable to I/O expansion chassis (6-feet maximum).
J15	Paddleboards and cables interfacing controllers (in J3 through J7) to peripheral devices.	J15	Paddleboards and cables interfacing controllers (in J3 through J7) to peripheral devices.
J16		J16	
J17		J17	
J18		J18	
J19		J19	
J20	Access to cold-start (CLDS-) memory mode (64KM+) and remote control lines (IPLSW-, PFD-, RSET-, RUN-).	J20	Serial I/O paddleboard with connector to TTY or CRT also provides access to cold-start (CLDS-) jumpers and remote control lines (IPLSW-, PFD-, RSET-, RUN-).
J21	Memory Service Module if backup power is not used.	J21	Memory Service Module if auxiliary power supply is not used.
-	AC cord for backup power supply.	-	AC cord for backup power supply.
-	Terminals for external 12-volt battery on rear of battery pack.	-	Terminals for external 12-volt battery on rear of battery pack.



88A00508A-E

Figure 3-3. GA-16/110 System in Jumbo Chassis (With MPP)

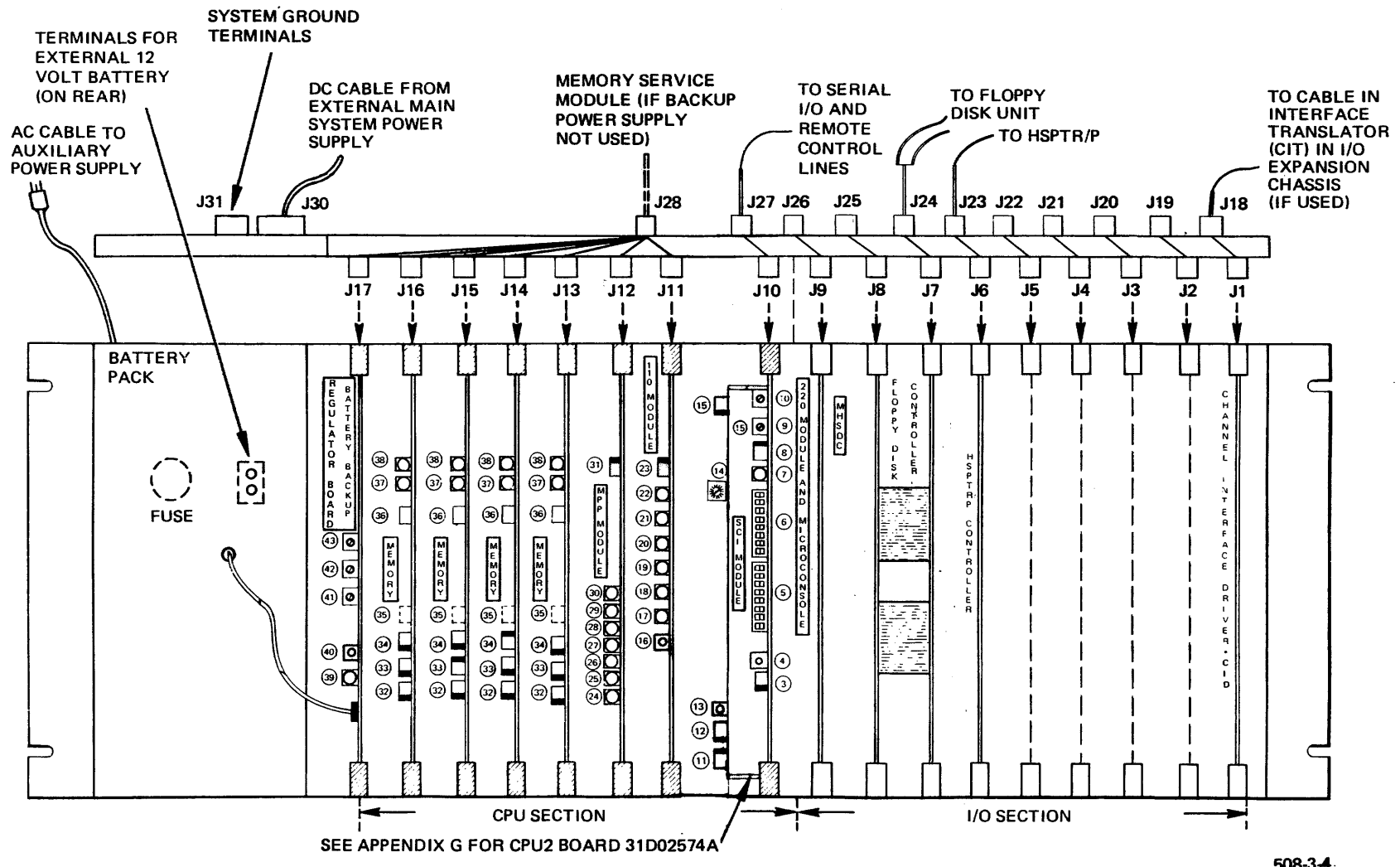


Figure 3-4. GA-16/220 System in Jumbo Chassis (With MPP)

Table 3-2. Connector Assignments for Jumbo Chassis Computer System

FRONT/MODULE (110)		FRONT/MODULE (220)	
J1	Cable interface driver (CID)	J1	Cable interface driver (CID).
J2	I/O controllers or shorting boards.	J2	I/O controllers or shorting boards.
J3		J3	
J4	Shorting boards are used in empty slots between controllers or CID to maintain interrupt chain continuity.	J4	Shorting boards are used in empty slots between controllers or CID to maintain interrupt chain continuity.
J5		J5	
J6		J6	
J7		J7	
J8		J8	
J9		J9	MHSDC controller, if used, should go into J9.
J10	CPU-1 module with optional piggy-back memory and IPL ROM.	J10	CPU-2 module with optional System Console Interface.
J11	Memory module (can't be serviced properly by MPP).	J11	CPU-1 module with optional piggy-back memory.
J12	Memory module, or memory parity protect module. J18 also used for battery backup regulator board.	J12	Memory module, or memory parity protect module. J18 also used for battery backup regulator board.
J13		J13	
J14		J14	
J15		J15	
J16		J16	
J17		J17	
-	Backup power supply battery pack (use regulator board in J18).	-	Backup power supply battery pack (use regulator board in J18).
REAR/INTERFACE (110)		REAR/INTERFACE (220)	
J18	Cable to I/O expansion chassis, (6-feet maximum).	J18	Cable to I/O expansion chassis (6-feet maximum).
J19	Paddleboards and cables interfacing I/O controllers (in J2 thru J10) to peripheral devices.	J19	Paddleboards and cables interfacing I/O controllers (in J1 thru J10) to peripheral devices.
J20		J20	
J21		J21	
J22		J22	
J23		J23	
J24		J24	
J25		J25	
J26	Access to cold-start (CLDS-) memory mode (64KM+), and remote control lines (IPLSW-, PFD-, RSET-, RUN-).	J26	Serial I/O paddleboard with connector for TTY or CRT. Also provides access to cold-start (CLDS-) jumper, and remote control lines (IPLSW-, PFD-, RSET-, RUN-).
J27		J27	
J28	Memory Service Module if backup power supply is not used.	J28	Memory Service Module if auxiliary power supply is not used.
J30	Power cable from external power supply.	J30	Power cable from external power supply.
J31	System grounding terminals.	J31	System grounding terminals.
-	AC cord for backup power supply.	-	AC cord for backup power supply.
-	Terminals for external 12-volt battery on rear of battery pack.	-	Terminals for external 12-volt battery on rear of battery pack.

3.2 MODULE DESCRIPTION

This section describes the modules which may be incorporated into a GA-16/110 or a GA-16/220 system. For detailed description of modules, refer to the GA-16/110/220 hardware manual (88A00509A).

3.2.1 POWER SUPPLY

Two types of power supply may be used with a GA-16/110/220 system.

1. A plug-in supply for use with self-contained systems installed in the compact chassis (Figures 3-1 and 3-2). This supply requires 115VAC, 47-63Hz input at 230 watts maximum. Its outputs are +5V @ 17A, +15V @ 3A, and -15V @ 2A.

CAUTION

Units shipped prior to 15 December 1976 have lesser capability of +5V at 8A (with convective cooling) +5V @ 15A (with external cooling fan), +15V @ 2.5A and -15V @ 1.5A.

2. An external supply, which is used for larger GA-16/110/220 systems, installed in a jumbo chassis (Figure 3-3 and 3-4). An external supply may also be used with a compact chassis if desired. The external supply input is 115V or 230VAC, 47-63Hz at 500 watts maximum. Outputs are +5V at 30A, +15V at 5A, and -15V at 5A.

The only control on the built-in supply, is the power switch (①) on Figure 3-1. The fuse (②) on Figure 3-1 is replaceable; however, a blown fuse indicates a system failure requiring maintenance. The external supply does not have a power switch. Power is applied when the supply is connected to an AC outlet. The AC outlet may be controlled in accordance with the user's requirements.

3.2.2 CPU-2 MODULE

The CPU-2 module is always used in conjunction with a CPU-1 module (described in Section 3.2.3) to form a GA-16/220 computer. The CPU-2 module contains additional microconsole switches and indicators. It also contains the logic needed to provide built-in serial I/O controllers, real-time clock (RTC), and console interrupt. The controls and indicators are described in Section 3.3, Table 3-3.

3.2.3 SYSTEM CONSOLE INTERFACE (SCI) MODULE (GA-16/220)

The system console interface (SCI) module is an optional module which is plugged into the CPU-2 module. It contains a 256-word random-access-memory (RAM) and a 512-word read-only-memory (ROM) necessary to provide an interactive utility program. A number of its subroutines may also be called from a user's program (described in Section 3.7). This program, called the console ROM, uses the serial I/O capabilities of the CPU-2 module. Using the SCI program, an operator at a TTY or CRT may perform such functions as shown in the following list.

1. Displaying and changing registers.
2. Displaying and changing memory locations.

3. Resetting I/O.
4. Stepping through a program in memory.
5. Starting a program in memory.

A detailed description of the SCI operator commands and system responses is contained in Section 3.6. The controls and indicators on the SCI module are described in Section 3.3, Table 3-3. The SCI module may also contain an Initial Program Load (IPL) ROM which is described in the following section.

3.2.4 MULTI-DEVICE INITIAL PROGRAM LOAD READ-ONLY-MEMORY (IPL ROM) FOR GA-16/220

The IPL ROM is an option which is mounted on the SCI module prior to delivery of a GA-16/220 system. The IPL ROM provides a set of bootstrap loaders (256-words) which permits the loading of a user's program from the teletype, or from a variety of other peripheral devices. A selector switch on the SCI module (14) on Figure 3-2 or 3-4, selects the device. The IPL button (13) causes the program to execute, thereby loading from the selected device. Operating procedures are described in Sections 3.4 and 3.5.2.

3.2.5 CPU-1 MODULE

The CPU-1 module contains the basic microprogrammed CPU logic. This module is used in both the GA-16/110 computer and the GA-16/220 computer (a GA-16/110 may be converted to a GA-16/220 by adding CPU-2 module). This module contains the microprocessor which contains the firmware for decoding and executing all machine instructions. These instructions and their CAP-16 assembly language equivalents are described in Section 4. The module also has provisions for installing a 2K static RAM and 64-word IPL ROM; usage of this configuration is described in Section 3.4.2 (dedicated GA-16/110/220). The microconsole control on the CPU-1 is the RESET button (16) in Figure 3-1 through 3-4. When used with CPU-2 module in a GA-16/220 system, the RESET button function is determined by the setting of the CPU Reset enable switch (15) in Figure 3-2 or 3-4, on the SCI and the cold start line. Controls and indicators for the CPU-1 module are described in Section 3.3, Table 3-3.

3.2.6 MEMORY PARITY PROTECT (MPP) MODULE

The MPP module is an option which may be installed in a GA-16/110/220 system as shown in Figure 3-3 and 3-4. The MPP provides the system programmer with the capability of protecting areas of memory from access by the CPU (program protect) or by direct memory access from a DMA controller (DMA protect). Detailed description of MPP usage is contained in Section 5. The MPP module contains the indicators necessary for identifying write protect violations, parity errors, and multiple errors. Parity error detection requires the installation of 18-bit memory modules (described in Section 3.2.7). The MPP module also contains the logic necessary for initiating either a CPU stall condition or a program interrupt via vector X'42' when parity errors occur (protect errors always cause an interrupt). The STALL switch (31) on Figure 3-3 and 3-4) is preset before the module is installed to select which action will occur. Controls and indicators for the MPP are described in Section 3.3, Table 3-3.

3.2.7 MEMORY MODULES

This section describes the memory modules which may be used in a GA-16/110/220 system.

3.2.7.1 General Description of Memory Modules

The GA-16/110/220 computer may be equipped with several different types of memory modules. The standard type of memory modules for general-purpose read/write applications are dynamic random-access solid-state memories (RAM) of 4, 8, 16, 32, or 64K word capacities. The programmable word length is 16 bits; however, for detection of parity error, memories using 18-bit words may be installed in either size (16-bit and 18-bit memories can be intermixed). The two extra bits are set for odd parity for upper and lower bytes. Parity bits are set during write operations and tested during read operations. Usage of 18-bit modules normally assumes the installation of an MPP module (Section 3.2.6). 18-bit memories may be used without an MPP module for parity generation and error detection. A parity override switch is provided to disable parity error detections, when MPP is not installed.

NOTE

If parity override is not enabled, a parity error will result in a stall condition on data fetches or a repeated attempt to execute instruction with instruction fetches. Indicators lights DERR and IERR (37) and (38) on Figure 3-1 through 3-4) are provided to indicate these conditions and are operative if MPP is not installed.

Because RAM memories are solid-state, they require continuous power (approximately 12 watts per 8K module) in order to hold their contents. To ensure that power is maintained, backup power supply may be installed (Section 3.2.8). The wiring of both the compact and jumbo chassis is designed to power memories by supplying ± 5 and $+12$ volts via converter/regulators from a 12-volt battery contained in a battery pack. A switch ((23)) selects whether power is from main power supply or backup power supply. If plug-in dynamic RAM is used without battery backup, a memory service module must be installed in the interface connector at the rear of the MIB (refer to Figure 3-1 thru 3-4). The memory service module powers the dynamic memory modules by converting the $+5$ and ± 15 volt power from the main power supply to $+5$, -5 , and $+12$ volts required by the memory modules. Static RAM's on piggyback memory boards do not require the memory service module.

Other types of memories which may be installed in a GA-16/110/220 system may include preprogrammed read-only-memories or a combination of ROM and RAM configured to a users requirements. A 2K word piggyback RAM memory and 64-word IPL ROM may be installed on the CPU-1 module; however, the 64-word IPL ROM is not used if the SCI is installed on the GA-16/220 system. Use of piggyback memory and IPL ROM without an SCI module in a GA-16/220 system is recommended only for dedicated user applications.

3.2.7.2 Setting Memory Addresses

Each 8K memory module has three switches (32), (33), and (34) in Figures 3-1 through 3-4), which are used to set memory boundaries in 8K increments. A 4K memory module contains an additional switch (35) which allows the setting of memory boundaries in 4K increments. Figure 3-5 shows a core map and illustrates the setting of switches on memory modules for both 8K and 4K boundaries.

NOTE

The conventions shown in Figure 3-5 are typical of core map representations throughout this manual. Low addresses are shown at the bottom and high addresses at the top. Unless otherwise specified, locations on core map are in hexadecimal numbering system. Hexadecimal locations in text will always be represented as X'nnnn' which corresponds to the CAP-16 assembler convention for defining a hexadecimal number.

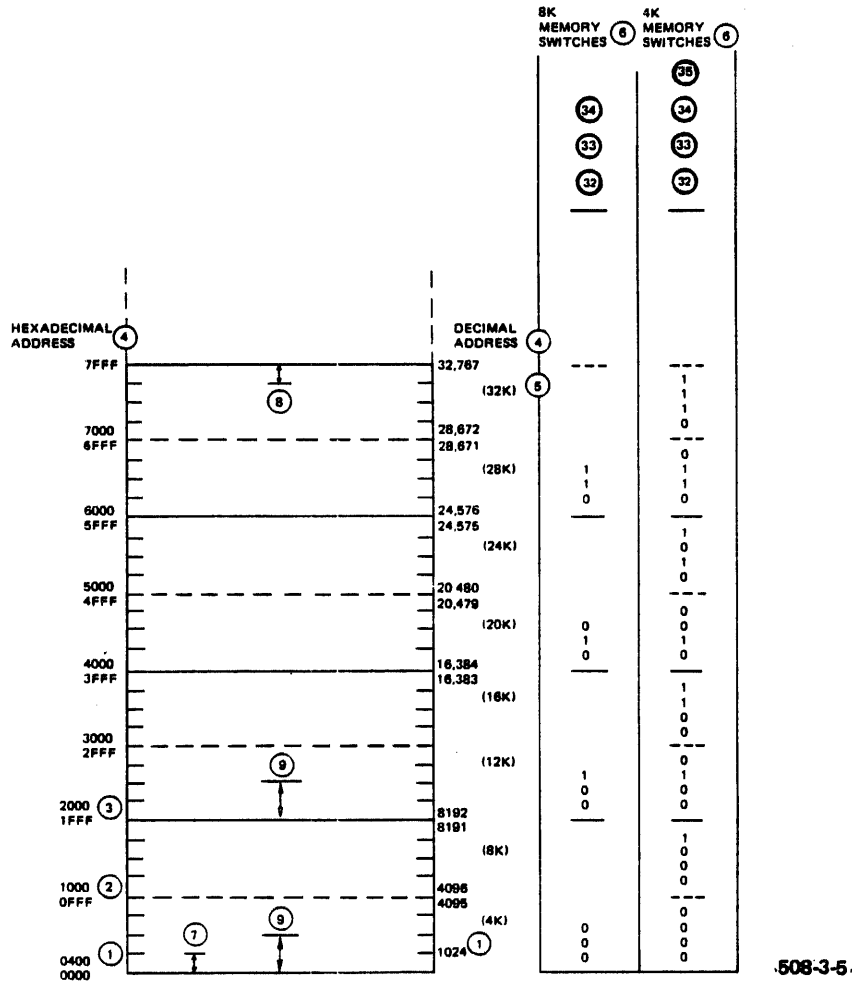
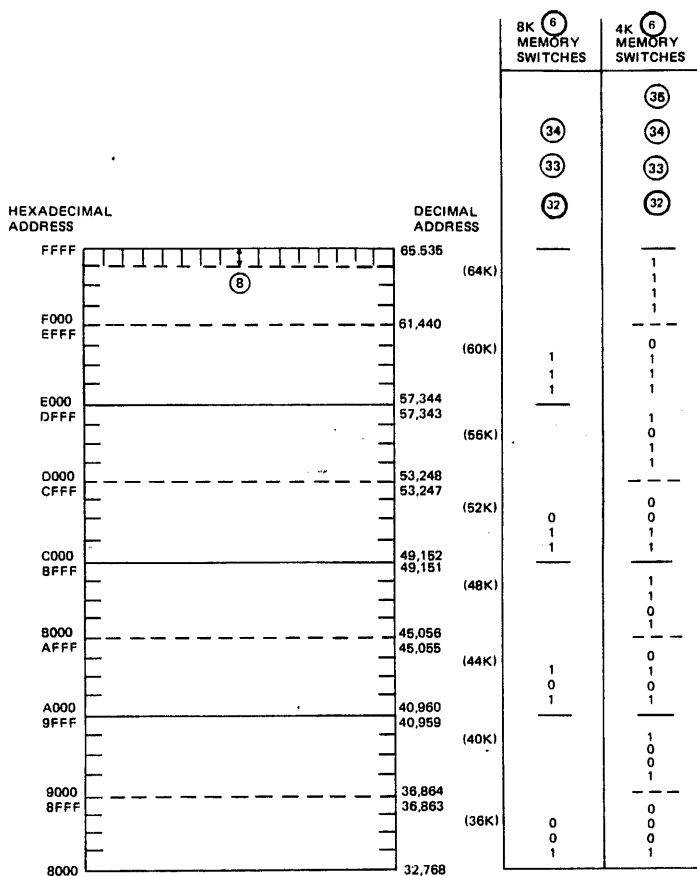


Figure 3-5. Memory Address Map (Sheet 1 of 2)



508-3-5.

NOTES:

- 1 Each small division represents 400 hexadecimal (X'400') or 1024 decimal (in common terminology, 1K) words of memory.
- 2 4K boundaries are represented by dashed lines.
- 3 8K boundaries are represented by solid lines.
- 4 Hexadecimal and decimal addresses shown specify both lower and upper addresses of each 4K block of memory. The decimal number in parenthesis represents the commonly used terminology for defining memory capacity (i.e., a 16 K system actually has 16,384 words, or an addressing range from 0 through 16,383 words, decimal, or X'0' through X'3FFF' when expressed in hexadecimal).
- 6 Memory switch settings are shown in the order they appear on the memory module, top to bottom. (As shown in Figures 3-1 through 3-4.) The up positions of the switch is 1.
- 7 Memory addresses X'00' through X'C1' are dedicated to the interrupt vector locations or are reserved for Series 16 operating systems which may be used, Figure 3-6.
- 8 On a GA-16/220, the upper 1K of each memory mode, locations X'7C00' through X'7FFF' when in 32K mode or locations X'FC00' through X'FFFF' when in 64K mode, is reserved for the SCI Console ROM and IPL ROM (Figure 3-7). These locations are independent of the amount of memory implemented by memory modules; however, if a memory module is set to include the reserved locations, those locations are not usable by that memory module. For example, the upper 1K of the fourth memory module in Figure 3-4 is not usable when operating in the 32K mode, since these locations are occupied by the SCI.
- 9 A 2K static RAM and a 64-word IPL ROM may be installed on the CPU-1 board. This memory (also referred to as the "piggyback" memory) physically occupies the first 2K of memory +64 words or 2K of memory beginning at the 8K boundary (set by jumpers). A RAM memory module may not be set to a boundary which includes the locations of the piggyback memory.

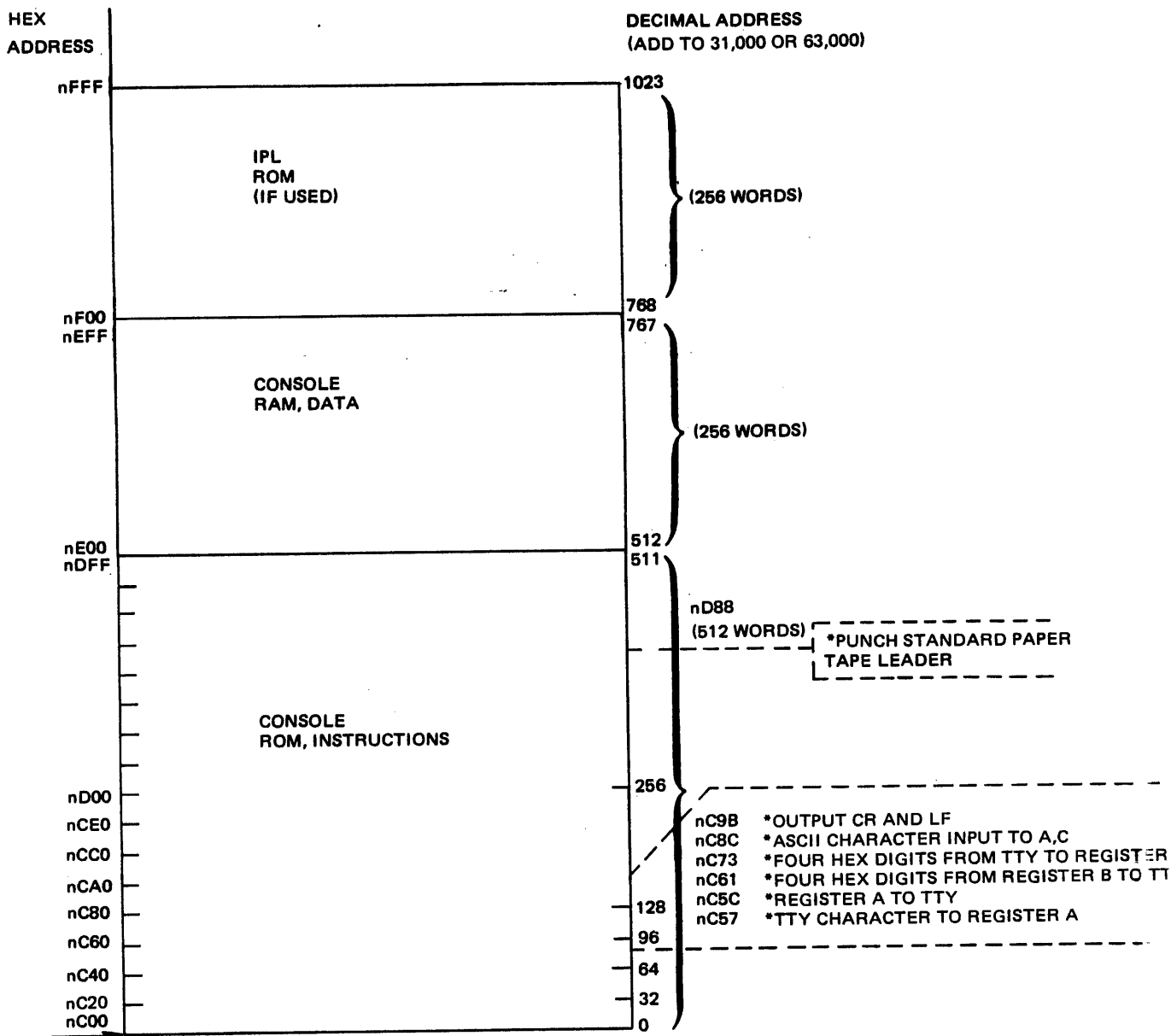
Figure 3-5. Memory Address Map (Sheet 2 of 2)

Figure 3-6 shows dedicated memory locations reserved for interrupt functions, and for GA-16 series operating systems when implemented. Figure 3-7 shows dedicated locations occupied by the SCI Console ROM and RAM and IPL ROM. The locations of closed subroutines which may be called from a user's program are also shown. These are described in more detail in Section 3.7. For a description of interrupts, refer to Section 2.6.

HEXADECIMAL LOCATIONS	DESCRIPTION
0080	MONITOR ENTRY POINT FOR GA-16 SERIES OPERATING SYSTEMS
007F	ISE
007E	P+I
007D	ISE
007C	P
007B	ISE
007A	P+I
0079	ISE STATUS
0078	P+I
0077	
0076	
...	
0048	} INTERRUPT VECTORS FROM I/O CONTROLLERS.
0047	CONSOLE INTERRUPT VECTOR
0046	SINGLE STEP, BREAK VECTOR
0045	TELETYPE NOT BUSY VECTOR
0044	TRAP INSTRUCTION VECTOR
0043	REAL TIME CLOCK (RTC) VECTOR
0042	MPP ERROR VECTOR
0041	RESTART VECTOR
0040	POWER FAIL VECTOR
003F	CAR
003E	SCR
...	
0021	CHANNEL ADDRESS REGISTER (CAR)
0020	SCAN CONTROL REGISTER (SCR)
001F	
...	
0018	} USED BY GA-16 SERIES OPERATING SYSTEMS
0017	
...	
0010	} USER PROGRAMS MAY USE
000F	
...	
0008	
0007	
...	
0001	} USED BY GA-16 SERIES OPERATING SYSTEMS
0000	

508-3-6.

Figure 3-6. Dedicated Memory Locations



NOTE: n=7 IN 32K MODE , F IN 64K MODE

*SCI CLOSED SUBROUTINES (ACCESSIBLE TO USER)

508-3-7

Figure 3-7. System Console Interface and IPL ROM Memory Locations

3.2.8 BACKUP POWER SUPPLY (BATTERY BACKUP)

Battery backup is an option which provides a battery pack, a line-operated charger, and a battery backup regulator module. The purpose of the battery backup is to supply +5 and +12 volt power to the memory modules (including +5V to 2K piggyback memory) at all times when power is applied, and to switch to batteries to preserve the contents of memory if the AC power fails. The length of time memory contents will be preserved is dependent on the size of memory. The standard battery backup unit is illustrated in Figure 3-1 through 3-4 is rated at 12 volts at 1.5A, and will maintain memory power for approximately 1.2 hours for 8K of memory and proportionally less for larger systems. A provision is available which permits using an external battery of greater capacity to extend the time power is maintained. The battery charger will trickle charge an external battery at a 0.5 ampere/hour rate. Most controls on the battery backup control module are for adjustment and will be preset. The only control of operator interest is the manual cutoff switch (40) on Figures 3-1 through 3-4). This switch permits an operator to deliberately disconnect the battery supply only when AC power is shut off to the backup power supply. The indicator light (39) indicates when power is applied to the memories. When battery backup is not installed, a module designated the memory service module (MSM) must be inserted into the connector at the rear of the MIB to provide -5V and +12V to plug-in dynamic RAM modules (not required with piggyback static RAM). The MSM makes alternative connections and conversions from the main power supply to supply necessary voltage to the memory modules.

3.3 CONTROLS AND INDICATORS

All possible controls and indicators which may be found on a typical GA-16/110/220 system are listed in Table 3-3. Controls and indicators which may be located on I/O controllers are not included. Controls and indicators are contained on several modules of a GA-16/110/220 system. The key numbers (circled on Table 3-3 and Figures 3-1 through 3-4) are identified by module as follows:

1 and 2 (PS)	Plug-in compact power supply
3 thru 10 (220)	CPU-2 module
11 thru 15 (220-SCI)	SCI module (option)
16 thru 23 (110)	CPU-1 module
24 thru 31 (MPP)	MPP module (option)
32 thru 34 (MEM)	8K memory module with 16-bit words
32 thru 34 } (MEM)	8K memory module with 18-bit words for parity error detection (option)
36 thru 38 }	
32 thru 35 (MEM)	4K memory module with 16-bit words
32 thru 38 (MEM)	4K memory module with 18-bit words for parity error detection (option)
39 thru 43 (BAT)	Backup power supply regulator module (option)

Table 3-4 provides switch settings for the IPL selector (IPL SLT) on the SCI module.

Table 3-3. GA-16/110/220 Controls and Indicators (Sheet 1 of 6)

Key	Label	Description and Function
① (PS)	ON OFF	The main power switch causes power to be applied to CPU and I/O controllers. Power is also applied to memory modules if memory service module is installed.
② (PS)	FUSE	The fuse holder allows replacement of main power supply fuse.
③ (220)	32K PGM	The 2-position switch selects memory addressing mode. When in the 32K position program addressing limit is 32K words. When in the PROGRAM position either 32K or 64K mode is selected by a program which sets the memory mode mask word (see Section 4.15.2.4).
④ (220)	CNSL INT	Console interrupt pushbutton, when pressed, causes a program interrupt through vector X'47" if (1) the console interrupt bit in the internal mask word is set (Section 4.15.2.1) and, (2) the interrupt system is enabled (refer to description of instructions INE and RISE in Section 4.13.2 and 4.10.9).
⑤ (220)	15 8	These miniature binary switches are called the console switches.
⑥ (220)	7 0	The console switches may be read by an RCSM or RCSR instruction in a program (Section 4.15.1). This capability permits writing programs which allow data entry and control via these switches. When switch is in position labeled OPEN, bit is set to 0.
⑦ (220)	DMA ACK	This indicator may blink periodically when direct memory access cycle stealing is occurring. It may illuminate continuously if very high DMA activity is in progress or if there is a malfunction. In a GA-16/220 system DMA activity is normally controlled by the multiple high-speed data channel (MHSDC) controller.
⑧ (220)	*TTY BAUD	The 2-position switch changes the baud rate of the built-in serial I/O controller to accommodate either a teletype (Model 33 automatic send receive, ASR, or equivalent) at 110 baud or a CRT terminal at 110 or 9600 baud.
⑨ (220)	*9600	This Trimpot permits fine adjustment of the 9600 baud transmission rate. It is adjusted only when the BAUD switch ⑧ is in the 9600 position.
⑩ (220)	*110	This Trimpot permits fine adjustment of the 110 baud transmission rate. It is adjusted only when the BAUD switch ⑧ is in the 110 position.

*These controls present an early model CPU2 boards; CPU2 board 31D02574A has a rotary baud rate selector (refer to Appendix G).

Table 3-3. GA-16/110/220 Controls and Indicators (Sheet 2 of 6)

Key	Label	Description and Function
⑪ (220-SCI)	BKDS	The 2-position switch enables or disables the TTY break capability. When switch is in the break-disable (BKDS) position, TTY breaks are ignored. When in the other position, TTY break is enabled so when a TTY operator presses the BREAK key on a teletype or CRT (provided unit is equipped with this key) either a processor (CPU) reset or an interrupt occurs depending on the setting of the BKINT switch ⑫.
⑫ (220-SCI)	BKINT	The 2-position switch sets the break mode for TTY or CRT, provided BKDS switch ⑪ is set so that break mode is enabled. When BKINT switch is set to break-interrupt position (BKINT) a TTY break causes a non-inhabitable interrupt via vector location X'46' (provided CPU is not operating under the SCI Console ROM program or the IPL ROM). Registers and status are saved in locations in the SCI program RAM. When switch is not in the BKINT position, a TTY break initiates a break reset which causes control to be transferred to the Console ROM and registers and status are not saved. I/O controllers are not reset.
⑬ (220-SCI)	IPL	Pushbutton initiates initial program load (IPL). When pressed, control is passed to the IPL ROM installed on the SCI module, and program loading occurs from a peripheral device. Selection of the peripheral device is accomplished by setting the IPL SEL selector switch ⑭ to the appropriate position for the IPL device installed with the system. This function may be remotely controlled via IPLSW line, or at auto-restart with cold start line low (at ground). CAUTION: IPL clears all memory locations.
⑭ (220-SCI)	IPL SEL	The IPL selector switch is a 16-position rotary switch. The switch is used to preselect which I/O device will be the source of an IPL when the IPL switch ⑬ is pressed. (In systems illustrated in Figure 3-2 and 3-4, the device would be floppy disk or teletype). Load-and-go operation is selectable for all devices, while load-and-stop operation may be selected for teletype, high-speed paper tape reader, or card reader. The switch positions and device selections are shown in Table 3-4.
⑮ (220-SCI)	CNSL	The 2-position switch determines both the function of the CPU reset button ⑯ and an auto-restart operation when power is applied. When in the console position (CNSL) the CPU and I/O are reset and control is transferred to the Console ROM. If in the other position, the status of the cold start line, CLDS, determines the operations as described for RESET pushbutton on the CPU-1 module ⑯.

Table 3-3. GA-16/110/220 Controls and Indicators (Sheet 3 of 6)

Key	Label	Description and Function
①⑥ (110)	RESET	<p>Pushbutton resets the CPU and the I/O system, and causes control to be transferred in accordance with the cold start line (CLDS-) and (on a GA-16/220) the CNSL switch ①⑤.</p> <p>On a GA-16/110, the cold start line determines the function of the RESET switch as follows:</p> <p style="padding-left: 40px;">cold start high: Control is transferred via auto-restart vector X'41'.</p> <p style="padding-left: 40px;">cold start low (grounded): Control is transferred to the IPL ROM.</p> <p>On a GA-16/220, the cold start line determines these functions only if the CNSL switch ①⑤ is not in the CNSL position. In the CNSL position, control is transferred to the console ROM on the SCI. This function may be remotely controlled via the RESET and SYRT lines, together.</p>
①⑦ (110)	IACK	<p>Interrupt acknowledge indicator blinks when control has passed to a routine via an interrupt vector. It normally blinks so rapidly as to be barely visible. If continuously illuminated, it indicates that the control has not returned from the interrupt processing routine, a "hung-up" condition.</p>
①⑧ (110)	ISE	<p>Interrupt system enabled indicator is illuminated when the ISE flip-flop is set, and means that the inhibitable interrupt system is enabled. Refer to description of instructions INE and RISE (Section 4).</p>
①⑨ (110)	OMA	<p>This indicator (also referred to as the OMA stall indicator) is illuminated when the operators monitor alarm (OMA) has timed out. The OMA must have been initially turned on by a PMA instruction (Section 4.13.7). When the OMA indicator illuminates, the RUN indicator ②① is extinguished and the CPU is in an idle state. To recover from an OMA, the operator must press the RESET button ①⑥.</p>
②⑦ (110)	FGND	<p>Foreground indicator is illuminated when the foreground registers are used and extinguished when background registers are used. Refer to instructions BMS and FMS (Sections 4.13.1 and 4.13.2) for setting foreground or background register usage.</p>
②① (110)	RUN	<p>The RUN indicator is illuminated when the CPU is in the run mode. If the WAIT indicator ②② is also illuminated, the CPU is in a wait condition as a result of executing a WAIT instruction. If MPP indicators ②⑦ through ③① are illuminated, memory parity error has caused the stall. In order for an MPP stall to occur, the STALL switch ③① must be in the stall position. To recover, the user must RESET the system. If the run indicator is extinguished, the system is in idle.</p>

Table 3-3. GA-16/110/220 Controls and Indicators (Sheet 4 of 6)

Key	Label	Description and Function
②① (110)	RUN (Cont)	<p><i>NOTE: If 18-bit memories are installed without installation of MPP module, a stall condition may occur if the parity override switch ③⑥ is not set. Refer to Section 3.2.7.1 and description of ③⑥, ③⑦, and ③⑧ in this table.</i></p> <p>A remote idle indicator may be implemented via the RUN line, and conversely grounding the RUN line will force the CPU into the idle condition.</p>
②② (110)	WAIT	Wait indicator illuminates when a WAIT instruction has been executed (Section 4.13.10).
②③ (110)	-	Unlabeled 2-position battery backup switch faces to the rear of CPU-1 module, and must be set prior to installation. In the backup position (down), +5VB power for the memories (including the piggyback RAM) originates in the backup power supply or from batteries (when AC power is disconnected). When switch is not in backup position, the power is obtained from the main power supply. A memory service module must also be installed in the rear connector (as shown in Figures 3-1 through 3-4) to obtain power from the main power supply for the 8K/4K RAM memories when backup power supply is not used.
②④ (MPP)	DPT	The DMA write protect indicator, when illuminated, indicates that an attempt has been made by DMA to write into a memory area (via the high-speed data channel) which has been DMA-protected (refer to Section 5).
②⑤ (MPP)	ME	Multiple error indicator illuminates when a second error occurs before software can process a previous error. Other indicators also may be illuminated.
②⑥ (MPP)	PPT	Program write protect indicator, illuminates when a program has attempted to write data in a memory area which has been program-protected (refer to Section 5).
②⑦ (MPP)	LPB	Lower parity bit indicator shows the contents of parity bit for the lower byte of a memory word (illuminated = 1. Content of bit must be compared with the lower byte to determine if error has occurred; even parity is error.
②⑧ (MPP)	UPB	Upper parity bit indicator shows contents of the parity bit for the upper byte of a memory word (illuminated = 1. Content of bit must be compared with the upper byte to determine if error has occurred; even parity is error.

Table 3-3. GA-16/110/220 Controls and Indicators (Sheet 5 of 6)

Key	Label	Description and Function
②⑨ (MPP)	DPY	DMA parity indicator illuminates when a parity error is detected during a DMA transfer.
③⑩ (MPP)	PPY	Program parity indicator illuminates when a parity error is detected when a program reads a memory location (data or instruction fetch). <i>NOTE: Indicators ②④ through ③⑩ remain illuminated until MPP status is reset under software control. Refer to Section 5.</i>
③① (MPP)	STALL	The 2-position switch faces to the rear of the MPP module and is preset prior to installation. This switch determines CPU action when a DMA or CPU parity error occurs. When switch is in STALL position, the memory bus is forced to a busy state, thereby halting the CPU. When not in the STALL position, control is passed to a routine via non-inhibitible interrupt vector X'42'. (All non-parity errors pass control via X'42' regardless of the setting of STALL switch, and routine must determine type of error.)
③② ③③ ③④ (MEM)	15 14 13	Three 2-position switches used on both 8K and 4K memory modules to set the memory to occupy locations starting at one of eight 8K boundaries. Refer to Section 3,2,7, Figure 3-5, for memory map and corresponding switch settings.
③⑤ (MEM)	12	The 2-position switch installed only on 4K memories. It selects the upper or lower 4K boundary within the 8K boundary set by switches ③②, ③③, and ③④. Refer to Section 3.2.7, for memory map and corresponding switch settings.
③⑥ (MEM)	PAR OVRD	The 2-position switch is installed only on 18-bit memories. This switch is effective only if MPP module is not installed and enables parity override which disables parity error detection when in DOWN position. When in UP position, parity error detection occurs. Action upon a parity error depends on whether or not an MPP module is installed: <ul style="list-style-type: none"> - When MPP module is not installed and switch is in the UP position, a parity error will cause a CPU stall condition on data fetches or repeated attempts to execute instruction fetches, and the DERR and IERR indicators will identify the error. - When an MPP module is installed, switch may be in either position and parity errors will be identified by the LPB, UPB, DPY, and PPY indicators ②⑦ through ③⑩ on the MPP module. Detection is enabled by PIO to the MPP module (Section 5). Action taken upon parity error detection then is determined by the setting of the stall switch ③① on the MPP module.

Table 3-3. GA-16/110/220 Controls and Indicators (Sheet 6 of 6)

Key	Label	Description and Function
③⑦ (MEM)	DERR	The data error indicator illuminates (in 18-bit memories) when a parity error occurs on a data fetch. The data comes out as '0000'. This condition is cleared by a system reset or by a break reset (see description of ①⑥ and ①②).
③⑧	IERR	This instruction error indicator (in 18-bit memories) illuminates when a parity error occurs on an instruction fetch. The instruction comes out as '0000' (WAIT). This condition is cleared by a system reset or by a break reset. <i>NOTE: If the Memory Parity Protect (MPP) option is included in a system, neither the IERR nor DERR indicators will illuminate unless there is a high failure rate. The MPP is managing the memory under program control.</i>
③⑨ (BAT)	-	Unlabeled indicator illuminates when power (either from AC line or from batteries) is applied to the memory modules. Light is extinguished when manual cut-off button ④⑩ is pressed (or if batteries are exhausted) provided AC power is disconnected from the auxiliary power supply.
④⑩ (BAT)	-	Unlabeled pushbutton provides manual cut-off of battery power to memory modules. Pressing this button will have no effect unless AC power is disconnected from auxiliary power supply. If AC power is disconnected, pressing this button will cut off power to memories. Power will not be restored to memories until AC power is reconnected. <i>NOTE: Application of power to auxiliary power supply is independent of built-in main power supply switch ①, or other means of controlling power to external main power supply.</i>
④① (BAT)	BAT CHG ADJ	This control is used to adjust the charge voltage to the battery pack to maintain a 0.5 ampere rate (maximum). Setting for 12-volt, 1.5 ampere battery supplied is 13.6 volts.
④② (BAT)	5VB	This control is used to adjust the +5-volt regulator.
④③ (BAT)	VDD	This control is used to adjust the +12-volt regulator.

Table 3-4. IPL Selector Switch on SCI Module (GA-16/220)

Position	Device
0	Teletype (Load and Go)
1	High-Speed Paper Tape Reader (Load and Go)
2	Card Reader (Load and Go)
3	3347 Disk (Load and Go)
4	3346 Removable Disk (Load and Go)
5	3349 Floppy Disk (Load and Go)
6	3343 Disk (Load and Go)
7	3346 Fixed Disk (Load and Go)
8	3342 Disk (Load and Go)
9	3341 Disk (Load and Go)
A	Teletype (Load and Stop)
B	High-Speed Paper Tape Reader (Load and Stop)
C	Card Reader (Load and Stop)
D	Option
E	Option
F	Option

NOTE: Refer to description of IPL SEL switch (14) in Table 3-3.

3.4 START-UP AND PROGRAM LOAD

This section provides start-up and program load procedures for basic configurations of the GA-16/110 and the GA-16/220 systems. These configurations are:

1. A general-purpose GA-16/220 system configured to use with one of the GA-16 series operating systems; this configuration will include the system console interface module equipped with IPL ROM for loading a program from various devices. It may include a 2K piggyback RAM (on the CPU-1 module) however, the IPL ROM on the piggyback will not be used.
2. A dedicated GA-16/110 or 220 system configured to use a special-purpose program; this configuration will use the 2K piggyback RAM (on the CPU-1 module) equipped with a single-device IPL, ROM and, in the case of a GA-16/220, does not include the SCI on the CPU-2 module. Circled numbers in procedures refer to switch designations in Figures 3-1 through 3-4, Table 3-3.

Figure 3-8 provides a general flow diagram for GA-16/110 and GA-16/220 System Start-Up.

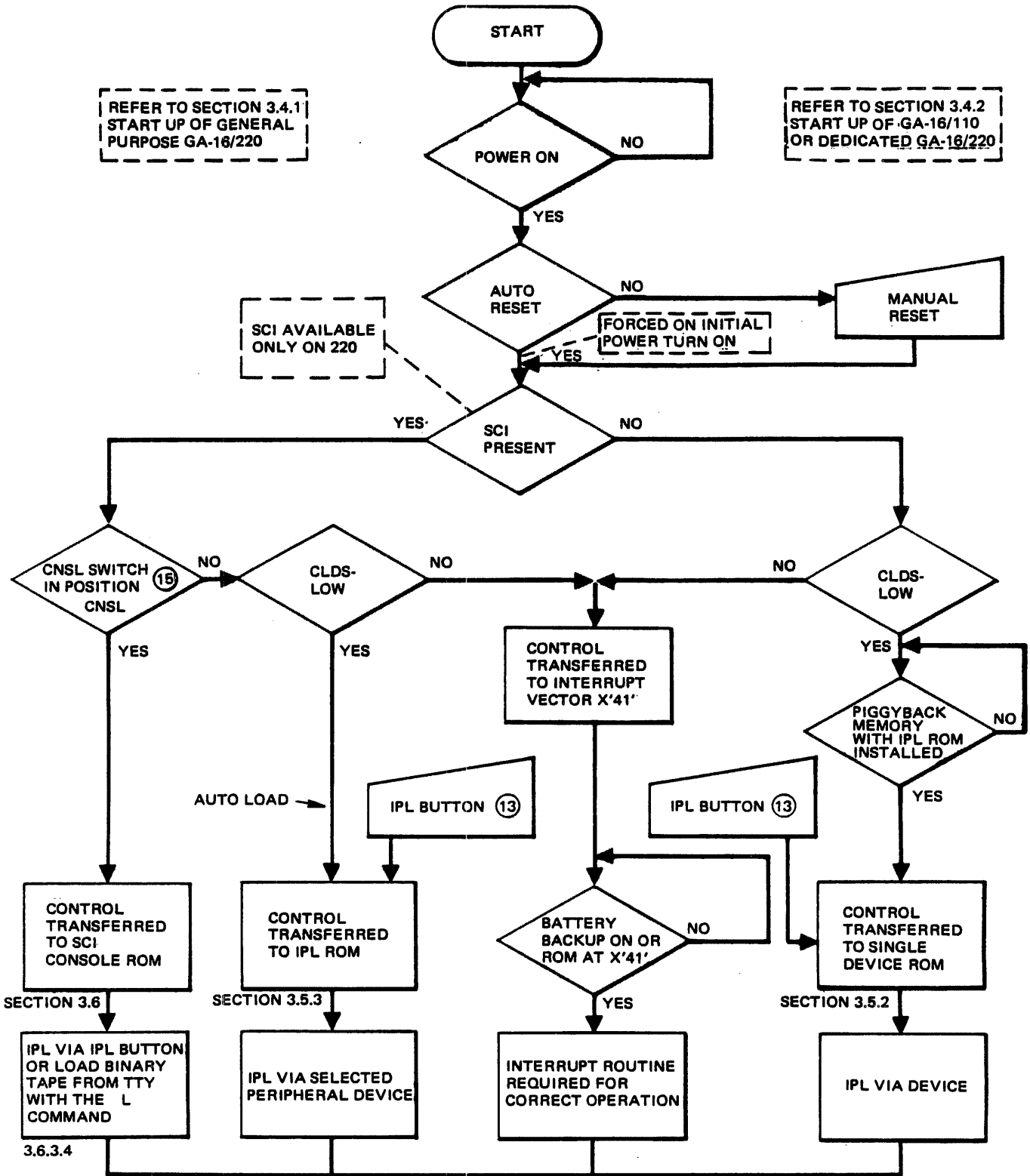


Figure 3-8. General System Start-Up Flow Diagram

508-3-8

3.4.1 START-UP OF GA-16/220 GENERAL-PURPOSE SYSTEM

3.4.1.1 Preconditions for System Start-Up

Power turn on, an automatic power restart, or pressing the RESET button (16) will result in one of three options. This is dependent upon the setting of the CNSL selector switch (15) and the status of the cold-start via hard-wired interface, CLDS.

1. CNSL switch in CNSL position: Control is transferred to the SCI Console ROM for operator interaction via console teletype.
2. CNSL switch not in CNSL position and CLDS low: Control is transferred to the IPL ROM and loading of a user's program or GA-16 operating system will begin from the peripheral device selected via the IPL SLT switch (14) in Table 3-3 and 3-4.
3. CNSL switch not in CNSL position and CLDS high: Control is transferred to interrupt vector X'41' which must contain the address of a routine which can effect some operation. This routine must be defined by a user program. It is also a requirement that battery backup be used to preserve the memory contents or that a ROM be implemented at location X'41'.

NOTE

Standard systems are shipped with CLDS hard-wired high. This line is generally used only for special systems applications.

3.4.1.2 Verify Memories Are Powered

When battery backup supply is installed, the operator should verify that AC-derived power is being supplied to memories before turning on the main power supply by the following procedure:

1. Verify that the battery backup switch (23) is in the correct position.
2. Verify that indicator (39) is illuminated.
3. Press manual cutoff switch (40) .

- Indicator should remain illuminated.

If indicator is extinguished, it signifies that memories were running on battery with no AC power to the backup power supply. AC supply mains should be checked. When power is supplied, indicator will come back on.

If battery backup is not installed, memory power will be applied when main power supply is turned on (Section 3.4.1.3).

3.4.1.3 Main Power Application Procedure

1. Ready the peripheral device.
 - a. If option 1 (Section 3.4.1.1) is selected, verify that console TTY is turned ON and in LINE position.
or
 - b. If option 2 is implemented, mount the program source media (paper tape, cards, disk, etc.) on the peripheral device, (refer to Section 3.5.3).
2. Apply power to main power supply.
 - a. Via main power connector for external supply
or
 - b. Via power switch (1) on built-in supply,
 - RUN indicator (21) will illuminate.
 - (Option 1) TTY will output a (CR) (LF) (proceed with Section 3.6).
 - (Option 2) Program will load from IPL device; (refer to Section 3.5.3).

If a load and stop IPL is used, control will return to the console ROM and operator interaction may occur in accordance with the procedures of Section 3.6.

 - (Option 3) user-defined.

3.4.2 START-UP OF GA-16/110 OR GA-16/220 DEDICATED SYSTEM WITHOUT SCI

3.4.2.1 Preconditions for System Start-Up

1. One of the following single-device IPL ROM's will be installed on the 1K or 2K piggyback memory. These are described in Section 3.5.
Version 1 - Teletype Version 2 - High-speed paper tape reader
2. An I/O controller must be installed in the I/O side of the MIB as follows:
 - a. Model 1582 for teletype (TTY).
 - b. Model 3321 for high-speed paper tape reader (PTR).

3. Power turn on, automatic restart, or pressing the RESET button (16) will result in one of two options, dependent upon the state of the cold start hardware interface, CLDS.
 - a. CLDS low: Control is transferred to the single-device IPL ROM.
 - b. CLDS high: Control is transferred to interrupt vector X'41' which must contain the address of a routine which can effect some operation. This routine must be defined by a user program. It is also a requirement that battery backup be used to preserve memory or that a ROM be implemented at location X'41' with a transfer vector to a user's dedicated program.

NOTE

Standard systems are shipped with CLDS hard-wired high. This line is generally used only for special systems applications.

3.4.2.2 Verify Memories Are Powered

When battery backup supply is installed, the operator should verify that AC-derived power is being supplied to memories before turning on the main power supply by the following procedure:

1. Verify that battery backup switch (23) is in correct position.
2. Verify that indicator (39) is illuminated.
3. Press manual cutoff switch (40) .
 - Indicator should remain illuminated.

If indicator is extinguished, it signifies that memories were running on battery with no AC power to backup power supply. AC supply mains should be checked. When power is supplied, indicator will come back on.

If battery backup is not installed, memory power will be applied when main power supply is turned on (Section 3.4.2.3).

3.4.2.3 Main Power Turn-On

NOTE

If IPL (Section 3.4.2.1) is implemented, mount the program source tape on the TTY or PTR and refer to Section 3.5.2 for actions which occur upon power turn on.

1. Apply power to main power supply.
 - a. Via power switch (1) on built-in power supply,
 - or
 - b. Via power connector if external supply is used.

In either case:

- RUN indicator illuminates.
- or
- RUN indicator illuminates and paper tape loads,
as described in Section 3.5.2.

3.5 IPL ROMS

As standard options, GA offers the following Initial Program Load (IPL) ROMs:

1. Single-device IPL — This IPL runs on a GA-16/110 or a dedicated GA-16/220 and mounts on a 2K piggyback memory. It is a complete 64-word mini PGS loader requiring no preamble on the media being loaded. It is an absolute load-and-go binary loader with check sum.

Version 1 - TTY

Version 2 - High-speed paper tape reader

2. Multi-device IPL — This IPL ROM is mounted on the GA-16/220 System Console Interface Module. It consists of a 256-word program which provides bootstrap loaders for the TTY, high-speed paper tape reader, card reader, floppy disk, moving arm disk (both fixed and removable platter) and head-per-track disk. Each of these bootstrap loaders accesses its designated peripheral to bring a PGS loader into the RAM. (The PGS loader must be in the zero sector of a disk or in the preamble of tapes and card decks.) The PGS loader then automatically loads the program and data which must be in the PGS format, i.e., relocatable with check sum and designated start address as described in Section 3.5.1.

The following sections describe PGS and mini-PGS formats, use of single-device ROM on a GA-16/110 or dedicated GA-16/220, and use of the multiple-device ROM on a general-purpose GA-16/220.

3.5.1 PGS AND MINI-PGS FORMAT

The PGS (Program Generation System) format is the standard for all GA program generation systems. All language translators (Macro Assembler, FORTRAN, BASIC, COBOL) output object code in PGS format; the CoreLoad Overlay Builder (CLOB) links PGS elements into a loadable PGS module; all utilities, subroutines, T&Vs, etc., are in PGS format. All GA device loaders require the PGS format. They perform check summing, permit specification of load address and automatically relocate program elements. PGS format is described in Appendix D.

The multi-device IPL on the System Console Interface Module (GA-16/220) contains a set of binary bootstrap loaders for various peripherals. They bring in PGS loaders in binary format from their respective devices. The loaders, in turn, proceed to load the programs, which must be in PGS format.

A mini-PGS format was developed especially to facilitate TTY or PTR IPLs in GA-16/110 or dedicated GA-16/220s with small piggyback RAMs. Because of limited memory in these systems, it is undesirable to use any portion of RAM to hold the PGS loader during IPL. Also, due to physical constrictions, it is not possible to commit the full PGS loader to ROM. Thus, the mini-PGS loader operates as a complete loader (not just bootstraps) fitting into 64 words of ROM on the 2K RAM boards.

The mini-PGS format differs from PGS mainly in its being absolute rather than relocatable. Mini-PGS still performs check summing for verified loading.

For systems with a mini-PGS IPL, the software package includes a mini-PGS punch routine. After developing his program in the full relocatable PGS format with any language (Macro Assembler, FORTRAN, BASIC, COBOL), the programmer uses the mini-PGS punch routine to produce a mini-PGS copy of the program he has generated. The resulting tape is loaded by the 64-word IPL ROMs on the piggyback RAMs.

The full PGS loaders handle both PGS and mini-PGS format. The mini-PGS loaders are restricted to the mini-PGS format.

3.5.2 SINGLE-DEVICE ROM ON GA-16/110 AND DEDICATED GA-16/220

In a small dedicated system using 2K piggyback RAM on the GA-16/110 module, one of the single-device IPLs described in Section 3.5 must be mounted on the piggyback memory board to load-and-go from a TTY or PTR.

The following procedures apply to TTY or PTR IPL GA-16/110 and on dedicated GA-16/220s without the System Console Interface Module:

1. The tape to be loaded must be in mini-PGS format. (See Section 3.5.1 for an explanation of PGS and mini-PGS formats.)
2. Place the paper tape in the TTY or PTR with the leader in the read position. Do not place the first character in the read position.
3. Auto-load is initiated upon power turn-on or by the operator by pressing the Reset button, provided the Cold Start Line is grounded. This does a complete system reset and forces the CPU to the IPL address for execution of the program stored in the ROM.

NOTE

The reason for grounding CLDS is that if Cold Start is not grounded, reset sends the CPU to X'41', whose content is probably invalid if power has been down.

Auto-load can also be initiated remotely by grounding:

- a. *Cold Start Line and (momentarily) Reset Line on the rear of the computer. This resets the CPU and auto-loads. It does not reset I/O.*

b. Cold Start Line and (momentarily) Power Fail Detect Line. (Grounding PFD is equivalent to the power-up or restart signal.) This auto-loads with a complete system reset.

4. With power on, the PTR automatically starts reading when the IPL is initiated. However, TTY start is different. When IPL is initiated, the ROM goes out to the TTY and attempts to start reading. The TTY reader must then be manually switched to "START". (This manual switching at the TTY need not be done immediately, since the IPL ROM continues indefinitely to apply "start" sequence to TTY until it detects the first character from the reader.)
5. Without interruption, either TTY or PTR read their tapes to the end and stop. The IPL ROMs then automatically transfer control to the start address of the program just loaded and begin execution. Both TTY and PTR IPLs are load-and-go.

3.5.3 GA-16/220 MULTI-DEVICE INITIAL PROGRAM LOAD ROM ON SCI

This section describes the characteristics of various devices when loading programs under the control of the IPL ROM installed on the SCI module. Loading of a program via the IPL ROM takes place under the following conditions:

1. Upon initial power turn-on or automatic power restart; provided option 2. (Section 3.4.1.1) has been implemented.
2. When an operator presses the RESET button (16) on the microconsole; provided option 2 (Section 3.4.1.1) has been implemented; (alternatively, an external RESET button may be connected via hardware interface, RSET line).
3. When an operator presses the IPL button (13); (alternatively an external IPL button may be connected via hardware interface, IPLSW).

3.5.3.1 Teletype (TTY)

Blank leader is ignored. Loading of binary tape begins at a predetermined location below the end of available memory. The IPL ROM looks for the highest X'nA00' address available; for a 32K memory this would be X'7A00'; for 8K memory this would be X'1A00'.

Loading continues until either feeding stops (tape runs out or reader is turned off), or 150 blank frames have been encountered. If 150 blank frames have gone by, it is assumed that either the module just loaded is itself a loader and what it is to load follows on the same piece of tape, or that the program is stand-alone. When in the load-and-go mode, execution will transfer to the starting load location; load-and-stop will print the starting address in hex and transfer to the Console ROM.

If feeding stops, control goes immediately to the Console ROM when in load-and-stop mode, or the IPL routine will wait for the reader to be started up again if in load-and-go mode. Refer to the GA-16/220/330 Stand-Alone Utilities document number 94A001531A for detailed operating procedures.

3.5.3.2 High-Speed Paper Tape Reader (PTR)

This device is functionally identical to the TTY. The only real difference is that there is no way for the operator to restart the PTR in load-and-go mode, so that the IPL routine will periodically try to restart it. Since this method is not particularly reliable, it is recommended that for the PTR, both loader and program be on the same tape (separated by at least 150 blank frames), or that load-and-stop be used.

3.5.3.3 Card Reader (CDR)

The CDR loader starts loading at the location 512 words below the end of available memory and continues loading until bit 9 in column 72 is encountered in any card. At this point, load-and-go mode will wait for CDR-ready and go to the loaded module, while load-and-stop will transfer control the Console ROM. Refer to GA-16/220/330 Stand-Alone Utilities document for detailed operating procedures.

3.5.3.4 Disk (DKx)

The devices are functionally identical. Sector 0, track 0, head 0, is loaded into the RAM above the Console ROM (7E00 in a 32K system, or FE00 in 64K). Control is immediately transferred to the sector just loaded which is assumed to be sector loader.

3341	10-Surface Disk	(DK1)
3342	Fixed-Head Disk	(DK2)
3343	20-Surface Disk	(DK3)
3346	Dual-Platter Disk, Removable Cartridge	(DK6)
3346	Dual-Platter Disk, Fixed Surface	(DK6F)
3347	Removable Cartridge Disk	(DK7)
3349	Floppy Disk	(DK9)

3.6 SCI CONSOLE ROM (GA-16/220)

The SCI Console ROM provides an interactive program which permits program diagnostic and development activities to be carried out via the teletype or CRT. The SCI Console ROM is entered by several methods:

1. Upon power-up, auto-restart, or when an operator presses RESET button (16), provided CNSL switch (15) is in the CNSL position (option 1, Section 3.4.1.1).
 - Carriage return (CR) and line feed (LF) occur on TTY.
 - The break interrupt vector X'46' is loaded so that subsequent entry may be made with TTY BREAK key.
2. When teletype operator presses BREAK key, provided the following hardware conditions are met:
 - a. BKDS switch (11) is not set in the BKDS position.
 - b. BKINT switch (12) is not set in the BKINT position.
 - Address of next instruction to be executed is printed.

3. When a program executes a TRAP instruction.
 - Address where trap occurred is printed.
4. When an IPL ROM loads from TTY, card reader, or paper tape reader in a load-and stop mode (refer to IPL SLT positions, Table 3-4) after loading is completed.
 - Starting address of loaded program is printed.
5. When a user's program uses one of the closed subroutines to read or write data on the teletype, or to punch a tape loader. This entry is not interactive and control will return to the user's program. Section 3.7 describes the SCI ROM closed subroutines.

On all entries except 5, the operator may use the Console ROM commands to restart a program, single step, examine memory locations, etc., as described in Section 3.6.1.

3.6.1 CONSOLE ROM COMMANDS

The user can accomplish various operations by means of commands, which he enters at the teletype. The Console ROM interprets and acts upon these commands. All commands have the general format:

[a [/aaaa]] c [,m] (CR)

where:

- a is a hexadecimal address of value constant (Section 3.6.2.1)
- c is a Console ROM command mnemonic (Section 3.6.2.2)
- m is a command modifier (Section 3.6.2.3)

(CR) is a carriage return

Each of the components of a command is discussed in the following paragraphs. Command components in brackets ([]) are optional.

All Console ROM commands are terminated by typing a carriage return. If, during the typing of a command, a user makes an error, a rubout may be typed to delete the entire command.

3.6.2 COMMAND COMPONENTS

3.6.2.1 Address or Value Constant

An address or value constant is a four-digit, hexadecimal value which is used as either an absolute word address or a data value, depending on the command currently being processed. When entering an address or value constant, leading zeros may be omitted. Only the last four digits of the string are accepted as the address or value. therefore, if an error is made, you can continue typing and enter the correct four digits as the last characters of the string.

Entering a slash (/) character terminates the accumulation of the current address or value constant and indicates the beginning of the accumulation of hexadecimal digits for the next address or value constant. A maximum of four addresses or value constants are allowed. If more than four address or value constants are input, or if no address or value is specified, Console ROM will output a question mark (?) followed by a carriage return and line feed, ignoring everything which has been previously entered. A new command line may then be entered.

3.6.2.2 Command Mnemonics

Each command mnemonic is specified by a single, non-hexadecimal character. The typing of a command mnemonic terminates the acceptance of address or value constant input and activates a particular command.

If the command mnemonic is not recognized, the Console ROM outputs a question mark (?) followed by a carriage return and line feed, ignoring everything which has been previously entered. A new command line may then be entered.

All valid Console ROM command mnemonics are shown in the following listing.

<u>MNEMONIC</u>	<u>FUNCTION</u>
Carriage Return (CR)	Memory display and alter (at TTY or CRT)
G	Go, program execution
I	Punch paper tape in binary
L	Load binary paper tape
M	Memory dump
R	Register display and alter
S	Single instruction execution
T	Trap set
Y	Set bias for relative addressing
Z	Fill memory
!	I/O reset

3.6.2.3 Command Modifier

The optional command modifier is used to indicate the addressing mode to be used in the performance of the indicated command. It is separated from the command mnemonic by a comma and is indicated by a single letter. For some commands, a modifier is not accepted and any attempt to enter one will be ignored. The modifier A specifies that absolute addressing is to be used. That is, the contents of the hexadecimal address entered as part of the command is to be used for referencing memory during command execution.

The modifier B specifies that relative addressing is to be used. That is, the bias (specified by command Y) is to be added to each hexadecimal address entered to compute the absolute address to be used for referencing memory during command execution.

After an addressing mode is specified, that mode (A or B) will remain in effect until explicitly changed by the modifier specification on another command. The initial addressing mode is absolute (A).

If the command modifier is not recognized, or if an invalid modifier is specified, the Console ROM will output a question mark (?) followed by a carriage return and line feed, ignoring everything which has been previously entered. A new command line may then be entered.

3.6.3 COMMAND DESCRIPTIONS

In the discussion of the various commands which follow, the following conventions are used:

␣ represents a space character

ⒸⒹ represents the RETURN key of the teletype or CRT.

Lower-case letters (e.g., b,e,n, etc.) are generic indications of the address or value components.

Letters or special characters must be specified as indicated. Command components in brackets [] are optional. Italics represent the portion typed by the operator.

3.6.3.1 Memory Display and Alter

Mnemonic: none

This command is implied by terminating a command (via a carriage return) before a modifier is specified or by the specification of a modifier.

Format: a [,m] (CR)

Operation: The Console ROM starts a new line and types the address (a) followed by an equal sign (=), the contents of location a, and a blank space. If the user responds with a space, the contents of the next sequential memory location will be displayed. If the user responds with a hexadecimal number, that value will immediately replace the displayed number in that memory location. If the user responds with a carriage return, the command is terminated.

Examples: *3FC0,A* (CR) Display the contents of location 3FC0.
3FC0=1642 Location 3FC0 contains the value 16A2; the space response requests that the next sequential location be displayed.
3FC1=D4BC *2DC5* Location 3FC1 contains the value D4BC; the value 2DC5 is to replace D4BC in location 3FC1 and the next sequential location is to be displayed.
3FC2=A208 (CR) Location 3FC2 contains the value A208; no change is made to location 3FC2 and the command is terminated.

3.6.3.2 Execute Program

Mnemonic: G

Format: [a]G [,m] (CR)

Operation: If an address (a) is specified, the Console ROM will start execution of the user's program at the specified address. If no address is specified, the Console ROM will start execution of the user's program at the address where the last trap occurred, or at the address following the location where the last single instruction execution occurred.

If no address was specified and no trap or single instruction execution has occurred, the Console ROM will respond with a question mark (?), ignore the command, and be ready to accept another command.

Examples: 3000Y,A (CR) Set relative addressing bias at location 3000.

⋮

100T,B (CR) Set trap at memory location 100 relative to bias.

⋮

10G,B (CR) Start execution of the program at memory location 10 relative to the bias.

⋮

B+0100 Location of trap (previously set) is printed followed by (CR) (LF) .

⋮

G (CR) A trap returns control to the Console ROM; execution is to continue at the location where the trap occurred.

3.6.3.3 Punch Memory In Binary

Mnemonic: I

Format: b/eI (CR)

The beginning address (b) is the first memory location to be punched and (e) is the last memory location to be punched.

Operation: Memory locations (b) through (e) are output to the teletype for punching on paper tape. When the command is terminated (via a carriage return) a 100-character leader is punched. It is during this time that the user must turn on the teletype punch. After the leader, memory locations (b) through (e) are punched in binary. Following the punching of data, a 100-character trailer is punched. The Console ROM is then ready for another command; at this time, the user must turn off the teletype punch.

NOTE

A carriage return line-feed is not output when the punch command is completed. Therefore, the teletype print head will be positioned at any random position when the command is completed. The user may input any non-hexadecimal character which is not a command mnemonic in order to effect a carriage return line-feed (see Section 3.6.2.2, paragraph 2).

Examples: 23FD/347CI (CR) When the (CR) is typed, a 100-character leader will be punched. Turn on the teletype punch at this time. Next, locations 23FD through 347C will be punched, followed by a 100-character trailer. When punching is complete, turn off the teletype punch.

3.6.3.4 Load Memory In Binary

Mnemonic: L

Format: aL (CR)

NOTE

This provides an alternate to IPL by permitting a bootstrap loader on paper tape to be read into memory.

Operation: The teletype paper tape reader is read and loaded directly into memory, starting at location 1. Only a binary paper tape can be properly processed. The paper tape must be placed in the teletype paper tape reader with the first non-blank frame over the read head (i.e., the leader frames will NOT be ignored). The reader must be turned on AFTER the load command has been terminated by a carriage return. Loading will continue until the tape reader is turned off (i.e., the trailer frames will be read and loaded; they will NOT terminate the loading process. After the loading process has been terminated by turning off the paper tape reader, the user must re-enter the Console ROM by pressing the RESET button. The Console ROM may also be re-entered by executing a break reset function.

Example: 80L (CR) Load from the teletype paper tape reader starting at location 80.

3.6.3.5 Memory Dump

Mnemonic: M

Format: b/eM [,mode] (CR)

The beginning address (b) is the first memory location to be printed and (e) is the address of the last memory location to be printed.

Operation: The contents of memory locations starting with location (b) and proceeding through location (e) are printed at the console teletype. The data is displayed eight locations per line, and each line is prefixed with the address of the first (left-most) value for that line. The contents of memory locations cannot be changed with this command.

If the memory address specified by (e) is less than the memory address specified by (b), printing will proceed from location (b) to location FFFF, then from location zero to location (e).

Examples: 10/22M,B (CR) Print contents of relative memory locations 10 thru 22.

```
B+0010=0782 D891 ...
B+0018=6088 103F ...
B+0020=5824 0135 0003
```

3.6.3.6 Register Display and Alter

Mnemonic: R

Format: [r] R (CR)

where: r must be in the range: $0 \leq r \leq 8$

<u>r value</u>	<u>register name</u>
0	A
1	X
2	Y
3	Z
4	B
5	C
6	D
7	E
8	S (status)

Operation: If r is specified, the Console ROM starts a new line and types the register name followed by an equal sign, the contents of the register, and a space. If the user responds with a space, the contents of the next sequential register will be displayed. If the user responds with a hexadecimal number, that value will replace the displayed number in that register. If the user responds with a carriage return, the command is terminated; if he enters a hexadecimal number, that value replaced the contents of the register before the command is terminated.

If r is not specified, all nine registers are displayed on a, single line. The value contained in each register is preceded by its name and an equal sign. No register values can be changed with this format of the command.

Examples: 3R (CR) Display the contents of register Z.

Z=3FC0 ␣ The contents of register Z is 3FC0; display the next register.

B=2ABC 2D5C␣ The contents of register B is 2ABC; replace the contents with 2D5C, then display the next register.

C=1044 1408 (CR) The contents of register C is 1044; replace the contents with 1408, then terminate the command.

3.6.3.7 Executing a Single Instruction

Mnemonic: S

Format: S $\text{\textcircled{CR}}$ or aS [,mode] $\text{\textcircled{CR}}$

where: a is the memory location of the single instruction to be executed.

Operation: If an address (a) is specified, the Console ROM will execute the instruction of the user's program at the location specified; control will return to the Console ROM. If no address is specified, no command terminator (carriage return) will be accepted, and the Console ROM will execute the instruction of the user's program at the address where the last trap occurred or at the address following the location where the last single instruction execution occurred (i.e., the next instruction), whichever occurred last. If no address was specified and no trap or single instruction execution has occurred, the Console ROM will respond with a question mark (?), ignore the commands, and be ready to accept another command.

When control returns to the Console ROM from a single instruction execution, all the current registers are saved (including the status) and the address of the next instruction to be executed (i.e., register P) is typed at the teletype. The Console ROM is then ready to accept another command.

NOTE

A single step through the following instructions require special procedures; in general, it is best to avoid attempts to single step through them.

WAIT: Specify 'a' to advance past location of WAIT instruction.

XEC: A TRAP (T) must previously be set past the XEC instruction location.

Any PIO instruction to the serial I/O controller device address X'3F': The SCI also uses this device and leaves it in RCV mode when it exits. Results may be unpredictable.

Examples:	94S, B $\text{\textcircled{CR}}$ B+0095	Execute the single instruction at relative location 94. The instruction at relative location 94 was executed; the next instruction to be executed is at relative location 95.
	S B+0024	Execute the next instruction (i.e., the single instruction at relative location 95). A single instruction was executed; the next instruction to be executed is at relative location 24 (a JMP, JSR, etc., must have occurred).

3.6.3.8 Set Trap

Mnemonic: T

Format: [a] T [,mode] (CR)

where: a is the address of the memory location where control is to return to the Console ROM BEFORE the instruction at that memory location is executed.

Operation: The Console ROM uses the GA-16 TRAP instruction to facilitate the setting of break points within the user's program. This facility enables control to return to the Console ROM when control reaches that point so that the user may examine or alter either memory locations or registers, or he may redirect execution control.

When a TRAP returns control to the Console ROM, all of the current registers are saved (including the status), the user's original data words are restored, the TRAP is deactivated, and the address of the memory location where the TRAP occurred is typed on the teletype. The Console ROM is then ready for a command to be input. A maximum of four TRAPs may be active at any one time. If the user attempts to set more than four TRAPs, the Console ROM will respond to each such TRAP command with question mark (?), ignore the command, and be ready to accept another command. The status of the four active TRAPs is not affected.

If the address (a) is not specified, a TRAP will be removed. When a TRAP is removed, the address of the restored memory location is typed on the teletype. If no TRAPs are active when a request for TRAP removal is made, the Console ROM will respond by typing a question mark (?), and be ready to accept another command.

Examples: 1000T,A (CR) Set a TRAP at absolute memory location 1000.

10T,B (CR) Set a TRAP at relative memory location 10.

0G (CR) Start user program execution at relative memory location 0 (note that the addressing mode of the previous command remains in effect).

B+0010 Control returns to the Console ROM upon encountering
: the TRAP which was set at relative memory location 10.
:

T (CR) Remove the TRAP. Address of the TRAP removed. (Note:
1000 It is assumed that the user specified some command which changed the addressing mode from relative to absolute.)

T (CR) Remove a TRAP.
? All TRAPs have been removed.

3.6.3.9 Set Relative Addressing Bias

Mnemonic: Y

Format: aY[,A] (CR)

where: a is the address bias to be used when the relative addressing mode is in effect.

Operation: The value of the bias to be used when the relative addressing mode is in effect is changed to the address specified. The addressing mode is NOT affected.

Examples: 3000Y,A The bias to be used in relative addressing is set to 3000.

7/11M,B The eleven-word block of memory beginning at relative location 7 (absolute location 3007) is to be listed.

3.6.3.10 Fill Memory

Mnemonic: Z

Format: b/e/nZ[,mode] (CR)

where: b is the address of the first memory location to be filled with the specified address value n, and (e) is the address of the last memory location to be filled with the specified value n.

Operation: Memory locations (b) through (e) (inclusive) are filled with the specified value (n).

If the memory address specified by (e) is less than the memory address specified by (b), filling will proceed from location (b), to the first location where the store of data is not effective (i.e., non-existent memory or ROM).

Examples: 23AB/24AB/OZ,A (CR) Fill absolute memory locations 23AB through 24AB with zeros.

55/63/1111Z,B (CR) Fill relative memory locations 55 through 63 with the hexadecimal value 1111.

3.6.3.11 I/O Reset

Mnemonic: !

Format: ! (CR)

Operation: A system-level I/O reset is performed (i.e., a CTRL 2 to device X'3' is performed). This will initialize (reset) all peripheral controllers (except the teletype) and will not affect the CPU.

3.7 CONSOLE ROM SYSTEM INTERFACES (GA-16/220)

This section describes the methods by which the console ROM may interact with a user's program.

3.7.1 USE OF INTERRUPT VECTORS

The Console ROM makes use of interrupt vectors at locations X'44' and X'46'.

Location X'44' and its associated cells X'7C' and X'7D' are used to service traps. When a G (3.6.3.2) command is given, the Console ROM checks to see if the user has specified any traps to be set. If any traps are to be set, the Console ROM, at this time, sets the appropriate traps after first saving the user's original object code. The contents of location X'44' is then saved and the vector is changed to the Console ROM's TRAP entry point.

When a trap interrupt through location X'44' occurs, the Console ROM examines the instruction causing the trap. If it is not a trap that the Console ROM set, control will go to the location pointed to by the saved trap vector; if a trap that the Console ROM set caused the interrupt, the address where the trap occurred will be printed on the teletype, those user locations containing the traps will be re-established with user code, and the location X'44' vector will be reset to the user's original contents.

Location X'46' is set by the Console ROM to point to its single step or break interrupt entry point whenever the RESET button is pressed or a G (3.6.3.2) or S (3.6.3.7) command is given. The user's contents are neither preserved or restored.

3.7.2 USE OF THE BREAK INTERRUPT FEATURES

The CPU has a BKDS switch and a BKINT switch. When the BKDS is set to enable the BREAK key and the BKINT switch is set to allow an interrupt upon the detection of a break signal, the user may interrupt his program at any point in time by pressing the teletype BREAK key. When this is done, an interrupt through location X'46' will occur, the Console ROM will save all registers and the status at the time of the interrupt, and then print at the teletype the address of the next instruction to be executed in the user program.

3.7.3 USE OF CONSOLE ROM CLOSED SUBROUTINES

Table 3-5 provides a list of the closed subroutines which a user may call from his program. Assembly coding must be used. For all Console ROM subroutines, the condition indicators (Z,P,O,L) will be changed.

The addresses shown in Table 3-5 are the full 16-bit (64K) addresses and should be used by the user in referencing these routines even in the 32K mode (since the high-order bit will be truncated). The addresses in parenthesis are the corresponding addresses for 32K mode, so that the user will be aware of possible address conflicts.

The Console ROM subroutines use base-relative references (Section 4.3.1.2); therefore, register D must be set to the base location X'FE00' in the ROM prior to entry of any subroutine. If the user program also uses register D, the user's calling sequence must save register D contents (in a memory address or an available register) before setting register D to X'FE00'. Upon return, register D contents may then be restored. The CAP-16 assembler does not provide any automatic means of referencing the subroutines.

The user must construct his own linkages to the addresses, usually via equate (EQU) directive in his program. A suggested method is, for example:

ROMDB	EQU	X'FE00'	Base address for subroutine data references
TTYIN	EQU	X'FC57'	Address of TTY input subroutine.
	:		
	:		
RTR		Z,D	Save program base
LDV		D,ROMDB	Load subroutine base
JSR		TTYIN	Read a character from TTY
RTR		D,Z	Restore program base

CAUTION

A note of caution should be observed — if for some reason, a new version of the Console ROM is made, the subroutine entry locations and base address may change.

Table 3-5. Console ROM Closed Subroutines

Hexadecimal Location	Function	Registers Affected; Comments
FC57 (7C57)	Read one character from the console TTY into register A.	A; data in bits 7-0, bits 15-0 = zero, parity bit not forced on.
FC5C (7C5C)	Write one character from A register to the console TTY.	None; data taken from bits 0-7 of register A.
FC61 (7C61)	Output the 4 hexadecimal digits in register B to the console TTY.	A,Y,B; uses RAM location FE03 (7E03)
FC73 (7C73)	Input hexadecimal digits into register b; stops on first non-hexadecimal character.	A,B,C; uses RAM locations FE03 (7E03), FE04 (7E04), and FE01 (7E01); on return: <ul style="list-style-type: none"> - Register B contains the last 4 hex digits input. - Register C contains the terminative non-hex character with parity in bits 7-0, bits 15-8 are zero. - Zero indicator is set if any valid digits were input; reset if not.
FC8C (7C8C)	Input an ASCII character.	A,C; uses RAM location FE04 (7E04) on return: <ul style="list-style-type: none"> - Register A contains the character without parity in bits 7-0, bits 15-8 are zero. - Register C contains the character with parity in bits 7-0, bits 15-8 are zero.
FC9B (7C9B)	Output carriage-return and line-feed to console TTY.	A; uses RAM location FE03 (7E03)
FD88 (7D88)	Punch standard length paper tape leader or trailer on console TTY paper tape punch.	A,X; uses RAM location FE03 (7E03)

3.8 BUS-16 STAND-ALONE UTILITY

BUS-16 is a powerful utility program which is similar in scope to the console ROM with enhancements. BUS-16 is provided in PGS or binary format on cards or tape. BUS-16 can be loaded in PGS format by the PGS LOADER or from disk; or in binary format by TTYB or HSPT bootstrap loaders. It can perform the functions of both the PGS LOADER and PGS PUNCHER. In addition, its control commands make it ideal for debugging another stand-alone program in core. Using BUS-16, the following functions can be performed by directives at the teletype keyboard.

1. Select octal or hexadecimal mode of operation.
2. Select absolute or relocatable address mode.
3. Load object program in PGS format.
4. Display and alter memory.
5. Display and alter registers.
6. Search memory for specified value.
7. Selectively execute object program.
8. Selectively list memory.
9. Punch PGS or binary object program.
10. Calculate sums and differences.

Since BUS-16 is relocatable, it may be loaded into any free area of core. Total memory required is approximately X'4C0' words.

Its execution address is the first location of the program.

For loading and execution procedures refer to GA-16/220/330 Stand-Alone Utilities, document number 94A01531A.

3.9 TEST AND VERIFY (T&V) PROGRAMS

A series of test and verify programs are available for checkout of the SPC-16 and GA-16 series computer systems as described in the following sections.

3.9.1 CPU T&V

The CPU T&V program for GA-16 series computers is available in binary or PGS format on paper tape. Description of the test and procedures for running the CPU T&V are provided in comments which begin the program listing (GA release #5T10A). For a GA-16/220 system, an introductory manual entitled "How to Use Your GA-16/220" (Document Number 88A00525A) is available. This manual is oriented to the installation, familiarization, and first-time use of a GA-16/220 system. The CPU T&V program is run to check out the system.

3.9.2 MPP T&V

A T&V program for GA-16 systems equipped with an MPP module is available. Description of the tests and procedures for running the MPP T&V are provided in comments which begin the program listing (GA release number 5T110A).

3.9.3 PERIPHERAL PRODUCT T&Vs

T&Vs for peripheral products are available for the SPC-16 series of computers. The procedures provided with these T&Vs are oriented toward use of a SPC-16 machine. There are several differences when running them on a GA-16 series computer. These differences are described in a reference document 94A01519A entitled Execution of Test and Verify Programs on the GA-16 Series Computers.

instruction repertoire **4**

This section describes the GA-16/110/220 instructions. A working program is illustrated at the end of Section 4 (Figure 4-8) in which many of the GA-16/110/220 instructions are used.

4.1 INSTRUCTION GROUPS

Table 4-1 shows the fourteen instruction groups and the entire instruction repertoire for the GA-16/110/220 computer systems. This instruction set provides many operations not commonly available on a minicomputer. These operations include:

- Bit instructions that will test, set, or reset any bit in memory, facilitating efficient use of logical variables.
- Byte instructions to load or store into memory any byte, or to switch the contents of two bytes in a word; this greatly simplifies handling character data.
- Register-to-register and literal (immediate)-to-register arithmetic and logic instructions. These instructions may generate results which are stored in a register, or the results may be reflected in status indicators only (indicating greater than, less than, zero, non-zero, plus, minus, etc.).
- Compare-memory-with-register instructions to reflect results in status indicators only; this allows efficient searching of memory for specified values.
- Capability of loading an instruction into a register and of executing it; this feature facilitates the use of procedure-only programs in read-only memory (ROM).
- Jump to a subroutine and return via the address stored in register E; the technique of saving the return address in a register decreases subroutine and interrupt processing overhead.
- Load all registers and status, or save all registers and status; each of these operations requires only one instruction and greatly simplifies coding of interruptable and re-entrant programs.

The GA-16/220 provides the following additional instruction capability:

- Single step and I/O reset under program control.
- Read microconsole data entry switches under program control.

Instructions having similar formats (and addressing modes, where applicable) are introduced as a group, followed by subsections which describe individual instructions and include examples of how the instructions are used.

Table 4-1. GA-16/110/220 Instruction Repertoire (Sheet 1 of 3)

Op Code	Variable Parameters	Description	Execution Time (μ s)	Discussed in Paragraph Number
<u>MEMORY REFERENCE</u>				
JMP	Addr	Jump unconditionally	1.55	4.4.1
JSR	Addr	Jump to subroutine	2.05	4.4.2
LDA	Addr	Load register A	2.60	4.4.3
STA	Addr	Store register A	2.60	4.4.4
<u>MEMORY REFERENCE WITH INDEXING</u>				
CMR	R,Addr,X	Compare memory with register	3.60	4.5.1
DECM	Addr,X	Decrement memory	3.60	4.5.2
INCM	Addr,X	Increment memory	3.60	4.5.3
LARS	Addr,X	Load all registers and status	11.50	4.5.4
LDBY	R,Addr,X	Load byte	3.60	4.5.5
LDR	R,Addr,X	Load register	2.60	4.5.6
RBIT	n,Addr,X	Reset bit n (n = 0 to 7)	3.60	4.5.7
SARS	Addr,X	Store all registers and status	15.20	4.5.8
SBIT	n,Addr,X	Set bit n (n = 0 to 7)	3.60	4.5.9
STBY	R,Addr,X	Store byte	3.55	4.5.10
STR	R,Addr,X	Store register	3.00	4.5.11
TBIT	n,Addr,X	Test bit n (n = 0 to 7)	3.60	4.5.12
<u>CONDITIONAL JUMP (SKIP) -- EXTENDED DISPLACEMENT</u>				
SKM	Addr	Skip if minus	2.05	4.6.1
SKN	Addr	Skip if non-zero	2.05	4.6.2
SKOF	Addr	Skip if overflow false (no overflow)	2.55	4.6.3
SKOT	Addr	Skip if overflow true	2.55	4.6.4
SKP	Addr	Skip if plus	2.05	4.6.5
SKR	Addr	Skip if link reset	2.05	4.6.6
SKS	Addr	Skip if link set	2.05	4.6.7
SKZ	Addr	Skip if zero	2.05	4.6.8
<u>REGISTER OPERATE and REGISTER OPERATE COMPARE</u> ^②				
(Rd = destination register; Rs = source register)				
ADD	Rd,Rs	Add registers	2.05	4.7.1
ADDC	Rd,Rs	Add register and compare	2.05	4.7.2
AND	Rd,Rs	AND registers	2.05	4.7.3
ANDC	Rd,Rs	AND registers and compare	2.05	4.7.4
OR	Rd,Rs	OR registers	2.05	4.7.5
ORC	Rd,Rs	OR registers and compare	2.05	4.7.6
RTR	Rd,Rs	Transfer register	2.05	4.7.7
SUB	Rd,Rs	Subtract registers (Rd-Rs)	2.05	4.7.8
SUBC	Rd,Rs	Subtract registers and compare	2.05	4.7.9
XOR	Rd,Rs	Exclusive-OR registers s and d	2.05	4.7.10
XORC	Rd,Rs	Exclusive-OR registers and compare	2.05	4.7.11

① Execution times will vary depending on addressing mode, indexing, and (for memory reference with indexing) number of words per instruction.

② Compare instructions do not affect the contents of the registers used.

Table 4-1. GA-16/110/220 Instruction Repertoire (Sheet 2 of 3)

Op Code	Variable Parameters	Description	Execution Time (μ s)	Discussed in Paragraph Number
<u>REGISTER OPERATE LITERAL and REGISTER OPERATE LITERAL COMPARE</u>				4.8
ADDV	R,value	Add value to register R	3.10	4.8.1
ADDVC	R,value	Add value to register R and compare	3.10	4.8.2
ANDV	R,value	AND value with register R	3.10	4.8.3
ANDVC	R,value	AND value with register R and compare	3.10	4.8.4
LDV	R,value	Load value into register R	3.10	4.8.5
ORV	R,value	OR value with register R	3.10	4.8.6
ORVC	R,value	OR value with register R and compare	3.10	4.8.7
SUBV	R,value	Subtract value from register R	3.10	4.8.8
SUBVC	R,value	Subtract value from register R & compare	3.10	4.8.9
XORV	R,value	Exclusive-OR value with register R	3.10	4.8.10
XORVC	R,value	Exclusive-OR value with register R and compare	3.10	4.8.11
<u>SUBROUTINE RETURN VIA INDIRECT VECTOR</u>				
RTNIV	Addr	Return from an NI interrupt	4.70	4.9
<u>REGISTER CHANGE</u>				4.10
ADDS	R	Add shift counter to register R	3.55	4.10.1
CMPL	R	Complement register R	3.05	4.10.2
DECR	R	Decrement register R	2.55	4.10.3
DSPL	R	Transfer register R to Data Bus	3.05	4.10.4
EXBY	R	Exchange bytes	2.55	4.10.5
EXIT	R	Exit from subroutine via register R	2.55	4.10.6
INCR	R	Increment register R	2.55	4.10.7
RCSW	R	Invalid instruction; use RCSR	2.55	4.10.8
RISE	R	Restore interrupt system enable via R	2.55	4.10.9
RLK	R	Add link to register R	3.05	4.10.10
RTRN	R	Return from subroutine via register R	2.55	4.10.11
TRS	R	Transfer register R to status register	2.55	4.10.12
TSR	R	Transfer status register to register R	2.55	4.10.13
XEC	R	Execute instruction contained in R	2.05	4.10.14
ZERO	R	Zero register R	2.55	4.10.15
ZLBY	R	Zero the left byte of register R	2.55	4.10.16
ZRBY	R	Zero the right byte of register R	2.55	4.10.17
<u>SHIFT LEFT</u>				4.11
SLC	R	Shift left circular in register R	2.55	4.11.1
SLCL	R	Shift left circular in register R thru link	3.05	4.11.2
SLIO	R	Shift left logical; insert one's	2.55	4.11.3
SLIZ	R	Shift left logical; insert zeros	2.55	4.11.4

Table 4-1. GA-16/110/220 Instruction Repertoire (Sheet 3 of 3)

Variable Op Code Parameters		Description	Execution Time (μs)	Discussed in Paragraph Number
<u>SHIFT RIGHT</u>				4.12
SRA	R,count	Shift right arithmetic count times	3.05+1.ON ^③	4.12.1
SRC	R,count	Shift right circular count times	3.05+1.ON ^③	4.12.2
SRCL	R,count	Shift right circular in register R thru link	3.05+1.ON ^③	4.12.3
SRLC	R,count	Shift right logical and count	3.05+1.ON ^③	4.12.4
<u>CONTROL</u>				4.13
BMS		Select background registers	2.55	4.13.1
FMS		Select foreground registers	2.55	4.13.2
INE		Enable interrupts	2.55	4.13.3
INH		Inhibit interrupts	2.55	4.13.4
LKR		Reset link	2.55	4.13.5
LKS		Set link	2.55	4.13.6
PMA		Pulse monitor alarm (reset OMA timer)	2.55	4.13.7
SYNC		Generate a sync pulse	2.55	4.13.8
TRAP		Interrupt the program sequence	8.80 ^④	4.13.9
WAIT		Halt execution of instructions	2.55	4.13.10
<u>PROGRAMMED I/O</u>				4.14.
CTRL	fun,device	Output function control pulse	2.05	4.14.1
DTIM	R,device	Transfer data from device into loca- tion pointed to by register R	4.10	4.14.2
DTIR	R,device	Transfer data from device to register R	3.05	4.14.3
DTOM	R,device	Transfer data pointed to in register R from memory to device	3.10	4.14.4
DTOR	R,device	Transfer data in R to device	2.55	4.14.5
TEST	fun,device	Test device	2.55(false) 3.05(true)	4.14.6
<u>READ CONSOLE SWITCHES (220 only)</u> ^⑤				4.15.1
RCSM	R	Read console switches to memory pointed to by register R	3.05	4.15.1.1
RCSR	R	Read console switches to register R	4.10	4.15.1.2
<u>MULTIPLY/DIVIDE</u>				
DIV	n	Divide	3.5+2.ON ^⑥	4.16.1
MPY	n	Multiply	3.5+1.5N ^⑥ +.5I ^⑦ per "1" bit	
<u>SPECIAL INSTRUCTIONS (220 only)</u>				
CTRL	2,X'3E'	I/O bus reset (IORST)		4.17.1
CTRL	1,X'3E'	Enable single instruction interrupt (This instruction must be in upper 1K of memory mode; that is, the SCI)		4.17.2

③ N is the number of shifts -1

④ Includes NI interrupt time

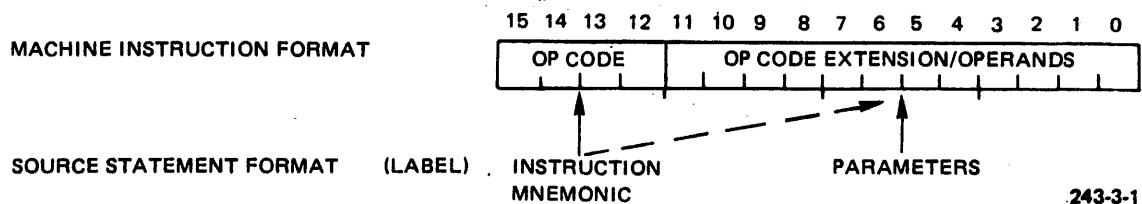
⑤ The GA-16/110 has no microconsole switches for data entry so these instructions have no effect

⑥ N is the total number of bits in the divisor or multiplier

⑦ I is the number of one bits in the multiplier

Examples in this section use CAP-16 assembly language syntax. A *source statement* consists of an instruction mnemonic (for example, JMP for jump instructions) followed by parameters that specify operands such as the address to be referenced by that instruction, the register to be loaded or stored, the register to be used as an index register, the bit to be tested, etc. A source statement may also include a label. A label identifies the memory location at which the machine-language representation of the source statement will be stored; it does not affect machine-language coding until it appears as a parameter in another source statement.

A CAP-16 assembly-language program consists of a series of source statements which are translated by the CAP-16 assembler into machine-language equivalents. The general relationship between machine language format and source-statement format for the GA-16/220 instructions is represented below:



Both formats are given for each instruction group. A machine instruction diagram is presented, which identifies the significance of bits or groups of bits within an instruction word. This is followed by the source-statement format, which lists the instruction mnemonics for that group and defines the parameters which are used. The source-statement parameters are explained in terms of their relationship to the bits in the machine instruction. A discussion of the addressing modes, addressing ranges, and effective address generation is included for instruction groups that reference memory.

4.1.1 CONVENTIONS

The sequence of operations involved in executing an instruction is discussed in each instruction's subsection. The indicators and general-purpose registers that are affected by the execution of each instruction are specified. In discussing the sequence of operations, the following conventions are observed:

- The sequence is shown by one or more arrows. For example, 1→Link means that the contents of the link indicator is replaced by a 1.
- Upper-case letters are used to represent the contents of a specific register. For example, D refers to the contents of register D.
- Bits within a word may be represented as subscripts. For example, B₃₋₀ refers to bits 3 through 0 of register B
- The term *effective address*, EA, refers to the address value calculated during execution of an instruction which addresses a memory location. EA is the value at any step in the evaluation process.

- A memory address is represented by an address symbol without parenthesis. An address symbol enclosed in parentheses refers to the contents of the location addressed by the symbol. For example, EA is an effective address; (EA) is the contents of the memory location addressed by EA; ((EA)) means the contents of the memory location addressed by EA points to another location (i.e., an indirect address).
- The symbol \$ refers to the absolute address from which the instruction currently being executed was fetched. \$+1 refers to the next sequential address. Thus, if the instruction at \$ is a one-word instruction which does not affect program sequencing, \$+1 is the address of the next instruction to be executed. If \$ points to a two-word instruction, \$ is the address of the first word and \$+1 the address of the second word. For a two-word instruction, \$+1 is the second word of the instruction and \$+2 is the address of the next instruction to be executed.
- In examples using base-relative addressing, the symbol \$\$ refers to the base address (that is, the contents of register D).
- A bit position represented as an "X" can be either "1" or "0".
- Hexadecimal numbers are represented by X'nnnn' (X'111', X'7FFF', etc.). Numbers less than X'A' do not have to be represented in this way, but often are for consistency.

4.1.2 ADDRESSING PARAMETERS

Two instruction groups (memory reference and memory reference with indexing) use some addressing parameters that may or may not be present. These parameters correspond to the three addressing stages as follows:

- [,m] Stage-1 addressing (absolute, base-relative, program-relative) - For both groups, the interpretation of this stage is normally left to the CAP-16 assembler. The assembler will choose the correct addressing mode using information given by other source statements (called assembler directives) for which no machine instructions are generated. First-stage addressing can be specified by appending a parameter (,m) to the source statement.
- [*] Stage-2 addressing (direct, indirect) - For both groups, an asterisk specifies indirect addressing; if no asterisk is coded, direct addressing is used.
- [,i] Stage-3 addressing (indexed) - The memory reference with indexing instructions can specify an indexed address by naming an index register as a parameter; if no index register is given, the address is not indexed.

These parameters are enclosed in brackets in the source statement format to indicate that the parameter may or may not be present in the source statement itself.

There is one other addressing parameter needed by three of the instruction groups which reference memory. This is an address expression which is used in determining the value placed into the address field of the instruction. The address expression may be a constant or it may include a label reference. The value of a constant is specified directly as a parameter in a source statement. The value assigned to a label, however, is independent of the source statement which includes it as an addressing parameter, since the label's value is determined by the location of the instruction in which the label appears.

A CAP-16 assembly-language program generally consists of two sections: a program section (PSECT) and a data section (DSECT). The PSECT normally contains all the executable instructions in the program as well as any constants used by the program that do not change during its execution. The DSECT normally reserves storage for variable data items on which the instructions in the PSECT operate.

Labels in a PSECT are assigned a value relative to the beginning of the PSECT. If PB (*program base*) symbolizes the absolute address of the first location in a PSECT, then PB+0, PB+1, and PB+2 identify the addresses of the first three locations. If these locations are given the labels ONE, TWO, and THREE, respectively, label ONE has the value PB+0, label TWO has the value PB+1, and label THREE has the value PB+2. When a label that has been given a program-relative value appears as a parameter in another source statement, that statement is normally assembled into machine code using program-relative addressing. For example, in the following program section:

<u>Value of Label</u>	<u>Label</u>	<u>Mnemonic</u>	<u>Parameters</u>
PB+0	ONE	JMP	THREE
PB+1	TWO	.	.
PB+2	THREE	.	.
PB+3	FOUR	JMP	TWO

the jump instructions would be assembled into machine code using program-relative addressing. The first instruction is located at PB+0; its program-relative origin is its location plus one (or PB+1). The location that the instruction references (THREE) has the address PB+2; the address field will be assembled to contain $(PB+2)-(PB+1)=1$. The fourth instruction's location is PB+3 and its origin is PB+4. It references the address PB+1 (TWO); therefore, the address field of the fourth instruction will contain $(PB+1)-(PB+4)=-3$ (in two's complement form). Since the absolute address PB is eliminated in address calculation, its value is irrelevant to the coding of the instruction. This means that program sections, comprised of instructions that use only program-relative addressing when referencing locations within the PSECT, may be located anywhere in memory without affecting the coding of individual instructions (that is, PB can have any value).

Labels within a DSECT are given a value relative to the beginning of the DSECT. If \$\$ symbolizes the address of the first location of DSECT, then \$\$+0, \$\$+1, and \$\$+2 identify the first three locations of the DSECT. For example, in the following program:

	DSECT		(Indicates data section)
DATA0	DS	1	(Defines labels DATA0 and DATA1 and allows one word
DATA1	DS	1	of memory storage for each.)
.			
.			
	PSECT		(Indicates program section)
.			
.			
LDA	DATA0		Load contents of location DATA0 into register A.
STA	DATA1		Store contents of register A at location DATA1.
.			
.			

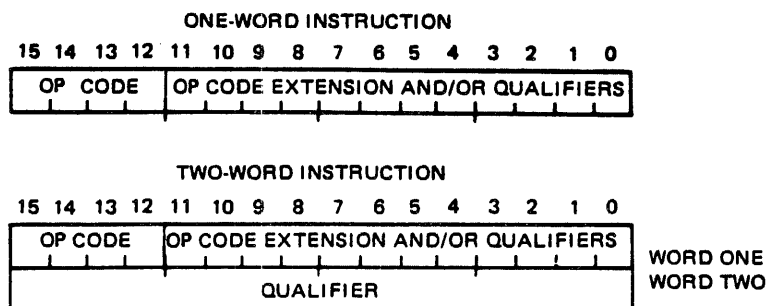
4.1.3 REGISTER IDENTIFIER CODES

Several instruction groups use register identifier codes. These codes are listed below; they are the same for all instructions and apply to both foreground and background register usage (refer to Section 4.13.1 and 4.13.2 for selection of register sets).

<u>Bit Code</u>	<u>Register</u>
000	A or A'
001	X or X'
010	Y or Y'
011	Z or Z'
100	B or B'
101	C or C'
110	D or D'
111	E or E'

4.2 INSTRUCTION FORMAT

Instructions are represented as one or two 16-bit binary words (depending on the op code) in the following general formats:



466-1.

The operations code (bits 15 to 12 of the first word), together with any operation code extension bits, uniquely identify each of the GA-16/220 instructions. The remaining bits are grouped into various qualifier fields. The number, grouping, and significance of qualifier fields depends on the instruction that is specified by the operation code and operation code extension bits. Qualifiers specify such things as the addressing mode that the instruction is to use, the address of the memory location to be referenced by the instruction, the register on which the instruction is to operate, etc. All instructions have an operation code in bits 15 to 12. Instructions requiring the most space for qualifiers are given the shortest operation codes (that is, use only bits 15 to 12 to specify the instruction). Some instructions have a combination of operation code extension and operands in bits 11 to 0, and instructions which do not require any operands use all 16 bits to specify an operation.

A two-word instruction is needed with memory reference with indexing instructions (Section 4.5) when an allowed address is too large to fit in the first word; in this case, the address will be contained in the second word. An instruction which includes a literal value operand (Section 4.8) will always be a two-word instruction with the literal value contained in the second word (that is, the operand for the instruction is in the location immediately following the op-code word). The subroutine return via indirect vector instruction (Section 4.9) is also a two-word instruction.

4.3 ADDRESSING MODES

The processing power of the GA-16/110/220 computer is a result of its flexible addressing scheme. Instructions reference memory locations to either alter program sequencing or to store, retrieve, or test data. Not all instructions reference memory, but those that do contain either explicit or implicit addressing information. During the execution of an instruction containing address information, the CPU uses the address information to determine the absolute address of the memory location to be accessed.

There are a number of methods by which the CPU may calculate an address. These methods are termed "addressing modes". Some instructions use only one addressing mode; for these instructions, the mode is implied by the group to which the instruction belongs. Other instructions allow a choice among several addressing modes; certain bits are set aside in these instructions to select the addressing mode.

All instructions that reference memory (except instructions using literal addressing) have a number of bits that serve as an address field. The address field contains an absolute address or relative address, which may be signed or unsigned.

Figure 4-1 identifies the instruction groups that reference memory and the addressing modes available to each group. As illustrated, an address evaluation requires from one to three stages, depending upon the addressing mode.

4.3.1 EFFECTIVE ADDRESS GENERATION - STAGE 1 (ABSOLUTE, BASE-RELATIVE, PROGRAM-RELATIVE, OR LITERAL)

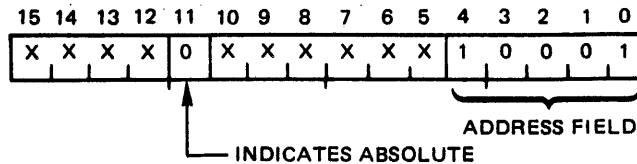
As shown in Figure 4-1, instructions that reference memory may use absolute, base-relative, program-relative, or literal addressing. These terms describe the various methods by which the CPU can determine an effective address in stage 1 of an address generation. For some instructions, the effective address that is determined in stage 1 (EA_1) is the final effective address (EA_{FINAL}). That is, stage 2 and stage 3 are not required. When stage 2 is required, the result of stage 1 is carried into stage 2.

Instruction Group And Function	Addressing Stages			
	Stage 1	Stage 2	Stage 3	
Memory Reference				
a. Move data items	Program Relative } EA ₁	Direct } EA ₂		= EA FINAL
b. Change program sequencing	Base Relative } EA ₁	Indirect } EA ₂		
Memory Reference with Indexing				
Reference word, byte or bit data.	Absolute } EA ₁	Direct } EA ₂	Non-Indexed } EA ₃	= EA FINAL
	Base Relative } EA ₁	Indirect } EA ₂	Indexed	
Conditional Jump (Skip)				
Conditionally change program sequencing	Program Relative EA ₁			= EA FINAL
Register Operate Literal and Register Operate Literal Compare				
Reference a literal value	Literal EA ₁			= EA FINAL

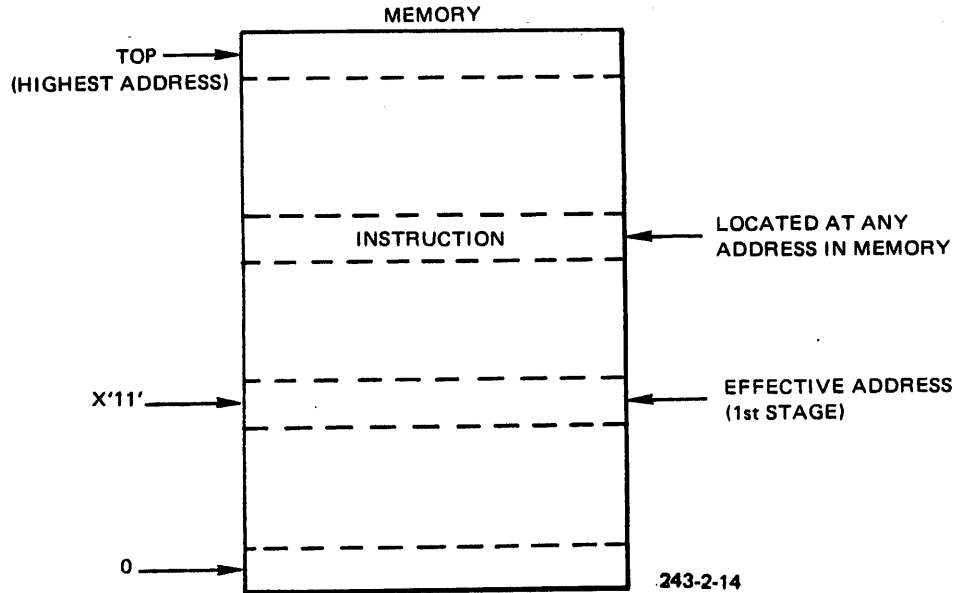
Figure 4-1. Address Stages for Instructions that Reference Memory

4.3.1.1 Absolute

The effective address is the address contained in the address field of the instruction. For example, a memory reference instruction with indexing:



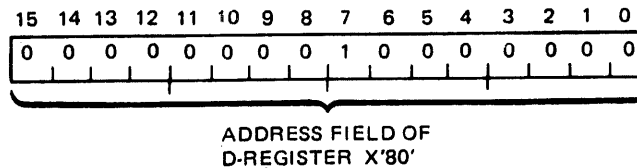
specifies the effective address, X'11'.



Absolute addressing is used by the memory reference with indexing instructions, Section 4.5.

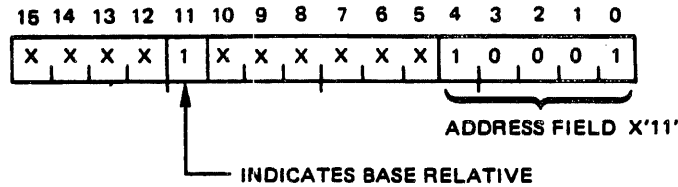
4.3.1.2 Base-Relative

The effective address is calculated as the sum of the contents of register D (called the *base address*) and the contents of the address field of the instruction. For example, if register D contains:

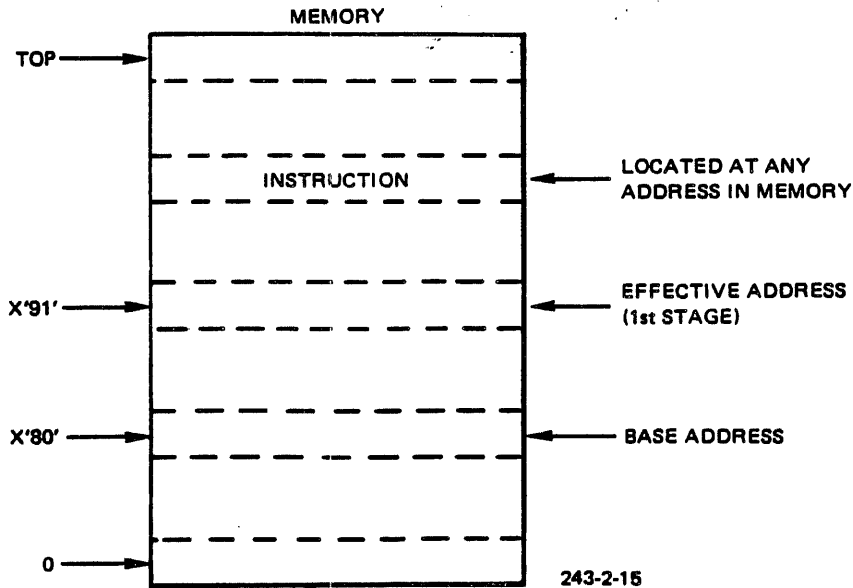


243-2-15

and the instruction is a memory reference instruction with indexing:



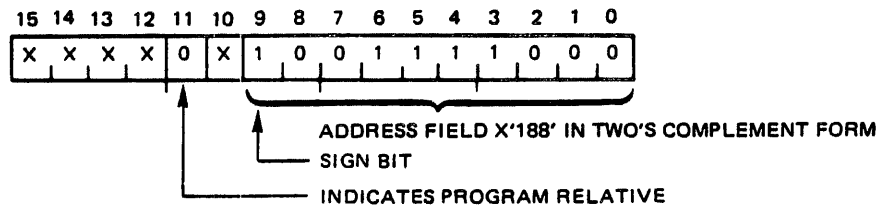
the effective address will be $X'80' + X'11' = X'91'$.



Base relative addressing is used by the memory reference instructions (Section 4.4) and the memory reference with indexing instructions (Section 4.5).

4.3.1.3 Program-Relative

The effective address is calculated as the sum of the address of the memory location following the one from which the instruction itself was fetched and the contents of the address field of the instruction (in two's complement form with the high-order bit of the address field specifying the sign). For example, if the instruction (a memory reference instruction is illustrated) is located at hex address 1FF and it appears as:



the address is determined as follows. The addressing origin is X'1FF'+X'1'=X'200' or, expressed in binary:

```
0000001000000000
```

Since the address field (for program-relative addressing) always contains a signed number, the binary digit indicated in the sign bit is extended to the left to form a 16-digit binary number. In the above example, the address field indicates the following two's complement number:

```
111111001111000
```

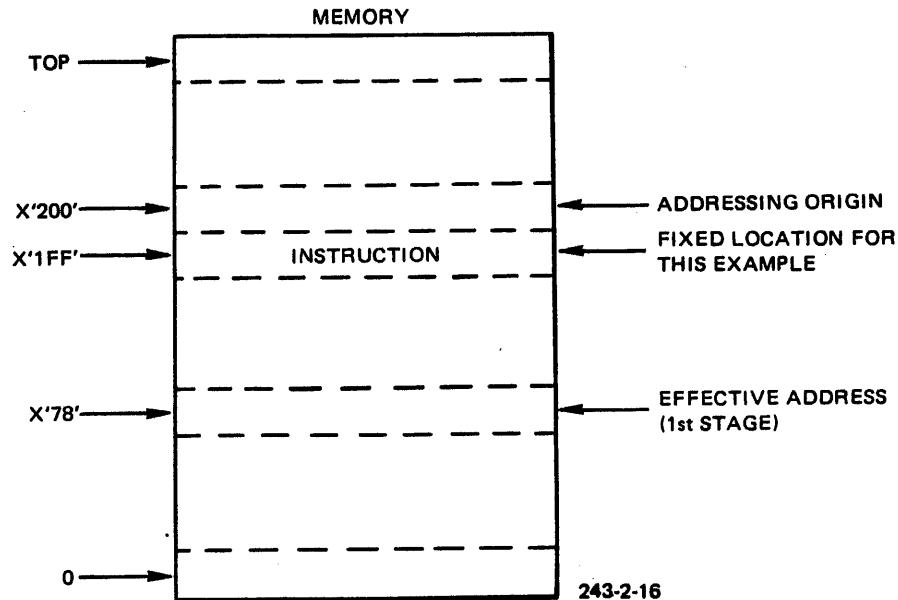
This is summed with the origin address, ignoring the carry.

```

0000001000000000
+1111111001111000
-----
0000000001111000 = X'78'

```

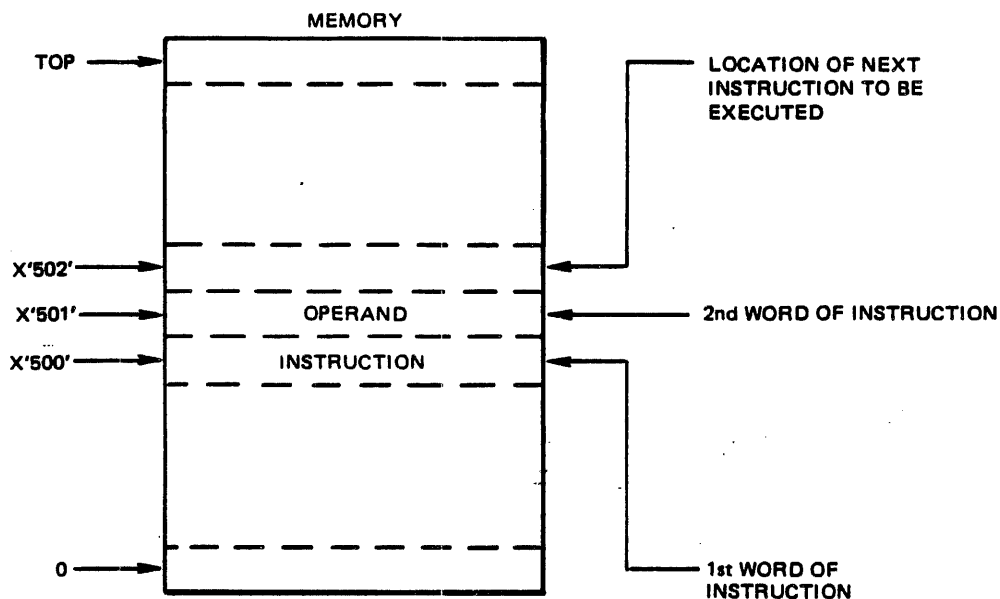
Thus, X'78' is the effective address.



Program relative addressing is used by the memory reference instructions (Section 4.4) and the conditional jump instructions (Section 4.6).

4.3.1.4 Literal

Instructions that use literal addressing are all two-word instructions. The effective address is simply the address of the second word of the instruction (i.e., the operand is contained in the second word of the instruction). The register operate literal and register operate literal compare instructions (Section 4.8) use literal addressing.



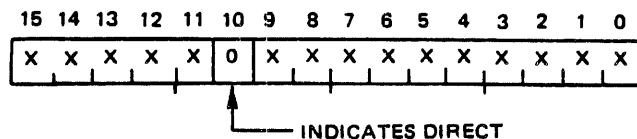
466-4.3.

4.3.2 EFFECTIVE ADDRESS GENERATION - STAGE 2 (DIRECT OR INDIRECT)

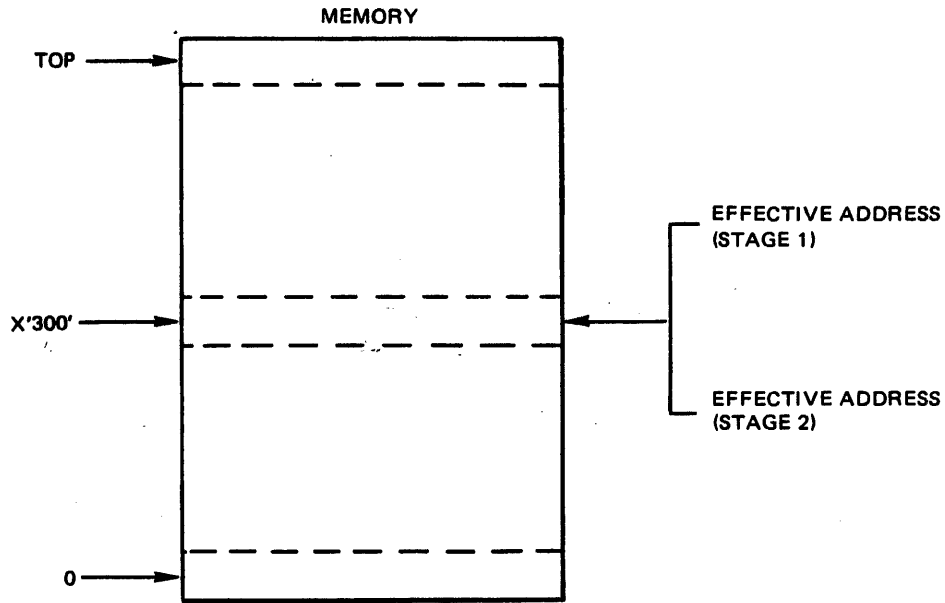
If a second address generation is required, the CPU must determine if the effective address from stage 1 is direct or indirect. In the examples that follow, the CPU interrogates only one bit in the instruction word during stage 2; the address field of the instruction is no longer relevant (it is used only to determine the effective address for stage 1).

4.3.2.1 Direct

The effective address that was generated in stage 1 is simply carried into stage 3 (that is, no operation is indicated in stage 2). The result of stage 1 is thus termed a *direct address*. For example, the following instruction specifies the omission of stage 2 of the operand address generation.



If the effective address resulting from stage 1 for the above instruction were X'300', the effective address carried into stage 3 would be X'300' (that is, $EA_1=EA_2=X'300'$).

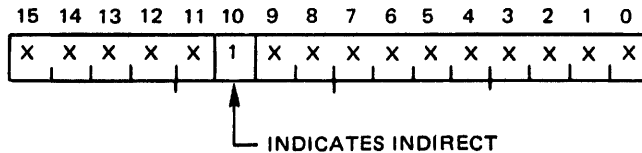


466-4-4.

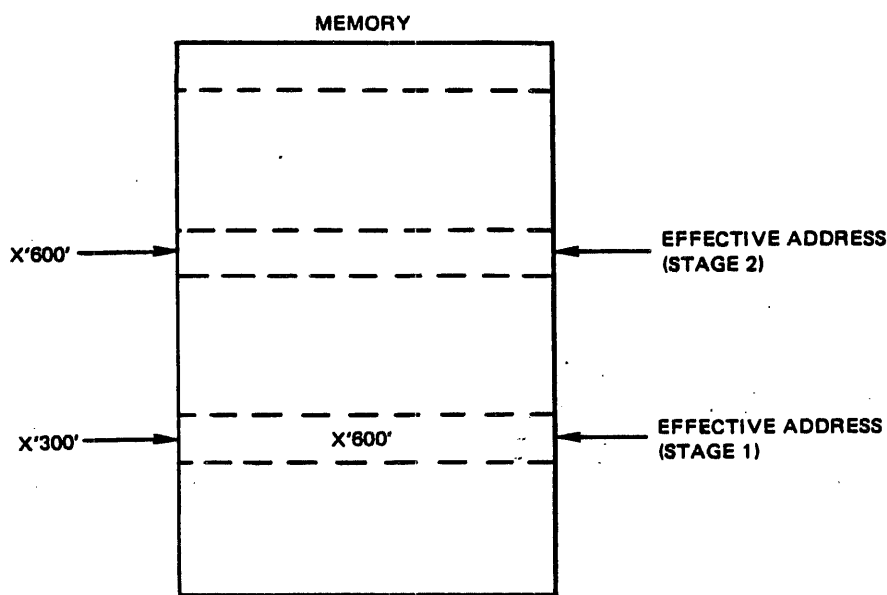
Direct addressing is used by memory reference instructions (Section 4.4), memory reference with indexing (Section 4.5), and conditional jump instructions (Section 4.6).

4.3.2.2 Indirect

The effective address generated in stage 1 is the address of a memory location that contains the effective address for stage 2. When indirect is indicated, an extra memory cycle is required to fetch the effective address for stage 2 (that is, the contents of the memory location identified by the effective address from stage 1). The effective address resulting from stage 1 is called an *indirect address*. For example, the following instruction specifies that the memory location identified by the effective address from stage 1 is to be accessed to retrieve the effective address that is to be carried into stage 3.



If the effective address from stage 1 were X'300' and memory location 300 contained X'600', then X'600' would be the effective address resulting from stage 2.



466-4-5.

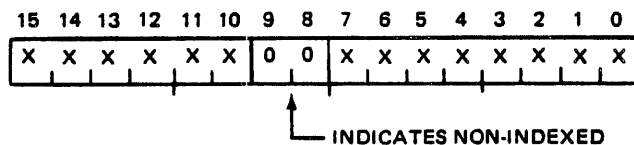
Indirect addressing is used with memory reference instructions (Section 4.4), memory reference with indexing (Section 4.5), return via indirect vector instruction (Section 4.9).

4.3.3 EFFECTIVE ADDRESS GENERATION - STAGE 3 (INDEXED OR NON-INDEXED)

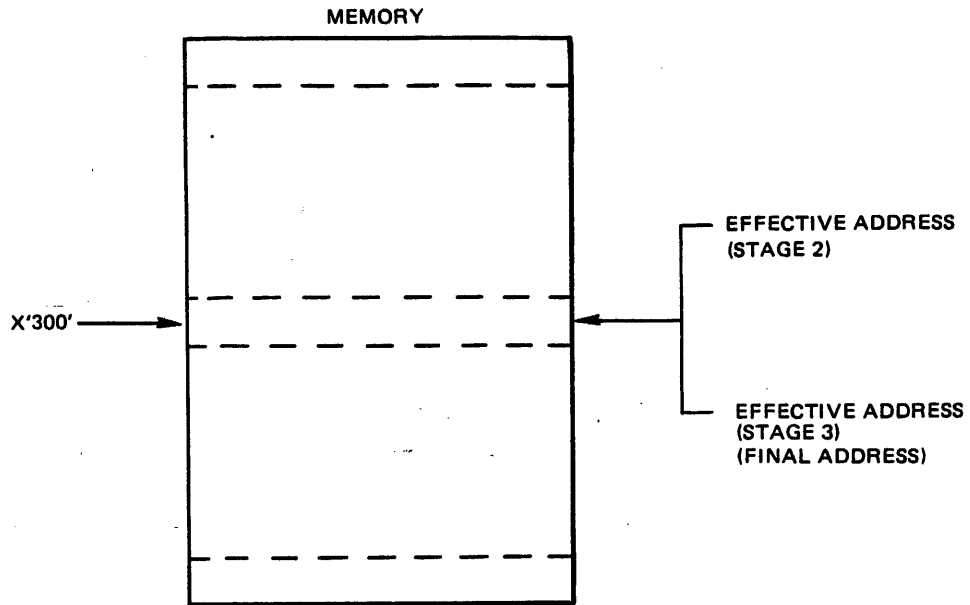
If stage 3 is indicated, the CPU must determine whether or not to add the contents of an index register to the effective address from stage 2. The address determined in stage 3 is the final effective address. Only the memory reference with indexing instructions (Section 4.5) provide a choice of indexed or non-indexed address generation, all other instruction groups which address memory are non-indexed.

4.3.3.1 Non-Indexed

The effective address resulting from stage 2 becomes the final effective address (that is, no operation is indicated in stage 3). For example, the following instruction specifies the omission of stage 3 of operand address generation.

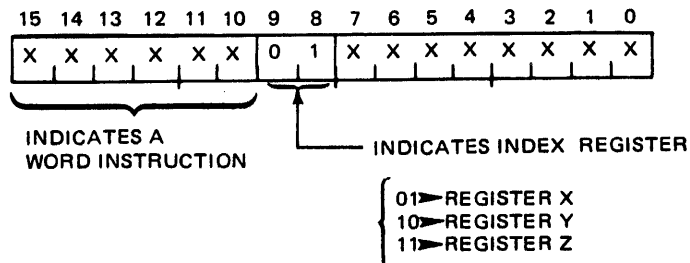


If the effective address resulting from stage 2 were X'300', then the result of stage 3 would also be X'300'. Thus, X'300' becomes the final effective address. On non-indexed instructions which reference bytes or bits, the upper byte of the effective address is accessed.

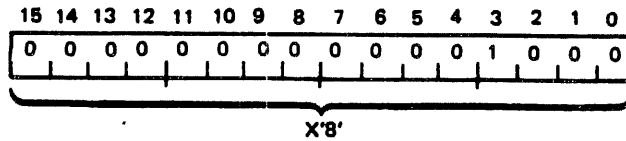


4.3.3.2 Indexed

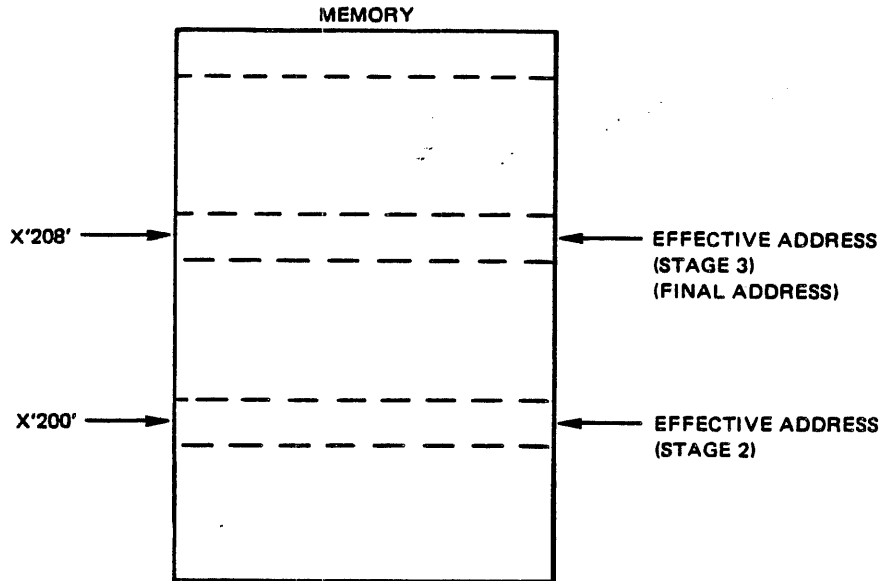
The effective address from stage 2 is modified by the contents of the index register (registers X, Y, or Z) indicated by the instruction, to determine the final effective address. If the instruction is one that operates on word data, the contents of the index register selected are simply added to the effective address from stage 2 to determine the final address. If the instruction is one that operates on byte or bit data, the contents of the index register are divided by two by shifting one bit right (bit 15 is set to 0), then added to the effective address from stage 2 to determine the operand address. The indexing range of byte and bit instructions is limited to 32K. If the contents of the index register are even (remainder=0), the left byte is selected; if odd (remainder=1), the right byte is selected. (For bit instructions, a 3-bit field in the instruction itself indicates which of the eight bits in the selected byte is to be referenced.) For example, the following word instruction specifies that the contents of register X are to be used in determining the operand address.



If register X contains

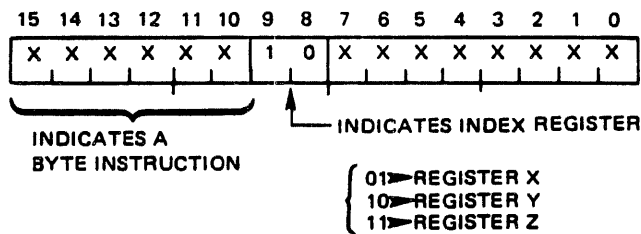


and the effective address from stage 2 is X'200', the operand address is X'200'+ X'8'=X'208'.

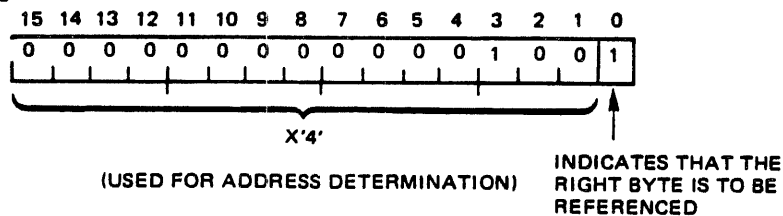


486-4.7.

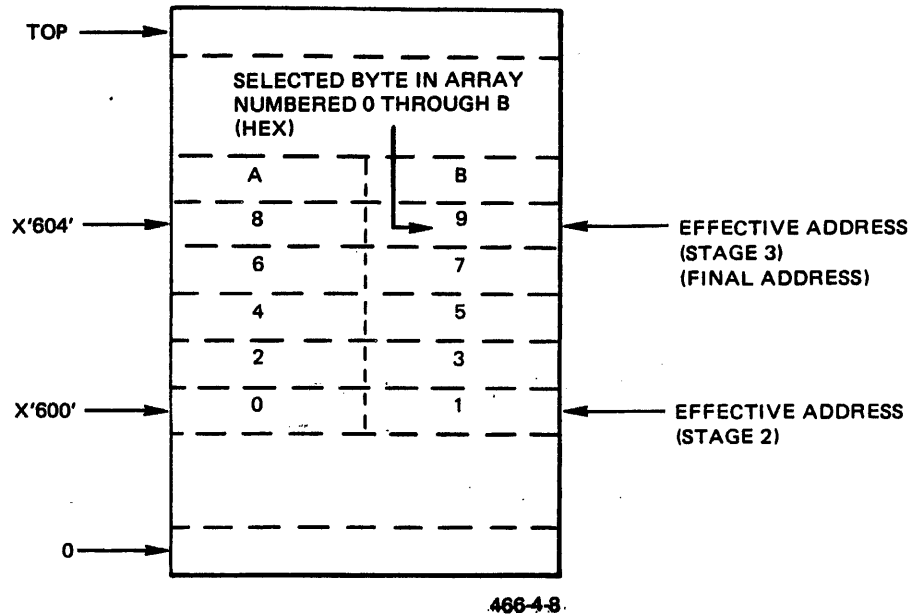
As an example of byte indexing, the following byte instruction specifies that the contents of register Y are to be used in determining the operand address.



If register Y contains



and the effective address from stage 2 is X'600', the operand address is X'600'+X'4'=X'604'.



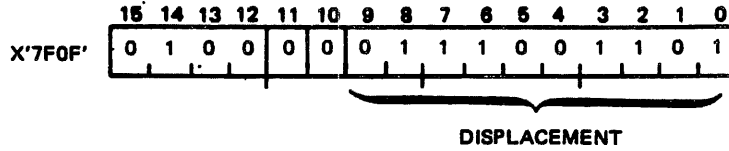
The actual contents of register Y is X'9'. Thus, in byte indexing, the bytes can be viewed as an array with the contents of the index register identifying the element in the array that is referenced.

4.3.4 EFFECTIVE ADDRESS WRAPAROUND

In the previous effective address generation examples it was assumed that the sum of the address origin (P counter +1, or base register), the displacement in the instruction, and the index register contents place the EA within the limits of 32K or 64K memory. Wraparound may occur if the EA range is above or below memory limits in a 32K or 64K system. In a smaller system a condition called effective address out-of-memory may also occur, as described in Section 4.3.5, when EA is above physical top of memory. The most common reason for wraparound or out-of-memory is improper specification of program or base locations when loading programs. Wraparound also can occur when EA calculations greater than X'7FFF' are applied to 32K mode operation. A deliberate wraparound is created when a negative displacement is coded in two's complement form.

The following examples illustrate wraparound and also give further insight into the mechanism by which addition of negative displacements (coded as two's complement) wraparound to place EA below the address origin. The examples also show what occurs when address calculations based on a 64K mode are applied to a machine operating in 32K mode.

1. A Load A (LDA) instruction is executed at location X'7F0F' and has the following format:



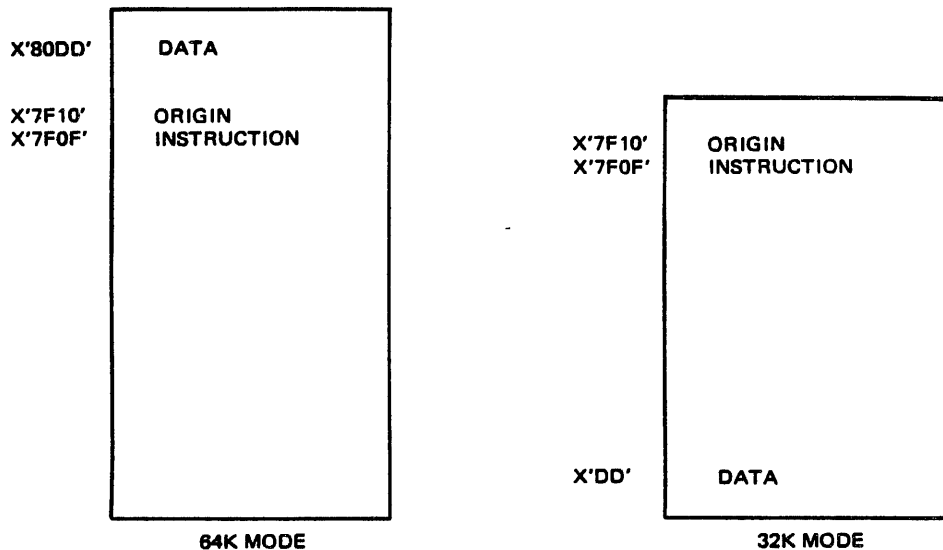
This instruction specifies program relative direct addressing mode with a displacement value (after extending sign bit) of X'01CD'.

The first stage addressing in binary is as follows:

X'7F0F'	0 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1
+X'0001'	1
=X'7F10'	0 1 1 1 1 1 1 1 0 0 0 1 0 0 0 0
+X'01CD'	0 0 0 0 0 0 0 1 1 1 0 0 1 1 0 1
=X'80DD'	1 0 0 0 0 0 0 0 1 1 0 1 1 1 0 1

The effective address calculations are always carried out using all 16 bits, then at the end of the calculations, the 16th bit is forced to zero if in 32K mode.

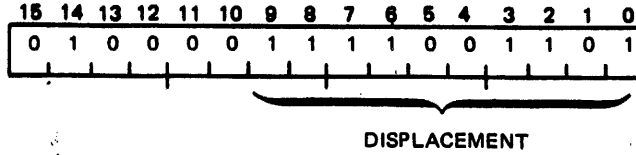
In the example, the difference between the effective address in 64K and 32K memory is illustrated below. The EA for 32K will probably lead to an error in execution.



466-4-9.

The data referenced is above the origin when in the 64K mode, but is below it if executed in the 32K mode. The effective address can be said to have wrapped around the top of core and is now in a lower core area.

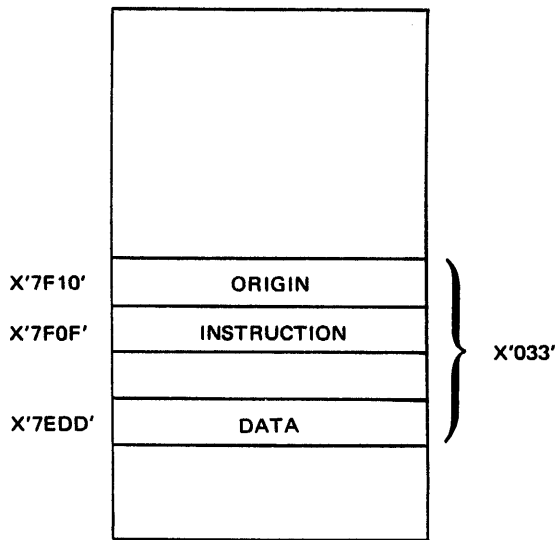
2. Using the same instruction with the sign bit = 1 to denote a negative displacement:



The displacement value after extending the sign bit is X'FFCD' and the calculations (beginning with the origin are as follows):

X'7F10'	0 1 1 1 1 1 1 1 0 0 0 1 0 0 0 0
+X'FFCD'	1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1
X'7EDD'	0 1 1 1 1 1 1 0 1 1 0 1 1 1 0 1

There will be no difference between 32K and 64K mode address. The equivalent negative displacement from the origin = X'033'.



466-4-10

3. Taking the two's complement of X'0033', the negative displacement from the origin of example 2 and coding it into the 10-bit displacement field:

0 0 0 0 1 1 0 0 1 1
1 1 1 1 0 0 1 1 0 0
+ 1
1 1 1 1 0 0 1 1 0 1
displacement

Gives the same binary number as shown in the beginning of example 2. It follows then, that if an address expression were evaluated by an assembler to be -X'0033', the assembler would code the two's complement into the instruction's displacement field. After extending the sign and adding to the P counter+1 the correct effective address in memory is obtained.

4. In this example, a two-word memory reference with indexing instruction is located at address X'11F7' (and X'11F8').

X'11F8'	1 0 0 0 0 1 0 1 0 1 1 1 0 0 0 1	X'8571'
X'11F7'	1 1 0 1 1 0 1 1 0 0 0 1 1 1 1 1	X'DB1F'

The instruction (an STR) specifies base relative direct addressing with register Z indexing. Register A will contain the data to be stored in the effective address. The base register D contains the following value:

D -	0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0	X'0C80'
-----	---------------------------------	---------

and the Z register contains the following:

Z -	0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1	X'0007'
-----	---------------------------------	---------

466-4-11

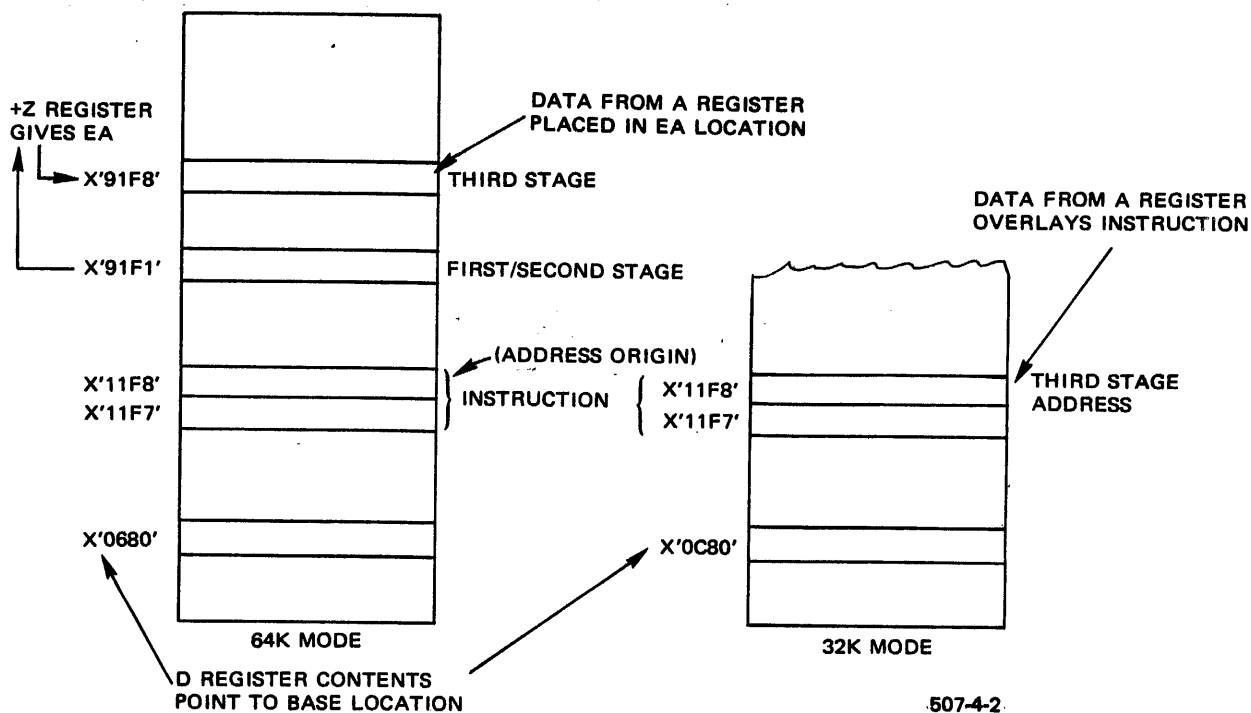
The first stage addressing calculation is as follows and, since this is direct addressing, the second stage result is the same:

1 0 0 0 0 1 0 1 0 1 1 1 0 0 0 1	X'8571'
+ 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0	X'0C80'
<u>1 0 0 1 0 0 0 1 1 1 1 1 0 0 0 1</u>	X'91F1'

The third stage adds the contents of register Z:

1 0 0 1 0 0 0 1 1 1 1 1 0 0 0 1	X'91F1'
<u>0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1</u>	X'0007'
1 0 0 1 0 0 0 1 1 1 1 1 1 0 0 0	X'91F8'

If in 32K mode the most significant bit is zero = X'11F8', the core maps for 64K or 32K are as follows:



4.3.5 EFFECTIVE ADDRESS BEYOND MEMORY

A situation may arise in which an effective address points to a location which is not occupied by real memory. This is different from the wraparound situation described in Section 4.3.4 because in wraparound we assume a memory whose upper limit is exceeded by an EA calculation either through a programming error, or normally whenever a negative displacement is converted to two's complement.

Address beyond memory may occur when operating in 32K mode with less than 32K of memory installed or in 64K mode with less than 64K of memory.

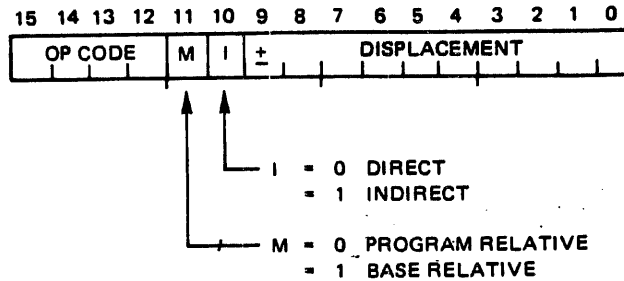
As an example, assume a machine with 16K of memory with a hexadecimal upper limit of X'3FFF'. If a memory reference or memory reference with index instruction results in an EA calculation of X'4C30', that location is beyond memory upper limit. The result then depends on the instruction as follows:

1. Write data into memory: No observable result.
2. Read data from memory: Zeros are returned.
3. Fetch instruction: Zeros are fetched; this is a WAIT instruction and the machine will stop on a WAIT condition.

If MPP is installed, a parity error interrupt will occur in situations 2 and 3 if parity error detection is enabled.

4.4 MEMORY REFERENCE INSTRUCTIONS

There are four instructions in the memory reference group. They are one-word instructions that cause data to be moved to/from register A or to effect a jump to another instruction or subroutine. Memory reference instructions have the following machine instruction format:



243-3-2

The CAP-16 source statement format for this instruction group is:

<u>Command</u>	<u>Parameters</u>
JMP JSR LDA STA	[*]address[,mode]

where:

- * if present, indicates indirect addressing (bit 10 = 1)
- if absent, direct addressing implied (bit 10 = 0)

address is an address expression

mode indicates addressing mode (bit 11)

- = 0 Program-Relative
- = 1 Base-Relative

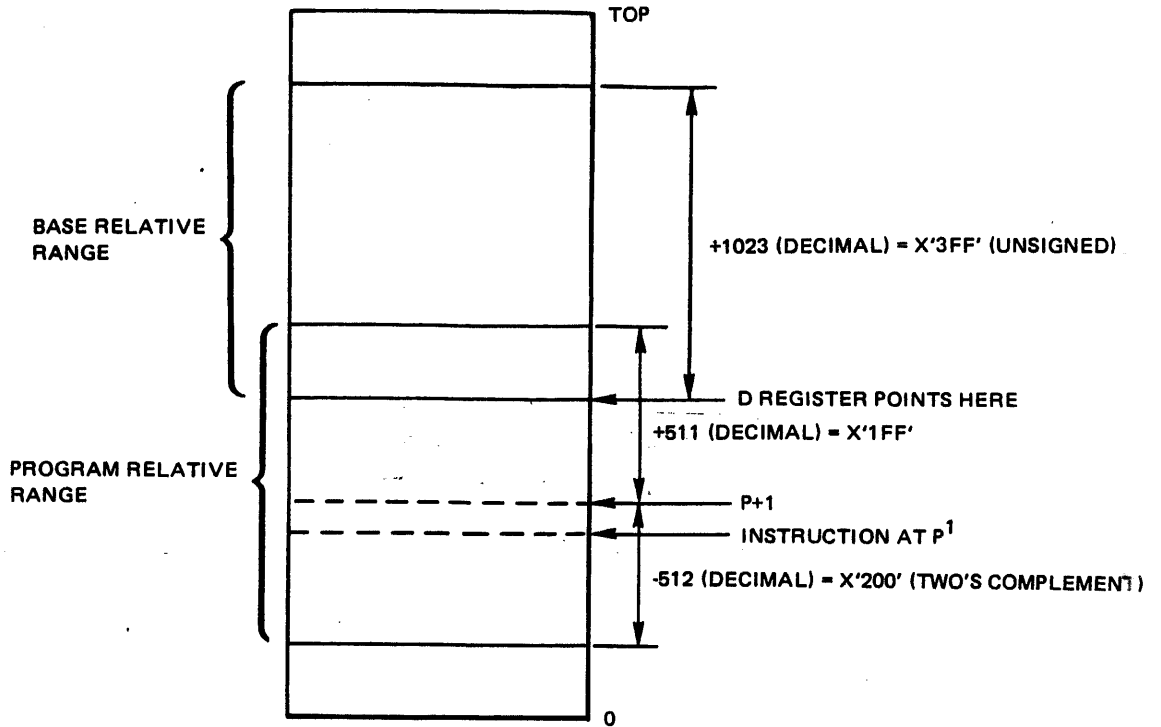
The address displacement (bits 9 to 0) is determined as follows:

1. If mode is absent from the CAP-16 source statement, the addressing mode is determined by the assembler:
 - Program-Relative: address - \$+1 → bits 9 to 0 (two's complement form)
 - Base-Relative: address - \$\$ → bits 9 to 0
2. If mode is specified: address → bits 9 to 0

NOTE

If mode is specified, the address expression is a displacement and must evaluate to a constant within the range 0 to 1023 for base-relative addressing and -512 to +511 for program-relative addressing (i.e., it may not include a relocatable label value).

The addresses, which may be accessed directly by memory reference instructions, are illustrated on the following page.



¹ MEMORY REFERENCE INSTRUCTION LOCATED ANYWHERE IN MEMORY, BUT IF INSTRUCTION LOCATION PLACES RANGE ABOVE TOP OR BELOW ZERO, EITHER EA WRAPAROUND OR EA OUT-OF-MEMORY MAY OCCUR FOR CERTAIN VALUES OF DISP. (SECTIONS 4.3.4 AND 4.3.5.)

507-4.3.

Effective address coding is summarized in Table 4-2.

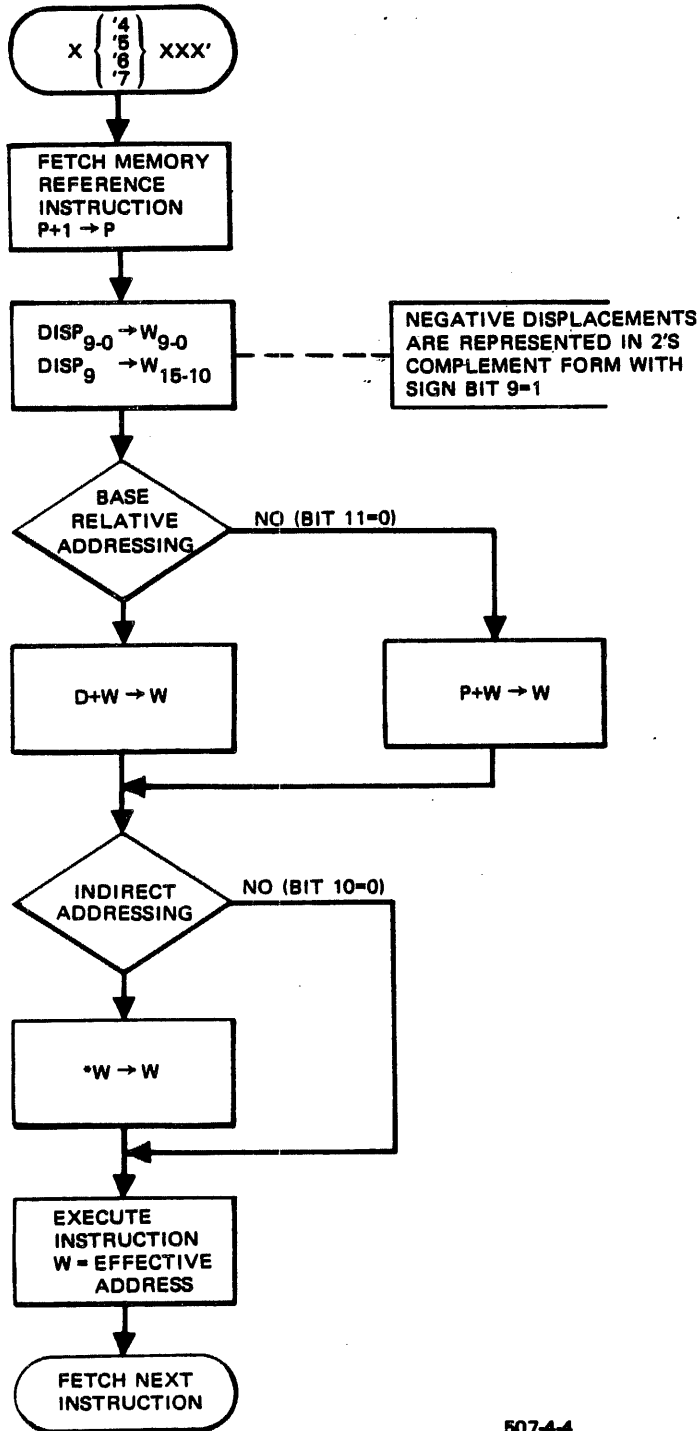
Table 4-2. EA Coding, Memory Reference Instructions

Type of Address	M	I	Range	EA ^①	Next Instruction	Additional Execution Time ^②
Program Relative	0	0	$-512 \leq \text{DISP} \leq +511$	$P+1 \pm \text{DISP}$	P+1	0
Program Relative, Indirect	0	1	$-512 \leq \text{DISP} \leq +511$	$(P+1 \pm \text{DISP})$	P+1	.500 μ s
Base Relative	1	0	$0 \leq \text{DISP} \leq 1023$	$D + \text{DISP}$	P+1	0
Base Relative, Indirect	1	1	$0 \leq \text{DISP} \leq 1023$	$(D + \text{DISP})$	P+1	.500 μ s

① Recall that, as described in paragraph 4.1.1, $D + \text{DISP}$ means that the final effective address is $D + \text{DISP}$: $(D + \text{DISP})$ means that the contents of memory location $D + \text{DISP}$ contains the final effective address.

② Instruction cycles plus additional cycles equals total processor cycles for the instruction. Add 500 ns for each additional cycle.

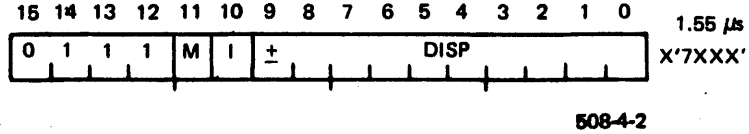
Upon executing a memory reference instruction, the effective address is generated as shown in Figure 4-2.



507-4-4

Figure 4-2. Effective Address Generation, Memory Reference Instructions

4.4.1 JUMP UNCONDITIONALLY (JMP)



The EA replaces the contents of register P. Program execution continues with the instruction at memory location EA.

32K Mode
EA₁₄₋₀ → P₁₄₋₀

64K Mode
EA₁₅₋₀ → P₁₅₋₀

Indicators affected: None
Registers affected: None

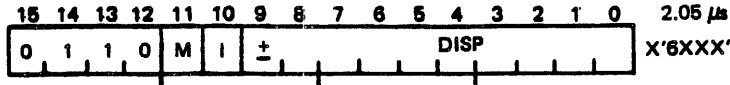
NOTE

This instruction is interruptable. Unlike the JMP instruction on a GA-16/440 or SPC-16, an interrupt may occur following the execution of the JMP instruction; therefore, the instruction at the EA could be pre-empted by an interrupt. Interrupt subroutine return will continue the program at the jumped-to-address.

Examples:

JMP	START	Program control is transferred to the instruction having the label START; the label must identify a memory location in the range -512 to +511 words from the JMP instruction location plus one.
JMP	*NEXT	Program control is transferred to the instruction whose address is contained in the memory location having the label NEXT. The instruction to be executed next may be located anywhere in memory.
	.	
	.	
	.	
NEXT	DC	START

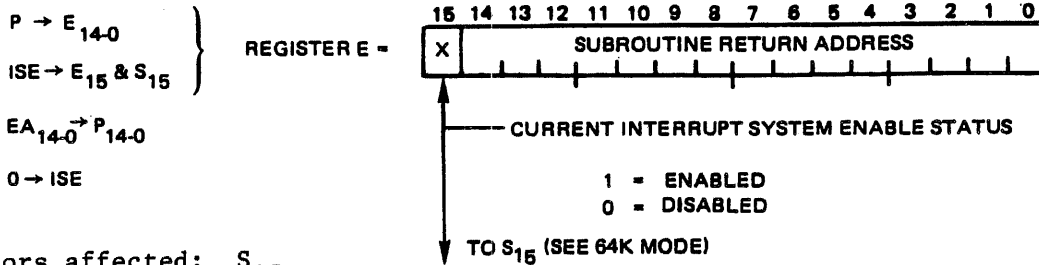
4.4.2 JUMP TO SUBROUTINE (JSR)



The JSR instruction is used to transfer control from a main program to a subroutine and disable the inhibitible (IN) interrupts (ISE = 0). The next instruction location (subroutine return address) and the ISE status of the main program are saved to resume main program execution sequence and ISE status after the subroutine executes an RTRN, E instruction.

32K Mode

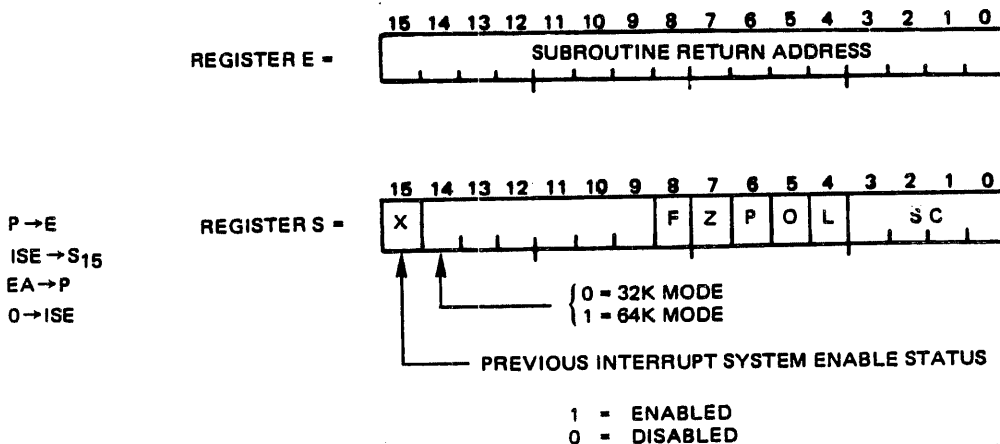
The contents of register P (containing the address of the next memory location) replaces the contents of register E, bits 14 to 0; the ISE status replaces bit 15 of register E and S₁₅. The EA then replaces the contents of register P and the ISE is turned off, disabling inhibitible interrupt requests.



Indicators affected: S₁₅
Registers affected: E

64K Mode

The contents of register P (containing the address of the next memory location) replaces the contents of register E. The ISE status replaces bit 15 of the status register. The EA then replaces the contents of register P and the ISE is turned off, disabling inhibitible interrupt requests.



Indicators affected: S₁₅
Registers affected: E

The contents of registers A,B,C,D,X,Y, and Z are not affected by the execution of a JSR; thus, parameters may be passed to the subroutine as the contents of these registers. (Other methods are shown in the examples in Section 4.5.13.)

The return from a subroutine is normally made using a RTRN E instruction (Section 4.10.11). The ISE of the calling program may be duplicated in the subroutine with the RISE E instruction (Section 4.10.9), or IN interrupts may be specifically enabled with INE instruction (Section 4.13.3). Otherwise, IN interrupts remain disabled during the execution of the subroutine.

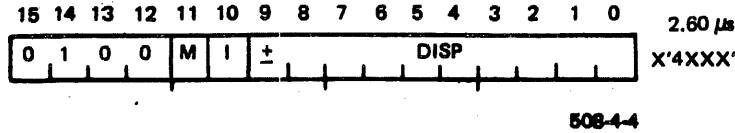
Since the address transferred to register E is $\$+1$, a return cannot be made properly from a subroutine called by a JSR instruction at the last location of memory.

When the CPU acknowledges an IN interrupt, it simulates a JSR instruction which indirectly references the memory location dedicated to the interrupting device. The address plus one of the last instruction executed before the interrupt occurred is stored in register E.

Examples: (32K Mode)

GO	JSR SUBRUT LDA MEAS	SUBRUT is a label in the program section. The address of the instruction labeled GO and the current interrupt status are stored in register E and S; the ISE is inhibited, and program control is transferred to location SUBRUT. Location SUBRUT must be in the range -512 to +511 words from the location of the instruction labeled GO (that is, $\$+1$).
	JSR *\$\$+1	The memory location given by $\$+1$ contains the address (anywhere in memory) of the subroutine. The address of the location following this JSR instruction and the current interrupt status are loaded into register E and S and the ISE is inhibited. Control is then transferred to the subroutine.

4.4.3 LOAD REGISTER A (LDA)



The contents of the location specified by the effective address replaces the contents of register A.

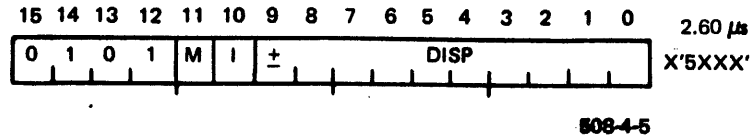
$(EA)_{15-0} \rightarrow A_{15-0}$

Indicators affected: None
Registers affected: A

Examples:

- LDA $\$-54$ The displacement is calculated as $(\$-54) - (\$+1) = -55$. Thus, the contents of location $\$-54$ (i.e., $\$+1-55$) is loaded into register A.
- LDA SMRT The contents of location SMRT are loaded into register A. If label SMRT is in the PSECT, it may identify any location in the range -512 to $+511$ locations from $\$+1$. If label SMRT is in the DSECT, it may identify any location up to 1023 words from the current base address.
- LDA 0,1 Base-relative addressing is specified. The displacement is 0, and the effective address is calculated as $D+0$; thus, the contents of the memory location pointed to by the current value in register D is loaded into register A.
- LDA *ADRS2 Indirect addressing is specified. ADRS2 may identify a location up to 1023 words from the base address or in the range -512 to $+511$ locations from $\$+1$. The contents of this location is used as an address, which may specify any location, and is loaded into register A.

4.4.4 STORE REGISTER A (STA)



The contents of register A replace the contents of the memory location specified by EA.

$$A_{15-0} \rightarrow (EA)_{15-0}$$

Indicators affected: None

Registers affected: None

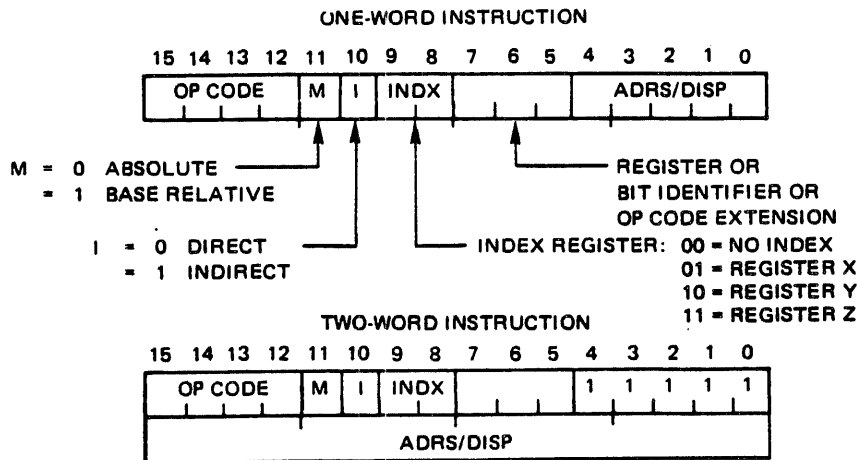
Example:

STA *VAL The contents of memory location VAL provide the address where the contents of register A will be stored. The final effective address (i.e., the contents at VAL) is taken as an absolute address and may specify any memory location.

4.5 MEMORY REFERENCE WITH INDEXING INSTRUCTIONS

The twelve instructions for memory reference with indexing provide the option of specifying an index as part of the address when referencing a memory location. Any word or byte of memory may be loaded to or stored from a specified register. Any bit in memory may be set, reset, or tested. Any word in memory may be incremented, decremented, or compared to the contents of any register. Instructions are also provided to load or store all registers and status, simplifying efficient, interruptable, re-entrant subroutine coding.

A memory reference with indexing instruction occupies one or two words, depending upon the value of bits 4 to 0 of the first word of the instruction. The one- and two-word formats are as follows:



486-4-21

The CAP-16 source statement format for this group is:

Command	Parameters
CMR ① DECM ② INCM ② LARS ② LDBY ① LDR ① RBIT ③ SARS ② SBIT ③ STBY ① STR ① TBIT ③	① register, [*]address[, index][, mode] ② [*]address[, index][, mode] ③ b, [*]address[, index][, mode]

where:

register specifies one of the general-purpose registers, coded into bits 7 to 5 for instructions which name a register (LDR, STR, CMR, LDBY, and STBY).

or

b specifies a bit position, 7 to 0, of the referenced byte in memory, coded into bits 7 to 5 for bit instructions (SBIT, RBIT, and TBIT).

or

register or b is absent for other instructions (LARS, SARS, INCM, and DECM), and bits 7 to 5 contain an operation code extension.

* if present, indicates indirect addressing (bit 10=1); if absent, indicates direct addressing (bit 10=0).

address is an address expression.

index if present, specifies one of the three index registers (bits 9 and 8):
= 01 register X; = 10 register Y; = 11 register Z. If absent, no index register is specified: = 00 no index.

mode indicates the addressing mode (absolute or base-relative) and instruction length as follows:

= 0 one-word instruction (bits 4 to 0 = ADRS) with absolute addressing (bit 11=0).

= 1 one-word instruction (bits 4 to 0 = DISP) with base-relative addressing (bit 11=1).

= 2 two-word instruction (first word bits 4 to 0 = X'1F' and second word bits 15 to 0 = ADRS) with absolute addressing (bit 11=0).

= 3 two-word instruction (first word bits 4 to 0 = X'1F' and second word bits 15 to 0 = DISP) with base-relative addressing (bit 11=1).

NOTE

If the computer is in the 32K mode, bit 15 of EA is forced to 0.

ADRS and DISP are determined as follows:

Absolute ADRS: address → bits 4 to 0 (or 15 to 0 of second word)

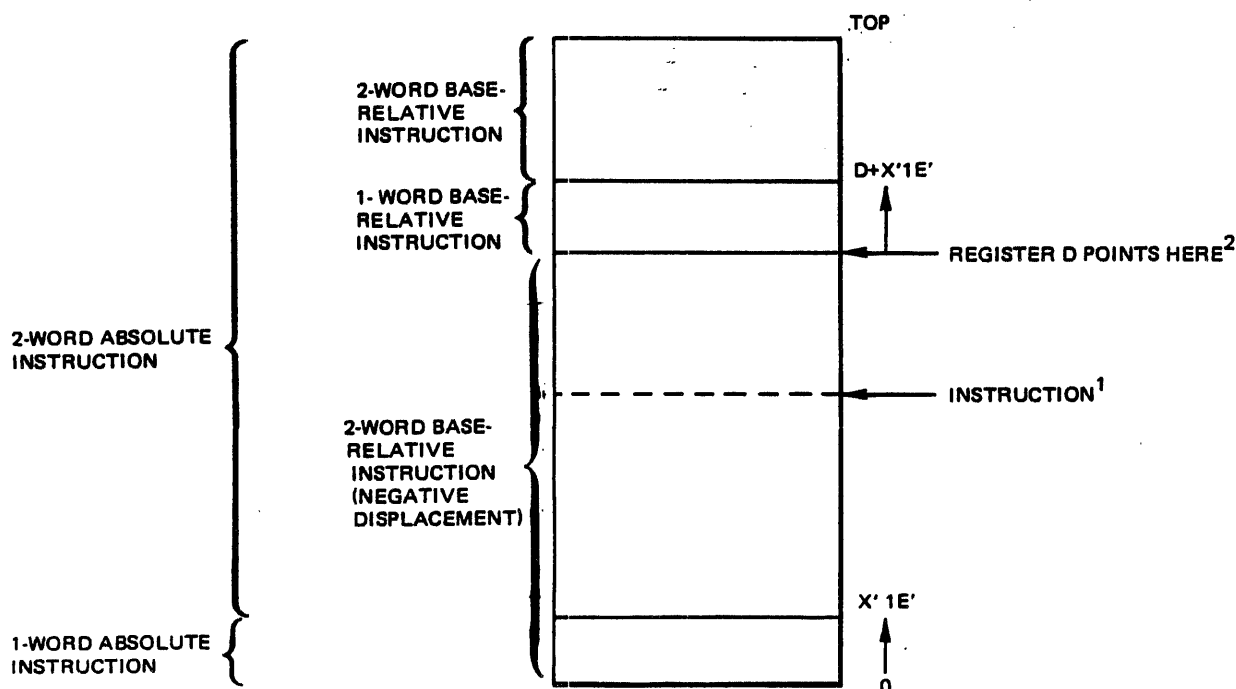
Base Relative DISP: if mode is not specified, (address - \$\$) → bits 4 to 0 (or 15 to 0 of second word)

Base Relative DISP: if mode is specified, address → bits 4 to 0 (or 15 to 0 of second word)

NOTE

If the programmer wishes to specify the addressing mode and the address contains a relocatable label value or a label value outside the range $0-30_{10}$, a two-word instruction (mode=2 or 3) must be specified.

Addressing ranges are illustrated below. Note that for indirect addresses, the indirectly addressed location must be in the range illustrated below, but the retrieved address may point to any location in memory.



¹ MEMORY REFERENCE WITH INDEXING INSTRUCTION LOCATED ANYWHERE IN MEMORY.

² ADDRESS WRAPAROUND OR ADDRESS OUTSIDE MEMORY MAY OCCUR IF EA CALCULATION PLACES ADDRESS RANGE ABOVE TOP (SECTION 4.3.4 AND 4.3.5).

466-4-22

Effective address coding is summarized in Table 4-3.

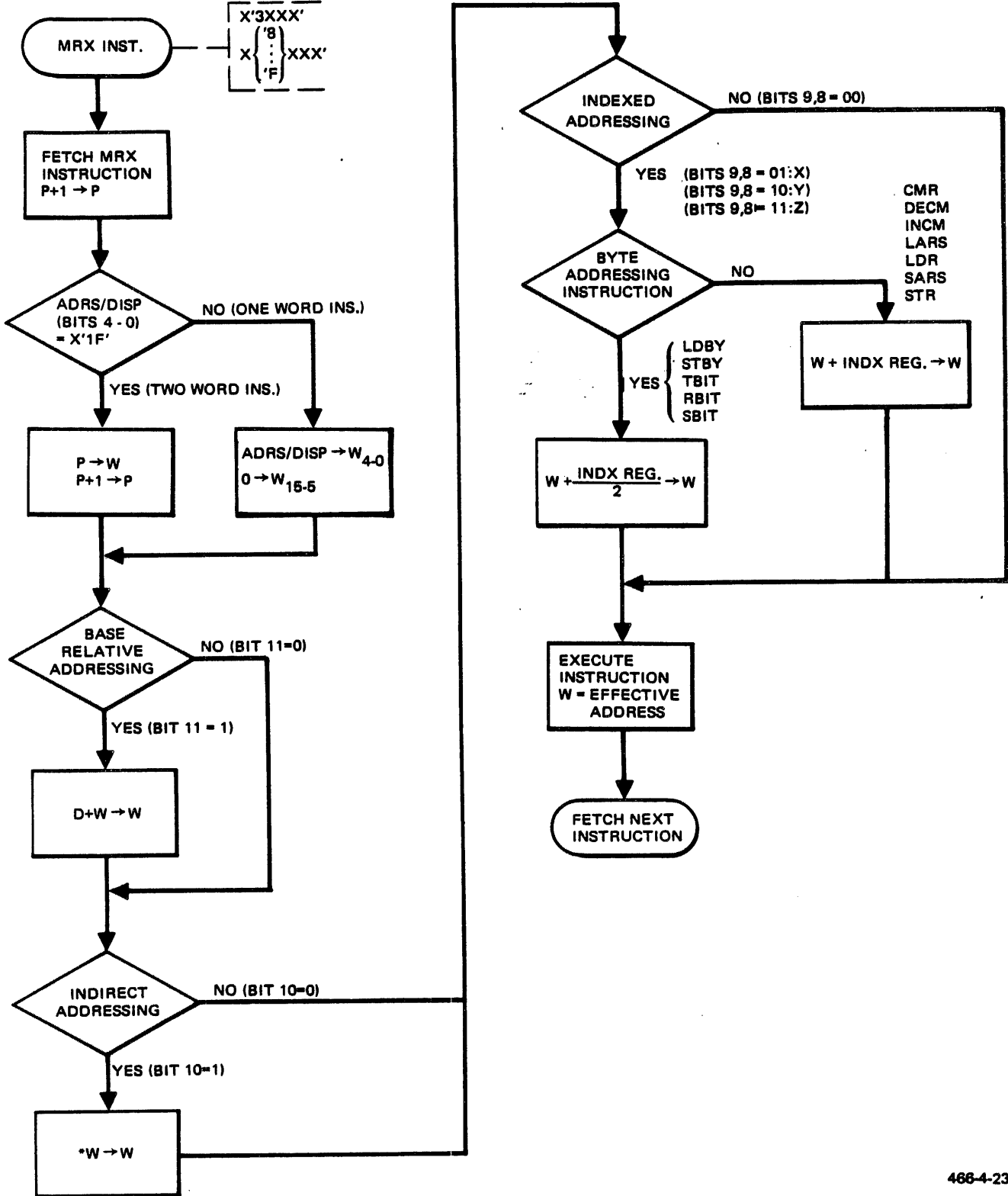
Table 4-3. Coding of Effective Address - Memory Reference with Indexing Instructions

Absolute	M	I	Indx	Adrs	EA ^①	Next Instruction	Additional Execution Time (μs) ^②
Direct	0	0	0	<1F	ADRS	P+1	0.0
Direct, Indexed	0	0	1-3	<1F	ADRS+INDX	P+1	0.0
Direct	0	0	0	=1F	(P+1)	P+2	0.500
Direct, Indexed	0	0	1-3	=1F	(P+1)+INDX	P+2	0.500
Indirect	0	1	0	<1F	(ADRS)	P+1	0.500
Indirect, Indexed	0	1	1-3	<1F	(ADRS)+INDX	P+1	0.500
Indirect	0	1	0	=1F	((P+1))	P+2	1.000
Indirect, Indexed	0	1	1-3	=1F	((P+1))+INDX	P+2	1.000
<u>Base-Relative</u>				<u>DISP</u>			
Direct	1	0	0	<1F	D+DISP	P+1	0.0
Direct, Indexed	1	0	1-3	<1F	D+DISP+INDX	P+1	0.0
Direct	1	0	0	=1F	D+(P+1)	P+2	0.500
Direct, Indexed	1	0	1-3	=1F	D+(P+1)+INDX	P+2	0.500
Indirect	1	1	0	<1F	(D+DISP)	P+1	0.500
Indirect, Indexed	1	1	1-3	<1F	(D+DISP)+INDX	P+1	0.500
Indirect	1	1	0	=1F	(D+(P+1))	P+2	1.000
Indirect, Indexed	1	1	1-3	=1F	(D+(P+1))+INDX	P+2	1.000

① INDX/2 for LDBY, STBY, SBIT, RBIT and TBIT instructions; indexing range is limited to 32K.

② Instruction cycle time plus additional execution time equals total processor time for instruction.

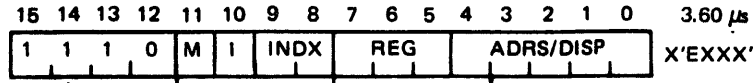
Upon execution of a memory reference with indexing instruction, the effective address is generated as shown in Figure 4-3.



466-4-23

Figure 4-3. Effective Address Generation, Memory Reference with Indexing Instructions

4.5.1 COMPARE MEMORY WITH REGISTER (CMR)



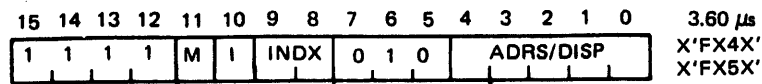
The contents of the location, as specified by the EA, are subtracted from the contents of the selected register, with the result of the subtraction being recorded only in the indicators. The zero indicator is set if the result of the subtraction is zero, and the plus indicator is set if the result is positive. An overflow occurs if the sign of the register and the sign of the contents of the EA are unlike, with the sign of the result also different from the sign of the register contents. The link will be set if there was a carry-out of bit 15.

$$R_{15-0} - (EA)_{15-0} \rightarrow \text{Data Bus}$$

Indicators affected: Zero, Plus, Overflow, Link
 Registers affected: None

Example: CMR A,10,,1 The contents of location D+10 are subtracted from the contents of register A, with the results appearing in the indicators only.

4.5.2 DECREMENT MEMORY (DECM)



508-4-6

The contents of the location specified by the EA is decremented by 1.

$$(EA)_{15-0}^{-1} \rightarrow (EA)_{15-0}$$

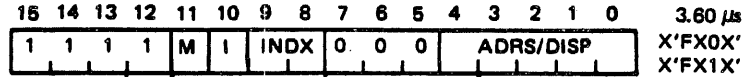
Indicators affected: Zero, Plus, Link (Overflow is not affected.)
 Registers affected: None

Example: DECM 0,X An absolute address is specified with indexing. Since the contents of register X are added to zero, register X contains the absolute address of the location to be referenced. The specified memory location is decremented.

If, in the above, example, the number contained in register X were, instead, contained with register D, the following instruction (which specifies base-relative addressing) would decrement the same memory location.

DECM 0,,1

4.5.3 INCREMENT MEMORY (INCM)



The contents of the location specified by the EA is incremented by 1.

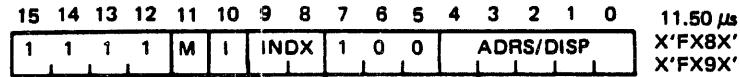
$$(EA)_{15-0} + 1 \rightarrow (EA)_{15-0}$$

Indicators affected: Zero, Plus, Link (Overflow is not affected.)

Registers affected: None

Example: INCM ADR+4 The contents of memory location ADR+4 are incremented.

4.5.4 LOAD ALL REGISTERS AND STATUS (LARS)



508-4-7

Registers A,X,Y,Z,B,C,D, and E, and the shift counter and indicators (register S) are loaded from the contents of nine consecutive locations specified by EA through EA+8. Bit 14 of register S (memory mode) is not loaded. Bit 15 of register S is loaded, but ISE itself is unaffected.

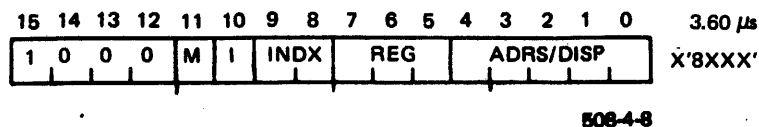
- | | | |
|--|--|--------------|
| (EA) ₁₅₋₀ → A ₁₅₋₀ | (EA+8) ₃₋₀ → Shift Counter | } Register S |
| (EA+1) ₁₅₋₀ → X ₁₅₋₀ | (EA+8) ₄ → Link | |
| (EA+2) ₁₅₋₀ → Y ₁₅₋₀ | (EA+8) ₅ → Overflow | |
| (EA+3) ₁₅₋₀ → Z ₁₅₋₀ | (EA+8) ₆ → Plus | |
| (EA+4) ₁₅₋₀ → B ₁₅₋₀ | (EA+8) ₇ → Zero | |
| (EA+5) ₁₅₋₀ → C ₁₅₋₀ | (EA+8) ₈ → Foreground | |
| (EA+6) ₁₅₋₀ → D ₁₅₋₀ | (EA+8) ₁₃₋₉ → S ₁₃₋₉ | |
| (EA+7) ₁₅₋₀ → E ₁₅₋₀ | (EA+8) ₁₅ → ISE Save | |

Indicators affected: Zero, Plus, Overflow, Link, Foreground, ISE Save

Registers affected: A,X,Y,Z,B,C,D,E

Example: LARS *STACK The registers and status indicators are loaded from nine consecutive memory locations; the address of the first location is contained at STACK (i.e., STACK is an indirect address).

4.5.5 LOAD BYTE (LDBY)



If an index register is specified, the index register divided by two is used to determine the EA. If the contents of the selected index register is odd, the right byte (bits 7 to 0) of the location specified by the EA replaces bits 7 to 0 of the selected register. If the contents of the selected index register are even or no index register is specified, the left byte (bits 15 to 8) of the location specified by the EA replace bits 7 to 0 of the selected register. In either case, bits 15 to 8 of the selected register are not affected.

If contents of index register are odd:

$$(EA)_{7-0} \rightarrow R_{7-0}$$

If contents of index register are even or no index register specified:

$$(EA)_{15-8} \rightarrow R_{7-0}$$

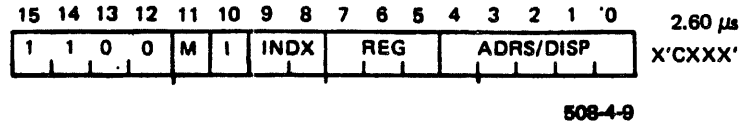
Indicators affected: None

Registers affected: Bits 7 to 0 of selected register

Example:

LDBY A,ADDR4,X If ADDR4 is a label in the DSECT, addressing is direct and base-relative with indexing. The displacement is ADDR4-\$\$\$. The EA is calculated as the contents of register D plus the displacement plus the contents of register X divided by 2. The contents of this location (bits 7 to 0 if X is odd or bits 15 to 8 if X is even) are loaded into register A, bits 7 to 0.

4.5.6 LOAD REGISTER (LDR)



The contents of the location specified by EA replaces the contents of the selected register.

$(EA)_{15-0} \rightarrow R_{15-0}$

Indicators affected: None

Registers affected: Selected register

- Examples:
- | | | |
|-----|------------|---|
| LDR | A,0 | Addressing is direct and absolute. The instruction is contained in a single word. The contents of memory location 0 are loaded into register A. |
| LDR | B,*0 | Addressing is indirect and absolute. The instruction is contained in a single word. The contents of memory location 0 are used as an address, and the contents of this addressed location are loaded into register B. |
| LDR | A,*30,X | Addressing is indirect and absolute with indexing, and the instruction is contained in a single word. The contents of memory location 30_{10} are used as an address; the value contained in register X is added to this address to specify the location whose contents are loaded into register A. |
| LDR | Z,*20,,1 | Addressing is indirect and base-relative. Since the first-stage address is within 31 locations of the base address, a one-word instruction is used where mode is specified. For either instruction, location $$$+20$ contains the address or the location whose contents are loaded into register Z. |
| LDR | Z,*\$\$+20 | |
| LDR | B,ADRS,Y,2 | Addressing is direct and absolute with indexing. The contents of the absolute memory location, whose address is the offset from ADRS by the contents of register Y, are loaded into register B. ADRS may identify any memory location, since a two-word instruction is specified, and may be relocatable. |

LDR C,*X'1000',X,3
 or
 LDR C,*\$\$+X'1000',X

Addressing is indirect and base-relative with indexing. Either of these source statements will cause a two-word instruction to be generated; the second word will contain X'1000'. When this instruction is executed, the effective address is generated as follows:

1. Add second word to base address:

$$1000_{16} + D = EA_1$$

2. Get address at indirect EA:

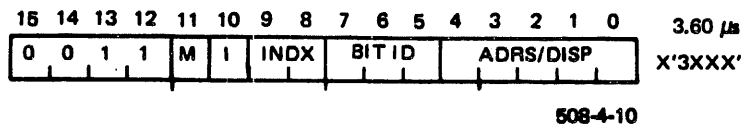
$$(EA_1) = EA_2$$

3. Add index:

$$EA_2 + X = \text{Final EA}$$

The contents of the location specified by the final EA replaces the contents of register C.

4.5.7 RESET BIT (RBIT)



A zero is placed in the specified bit of the location determined by the EA. The previous status (zero, non-zero) of the specified bit is shown in the zero indicator. This is done by complementing the bit, so that:

If bit was 1, zero indicator = 0 for non-zero status
 If bit was 0, zero indicator = 1 for zero status

If the address is to be indexed, the index register divided by two is used in determining the EA. If the contents of the selected index register are odd, the specified bit (7 to 0) of the right byte is reset. If the contents of the selected index register is even, or if no index register is selected, the specified bit (7 to 0) of the left byte is reset.

If index is odd:

$\overline{(EA)}_b \rightarrow$ Zero Indicator
 0 $\rightarrow (EA)_b$

If index is even (or no index):

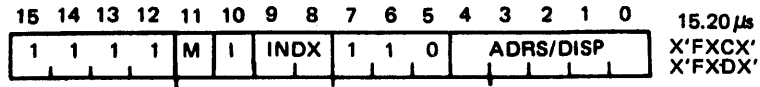
$\overline{(EA)}_{b+8} \rightarrow$ Zero Indicator
 0 $\rightarrow (EA)_{b+8}$

Indicators affected: Zero
 Registers affected: None

Example: RBIT 2,4,Y,3

The beginning address of the bit array is calculated as the contents of register D plus 4, and the value of register Y divided by two, identifies the word element in which the specified bit is contained. If the contents of register Y are odd, bit 2 at the EA is reset; if the contents of register Y are even, bit 10 (i.e., bit 2 of the left byte) at the EA is reset. The zero indicator will contain zero if the specified bit was one before the operation, or will contain one if the specified bit was zero before the operation.

4.5.8 STORE ALL REGISTERS AND STATUS (SARS)



508-4-11

Registers A,X,Y,Z,B,C,D, and E, and the shift counter and status indicators (register S) replace the contents of nine consecutive memory locations specified by EA through EA+8.

A ₁₅₋₀	→ (EA) ₁₅₋₀	} Register S	Shift Counter	→ (EA+8) ₃₋₀
X ₁₅₋₀	→ (EA+1) ₁₅₋₀		Link	→ (EA+8) ₄
Y ₁₅₋₀	→ (EA+2) ₁₅₋₀		Overflow	→ (EA+8) ₅
Z ₁₅₋₀	→ (EA+3) ₁₅₋₀		Plus	→ (EA+8) ₆
B ₁₅₋₀	→ (EA+4) ₁₅₋₀		Zero	→ (EA+8) ₇
C ₁₅₋₀	→ (EA+5) ₁₅₋₀		Foreground	→ (EA+8) ₈
D ₁₅₋₀	→ (EA+6) ₁₅₋₀		S ₁₃₋₉	→ (EA+8) ₁₃₋₉
E ₁₅₋₀	→ (EA+7) ₁₅₋₀		32/64K MEM Mode	→ (EA+8) ₁₄
			ISE, Save Status	→ (EA+8) ₁₅

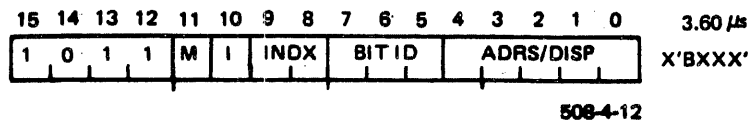
Indicators affected: None
 Registers affected: None

Example:

SARS *0,,0

The contents of all registers and indicators are stored into nine consecutive memory locations. The address of the first memory location is contained in location 0 (i.e., 0 is an indirect address).

4.5.9 SET BIT (SBIT)



A one is placed in the specified bit of the location determined by the EA. The previous status (zero, non-zero) of the specified bit is shown in the zero indicator. This is done by complementing the bit, so that:

if bit was 1, zero indicator = 0 for non-zero status
 if bit was 0, zero indicator = 1 for zero status

If the address is to be indexed, the index register, divided by two, is used in determining the effective address. If the contents of the selected index register are odd, the specified bit (7 to 0) of the right byte is set. If the contents of the selected index register are even, or if no index register is selected, the specified bit (7 to 0) of the left byte is set.

If index is odd:

$(\overline{EA})_b \rightarrow$ Zero Indicator

$1 \rightarrow (EA)_b$

If index is even (or no index):

$(\overline{EA})_{b+8} \rightarrow$ Zero Indicator

$1 \rightarrow (EA)_{b+8}$

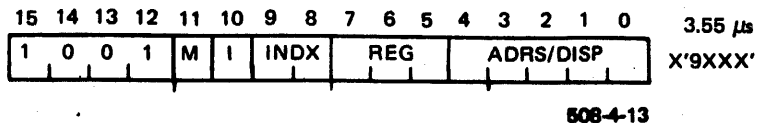
Indicators affected: Zero
 Registers affected: None

Example:

SBIT 4,BITS,Z

BITS identifies the beginning address of a bit array, and the value of register Z, divided by two, identifies the word that contains the referenced bit. If the contents of register Z is odd, bit 4 at the EA is set. If the contents of register Z is even, bit 12 (i.e., bit 4 of the left byte) is set. The zero indicator will contain zero if the specified bit was one before the operation, or will contain one if the specified bit was zero before the operation.

4.5.10 STORE BYTE (STBY)



If an index register is specified, the index register, divided by two, is used to determine the EA. If the contents of the selected index register are odd, bits 7 to 0 of the selected register replace the right byte (bits 7 to 0) of the location specified by the EA. If the contents of the selected register are even, or if no index register is specified, bits 7 to 0 of the selected register replace the left byte (bits 15 to 8) of the location as specified by the EA.

If contents of selected index register are odd:

$$R_{7-0} \rightarrow (EA)_{7-0}$$

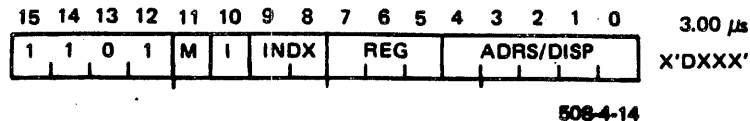
If contents of index register are even or if no index register is specified:

$$R_{7-0} \rightarrow (EA)_{15-8}$$

Indicators affected: None

Registers affected: None

4.5.11 STORE REGISTER (STR)



The contents of the selected register replace the contents of the location specified by the EA.

$$R_{15-0} \rightarrow (EA)_{15-0}$$

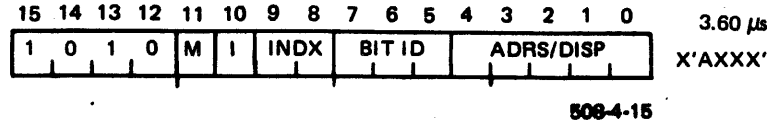
Indicators affected: None
 Registers affected: None

Example:

STR A,LOC

Where $LOC \geq X'1F'$ and is in the DSECT. Addressing is direct and base-relative; a 2-word instruction is generated. Location LOC may be any forward reference from the base address. (Although the programmer will rarely use the capability, the location could also be any reverse reference from the base address, since the second word of the instruction may specify a negative displacement in two's complement form.) When this instruction is executed, the contents of register A are stored at location LOC.

4.5.12 TEST BIT (TBIT)



The complement of the specified bit in the location determined by the EA is loaded into the zero indicator to indicate the status of the bit. If an index register is specified, the register, divided by two, is used in determining the EA. If the contents of the selected index register is odd, the status of the specified bit (7 to 0) of the right byte is reflected in the zero indicator. If the contents of the selected index register is even (or if no register was selected), the status of the specified bit (7 to 0) of the left byte is reflected in the zero indicator.

If index is odd:

$\overline{(EA)}_b \rightarrow$ Zero Indicator

If index is even (or no index):

$\overline{(EA)}_{b+8} \rightarrow$ Zero Indicator

Indicators affected: Zero
Registers affected: None

Example:

TBIT 3,OME,Y

The EA is the address OME, plus the contents of register Y, divided by two. If register Y is odd, the status of bit 3 is placed into the zero indicator. If register Y is even, the status of bit 11 (bit 3 of the left byte) is placed into the zero indicator. The contents of the memory location addressed is not changed.

4.5.13 PROGRAMMING EXAMPLES

1. Beginning at the location pointed to by the base address, this instruction sequence performs an operation (not defined in the example) on consecutive elements of an array. When a test for exit is successful, the routine exits.

```

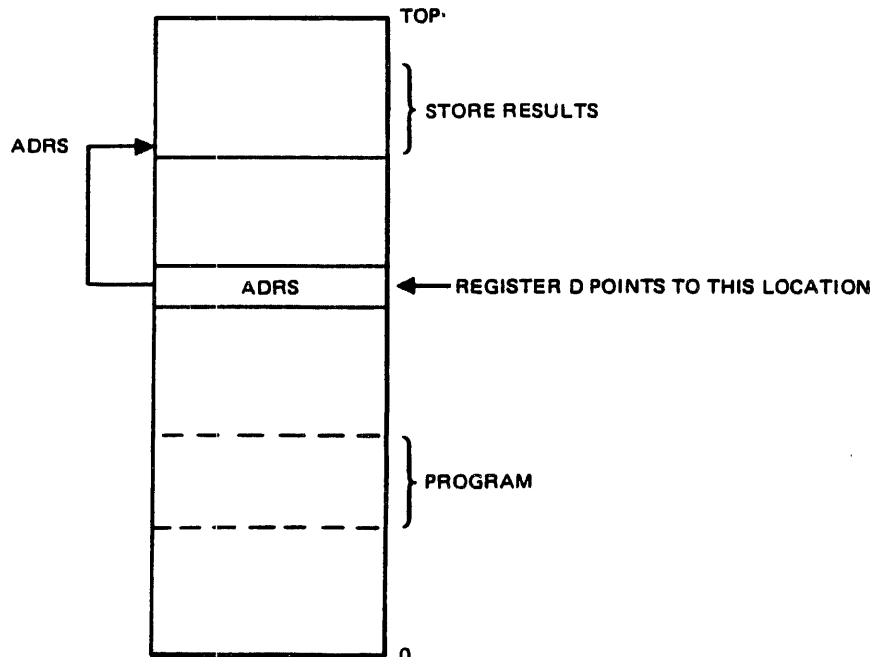
ZERO    Y           Zero register Y
LOOP    LDR    A,$$,Y   Load element from location $$+Y.
        .           (Perform operations and test for exit.)
        INCR   Y           Increment register Y.
        JMP    LOOP      Jump to load next element.
    
```

2. This sequence loads two-word elements of an array, placing the first word into register B and the second word into register C. The beginning address of the array is accessed indirectly. The indirect address is the contents of the memory location at the base address. Upon execution of the LDR instruction, the indirect address D+0 contains the address of the beginning of the array. Then the value of the index (register Y) is added to locate the array element.

```

ZERO    Y           Zero the index register to begin.
LOOP    LDR    B,*0,Y,1  Load register B with the first word.
        INCR   Y           Increment index.
        LDR    C,*0,Y,1  Load register C with the second word.
        .           (Perform operations and test for exit.)
        INCR   Y           Increment to the next first word.
        JMP    LOOP      Jump to load the next first word.
    
```

3. The results of a number of computations are to be stored sequentially, beginning at a memory location which is accessed indirectly. This is illustrated as follows:



Main Program:

```

ZERO    X           Zero index to start STORE at beginning of STORE array.
:
:           Do first computation, leaving result in register A.
JSR     STORE       Call subroutine STORE.
:
:           Do second computation, leaving result in register A.
JSR     STORE       Call subroutine STORE.
:
:           (Continue computations, calling STORE for each result.)

```

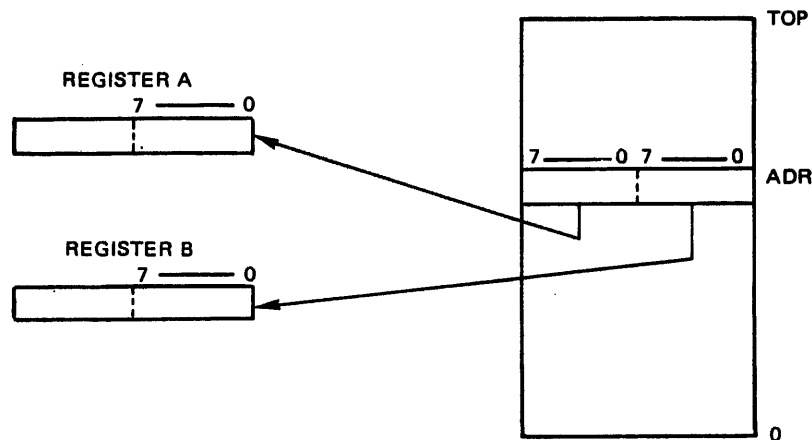
Subroutine:

```

STORE   STR    A,*O,X,1  Store result in register A.
        INCR   X           Increment index.
        RTRN   E           Return to main program.

```

4. This sequence loads the left byte of location ADR into register A, then loads the right byte of the same location into register B.



243-3-23

```

ZERO    X           Zero index.
LDBY    A,ADR[,X] 1 Load left byte.
INCR    X           Increment index.
LDBY    B,ADR,X     Load right byte.

```

¹Has same effect if omitted

5. This sequence stores bytes into consecutive byte locations. The beginning address of the byte array is located indirectly through the base address.

```

LOOP    ZERO    X           (Perform computation, leaving result in right byte
:                                     of register A.)
:                                     Store byte result.
STBY    A,*$$,X
INCR    X           Increment index.
:                                     (Test for exit.)
JMP     LOOP       Loop to compute next byte.

```

6. A subroutine may use the instruction SARS to start and LARS prior to returning, then perform any computations using registers/indicators without interfering with the contents/status set by the calling program.

Calling program:

```

      JSR    SUBR
      .
      .

```

Subroutine:

```

SUBR  SARS    SAVER    Store all registers/status in a buffer (9 words)
                        called SAVER.
      .
      .              (Perform computations.)
      LARS    SAVER    Restore calling program's registers/status.
      RTRN    E        Return to calling program.

```

7. This sequence shows one method of passing parameters to a subroutine. The beginning address of the parameter list (i.e., the address of the first parameter) is put in the memory location immediately following the JSR instruction. Upon executing the JSR, the address of this location is stored in the E register as the return address. The subroutine transfers this address to the D register so that the subroutine's base address points to the memory location immediately following the JSR instruction in the main program. By indirectly addressing this location, with base relative mode, the subroutine obtains the beginning address of the parameter list, and adds an index to this indirect address to fetch the individual parameters. To return to the calling program, register E is incremented by one to point to the next executable instruction in the main program. Then a RTRN returns to the main program.

Main program:

```

      JSR    SUBR
      DC     LIST    Define beginning location of parameters.
HOH   .
      .

```

Subroutine:

```

SUBR  SARS    *SAVEP   Store register/status in nine locations beginning
                        with adrs in SAVEP.
      RTR    D,E      Change base adrs to Main return adrs.
      ZERO   Y        Zero index.
LOOP  LDR     A,*0,Y,1 Load first parameter.
      .
      .              (Perform operations and test for jump to EXIT.)
      INCR   Y        Increment index.
      JMP    LOOP     Jump to load next parameter.
EXIT  LARS    *SAVEP   Restore original register contents.
      INCR   E        Increment E to return at location HOH.
      RTRN   E        Return to main program at HOH.

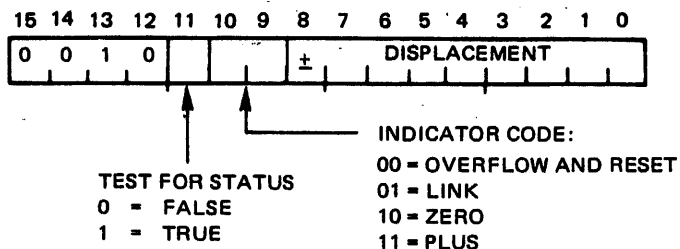
```

4.6 CONDITIONAL JUMP INSTRUCTIONS (SKIP)

Each Conditional Jump instruction tests one of the indicators (Zero, Plus, Overflow, or Link) for either a true (1) or false (0) state. If the condition being tested is met, program control is transferred to the location specified by the instruction; if the condition is not met, execution continues with the next instruction in the program sequence.

Testing the Overflow indicator also resets it; testing the other indicators does not affect their status.

There are eight instructions in this group; each has the following machine format:



508-4-17

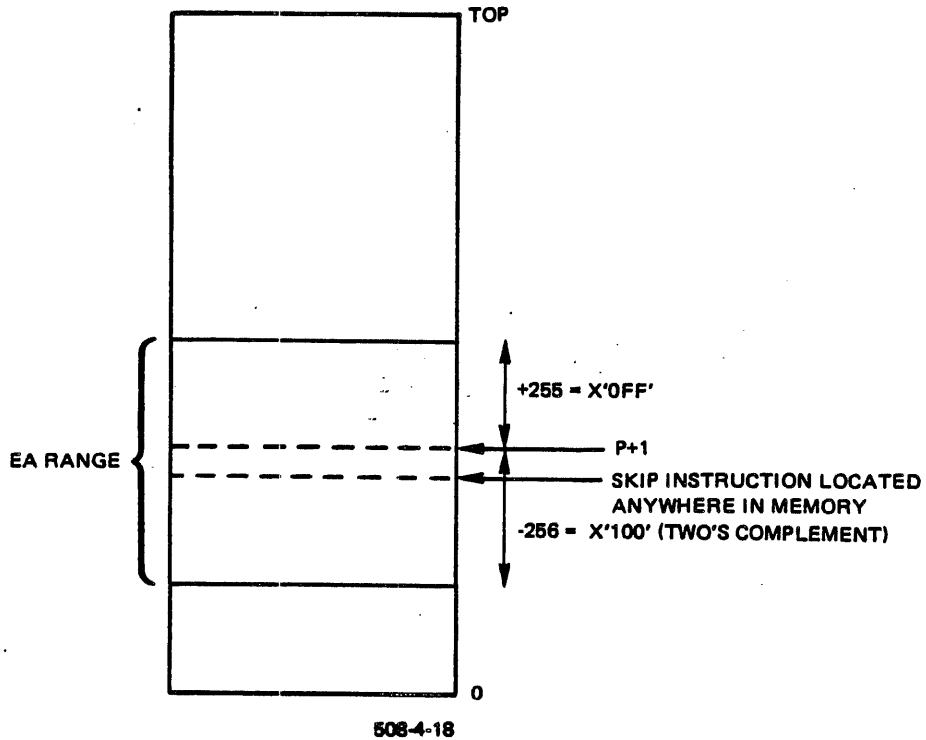
The CAP-16 source-statement format for this instruction group is as follows:

<u>Command</u>	<u>Parameters</u>
SKM SKN SKOF SKOT SKP SKR SKS SKZ	address expression

The addressing mode for this instruction group is always program-relative and direct. The displacement field of the skip instruction is a signed 9-bit value. If the skip is forward (to a higher address) the displacement field will contain a positive value (with bit 8=0) equal to the difference between the EA and the instruction location +1 (P+1). If the skip is reverse (to a lower address) the displacement field will contain the two's complement (with bit 8=1) of the difference between the EA and P+1.

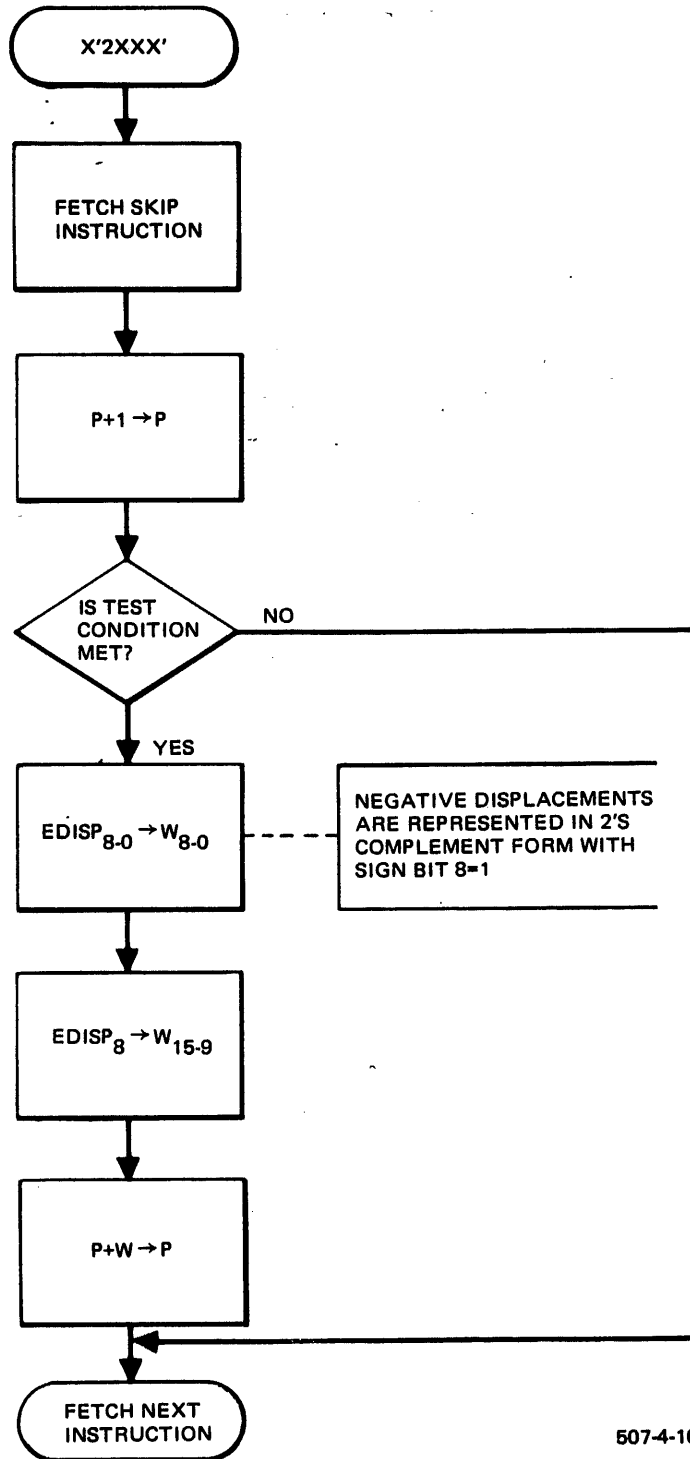
The effective address determines where control is to be transferred if the status condition (bit 11) of the selected indicator (bits 10 and 9) is met.

The addressing range is illustrated below:



Address wraparound or out-of-memory condition may occur if the instruction location permits EA range to exceed Top or go below 0 (Sections 4.3.4 and 4.3.5).

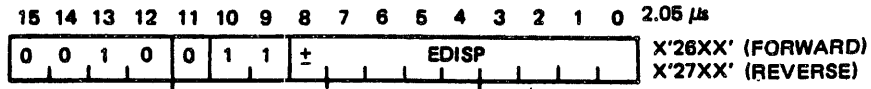
The effective address is generated using the signed displacement bits 8 to 0. Bit 8 is extended to bits 15 through 9 and added to P+1 to compute the effective address. Effective address generation is diagrammed in Figure 4-4.



507-4-16

Figure 4-4. Effective Address Generation, Conditional Jump Instructions (SKIP)

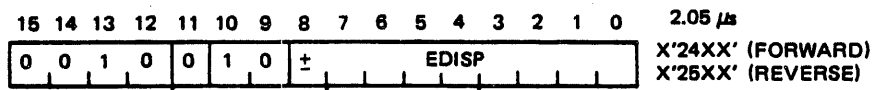
4.6.1 SKIP IF MINUS (SKM)



If the plus indicator is not set (negative condition exists), control is transferred to the EA. If the plus indicator is set (positive condition exists), execution continues with the next instruction.

Indicators affected: None
Registers affected: None

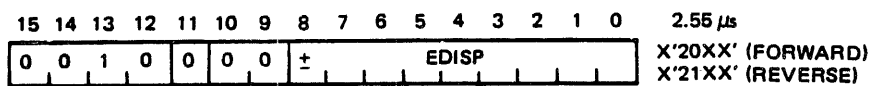
4.6.2 SKIP IF NOT ZERO (SKN)



If the zero indicator is not set (non-zero condition exists), control is transferred to the EA. If the zero indicator is set (zero condition exists), execution continues with the next instruction.

Indicators affected: None
Registers affected: None

4.6.3 SKIP IF OVERFLOW FALSE (SKOF)

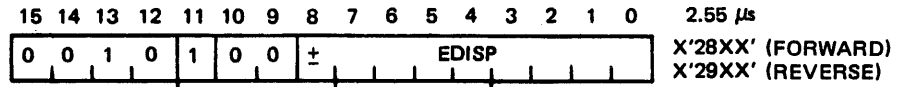


508-4-19

If the overflow indicator is not set (no overflow condition), control is transferred to the EA. If the overflow indicator is set (overflow condition exists), execution continues with the next instruction. The overflow indicator is reset.

Indicators affected: 0 → Overflow
Registers affected: None

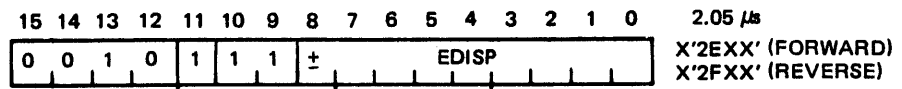
4.6.4 SKIP IF OVERFLOW TRUE (SKOT)



If the overflow indicator is set (overflow condition exists), control is transferred to the EA. If the overflow indicator is not set (no overflow condition) execution continues with the next instruction. The overflow indicator is reset.

Indicators affected: 0 \rightarrow Overflow
Registers affected: None

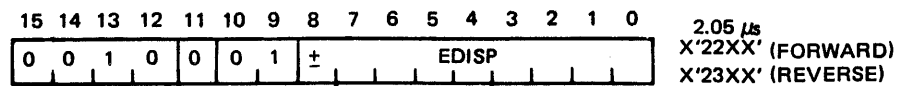
4.6.5 SKIP IF PLUS (SKP)



If the plus indicator is set (positive condition exists), control is transferred to the EA. If the plus indicator is not set (negative condition exists), execution continues with the next instruction.

Indicators affected: None
Registers affected: None

4.6.6 SKIP IF LINK RESET (SKR)

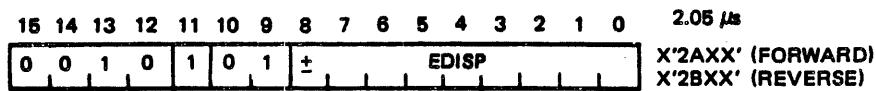


508-4-20

If the link indicator is not set (=0), control is transferred to the EA. If the link indicator is set (=1), execution continues with the next instruction.

Indicators affected: None
Registers affected: None

4.6.7 SKIP IF LINK SET (SKS)

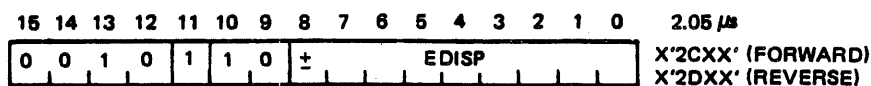


If the link indicator is set (=1), control is transferred to the EA. If the link indicator is not set (=0), execution continues with the next instruction.

Indicators affected: None

Registers affected: None

4.6.8 SKIP IF ZERO (SKZ)



508-4-21

If the zero indicator is set (zero condition exists), control is transferred to the EA. If the zero indicator is not set (non-zero condition exists), execution continues with the next instruction.

Indicators affected: None

Registers affected: None

Example:

SKZ START

The zero indicator is tested; if it is set (=1, indicating a zero condition), control is transferred to location START. If it is not set (=0, indicating a non-zero condition), program execution continues with the next instruction. START may be any location in the range -256 to +255 words from \$+1.

4.6.9 PROGRAMMING EXAMPLES

The elements of a byte array belong to one of two groups, identified by a flag in bit 7 (Group A=0, Group B=1). A computation is to be performed using bytes in Group A only. Bit 7 of each byte is tested, with the result placed into the zero indicator:

Group A — If bit 7=0, zero indicator is set (=1)
 Group B — If bit 7=1, zero indicator is reset (=0)

Base-relative, indirect addressing is specified for the TBIT instruction, with the byte array accessed indirectly through the base address location.

	ZERO	X	Zero byte index to begin.
LOOP	TBIT	7,*0,X,1	Test bit 7.
	SKN	NEXT	If Group B byte, go on to next byte.
	LDBY	C,*0,X,1	Load Group A byte into register C.
	:		(Perform computations.)
NEXT	:		(Test for done.)
	INCR	X	Increment index for next byte
	JMP	LOOP	Jump to test next byte.

4.7 REGISTER OPERATE and REGISTER OPERATE COMPARE INSTRUCTIONS

Register operate and register operate compare instructions are all one-word, single-cycle, register-to-register operations.

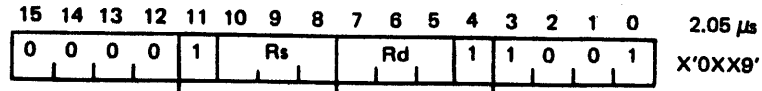
Register operate instructions may be used to:

1. Transfer the contents of one register to another.
2. Add or subtract the contents of two registers.
3. Perform one of logical operations AND, XOR, or OR with the contents of two registers.

The results of a register operate instruction are also reflected in the indicators.

The register operate compare instructions perform the same arithmetic and logical operations without changing register contents; the results appear in the indicators only.

4.7.1 ADD REGISTERS (ADD)



The contents of the source register are added to the contents of the destination register. The destination register retains the sum; the contents of the source register remain unchanged.

$$Rd_{15-0} + Rs_{15-0} \rightarrow Rd_{15-0}$$

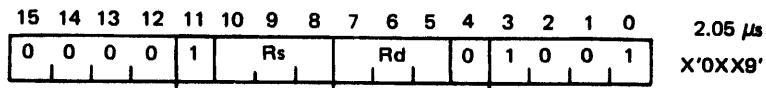
Indicators affected: Zero, Plus, Overflow, Link
 Registers affected: Destination Register

Example:

ADD A,B

The contents of register B are added to the contents of register A; the result is placed in register A. The contents of register B remain unchanged.

4.7.2 ADD COMPARE REGISTERS (ADDC)



508-4-22

The contents of the source register and the contents of the destination register are added together, and the results are reflected in the indicators. Neither the source nor destination register is changed.

$$Rd_{15-0} + Rs_{15-0} \rightarrow \text{Data Bus}$$

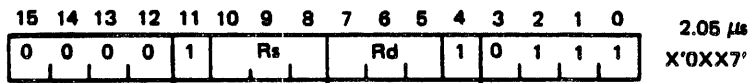
Indicators affected: Zero, Plus, Overflow, Link
 Registers affected: None

Example:

ADDC A,B

The contents of register A are added to the contents of register B. The results are reflected in the zero, plus, overflow and link indicators.

4.7.3 AND REGISTERS (AND)



The contents of the source register are logically ANDed (\wedge symbolizes the AND function) with the contents of the destination register and the result replaces the contents of the destination register.

$$Rd_{15-0} \wedge Rs_{15-0} \rightarrow Rd_{15-0}$$

Indicators affected: Zero, Plus

Registers affected: Destination Register

Example:

AND A,B

If:

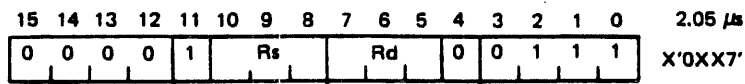
A = 0 ... 01010

B = 0 ... 01100

Result = 0 ... 01000

The result replaces the contents of register A. The contents of register B are unchanged.

4.7.4 AND COMPARE REGISTERS (ANDC)



508-4-23

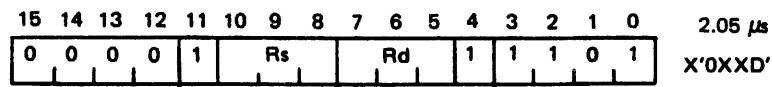
The contents of the source register are logically ANDed with the contents of the destination register, and the results are reflected in the indicators. Neither the source nor the destination register is changed.

$$Rd_{15-0} \wedge Rs_{15-0} \rightarrow \text{Data Bus}$$

Indicators affected: Zero, Plus

Registers affected: None

4.7.5 OR REGISTERS (OR)



The contents of the source register are logically ORed (V symbolizes the OR function) with the contents of the destination register and the result replaces the contents of the destination register.

$$Rd_{15-0} \vee Rs_{15-0} \rightarrow Rd_{15-0}$$

Indicators affected: Zero, Plus
 Registers affected: Destination Register

Example:

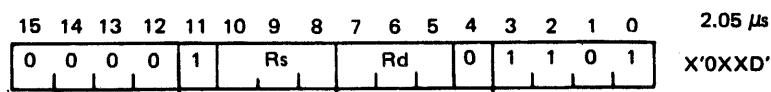
OR A,B

If:

A = 0 ... 01010
 B = 0 ... 01100
 Result = 0 ... 01110

The result replaces the contents of register A. Register B is unchanged.

4.7.6 OR COMPARE REGISTERS (ORC)



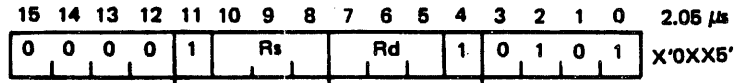
508-4-24

The contents of the source register are logically ORed with the contents of the destination register, and the results are reflected in the indicators. Neither the source nor destination register is changed.

$$Rd_{15-0} \vee Rs_{15-0} \rightarrow \text{Data Bus}$$

Indicators affected: Zero, Plus
 Registers affected: None

4.7.7 TRANSFER REGISTER (RTR)



The contents of the source register replace the contents of the destination register. The contents of the source register remain unchanged.

$$Rs_{15-0} \rightarrow Rd_{15-0}$$

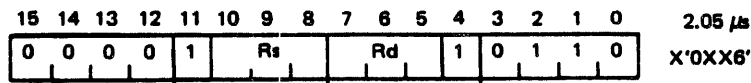
Indicators affected: Zero, Plus
Registers affected: Destination Register

Example:

RTR D,X

The contents of register X replace the contents of register D. Register X is unchanged

4.7.8 SUBTRACT REGISTERS (SUB)



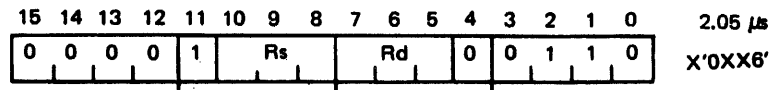
508-4-25

The contents of the source register are subtracted from the contents of the destination register. The result is stored in the destination register; the source register is unchanged.

$$Rd_{15-0} - Rs_{15-0} \rightarrow Rd_{15-0}$$

Indicators affected: Zero, Plus, Overflow, Link
Registers affected: Destination Register

4.7.9 SUBTRACT COMPARE REGISTERS (SUBC)



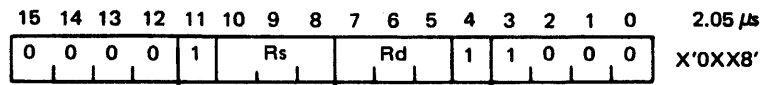
The contents of the source register are subtracted from the contents of the destination register, and the results are reflected in the indicators. Neither the source nor destination register is changed.

$Rd_{15-0} - Rs_{15-0} \rightarrow$ Data Bus

Indicators affected: Zero, Plus, Overflow, Link

Registers affected: None

4.7.10 EXCLUSIVE-OR REGISTERS (XOR)



508-4-26

The contents of the source register are logically Exclusive-ORed (Ψ symbolizes the XOR function) with the contents of the destination register and the result replaces the contents of the destination register.

$Rd_{15-0} \Psi Rs_{15-0} \rightarrow Rd_{15-0}$

Indicators affected: Zero, Plus

Registers Affected: Destination Register

Example:

XOR A,B

If:

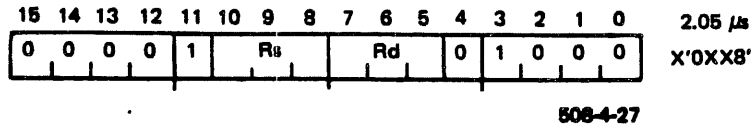
A = 0 ... 01010

B = 0 ... 01100

Result = 0 ... 00110

The result replaces the contents of register A. The contents of register B are not changed.

4.7.11 EXCLUSIVE-OR COMPARE REGISTERS (XORC)



The contents of the source register are logically Exclusive-ORed with the contents of the destination register, and the results are reflected in the indicators. Neither the source nor destination register is changed.

Rd₁₅₋₀ ∇ Rs₁₅₋₀ \rightarrow Data Bus

Indicators affected: Zero, Plus
Registers affected: None

4.7.12 PROGRAMMING EXAMPLES

- This sequence subtracts the number in register A from the number in register B, and places the result in register C if result ≥ 0 or clears register C if result < 0 .

	RTR	C,B	Transfer contents of register B to register C.
	SUB	C,A	Perform subtraction.
	SKP	NEXT	If result is positive, done.
	ZERO	C	If result is negative, zero register C.
NEXT	:		

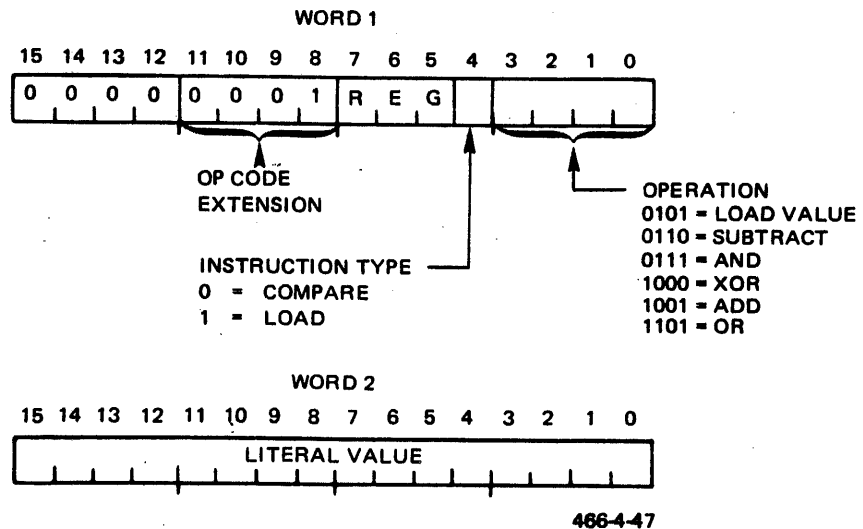
- This sequence tests for overflow before adding the A and B registers.

	ADDC	A,B	Add compare registers A and B.
	SKOT	OFL	Skip if overflow true.
	ADD	A,B	Perform addition, leaving result in register A.
OFL	:		Overflow routine.

4.8 REGISTER OPERATE LITERAL and REGISTER OPERATE LITERAL COMPARE INSTRUCTIONS

Register operate literal instructions perform load-register, arithmetic (addition, subtraction), and logical operations using a specified register and a literal value. The results of an operation may be retained in the specified register and shown in the indicators or may be shown in the indicators only.

There are eleven register operate literal instructions in this group. Each instruction occupies two words and has the following format:



If bit 4=1, the operation determined by bits 3 to 0 is performed. The result replaces the contents of the specified register, and the properties of the result are placed in the indicators. If bit 4=0, the operation determined by bits 3 to 0 is performed and the conditions of the result are placed in the indicators only; the specified register is not changed.

The results of load value to register (LDV) and logical operations are shown in the zero and plus indicators. The results of arithmetic operations are shown in the zero, plus, overflow and link indicators.

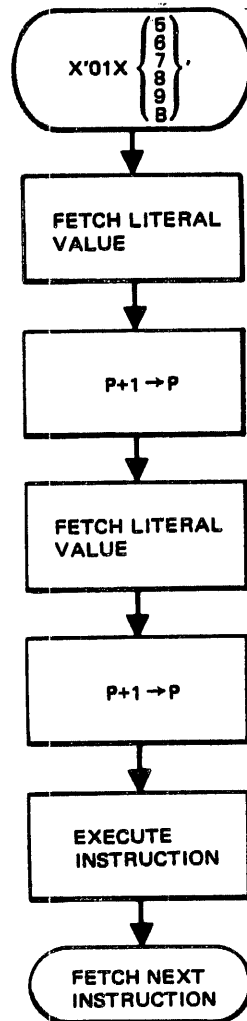
The CAP-16 source statement format for this instruction group is as follows:

<u>Command</u>	<u>Parameters</u>
ADDV	R, lit
ADDVC	
ANDV	
ANDVC	
LDV	
ORV	
ORVC	
SUBV	
SUBVC	
XORV	
XORVC	

where:

- R specifies one of the eight general-purpose register; it determines the coding of bits 7 to 5 in the first word of the instruction.
- lit is the literal value. The literal value may be coded in CAP-16 assembly language in decimal, hexadecimal or octal form or may be represented by any allowable symbol or expression. The literal value comprises the second word of the instruction.

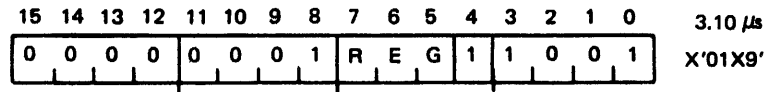
Literal addressing mode is used for all register operate literal and register operate literal compare instructions. The effective address is generated as the address of the next sequential location. Effective address generation is shown in Figure 4-5.



507-4-29

Figure 4-5. Effective Address Generation, Literal Addressing Mode

4.8.1 ADD VALUE TO REGISTER (ADDV)



The second word of the instruction (i.e., the literal) is added to the contents of the register specified and the sum replaces that register's contents.

$$R_{15-0}^{+(P+1)}_{15-0} \rightarrow R_{15-0}$$

Indicators affected: Zero, Plus, Overflow, Link

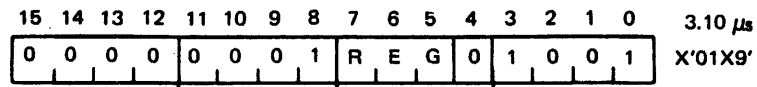
Registers affected: Specified Register

Example:

ADDV A,X'1F02'

The binary equivalent of X'1F02' is added to the contents of register A, and the result is placed in register A.

4.8.2 ADD COMPARE VALUE WITH REGISTER (ADDVC)



508-4-29

The second word of the instruction (i.e., the literal) and the contents of the specified register are added together; the indicators are set according to the results.

$$R_{15-0}^{+(P+1)}_{15-0} \rightarrow \text{Data Bus}$$

Indicators affected: Zero, Plus, Overflow, Link

Registers affected: None

Example:

ADDVC X,50

The binary equivalent of (decimal) 50 is added to the contents of register X with the results appearing in the indicators only. Register X is not changed.

4.8.3 AND VALUE WITH REGISTER (ANDV)



The second word of the instruction (i.e., the literal) is logically ANDed with the contents of the register specified; the result replaces the register contents.

$$R_{15-0} \wedge (P+1)_{15-0} \rightarrow R_{15-0}$$

Indicators affected: Zero, Plus
Registers affected: Specified Register

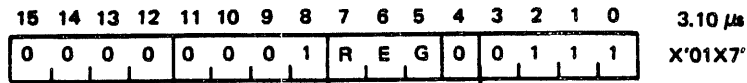
Example:

ANDV B, X'0063'

X'0063' = 0 ... 01100011
If register B = 0 ... 00011111
Result = 0 ... 00000011

The result is placed into register B.

4.8.4 AND COMPARE VALUE WITH REGISTER (ANDVC)



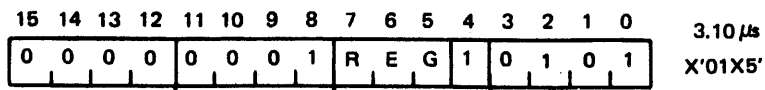
508-4-28

The second word of the instruction (i.e., the literal) and the contents of the register specified are logically ANDed; the indicators are set according to the results.

$$R_{15-0} \wedge (P+1)_{15-0} \rightarrow \text{Data Bus}$$

Indicators affected: Zero, Plus
Registers affected: None

4.8.5 LOAD VALUE TO REGISTER (LDV)



The second word of the instruction (i.e., the literal) replaces the contents of the register specified.

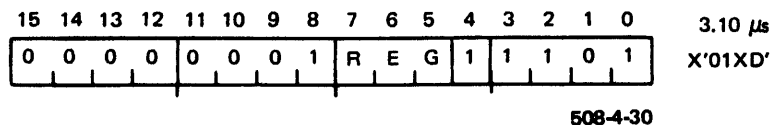
$$(P+1)_{15-0} \rightarrow R_{15-0}$$

Indicators affected: Zero, Plus
 Registers affected: Specified Register

Example:

LDV X,10 The binary equivalent of decimal 10 is loaded into register X.

4.8.6 OR VALUE WITH REGISTER (ORV)

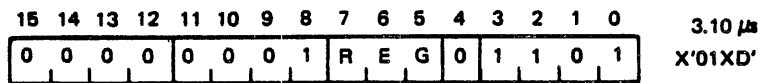


The second word of the instruction (i.e., the literal) is logically ORed with the contents of the register specified; the results are placed in the specified register.

$$R_{15-0} \vee V(P+1)_{15-0} \rightarrow R_{15-0}$$

Indicators affected: Zero, Plus
 Registers affected: Specified Register

4.8.7 OR COMPARE VALUE WITH REGISTER (ORVC)

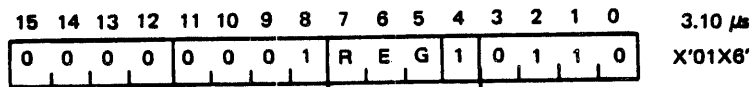


The second word of the instruction (i.e., the literal) and the contents of the register specified are logically ORed; the indicators are set according to the results.

$R_{15-0}^{V(P+1)}_{15-0} \rightarrow$ Data Bus

Indicators affected: Zero, Plus
Registers affected: None

4.8.8 SUBTRACT VALUE FROM REGISTER (SUBV)

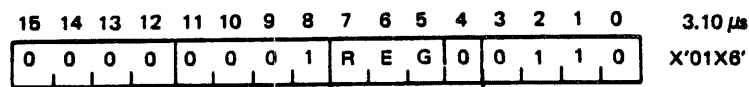


The second word of the instruction (i.e., the literal) is subtracted from the contents of the register specified; the result replaces that register's contents.

$R_{15-0}^{-(P+1)}_{15-0} \rightarrow R_{15-0}$

Indicators affected: Zero, Plus, Overflow, Link
Registers affected: Specified Register

4.8.9 SUBTRACT COMPARE VALUE WITH REGISTER (SUBVC)



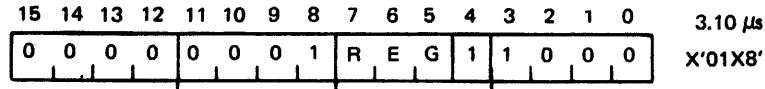
508-4-31

The second word of the instruction (i.e., the literal) is subtracted from the contents of the register specified; the indicators are set according to the results.

$R_{15-0}^{-(P+1)}_{15-0} \rightarrow$ Data Bus

Indicators affected: Zero, Plus, Overflow, Link
Registers affected: None

4.8.10 EXCLUSIVE-OR VALUE WITH REGISTER (XORV)

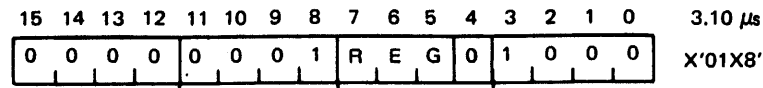


The second word of the instruction (i.e., the literal) is logically Exclusive-ORed with the contents of the register specified; the results are placed in the specified register.

$$R_{15-0} \vee^{(P+1)}_{15-0} \rightarrow R_{15-0}$$

Indicators affected: Zero, Plus
 Registers affected: Specified Register

4.8.11 EXCLUSIVE-OR COMPARE VALUE WITH REGISTER (XORVC)



508-4-32

The second word of the instruction (i.e., the literal) and the contents of the register specified are logically Exclusive-ORed; the indicators are set according to the results.

$$R_{15-0} \vee^{(P+1)}_{15-0} \rightarrow \text{Data Bus}$$

Indicators affected: Zero, Plus
 Registers Affected: None

4.8.12 PROGRAMMING EXAMPLES

1. This routine searches for a specific item in an unordered list of 100 items. The item to be found in the list is contained in register A. The list is stored in memory, and the beginning address of the list is contained in the memory location pointed to by the base address \$\$\$. Each element is accessed by indexing this indirect address, using the contents of register Y for the index. The search is started at the end of the list and works backwards. The first index used, the highest element number, is loaded into register Y by the LDV instruction. The index is decremented to the next list item after a comparison has been made by the CMR instruction. Upon successful completion of the search (FOUND), the absolute address of the list item is placed in register B. If the item is not in the list, B remains zero.

	ZERO	B	Initialize answer register.
	LDV	Y,99	Start at end of the list.
SEARCH	CMR	A,*0,Y,1	Compare item with a list element.
	SKZ	FOUND	If item equals list element, jump to FOUND.
	DECR	Y	Else go to next list element.
	SKM	EXIT	If list is exhausted, item is not in list.
	JMP	SEARCH	If more list items, go to next one.
FOUND	LDR	B,0,,1	Put beginning address of list in answer register.
	ADD	B,Y	Add index position of element in list; answer is
EXIT	:		now in register B.

2. ANDVC may be used to test a byte or a bit of a register for zero. *reg* is the register specified.

To test right byte:

ANDVC	<i>reg</i> ,X'00FF'	
SKZ	ZERO	Jump if right byte is zero.

To test left byte:

ANDVC	<i>reg</i> ,X'FF00'	
SKZ	ZERO	Jump if left byte is zero.

To test bit 'n' of register *reg* for zero ($0 \leq n \leq 15$), the literal is a mask composed of 15 zeros and a one in the bit position to be tested:

ANDVC	<i>reg</i> ,lit	where lit = 2^n .
SKZ	ZERO	

For example, to test bit 3 of register A:

ANDVC	A,X'0008'	$8=2^3$
SKZ	ZERO	

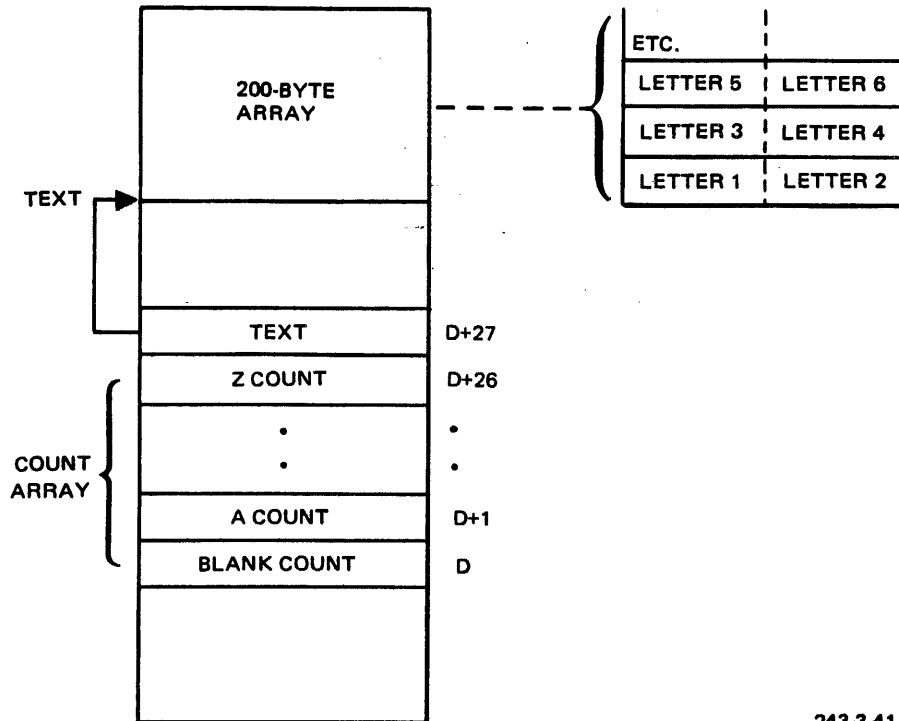
3. A function of NUM, with NUM previously loaded into register A, is to be completed according to the following formula:

$f(\text{NUM}) = \text{NUM} + f(\text{NUM} - 2)$	for NUM ≥ 10
$f(\text{NUM}) = 0$	for NUM < 10

In the sequence, successive values of NUM are added to the subtotal in register B until NUM becomes less than 10.

	ZERO	B	Zero answer register.
LOOP	SUBVC	A,10	} Test for NUM less than 10.
	SKM	DONE	
	ADD	B,A	Add NUM to subtotal.
	SUBV	A,2	Subtract 2 from NUM.
	JMP	LOOP	Jump to test new NUM value.
DONE	:		Answer is in register B.

4. This sequence counts the number of times each letter of the alphabet occurs in a piece of text. The ASCII text (see Appendix B) is contained in a 200-byte array labeled TEXT; the beginning address (TEXT) is contained in location D+27. A loop counter in register X is used to exit when all letters have been counted. The letter counts will be tallied in 27 locations beginning at D. (It is assumed that these locations initially contain zeros, and that TEXT contains only the characters A to Z and blank.) This is illustrated as follows:



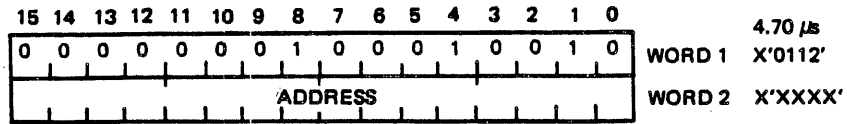
243-3-41

	ZERO	X	Zero TEXT array index to start.
LOOP	LDBY	Y,*27,X,1	Load letter into register Y.
	ANDV	Y,X'1F'	Strip three high-order bits to get count array index.
	INCM	0,Y,1	Add one to letter count.
	INCR	X	Next letter.
	SUBVC	X,200	End of table test.
	SKM	LOOP	If not done, fetch next letter.
DONE	:		
	:		

4.9 SUBROUTINE RETURN VIA INDIRECT VECTOR (RTNIV)

The primary use of this instruction is to return from subroutines entered via non-inhibitible (NI) interrupts. To return from inhibitible (IN) interrupts, see RTRN, Section 4.10.11. Another function of RTNIV is to effect orderly context switching. The memory mode mask word (32K/64K) will not take effect until an RTNIV instruction is executed (Section 4.15.2.4).

The RTNIV instruction has the following format:



508-4-33

CAP-16 source statement for an RTNIV instruction is as follows:

<u>Command</u>	<u>Parameter</u>
RTNIV	address

where: address is the first address of the two-word dedicated memory location for the particular NI interrupt (Table 4-4). Alternatively, value may be the first address of any two-word buffer which has previously been loaded with address of the next instruction and the ISE status.

Table 4-4. Dedicated Memory for NI Interrupts

NI Interrupt	Vector	P-Register Save (Address)	ISE Save
Power Fail	X'40'	X'78'	X'79'
Auto Restart	X'41'	None ^①	None
Memory Parity Protect	X'42'	X'7A'	X'7B'
TRAP and Undefined Op Code	X'44'	X'7C'	X'7D'
Single Step/Break	X'46'	X'7E'	X'7F'

① To use an RTNIV to return from an auto restart, address must be predefined by an operating system program or the interrupt service routine.

32K Mode

The contents of bits 15 to 0 of the location specified by word two are transferred to register P. The content of bit 15 of the next location is transferred to the ISE.

$$((EA))_{15-0} \rightarrow P_{15-0} \quad (\text{Bit 15 is forced to zero})$$

$$((EA))+1)_{15} \rightarrow ISE$$

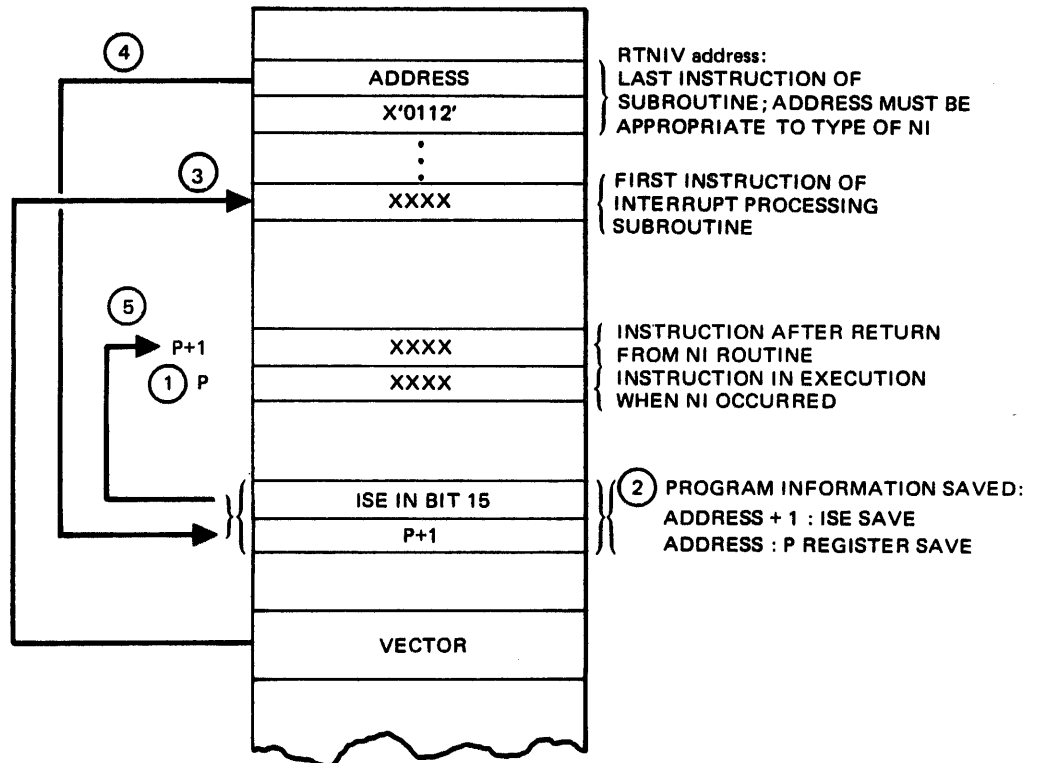
64K Mode

The contents of the location specified by word two are transferred to register P. The content of bit 15 of the next location is transferred to the ISE.

$$((EA))_{15-0} \rightarrow P_{15-0}$$

$$((EA))+1)_{15} \rightarrow ISE$$

An example of the use of an RTNIV for an NI subroutine return is illustrated for TRAP instruction (Section 4.13.9). The following map shows a general case:

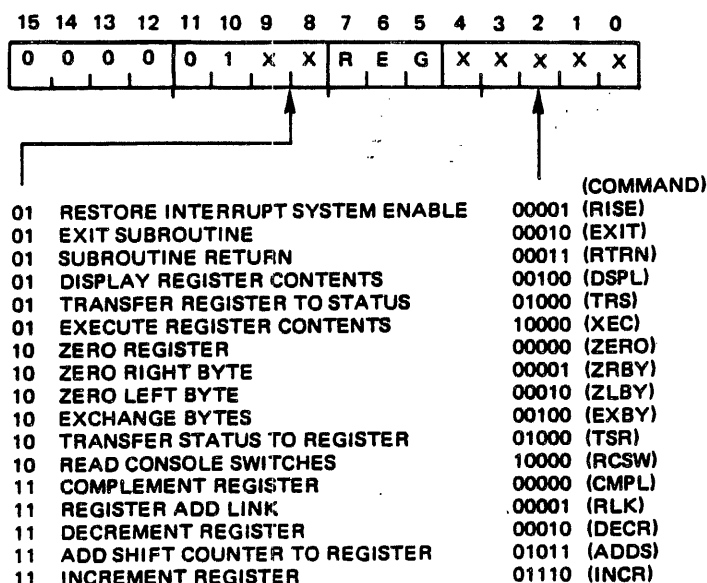


CIRCLED NUMBERS IDENTIFY SEQUENCE OF OPERATIONS

4.10 REGISTER CHANGE INSTRUCTIONS

Register change instructions manipulate data in single registers and perform register indicator transfer operations. This group also includes two instructions for subroutine management.

There are 17 register change instructions and each of these instructions has the following format:

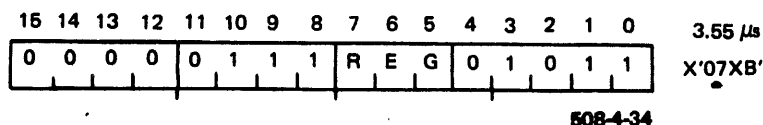


468-4-87

The DSPL and RCSW instructions are not used on a GA-16/110/220. The CAP-16 source statement format for this instruction group is as follows:

Command	Parameters																		
<table style="border: none; width: 100%;"> <tr><td style="width: 100%;">ADDS</td><td rowspan="17" style="font-size: 4em; vertical-align: middle; padding-left: 10px;">}</td></tr> <tr><td>CMPL</td></tr> <tr><td>DECR</td></tr> <tr><td>DSPL</td></tr> <tr><td>EXBY</td></tr> <tr><td>EXIT</td></tr> <tr><td>INCR</td></tr> <tr><td>RCSW</td></tr> <tr><td>RISE</td></tr> <tr><td>RLK</td></tr> <tr><td>RTRN</td></tr> <tr><td>TRS</td></tr> <tr><td>TSR</td></tr> <tr><td>XEC</td></tr> <tr><td>ZERO</td></tr> <tr><td>ZLBY</td></tr> <tr><td>ZRBY</td></tr> </table>	ADDS	}	CMPL	DECR	DSPL	EXBY	EXIT	INCR	RCSW	RISE	RLK	RTRN	TRS	TSR	XEC	ZERO	ZLBY	ZRBY	<p style="text-align: center;">Register (One of the eight general-purpose registers coded into bits 7 to 5)</p>
ADDS	}																		
CMPL																			
DECR																			
DSPL																			
EXBY																			
EXIT																			
INCR																			
RCSW																			
RISE																			
RLK																			
RTRN																			
TRS																			
TSR																			
XEC																			
ZERO																			
ZLBY																			
ZRBY																			

4.10.1 ADD SHIFT COUNTER TO REGISTER (ADDS)



The contents of the shift counter (bits 3-0 of the status register) are added to the specified register.

$$S_{3-0} + R_{15-0} \rightarrow R_{15-0}$$

Indicators affected: Zero, Plus, Overflow, Link

Registers affected: Specified Register

ADDS is normally used after the SRLC instruction (Section 4.12.4). The SRLC instruction shifts bits from bit 0 into the link indicator, stopping if a one is shifted into the link (or until the specified count is exhausted). A shift count is tallied in the shift counter, where 0 = 1 shift, 1 = 2 shifts, etc. The purpose of ADDS is to read the shift counter following execution of the SRLC instruction.

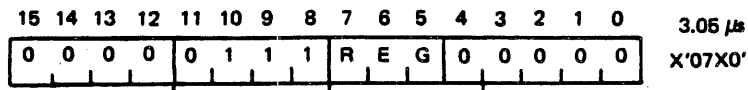
The value of the shift counter (bits 3 to 0) specifies the bit position (15 to 0) of the one bit that was shifted into the link indicator (i.e., the first low-order one bit). The value of the shift counter plus one specifies the number of shifts that were performed. (The link indicator should be tested to determine if the operation halted as a result of the shift count being exhausted or by a one being shifted into the link.)

Example:

To find the bit position of the first low-order one bit in register A and place the bit position in register B:

ZERO	B	Zero answer register.
SRLC	A,16	Shift until lower-order one bit is in link indicator.
SKR	NONE	Exit if link=0.
ADDS	B	Add shift count to register B.
NONE	:	

4.10.2 COMPLEMENT REGISTER (CMPL)



The contents of the register specified is replaced by its one's complement.

$$\bar{R}_{15-0} \rightarrow R_{15-0}$$

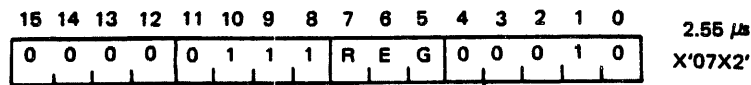
Indicators affected: Zero, Plus, Link
 If the initial contents of the register = 0, 0→L
 If the initial contents of the register ≠ 0, 1→L

Registers affected: Specified Register

Example:

CMPL B If register B contains 1110001011000000
 The one's complement: 0001110100111111 replaces
 the value in register B and the link is set.

4.10.3 DECREMENT REGISTER (DECR)



508-4-35

The specified register is decremented by one.

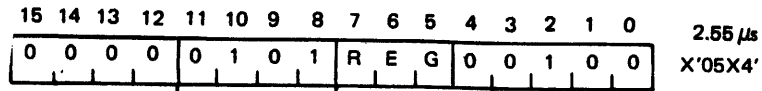
$$R_{15-0}^{-1} \rightarrow R_{15-0}$$

Indicators affected: Zero, Plus, Link (Overflow is not affected.)
 Registers affected: Specified Register

Example:

DECR B Subtract one from the contents of register B.

4.10.4 DISPLAY REGISTER CONTENTS (DSPL)



This instruction places data on the data bus and does a strobe. Because there is no register display device normally installed, this instruction generates no observable result. The instruction is provided for compatibility with the GA-16/440 and SPC-16 series computers.

$R_{15-0} \rightarrow$ data bus (and strobe signal generated)

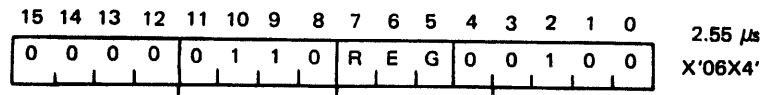
Indicators affected: None

Registers affected: None

NOTE

On a GA-16/220 with SCI, the subroutine at location X'FC61' (Table 3-5) to output (four) ASCII characters to a TTY could be called from a user's program to display data, if timing permits, and registers A, Y, and B can be used. Also D register contents must be saved, then D set to X'FE00' before making the subroutine call.

4.10.5 EXCHANGE BYTES (EXBY)



508-4-35

The left and right bytes of the register specified are interchanged.

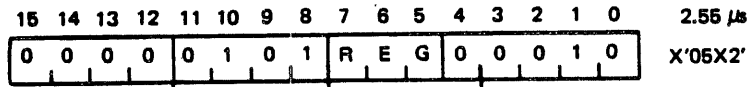
$R_{7-0} \leftarrow \rightarrow R_{15-8}$

Indicators affected: None

Registers affected: Specified register

Example: .EXBY C If register C contains 1111111100000000, after exchanging left and right bytes, register C contains 0000000011111111.

4.10.6 EXIT FROM SUBROUTINE (EXIT)



This instruction is used to transfer program execution to a location specified by the contents of the register specified. There is no effect on the ISE. (To change ISE, see RTRN, Section 4.10.11.)

32K MODE

Bits 14 to 0 of the register specified are transferred to the corresponding position in register P. ISE is unchanged.

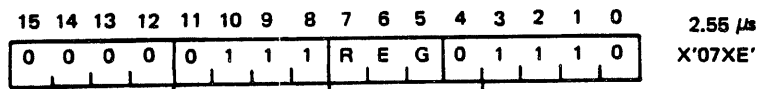
$R_{14-0} \rightarrow P_{14-0}$ ISE unchanged

64K MODE

The contents of the register specified are transferred to register P. ISE is unchanged.

$R_{15-0} \rightarrow P_{15-0}$ ISE unchanged

4.10.7 INCREMENT REGISTER (INCR)



508-4-36

The specified register is incremented by one.

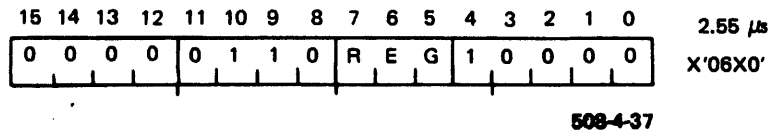
$R_{15-0}^{+1} \rightarrow R_{15-0}$

Indicators affected: Zero, Plus, Link (Overflow is not affected.)
 Registers affected: Specified Register

Example:

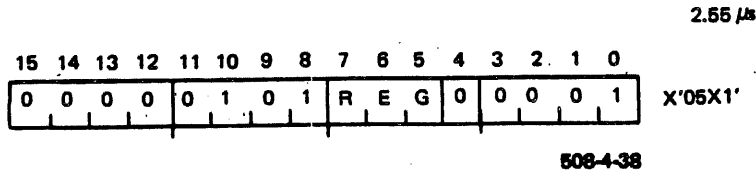
INCR A One is added to the contents of register A.

4.10.8 RCSW (INVALID INSTRUCTION)



The RCSW instruction cannot be used on a GA-16/220. Any program containing RCSW must be modified for execution on the GA-16/220. Use either Read Console Switches into Memory (RCSM) or Read Console Switches into Register (RCSR). (See Sections 4.15.1.1 and 4.15.1.2.)

4.10.9 RESTORE INTERRUPT SYSTEM ENABLE (RISE)



32K MODE

This instruction transfers the content of bit 15 of the register selected (by REG bits 7,6,5) to the ISE. (Logical one enables IN interrupts.)

$R_{15} \rightarrow ISE$

64K MODE

This instruction transfers the content of bit 15 of the status register to the ISE. The REG bits (7 to 5) in the instruction are disregarded.

$S_{15} \rightarrow ISE$

Indicators affected: None
 Registers affected: None

The RISE instruction is used to restore the interrupt status following the execution of a JSR if the interrupt status within the subroutine is to be the same as that in the calling routine. In the 32K mode, register E is normally specified, since the interrupt system enable status was stored in register E, bit 15, during the execution of the jump-to-subroutine (JSR) instruction.

If the transferred status enables the interrupt system, IN interrupts may occur immediately following execution of the RISE instruction.

Example:

Calling Program

```

    :
    : JSR    SUBR
    :
    :
```

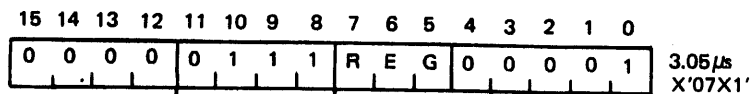
Subroutine

```

SUBR  SARS  temp
      RISE  E
      :
```

Subroutine SUBR will now have the same interrupt status as the calling program.

4.10.10 ADD LINK TO REGISTER (RLK)



508-4-39

The content of the link indicator is added to the contents of the register specified. This instruction would typically be used to set up the dividend for a hardware DIV instruction (Section 4.16.1) as shown in examples below. It can also be used when multiple programmable registers are used for multiplication of very large numbers by successive addition.

$$R_{15-0} + L \rightarrow R_{15-0}$$

Indicators affected: Zero, Plus, Overflow, Link

Registers affected: Specified Register

Examples:

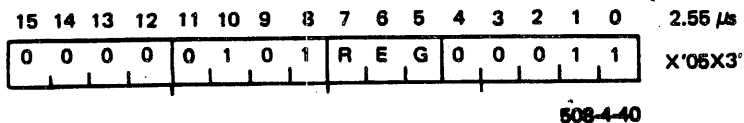
SINGLE PRECISION DIVIDEND

LDR	C, DIVS	Put unadjusted single precision dividend into register C
ZERO	B	Be sure upper bits of dividend are zero
ADD	C,C	Shift bits left; bit 15 into link
RLK	B	If the link indicator is set (=1), register B is incremented. If the link indicator is reset (=0), the value in register B is not changed.

DOUBLE PRECISION DIVIDEND

LDR	C, DIVL	Put least significant bits of dividend into register C
LDR	B, DIVM	Put most significant bits of dividend into register B (cannot exceed 14 bits)
ADD	B,B	Shift MSBs left one bit
ADD	C,C	Shift LSBs left one bit; (bit 15 into link)
RLK	B	Set bit 0 of register B from link (as described in previous example)

4.10.11 RETURN FROM SUBROUTINE (RTRN)

32K MODE

The contents of the register specified (normally register E) are transferred to register P and to the interrupt system enable.

$R_{14-0} \rightarrow P_{14-0}$

$R_{15} \rightarrow ISE$

64K MODE

The contents of the register specified are transferred to register P and the content of bit 15 of the status register is transferred to the ISE.

$R_{15-0} \rightarrow P_{15-0}$

$S_{15} \rightarrow ISE$

Indicators affected: None
Registers affected: None

If the restored status enables the interrupt system, interrupts may occur immediately following execution of the RTRN instruction.

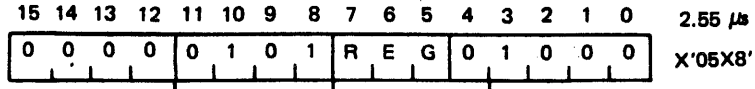
To return from NI interrupts, use an RTNIV instruction (Section 4.9). To return from a subroutine and not change ISE, see EXIT (Section 4.10.6).

Example:

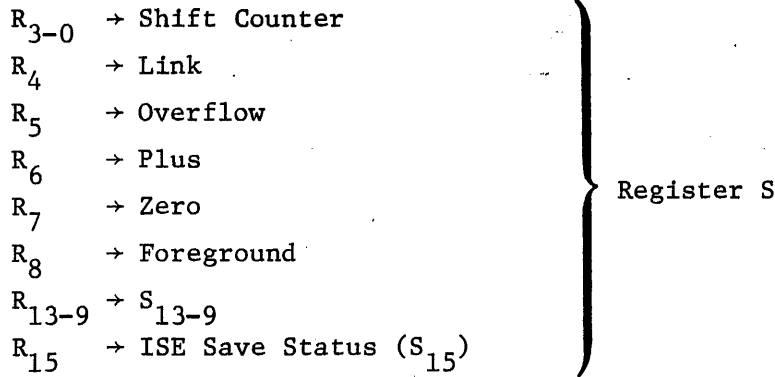
RTRN E

Control is returned to the calling program at the location specified by the contents of register E, bits 14 to 0. The ISE status of the calling program prior to the jump-to-subroutine, which was placed into bit 15 of register E, is placed back into the ISE, restoring the interrupt status of the calling program.

4.10.12 TRANSFER REGISTER TO STATUS (TRS)

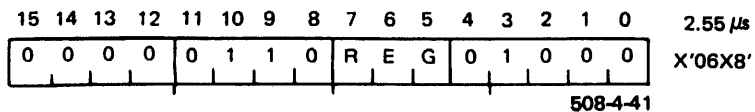


The appropriate bits of the register specified are loaded into the indicators and shift counter. This instruction changes the state of the indicators and may, therefore, affect the mode (foreground, background). The 32/64K mode indicator (S₁₄) is unaffected. S₁₅ (previous ISE status; also called ISE save status) is changed but ISE itself is unaffected.

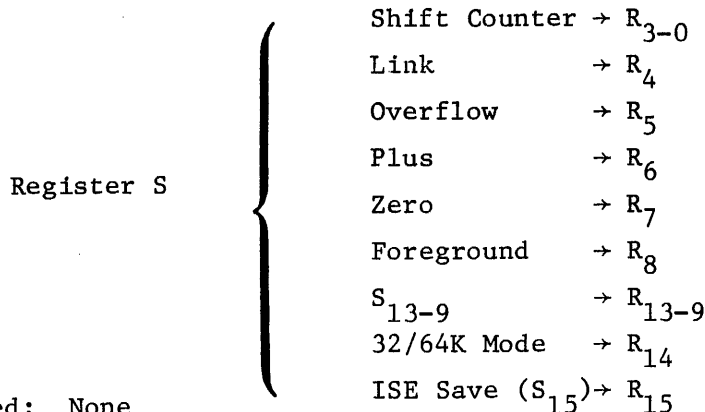


Indicators affected: Zero, Plus, Overflow, Link, Foreground, ISE Save
 Registers affected: None

4.10.13 TRANSFER STATUS TO REGISTER (TSR)

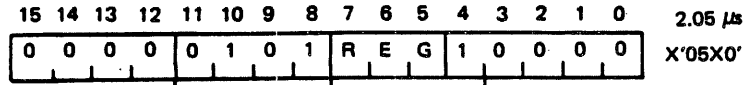


The contents of the indicators and the shift counter are transferred to the register specified.



Indicators affected: None
 Registers affected: Specified Register

4.10.14 EXECUTE REGISTER CONTENTS (XEC)



The contents of the register specified are loaded into the instruction register (register I) and executed as a single-word instruction. Only single-word instructions can be executed with the XEC instruction. This excludes two-word memory reference and indexing instructions (i.e., ADDR/DISP = X'1F') and all other instructions which require two words. The time of the normal instruction execution must be added to the XEC time for the total instruction execution time.

$R_{15-0} \rightarrow I_{15-0}$; Execute I

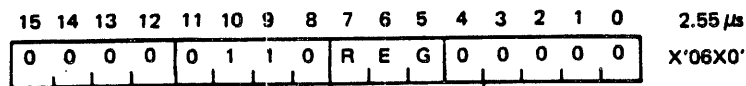
Indicators affected: The same indicators as would be affected by the normal execution of the instruction.

Registers affected: The same registers as would be affected by the normal execution of the instruction.

NOTE

Interrupts may occur before register contents are executed.

4.10.15 ZERO REGISTER (ZERO)



508-4-42

The register specified is cleared.

$0 \rightarrow R_{15-0}$

Indicators affected: None

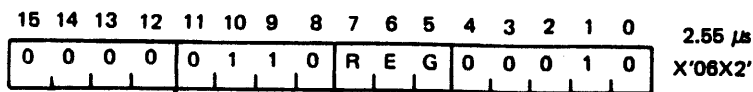
Registers affected: Specified Register

Example: ZERO X The contents of register X are replaced with zeros.

The double-word instruction (Section 4.8.5) may also be used.

LDV X,0 Place all zeros in register X and, in addition, affects the zero and plus indicators.

4.10.16 ZERO LEFT BYTE (ZLBY)



The left byte of the register specified is cleared. The right byte is not affected

0 \rightarrow R₁₅₋₈

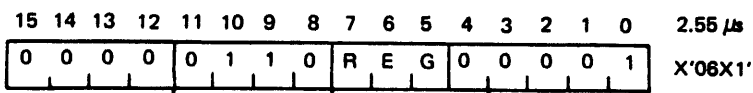
Indicators affected: None

Registers affected: Bits 15 to 8 of the register specified.

Example:

ZLBY A If register A contained X'FFFF', ZLBY would
replace that value with X'00FF'.

4.10.17 ZERO RIGHT BYTE (ZRBY)



508-4-43

The right byte of the register specified is cleared. The left byte is not affected.

0 \rightarrow R₇₋₀

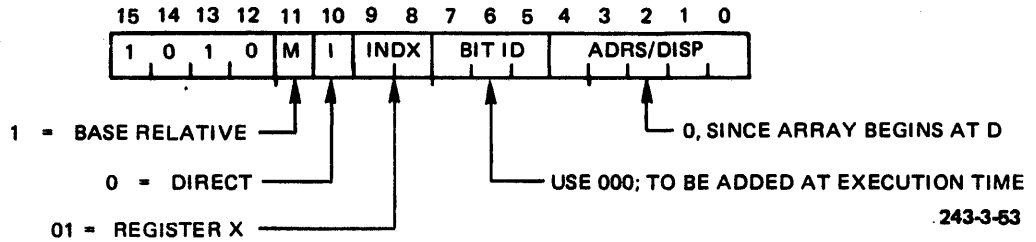
Indicators affected: None

Registers affected: Bits 7 to 0 of the register specified.

Example:

ZRBY C If register C contained X'FFFF', ZRBY would
replace this value with X'FF00'.

The TBIT instruction will be defined in the subroutine TSTB as a literal value with the bits set as follows:



This forms the number: binary — 1010100100000000
 hexadecimal — A 9 0 0

Subroutine TSTB must take out the three lower bits of register X and put them in bits 7 to 5 of the TBIT instruction. Then it shifts off the three bits, leaving the correct byte address in register X.

It is assumed that the main program will not need the original contents of any of the registers or indicators changed by the subroutine, e.g., that it will branch to a new operation depending upon whether the bit tested is zero or one.

Main:

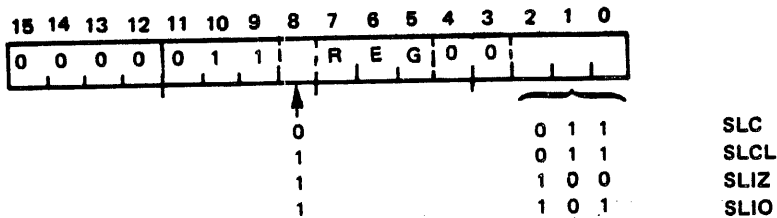
	:	Calculations define word, byte, and bit in bit
	:	array (399 to 0) and place binary representation
	:	in register X.
JSR	TSTB	Call subroutine to test bit.
SKN	BONE	Skip to routine for bit=1.
	:	Continue when bit=0.
	:	Routine to process bit=1.
BONE		

Subroutine:

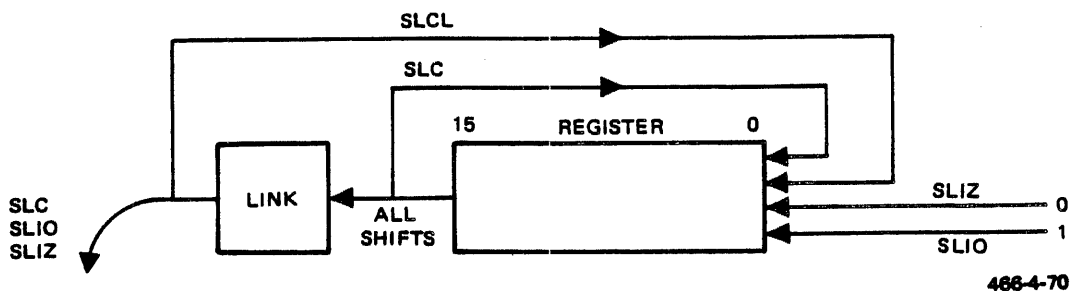
TSTB	LDV	A,X'0007'	Set up mask for bit position.
	AND	A,X	Put bit position (only) in register A.
	SRC	A,11	Shift bit position to bits 7 to 5 for TBIT instruction.
	ORV	A,X'A900'	Fill in rest of TBIT instruction.
	SRA	X,3	Make byte index register X.
	XEC	A	Execute TBIT instruction.
	RTRN	E	Result is in zero indicator.

4.11 SHIFT LEFT INSTRUCTIONS

Shift left instructions shift the selected register one bit position left. The most significant bit of the register is always transferred to link. The zero and plus indicators also monitor the results of all left shifts. There are four instructions in the shift left group. They have the following format:



The shift patterns performed by the shift left instructions are as follows:



The CAP-16 source statement for this instruction group is as follows:

<u>Command</u>	<u>Parameters</u>
SLC	Any Register
SLCL	
SLIZ	
SLIO	

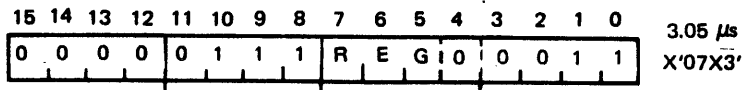
4.11.1 SHIFT LEFT CIRCULAR (SLC)



The register selected is shifted left one bit position. Bit 15 is shifted into both link and bit 0. The bit shifted out of link is lost.

Indicators affected: Zero, Plus, Link
Registers affected: Selected Register

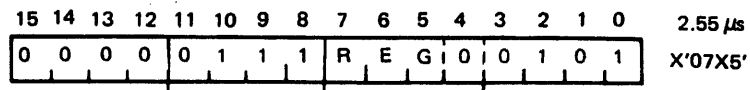
4.11.2 SHIFT LEFT CIRCULAR THROUGH LINK (SLCL)



The register selected is shifted left one bit position. Link is shifted into bit 0. Bit 15 is shifted into link.

Indicators affected: Zero, Plus, Link
Registers affected: Selected Register

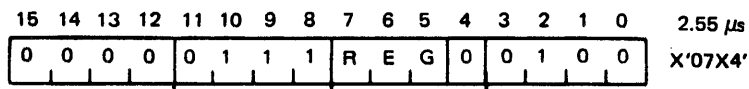
4.11.3 SHIFT LEFT INSERT ONE (SLIO)



The register selected is shifted left one bit position. Bit 15 is transferred to link, the bit shifted out of link is lost, and a one is shifted into bit 0.

Indicators affected: Zero, Plus, Link
Registers affected: Selected Register

4.11.4 SHIFT LEFT INSERT ZERO (SLIZ)



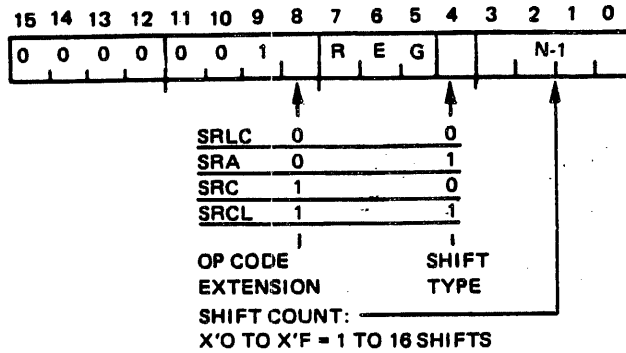
508-4-44

The register selected is shifted left one bit position. Bit 15 is transferred to link and a zero is shifted into bit 0. The bit shifted out of link is lost.

Indicators affected: Zero, Plus, Link
Registers affected: Selected Register

4.12 SHIFT RIGHT INSTRUCTIONS

Shift right instructions shift the bits of the selected register a specified number of positions to the right. Results of the shift are reflected in the zero, plus and link indicators. The shift proceeds bit by bit until the shift count has been satisfied; the shift right logical and count, however, will terminate as soon as a one bit is shifted into the link. For all other shifts, the number specified by the shift count (n) is always one less than the actual number of bits shifted. There are four instructions in the shift right group; each has the following format:



466-4-72

The CAP-16 source statement format for this instruction group is as follows:

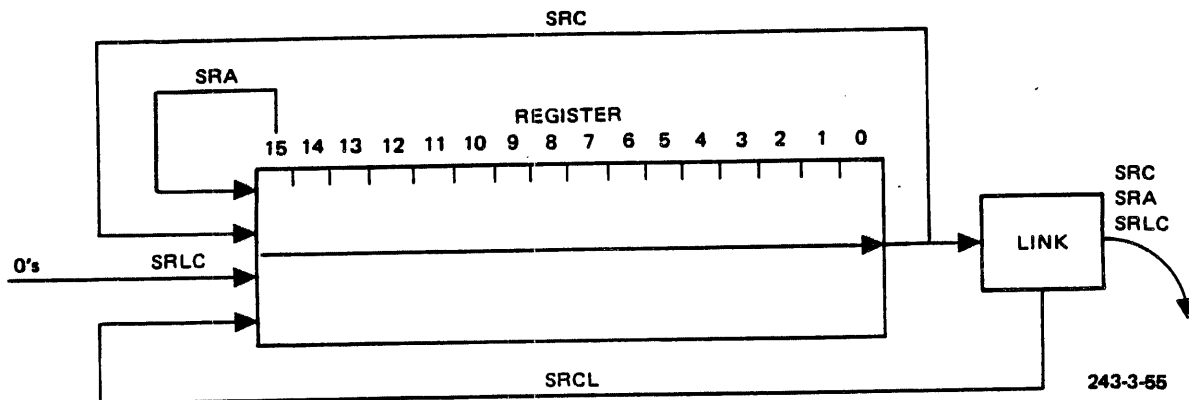


where:

Register is a general-purpouse register, coded into bits 7 to 5.

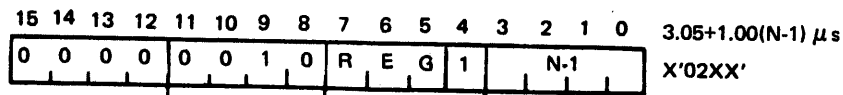
Number is the actual number of shifts to be performed. The assembler subtracts one from the number (n); therefore n-1 is coded into bits 3 to 0 of the machine instruction.

The shift patterns performed by the shift instructions are diagrammed as follows:



The processor time required to execute a shift instruction is a function of the number of shifts performed.

4.12.1 SHIFT RIGHT ARITHMETIC (SRA)



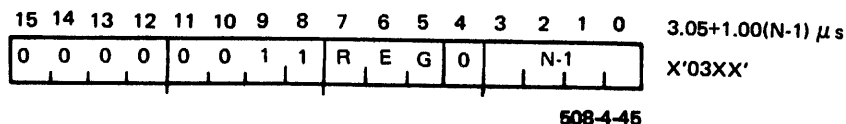
The register selected is shifted right (N) positions as specified by the shift count. The sign position (bit 15) is maintained and propagated to the adjacent right bit positions as specified by the shift count. Bits are shifted out of bit 0 into the link. Bits shifted out of the link are lost.

Indicators affected: Zero, Plus, Link
Registers affected: Selected Register

Examples:

- SRA A,10 If register A originally contains 0000111100001111, then after completing ten shifts, register A contains 0000000000000011 and the link contains the original bit 9 (=1).
- SRA B,15 If register B originally contains 1000000000000000, then after completing fifteen shifts, register B contains 1111111111111111 and the link contains the original bit 14 (=0).

4.12.2 SHIFT RIGHT CIRCULAR (SRC)



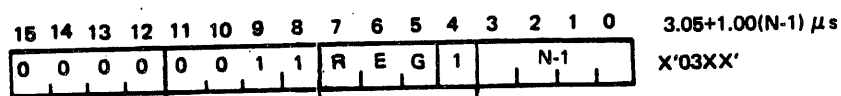
The register selected is shifted right (N) positions as specified by the shift count. Bit 0 is shifted into bit 15, and bit 0 is shifted into the link. Bits shifted into bit 15 move to the right for each subsequent shift. Bits shifted out of the link are lost.

Indicators affected: Zero, Plus, Link
Registers affected: Selected Register

Examples:

- SRC A,1 If register A originally contains 1111000011110000, then after completing one shift, register A contains 0111100001111000 and the link contains the original bit 0 (=0).
- SRC A,5 If register A originally contains 1111000011110000, then after completing five shifts, register A contains 1000011110000111 and the link contains the original bit 4 (=1).

4.12.3 SHIFT RIGHT CIRCULAR THROUGH LINK (SRCL)



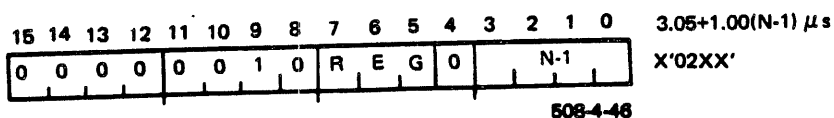
The register selected is shifted right circular (N) positions as specified by the shift count. The content of the link is shifted into bit 15. Bits are shifted out of bit 0 into the link.

Indicators affected: Zero, Plus, Link
 Registers affected: Selected Register

Example:

SRCL A,5 If register A originally contains 0000000011110000 and the link is set (=1), then after completing five shifts, register A originally contains 0000100000000111 and the link contains the original bit 4 (=1).

4.12.4 SHIFT RIGHT LOGICAL AND COUNT (SRLC)



The register selected is shifted right (N) positions as specified by the shift count or until the link contains one, whichever occurs first. For each shift made, a zero is placed into bit 15. Bits are shifted out of bit 0 into the link. Bits shifted out of the link are lost.

The shift counter will contain the bit position of the one bit that terminates the shifting; this number is also the number of shifts performed minus one. The value in the shift counter may be read by the ADDS instruction (Section 4.10.1) or by the TSR instruction (Section 4.10.13).

Indicators affected: Zero, Plus, Link
 Registers affected: Selected Register

Examples:

SRLC B,10 If register B originally contains 1101000000000000, then after completing ten shifts, register B contains 0000000000110100 and the link contains the original bit 9 (=0). The shift counter will contain nine.

SRLC C,6 If register C originally contains 1000000000011100, then only three shifts will be made. After completion, register C contains 0001000000000011 and the link contains the original bit 2 (=1). The shift counter will contain two.

4.12.5 PROGRAMMING EXAMPLE

This program will count the number of ones in register A and leave the count in register X. Upon completion, register A contains all zeros.

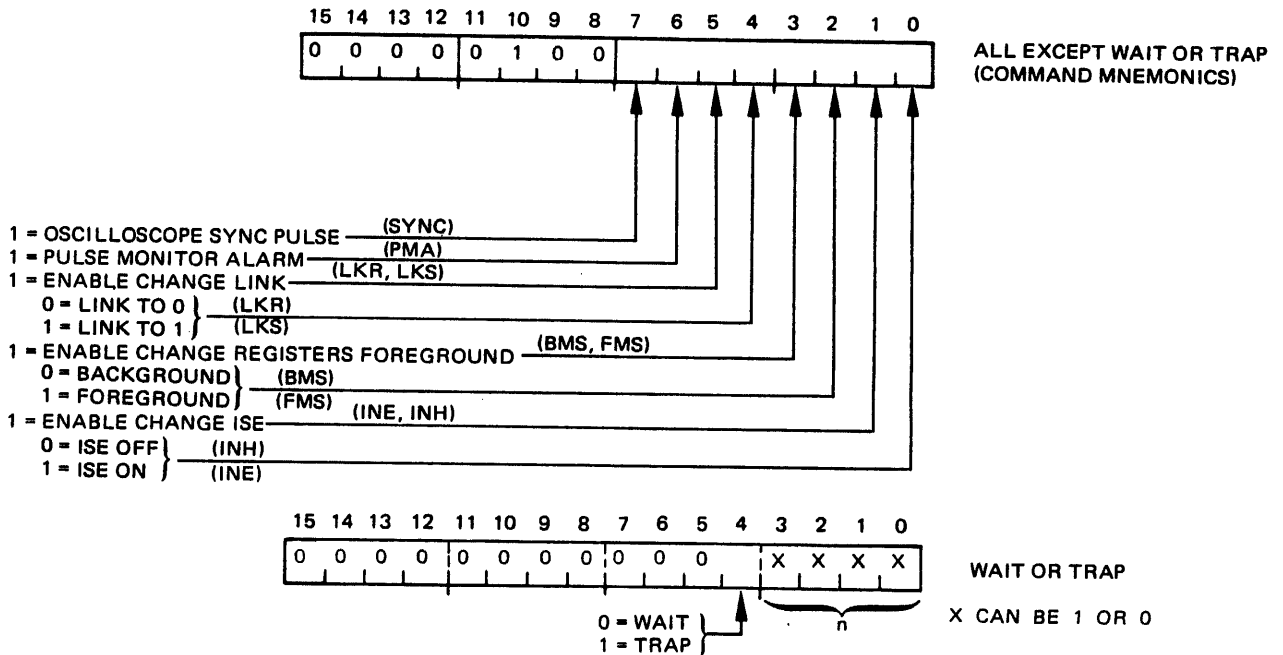
	ZERO	X	Zero count to begin.
LOOP	SRLC	A,16	Shift until link contains a one.
	SKR	DONE	Skip if no one was shifted into link.
	INCR	X	Count the ones.
	JMP	LOOP	Go to shift again.
DONE	:		
	:		

4.13 CONTROL INSTRUCTIONS

The control instructions perform the following functions:

- Enable/inhibit interrupt system
- Set foreground/background mode
- Pulse the operations monitor alarm (OMA) timer
- Generate a sync pulse
- Set or reset the link indicator
- Cause program execution to "halt" in a wait state
- Cause program to TRAP (interrupt) out of program sequence

The instructions in this group have the following formats:

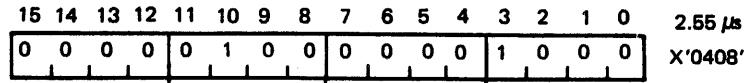


The source statement format for this group is as follows:

<u>Command</u>	<u>Parameters</u>
BMS	
FMS	
INE	
INH	
LKR	
LKS	
PMA	
SYNC	
TRAP	[number]
WAIT	[number]

No mnemonic parameters are specified for control instructions except for WAIT and TRAP, n=X'0-F'. As shown, all sixteen bits are used as an operations code, with the exceptions of WAIT and TRAP.

4.13.1 SET BACKGROUND MODE (BMS)



Following execution of this instruction, the eight background registers (A', X', Y', Z', B', C', D', E') are used by all instructions that specify registers. The foreground registers (A, X, Y, Z, B, C, D, E) are not addressable while the computer is operating in the background mode.

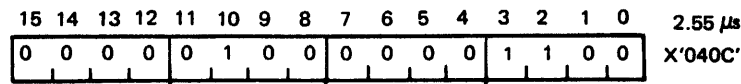
0 \rightarrow Foreground (S_8)

Indicators affected: Foreground (S_8)

Registers affected: Set of eight background mode registers become accessible;
set of eight foreground mode registers are not accessible.

Background mode may also be set by execution of the LARS (4.5.4) or the TRS (4.10.12) instructions.

4.13.2 SET FOREGROUND MODE (FMS)



508-4-47

Following execution of this instruction, the eight foreground registers (A, X, Y, Z, B, C, D, E) are used by all instructions that specify registers. The background registers (A', X', Y', Z', B', C', D', E') are not addressable while the computer is operating in the foreground mode.

1 \rightarrow Foreground (S_8)

Indicators affected: Foreground

Registers affected: Set of eight foreground mode registers become accessible;
set of eight background mode registers are not accessible.

The computer will always be placed in the foreground mode by pressing the console RESET (16) switch (Section 3.3), and may be placed in foreground by execution of the LARS (4.5.4) or the TRS (4.10.12) instructions.

4.13.3 ENABLE INTERRUPTS (INE)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	2.55 μ s
0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	X'0403'

Following execution of this instruction, inhibitible (IN) interrupt requests may interrupt the normal program sequence and demand service by an interrupt subroutine. The interrupt system will allow an IN interrupt immediately following execution of the INE instruction.

1 \rightarrow ISE

Indicators affected: None
Registers affected: None

4.13.4 INHIBIT INTERRUPTS (INH)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	2.55 μ s
0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	X'0402'

Following execution of this command, normal program sequencing will not be interrupted by IN interrupt requests. IN interrupts are inhibited immediately following execution of the INH instruction. INH does not affect the non-inhibitible (NI) interrupts. NI interrupts are summarized in Section 4.9.

0 \rightarrow ISE

Indicators affected: None
Registers affected: None

4.13.5 RESET LINK (LKR)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	2.55 μ s
0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	X'0420'

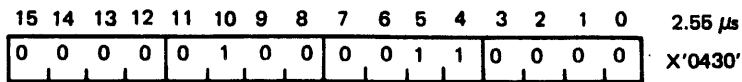
508-4-48

Execution of this instruction places a zero into the link.

0 \rightarrow Link

Indicators affected: Link
Registers affected: None

4.13.6 SET LINK (LKS)



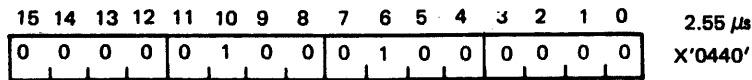
Execution of this instruction places a one into the link.

1 \rightarrow Link

Indicators affected: Link

Registers affected: None

4.13.7 PULSE OPERATIONS MONITOR ALARM (PMA)

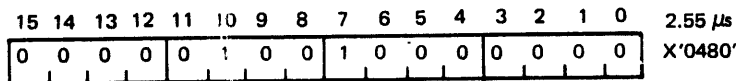


This instruction resets the operations monitor alarm (OMA) timer each time the instruction is executed. Failure to execute this instruction within 150-300 ms of the previous execution causes the computer to automatically switch from RUN mode to IDLE mode and the safe signal to be removed from the safe line (SFEC-). Recovery from this mode can be effected only by operator intervention. The OMA timer will not initially start until a PMA instruction is executed for the first time. Thereafter, it must be periodically executed to prevent the OMA from becoming activated. This alarm is also described in Section 4.15.4, Special Features and Standard I/O.

Indicators affected: None

Registers affected: None

4.13.8 GENERATE SYNC PULSE (SYNC)



508-4-49

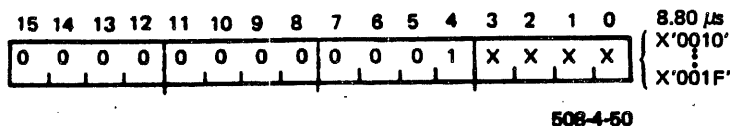
Execution of this instruction places a 120 ns pulse on the SYNC-line on the I/O cable. It is the purpose of this instruction to provide a synchronizing pulse for hardware/software debugging and is usually used to synchronize an oscilloscope.

Pulse \rightarrow Sync

Indicators affected: None

Registers affected: None

4.13.9 PROGRAM SEQUENCE INTERRUPT (TRAP)/RESERVED OP CODES

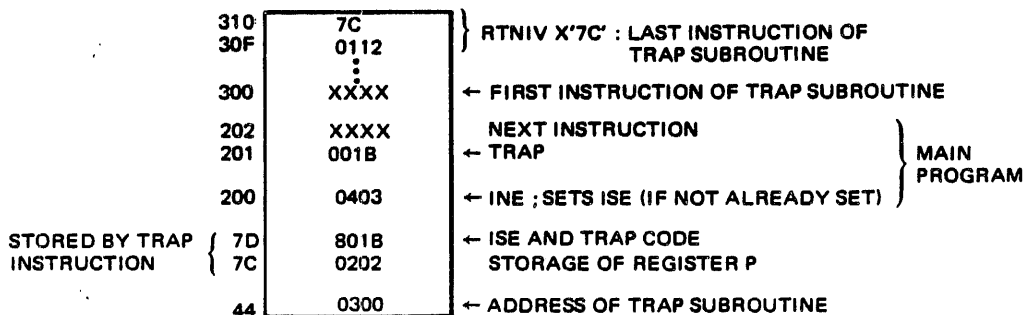


Execution of this instruction generates a non-inhabitable (NI) firmware interrupt through vector address 44. All op codes from X'0010' to X'001F' perform traps. The status of ISE is stored in bit position 15 of location 7D. The least significant 15 bits of the TRAP instruction are transferred to the corresponding bit positions of location 7D. The contents of register P are stored in location 7C. The next instruction address is the contents of location 44.

- ISE → 7D₁₅
- (TRAP Code)₁₄₋₀ → 7D₁₄₋₀
- P → 7C
- (44) → P
- 0 → ISE

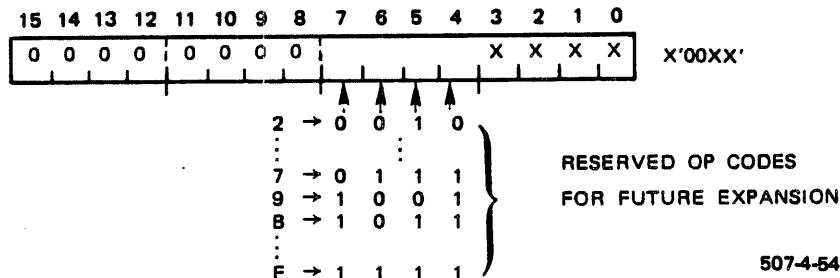
Indicators affected: None
 Registers affected: None

The operation of the TRAP instruction is illustrated in the following core map:



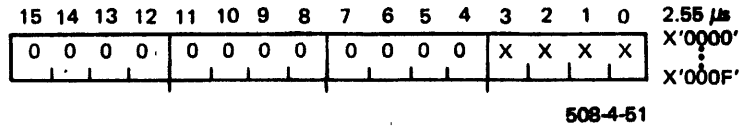
Reserved Op Codes

Certain op codes have been reserved for future expansion. They are listed below. They should never be used, as it may result in future operating system incompatibilities.



Execution of any reserved op code will perform the same functions as a TRAP instruction. For a more detailed discussion of interrupts, see Section 2.6.

4.13.10 WAIT (WAIT)



Execution of this instruction causes program execution to halt; the microconsole WAIT and RUN indicators are illuminated. The WAIT instruction operates by preventing the program counter (register P) from advancing; the processor continues to cycle the same memory location (i.e., the location containing the WAIT instruction).

An operand entry may be coded in bits 3 to 0 of the WAIT instruction.

NOTE

This value is in register I upon execution; however, there is no means to examine this value on a GA-16/110/220.

Both interrupts and direct memory transfers can be serviced while the computer is in the WAIT mode. If an interrupt occurs, control is transferred to the interrupt service routine as usual. The interrupt routine, after completion, returns control to the location containing the WAIT instruction provided the return address is taken as the stored contents of register P (in register E for IN interrupts or in a dedicated address for NI interrupts).

The program can be advanced past the WAIT condition if the CPU is in the RUN mode provided the following conditions are met:

- The TTY break capability is available (Section 3.7.2).
- A pending I/O interrupt routine contains an instruction which advances the stored program counter past the WAIT instruction prior to return.

Otherwise, the WAIT instruction should only be used if CPU reset is permissible following the WAIT instruction.

4.14 PROGRAMMED INPUT/OUTPUT INSTRUCTIONS

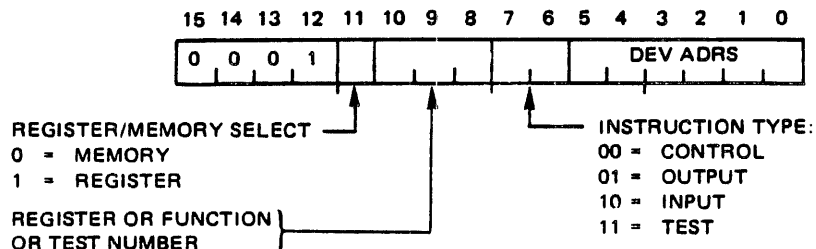
Programmed I/O instructions (PIO) cause information to be transferred over the I/O-Bus between the CPU and a device (typically an I/O controller for a peripheral) installed in the I/O section or expansion chassis. Programmed I/O instructions are also used to transfer information between the CPU and a teletype, console switches, or an internal mask word. (This latter type of programmed I/O is also called standard I/O because the functions are carried out within the CPU.)

The type and sequence of information which may be transferred is dependent on the design of the I/O controller, which in turn is determined by the peripheral it controls. Generally, the types of PIO information are as follows:

- Send one of eight possible control signals (CTRL instruction) to an I/O controller to control interrupt capability, prepare it or its peripheral for some operation, or initiate an operation such as a Direct Memory Access (DMA) transfer via the Multiple High-Speed Data Channel (MHSDC).
- Test the condition of one of eight possible functions (TEST instruction) of an I/O controller which causes a program branch when the condition tested is satisfied. Alternatively, an interrupt may be used to effect a branch on some conditions.
- Output a data word (or byte) from a register (DTOR instruction) or from a memory location specified in a register (DTOM instruction) to an I/O controller. The I/O controller may either transfer this data to the peripheral or it may treat the word as a control signal.
- Input a data word (or byte) from a peripheral device via its I/O controller (or from the I/O controller itself to indicate a status condition) to either a register (DTIR instruction) or memory location contained in a register (DTIM instruction).
- On a GA-16/220 only, input a data word entered via console switches to either a register (RCSR instruction) or to a memory location specified in a register (RCSM instruction).

The purpose of this section is to describe the general coding of the I/O instructions. Device-specific I/O is treated in Section 6 (and in controller manuals) which also describes MHSDC and DMA transfers. Inhibitible (IN) interrupts which are closely related to I/O operation are described in Section 2.6.

All instructions in the programmed input/output group have the following machine instruction format:



466-4-82

The CAP-16 source statement format for the Programmed Input/Output instructions is as follows:

<u>Command</u>	<u>Parameters</u>
CTRL	function, device address
DTIM } DTIR } DTOM } DTOR }	register, device address
RCSR } RCSM }	register
TEST	function, device address

where:

- register specifies one of the general-purpose registers for data in/out instructions (DTOR, DTIR, DTOM, DTIM, RCSR, RCSM).
- function specifies one of eight control functions.
or
specifies one of eight tests.
- device address specifies one of 64 possible device addresses. Device address X'3E' is implied for RCSR and RCSM. (RCSR and RCSM are described in Section 4.15.1 and 4.15.2.)

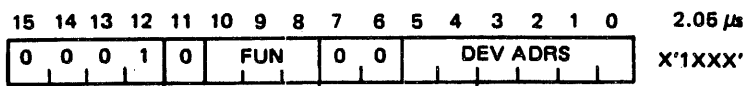
For a data word to/from a register, the destination/source register is specified in bits 10 to 8. For a data word to/from a memory location, the contents of the register specified is taken as the address of the memory location.

The instructions presented in this section may be used to control any peripheral interfaced to the GA-16/110/220. A peripheral may constitute one or more devices and each device has a unique device select code or combination of codes. The individual device controller is responsible for recognizing its own select code(s) and interpreting and performing the required functions when it is selected. For example, TEST 2 sent to device X is a hardware function of X and may not be the same operation as TEST 2 sent to device Y.

When the CPU executes an I/O instruction, it outputs the device select code (DEV ADRS) specified in the instruction, together with control signals or data appropriate to the instruction via the I/O bus. The specified unit, having recognized its device select code, responds with signals (and sometimes data) that are sent back to the CPU via the I/O bus.

The following subsections describe each of the programming I/O instructions with general examples. Additional examples specific to particular devices are included in Section 6, which summarizes individual peripheral controller instructions and data.

4.14.1 OUTPUT CONTROL FUNCTION (CTRL)



One of eight possible control functions (specified by the FUN bits) is decoded by the device controller addressed. Typically, a control function may be used to put a device in a particular mode (e.g., transmit or receive) or to enable/disable interrupt masks within a device controller.

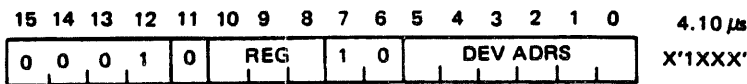
Instruction \rightarrow I/O BUS OUTPUT LINES (controller decodes DEV ADRS and FUN bits)
 CNTL- \rightarrow I/O Bus (for more detail see Section 6)

Indicators affected: None
 Registers affected: None

If the control function 0 enables interrupts and device X'08' is the paper tape reader controller:

CTRL 0,X'08' Execution of this instruction will enable interrupts from the paper tape read controller when certain conditions in the paper tape reader are satisfied.

4.14.2 DATA TRANSFER IN TO MEMORY (DTIM)



508-4-52

The contents of the specified register are used as the address of the memory location whose contents are replaced by a word of data from the addressed device.

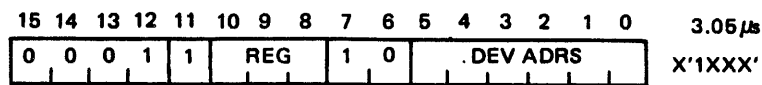
I/O BUS DATA INPUT LINES \rightarrow *R₁₅₋₀

Indicators affected: None
 Registers affected: None

Example:

DTIM C,X'17' Register C contains the address of the memory location whose contents will be replaced by data read from the device controller whose device select code is X'17'.

4.14.3 DATA TRANSFER IN TO REGISTER (DTIR)



A word of data from the addressed device controller replaces the contents of the specified register. If the selected device controller inputs a word of less than 16 bits, the extra bits will contain zeros.

I/O BUS DATA INPUT LINES \rightarrow R₁₅₋₀

Indicators affected: None

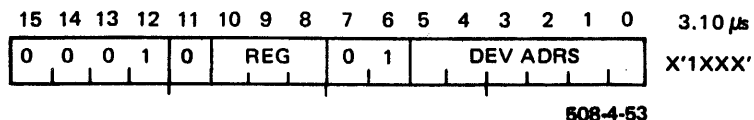
Registers affected: Specified Register

Example:

DTIR B, X'21'

A word of data from the device controller whose select code is X'21' is input to register B.

4.14.4 DATA TRANSFER OUT FROM MEMORY (DTOM)



The register specified contains the address of the memory location from which a word of data will be output to the address device.

*R₁₅₋₀ \rightarrow I/O BUS DATA OUTPUT LINES

Indicators affected: None

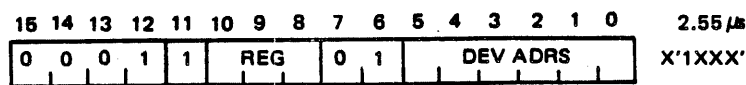
Registers affected: None

Example:

DTOM B, X'12'

Register B contains the address of the memory location whose contents are output to the device controller whose select code is X'12'.

4.14.5 DATA TRANSFER OUT FROM REGISTER (DTOR)



The contents of the specified register are output to the addressed device controller.

R_{15-0} \rightarrow I/O BUS DATA OUTPUT LINES

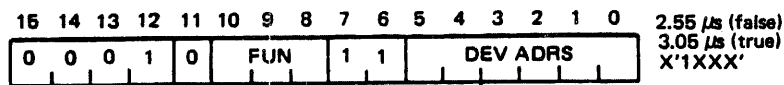
Indicators affected: None
Registers affected: None

Example:

DTOR A,X'10'

The contents of register A are output to the device controller whose select code is X'10'.

4.14.6 TEST DEVICE (TEST)



508-4-54

One of the eight possible tests (specified by the FUN bits) is performed by the device controller selected. The result is returned via the TEST- line. If the test condition is met, the next instruction will be skipped.

Instruction \rightarrow I/O Bus Output Lines (controller decodes DEV ADRS and FUN bits)
CPU \leftarrow TEST- (for more detail see Section 6)

If the test condition is not met (TEST-line low) no skip is made. The test functions performed might be typically labeled "ready," "operation complete," "error," etc., and could be for a condition in the peripheral device or within the device controller.

TEST FALSE P+1 \rightarrow P
TEST TRUE P+2 \rightarrow P

Certain conditions which may be tested via the TEST instruction may also generate an inhibitable interrupt when the condition is satisfied. The capability to generate an IN interrupt depends on the design of the controller. A controller designed for interrupt capability usually provides a means of enabling or disabling interrupts by means of a CTRL instruction or by setting an internal interrupt mask word. The interrupt system enable (ISE) flip-flop in the CPU must also be set. In this way either test driven or interrupt driven I/O may be selected under program control in

accordance with a software design philosophy. When a controller's interrupt capability is enabled, an interrupt routine must be provided which begins at an address contained in the interrupt vector for the controller. When the condition is satisfied the following occurs:

P→E; ISE→S₁₅

(interrupt vector)→P

When several conditions may cause an interrupt a series of TEST instructions may be used in the interrupt routine to determine which condition caused the interrupt. Alternatively, depending on the controller design, it may be necessary to input a status word via a DTIR or DTIM instruction.

Example:

For a device controller whose address is X'8' (high-speed paper tape reader), test condition 0 corresponds to READY and means data is ready for input. The following sequence first inhibits interrupt capability, then repeatedly tests for ready. When ready is true, data is input to register B.

INH		Inhibit interrupt system.
TEST	0,X'8'	Test for ready.
JMP	\$-1	If test false, continue testing.
DTIR	B,X'8'	Input data if test is true.

4.15 SPECIAL FEATURES

Features on the GA-16/110 and GA-16/220 deserving special mention are:

- Read Console Switches (GA-16/220 only)
- Internal Mask Words (GA-16/220 only)
- Console Interrupt (GA-16/220 only)
- Real-Time Clock (GA-16/220 only)
- Memory Mode Change (Jumper on GA-16/110; program control on GA-16/220)
- Teletype Controller (GA-16/110 external; GA-16/220 internal)
- Operations Monitor Alarm
- Power Fail Interrupt
- Automatic Restart Interrupt

NOTE

Two special instructions are provided for the GA-16/220 which work in conjunction with the SCI program. They are I/O reset and enable single-step interrupt, described in Section 4.17.

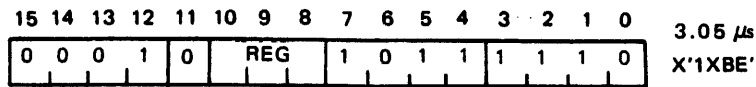
4.15.1 READ CONSOLE SWITCHES (GA-16/220)

These instructions are used to read the GA-16/220 console switch register into a memory location specified by a register (RCSM) or into a register (RCSR). Console data entry switches are used to enter the data as described in Section 3.3, Table 3-3, ⑤ and ⑥. Placing a switch in the "OPEN" position sets the corresponding bit=0. Placing the switch in the other position sets bit=1. A program using these instructions will usually contain routines to decode the data and perform some operation based upon the contents of the data. These instructions may be executed on a GA-16/110 without creating an error condition; however, without the data entry switches, no effect would be obtained.

NOTE

- *These instructions are an alternative to DTIM and DTIR addressed to device address X'3E' as described in Section 4.14.*
- *On a GA-16/220 with SCI, the subroutine at location X'FC73' (Table 3-5) which inputs four hexadecimal characters via TTY could be called from a user's program, if timing permits and registers A, B, and C can be used. D register contents must be saved then set to X'FE00' before subroutine call.*

4.15.1.1 Read Console Switches Into Memory (RCSM)



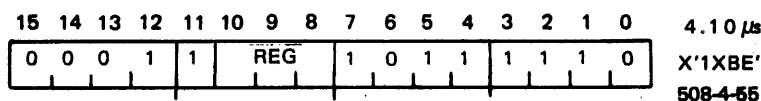
The contents of the console switches on the microconsole are loaded into the memory address indicated in the register specified. The device address is always X'3E'.

(CNSL SW)₁₅₋₀ → (*R)₁₅₋₀

Indicators affected: None

Registers affected: None

4.15.1.2 Read Console Switches Into Register (RCSR)



The contents of the console switches on the microconsole are loaded into the register specified. The device address is always X'3E'.

(CNSL SW)₁₅₋₀ R₁₅₋₀

Indicators affected: None

Registers affected: Specified register

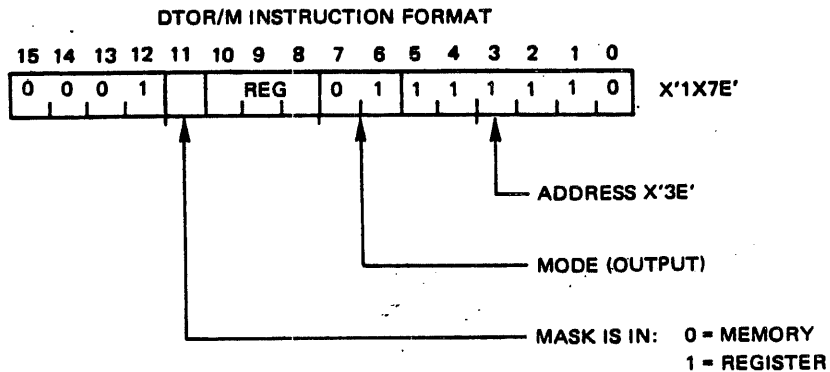
4.15.2 INTERNAL MASK WORDS (GA-16/220)

PIO control commands (CTRL 7 to 0) are normally used to enable or disable IN interrupt masks for peripheral devices with an external controller housed in the I/O chassis. However, three IN interrupts have their logic mechanized internal to the CPU and are controlled by outputting (DTOR/DTOM) a mask word to a mask register. The three internal interrupts are:

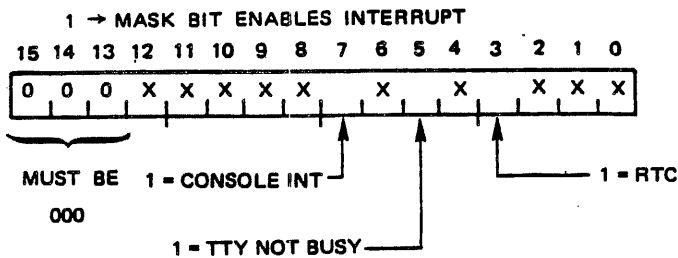
- Console Interrupt (CNSL INT)
- Real-Time Clock (RTC)
- Teletype (TTY)

An additional mask word can be output to select the 32K or 64K memory mode. The appropriate mask word is loaded into a register or memory location and output (DTOR/DTOM) to device X'3E' (the mask register). A one will enable and a zero will disable the mask or effect a mode change. Figure 4-6 shows the DTOR and DTOM instruction format, and the internal interrupt and the memory mode mask words.

ASSEMBLER MNEMONICS: DTOR R, X'3E'
DTOM R, X'3E'

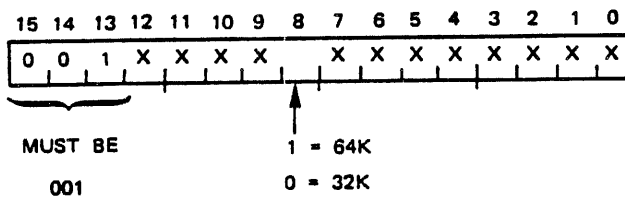


INTERNAL INT. MASK WORD



NOTE: POWER UP DISABLES ALL INTERNAL INTERRUPTS (ISE = 0) AND RESETS INTERNAL MASK BITS

MEMORY MODE MASK WORD



NOTE: PROVIDED 32K ↔ PROG SWITCH IS IN PROG POSITION;
SYSTEM RESET SETS 32K MODE.
AN IPL SETS 32K MODE (BUT THE IPL ROM IN THE SCI
SETS IT BACK TO 64K)
IN AND NI INTERRUPTS SET 64K MODE

486-4-87

Figure 4-6. Internal Mask Words (GA-16/220 Only)

4.15.2.1 Console Interrupt (CNSL INT)

A console interrupt switch is mounted on the CPU-2 module of the GA-16/220 (see Section 3.3, CNSL INT switch). This switch allows an operator to generate an inhibitable interrupt from the front panel. The conditions necessary to generate this interrupt are as follows:

- ISE set = 1 (see RISE, INE, RTRN and RTNIV instructions)
- CNSL INT bit in internal mask word set = 1

The console interrupt uses dedicated memory location X'47' for its transfer vector.

4.15.2.2 Real-Time Clock Interrupt (RTC)

This feature provides a periodic interrupt (one millisecond) when the interrupt mask from the RTC is enabled. An operating system (such as RTOS-16) may use this interrupt to simulate a real-time clock for multiprogramming applications. The interrupt may also be used to protect against unusual conditions by periodically returning control to a monitor program.

The RTC interrupt can be enabled or disabled under program control. The state of the interrupt mask for the RTC is controlled by the internal mask word (device select code X'3E'), which is illustrated in Figure 4-6.

The RTC derives its time base from the internal crystal-controlled master clock. Every one millisecond, an RTC interrupt is requested if this interrupt level and the master interrupt system (ISE) are enabled. The interrupt forces a JSR indirect through memory location X'43'. Location X'43' must contain the address of the interrupt service routine for the RTC.

NOTE

The first RTC interrupt will occur immediately after enabling the RTC. The next interrupt will occur from 0 to 1 ms after ISE is re-enabled after return from processing the first interrupt. Thereafter RTC interrupts will occur at 1 ms intervals. From this it follows that two interrupts are required after enabling before the RTC is synchronized.

CAUTION

The RTC mask bit should not be turned off while ISE is on.

4.15.2.3 Teletype Interrupt

The GA-16/220 CPU contains a built-in serial I/O controller for a teletype. When the TTY BUSY bit in the mask word is set, it enables the interrupt capability for the TTY controller. When the controller has a byte of data (input) or is ready to receive a byte of data (output), NOT BUSY is true; and interrupt occurs, which forces a JSR indirect through dedicated memory location X'45' to the address of the routine to service the TTY interrupt. If the TTY BUSY bit in the mask word is reset, the interrupt is disabled, and NOT BUSY true causes a branch to P+2 (when P is address of TEST 0, X'3F' instruction). Further detail on the teletype controller is described in Section 4.15.3.

4.15.2.4 Memory Mode Change Mask

Both the GA-16/110 and 220 have two basic addressing modes: 32K and 64K. The 32K mode operates identically with GA's SPC-16/40 series computers and, therefore, allows for software compatibility. The 64K mode allows program execution in 64K of memory by making the P counter a full 16 bits (ISE is removed from P₁₅). Many instructions and their functions are affected by the mode change as tabulated in Table 4-5. If the 32K mode is used, only 32K address translations will take place; i.e., the upper half of memory range (32K to 64K) will not be addressed.

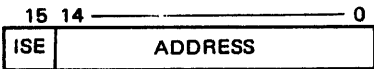
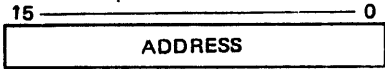
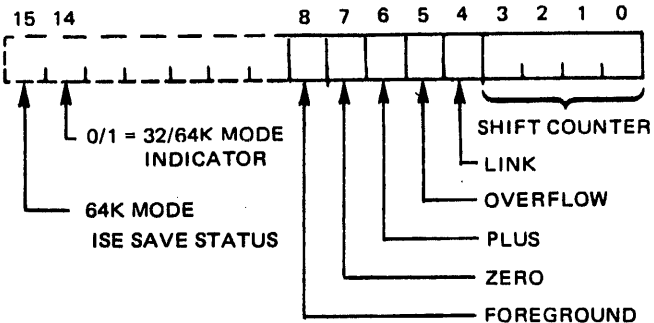
On a GA-16/110, memory mode is set by a jumper (see Section 4.15.7) while a GA-16/220 changes mode in accord with the following: an IPL places the CPU in 32K mode. It is placed in 64K mode by either NI or IN interrupts. Under program control, the CPU can be placed in either 32K or 64K mode by outputting (DTOR/M) the memory mode mask word to address X'3E'. This sets a 32K or 64K mode flip-flop. To activate the mode, it is necessary to execute an RTNIV instruction (see Section 4.9). If a DTOR/M has been executed to place the CPU in a 32K mode, and an IN or NI interrupt occurs before the RTNIV has been executed, the CPU will remain in the 64K mode. In returning to the interrupted program, it is not necessary to re-execute another DTOR/M command. Executing an RTNIV will activate the mode selected by the DTOR/M that was issued prior to the interrupt.

The memory mode indicator is part of the status register (S₁₄). Although CPU instructions (LARS and TRS) can load register S, they do not load the memory mode indicator nor change the memory mode.

4.15.3 SERIAL I/O CONTROLLER

The GA-16/220 has a built-in serial I/O controller for communicating directly with a teletype device such as the ASR-33 Teletype while a GA-16/110 may provide equivalent capability by installing a Model 1582 TTY Controller. The ASR Model 33 Teletype (TTY) has a keyboard, printer, paper tape reader and paper tape punch. The teletype can be operated independently (LOCAL mode) or it can be used on-line (LINE mode) to input or output data to the CPU. The maximum data transmission rate is 110 baud, approximately ten characters per second. The teletype controller can interrupt when it requires service, or an executing program can test the teletype to see if it is ready to be serviced. In LOCAL mode the teletype reads or punches standard ASCII character codes. These codes are compatible with other peripheral units which handle eight-channel paper tape. In LINE mode, any eight-level code may be read or punched.

Table 4-5. Effects of Memory Mode Change

Operation	32K Mode	64K Mode
JSR Instruction or IN Interrupt	$P_{14-0} \rightarrow E_{14-0}$ $ISE \rightarrow E_{15}$ and S_{15}	$P_{15-0} \rightarrow E_{15-0}$ $ISE \rightarrow S_{15}$
NI Interrupt	$P_{14-0} \rightarrow (ADDR)^1$ $ISE \rightarrow (ADDR+1)_{15}$	$P_{15-0} \rightarrow (ADDR)^1$ $ISE \rightarrow (ADDR+1)_{15}$
RTRN Instruction	$R_{15} \rightarrow ISE$ $R_{14-0} \rightarrow P_{14-0}$	$S_{15} \rightarrow ISE$ $R_{15-0} \rightarrow P_{15-0}$
RISE Instruction	$R_{15} \rightarrow ISE$	$S_{15} \rightarrow ISE$
RTNIV Instruction	$((EA)+1)_{15} \rightarrow ISE$ $((EA))_{15-0} \rightarrow P_{15-0}$ (Bit 15 is forced to 0)	$((EA)+1)_{15} \rightarrow ISE$ $EA_{15-0} \rightarrow P_{15-0}$
EXIT Instruction	$R_{14-0} \rightarrow P_{14-0}$ ISE unchanged	$R_{15-0} \rightarrow P_{15-0}$ ISE unchanged
P Register (P) Contents		
Status Register (S) Contents	<p>Same in both modes except bits 14 and 15 as shown:</p> 	

¹ Each NI interrupt has a pair of low core locations (beginning at ADDR) dedicated for storage of register P and ISE status. Refer to Section 4.9 and Table 4-4.

As an alternative to teletype such as the model 33, a teletype-compatible CRT terminal such as GA model 3381 video display terminal may be used at 110 or 9600 baud (baud rates are selected by baud select switch on the CPU-2 module, Section 3.3, Table 3-3,

⑧.

The controller for the console teletype is housed in the computer chassis as an internal serial/parallel and parallel/serial converter operating on the I/O-Bus. This internal controller operates in an identical fashion as an external Model 1582 controller. In addition to the internal TTY controller, there is a TTY paddleboard mounted external to the CPU. It provides electrical isolation and a 20-ma, 3-wire current loop between the CPU and the TTY. (An RS-232C paddleboard is also available.)

The operational status of the console teletype is reflected by the state of one indicator on the TTY controller: TTY NOT BUSY. One test instruction code has been assigned to interrogate the state of TTY NOT BUSY. To enable or disable the TTY interrupt, see Section 4.15.2.3. The mask word is implemented in the Model 1582 TTY Controller when used on a GA-16/110.

TTY NOT BUSY is a service request; it is true whenever the TTY controller is "not busy" transmitting a character to or from the teletype. When TTY NOT BUSY changes from false to true, an interrupt flip-flop on the TTY controller is set, thereby requesting an interrupt if the TTY interrupt mask is enabled. Upon receiving acknowledgement of the interrupt from the CPU, the interrupt flip-flop is cleared. TTY NOT BUSY then goes false when the controller's character buffer has been filled by data from the teletype or from the CPU (depending on the operating mode). TTY NOT BUSY also goes false when the teletype's operating mode is changed.

The teletype controller operates in four modes (summarized in Table 4-6):

- TRANSMIT
- RECEIVE ONLY
- RECEIVE AND ECHO
- BREAK

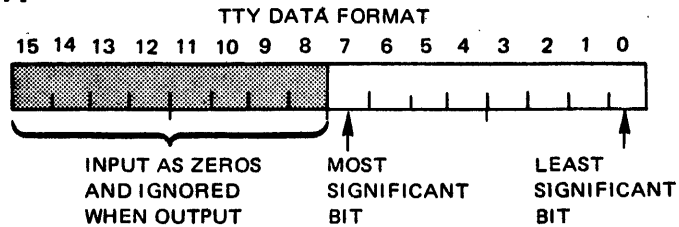
Mode selection is accomplished by four control (CTRL) instructions. Additionally, when the CPU is initialized, the console teletype is placed into the RECEIVE ONLY mode in a "busy" state (NOT BUSY false). The mode can be changed by executing a CTRL instruction.

During TRANSMIT mode operation, the controller sends CPU data to the teletype's printer and/or paper tape punch. After the data is sent to the teletype (via a DTOR/DTOM instruction), TTY NOT BUSY goes true and initiates another service request.

During RECEIVE ONLY mode operation, the controller sends teletype data to the CPU. During RECEIVE & ECHO operation, this data is also sent to the teletype printer and/or punch. In either of these receive modes, a TTY NOT BUSY service request is initiated after the teletype has sent a character to the controller. The service request is removed when the character has been input.

The BREAK mode is intended to be a "transmission ended" mode, during which a timer may cause the teletype to shut off if the mode is not changed within a predetermined time period.

Any of the four data in/out instructions (DTOR, DTIR, DTOM, DTIM) may be used with the teletype device select code X'3F' to transfer byte data between the computer and the console teletype. The following data format is used:



466-4-88

Table 4-6. TTY Modes

Mode	Description	To Remove Service Request	Specific Rules
TRANSMIT	Transmit to printer and/or punch	<ol style="list-style-type: none"> 1. Output character 2. Change mode 	<ol style="list-style-type: none"> 1. New character starts when DTOR or DTOM occurs. 2. Service request occurs when not busy.
RCV ONLY	Teletype input to processor buffer only	<ol style="list-style-type: none"> 1. Input character 2. Receive subsequent sync character 	<ol style="list-style-type: none"> 1. New character starts when not busy and mark-to-space transition occurs.
RCV & ECHO	<p>Teletype input to processor buffer and to teletype printer and/or punch</p> <p>Characters are printed/punched as they are struck on the keyboard independently of input to CPU.</p>	<ol style="list-style-type: none"> 3. Change from one receive mode to another receive mode 	<ol style="list-style-type: none"> 1. Mark=current flow Mark=logical one. Space no current flow=logical zero 2. To avoid data overrun, character must be input within 4 ms after service request. 3. Service request occurs when not busy and character has been received.
BREAK	<p>Transmit line to teletype is forced to quiescent condition (space=open=zero). This mode remains active until system is reset or a different mode is invoked. Transmit mode must have been established to have BREAK mode effective.</p>		

The console teletype I/O instructions are summarized in Table 4-7. The mode (initialized to RECEIVE ONLY) can be changed by executing a CTRL instruction.

Table 4-7. Instruction Summary for Serial I/O Controller

Device Select Code = X'3F'; Interrupt Vector = X'45';
Data Channel Locations = Not Applicable

Instruction			Function
10FF	TEST	0,X'3F'	Skip if TTY NOT BUSY is true
103F	CTRL	0,X'3F'	TRANSMIT mode
123F	CTRL	2,X'3F'	RECEIVE ONLY mode
143F	CTRL	4,X'3F'	RECEIVE & ECHO mode
103F	CTRL	6,X'3F'	BREAK mode
①18BF	DTIR	A,X'3F'	Input character byte to register A
①10BF	DTIM	A,X'3F'	Input character byte to memory (location contained in register A)
①187F	DTOR	A,X'3F'	Output character byte from register A
①107F	DTOM	A,X'3F'	Output character byte from memory (location contained in register A)

① These instructions shown with register A used. Register A could be replaced by a X,Y,Z,B,C,D, or E.

A working program using the teletype is shown in Figure 4-7. This program operates with TTY busy interrupt inhibited. (The DSPL instruction produces no observable result.)

```

0001      TITLE 'TTY EXERCISE PROGRAM'
0002      DATE '04/01/76'
0003      PRINT X'27'
0000      0004      DSECT
0000      0005      TYBUF DS      25          ALLOCATE CHAR BUFFER - 50 BYTES
0000      0019      0006      DSIZE EQU      $-TYBUF
0000      0007      PSECT
0000      0008      DS      DSIZE
0019      01D5      0009      BEGIN LDV      D,TYBUF      GET BUFFER ADDRESS
001A      0000      D
001B      143F      0010      CTRL      4,X'3F'      TTY IN RCV ECHO MODE
001C      0620      0011      ZERO      X
001D      0600      0012      ZERO      A          CLEAR REGISTER A
001E      011D      0013      ORV      A,X'0080'
001F      0080
0020      187E      0014      DTOR      A,X'3E'      TURN OFF TTY INTERRUPT
0021      0602      0015      INCH     ZLBY      A          CLEAR UNUSED BYTE AND BEGIN INPUT
0022      0106      0016      SUBVC    A,X'87'      TEST FOR BELL
0023      0087
0024      2C11      0017      SKZ      OUTY      BELL, DO OUTPUT
0025      0126      0018      SUBVC    X,X'32'      TEST FOR INDEX=50
0026      0032
0027      2C0E      0019      SKZ      OUTY      LAST CHARACTER, DO OUTPUT
0028      0106      0020      SUBVC    A,X'8D'      TEST FOR CR
0029      008D
002A      2C0B      0021      SKZ      OUTY      CR, DO OUTPUT
002B      10FF      0022      TEST      0,X'3F'      TEST FOR BUSY
002C      73FE      0023      JMP      $-1
002D      18BF      0024      DTIR      A,X'3F'      INPUT CHARACTER
002E      9900      0025      STBY      A,TYBUF,X      PLACE IN MEMORY BYTE
002F      072E      0026      INCR      X          INCREMENT INDEX
0030      0680      0027      ZERO      B          OPTIONAL CODE TO DISPLAY CHAR COUNT
0031      099D      0028      OR        B,X          AND CHAR
0032      0684      0029      EXBY      B
0033      089D      0030      OR        B,A
J034      0584      0031      DSPL      B          DISPLAY COUNT AND CHAR HEX
J035      73EB      0032      JMP      INCH      GET ANOTHER CHARACTER
0033      EJECT

```

466-4-89

Figure 4-7. TTY Exercise Program (Sheet 1 of 2)

88A00508A-E

0036	103F	0034	OUTY	CTRL	O,X'3F'	TTY IN TRANSMIT
0037	0600	0035		ZERO	A	
0038	011D	0036		ORV	A,X'8D'	SET CR CHARACTER
0039	008D					
003A	10FF	0037		TEST	O,X'3F'	
003B	73FE	0038		JMP	\$-1	
003C	187F	0039		DTOR	A,X'3F'	
003D	0600	0040		ZERO	A	
003E	011D	0041		ORV	A,X'8A'	SET LF CHARACTER
003F	008A					
0040	10FF	0042		TEST	O,X'3F'	
0041	73FE	0043		JMP	\$-1	
0042	187F	0044		DTOR	A,X'3F'	OUTPUT LF
0043	0995	0045		RTR	B,X	STORE NO CHAR INPUT
0044	0620	0046		ZERO	X	
0045	10FF	0047	OUTCH	TEST	O,X'3F'	
0046	73FE	0048		JMP	\$-1	
0047	8900	0049		LDBY	A,TYBUF,X	GET CHAR FROM MEM BYTE
0048	187F	0050		DTOR	A,X'3F'	OUTPUT TO TTY
0049	072E	0051		INCR	X	INCREMENT INDEX
004A	0524	0052		DSPL	X	DISPLAY COUNT
004B	0986	0053		SUBC	B,X	
004C	2C01	0054		SKZ	DONE	YES, FINISH UP
004D	73F7	0055		JMP	OUTCH	NO, OUTPUT ANOTHER CHARACTER
004E	0195	0056	DONE	LDV	B,X'AAAA'	
004F	AAAA					
0050	0584	0057		DSPL	B	DISPLAY END PATTERN
		0058		WAIT		FINISHED
0051	0000					
	0019 P	0059		END	BEGIN	
	0000 ERRS					

```

A      0000
B      0004
BEGIN 0019 P
BI$    0003
BO$    0004
C      0005
CC$    0000
CI$    0008
COND$  0000
D      0006
DM$0   0000
DM$1   1000
DM$2   2000
DM$3   3000
DONE   004E P
DSIZE  0019
DV$0   0000
DV$1   0400
DV$2   0800
DV$3   0C00
E      0007
INCH   0021 P
IRTNS$ 8000
ISS$   0006
LBS$   0009
LOS$   0005
OMS$   0007
OUTCH  0045 P
OUTY   0036 P
SI$    0001
SL$    000A
SO$    0002
SY$    0000
TYBUF  0000 D
UNCON$0020
X      0001
Y      0002
Z      0003
$LOAD

```

Figure 4-7. TTY Exercise Program (Sheet 2 of 2)

4.15.4 OPERATIONS MONITOR ALARM (OMA)

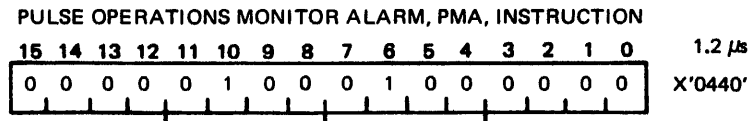
The operations monitor alarm (OMA) feature can be used to provide automatic processing shutdown if program control fails to return periodically to a system program. The OMA feature comprises an alarm counter that is initially enabled by execution of a pulse monitor alarm (PMA) instruction. The alarm counter is not armed until PMA is executed for the first time; but, thereafter, it must be periodically reset by a PMA instruction. Failure to issue a PMA instruction within 150 to 300 ms causes an OMA timeout.

If the OMA times out, the computer automatically switches from the RUN mode to the IDLE mode. In addition, the SFEC signal on the I/O bus goes false, indicating that the system is in an "unsafe" condition. This signal can be used to control an audio-visual alarm or automatic switchover.

The OMA may be cleared by an auto-restart interrupt sequence or by pressing the RESET (Table 3-3, (16)), switch on the microconsole. When cleared manually with RESET, the 220 will either enter the auto-restart routine or enter the console ROM program. See description of RESET (16) and (on a 220) CNSL (15) in Table 3-5.

The OMA indicator on the front panel will turn on when the alarm has time out (Section 3.3, Table 3-3, (19)).

The PMA instruction is classified as a control instruction described in Section 4.13.7 Its format is as follows:



466-4-90

4.15.5 NI POWER-FAIL INTERRUPT

When the primary AC input to the GA-16/220 drops below a predetermined value (less than 105 vac under full load), the Power-Fail Detect circuit request a Power-Fail Interrupt via the power-fail interrupt transfer vector at location 40. The regulated direct current voltages are guaranteed to be good at this time. This interrupt is of highest priority and cannot be inhibited.

At the same time that the interrupt is requested, the power fail timer is initialized to time out in 90 - 150 μ s. Therefore, after the power-fail interrupt occurs, the processing routine has 90 - 150 μ s to execute. After 90 - 150 μ s the power fail timer times out and the CPU goes to idle; the SFEC line indicates a safe condition and SYRT- is on.

4.15.6 AUTOMATIC RESTART INTERRUPT

When the computer is powered up and the primary AC power input goes above a predetermined value (guaranteed not to be greater than 105 vac), a powering-up sequence is initiated. During the powering-up sequence, the entire system is initialized. Initialization synchronizes the CPU timing, turns off ISE, and resets the mask register for RTC, CI, and TTY. The interrupt then occurs. If the 32K↔PROG switch is in the PROG position, the CPU will be set to 64K mode by the interrupt, but may be forced back to 32K by pressing RESET or by pressing IPL switch. (If IPL ROM or SCI is entered, it will set the system back to 64K.) Location X'41' has been assigned to the auto-restart's interrupt vector. This location must contain the address of the routine responsible for servicing the auto-restart interrupt.

4.15.7 GA-16/110 MEMORY MODE SET

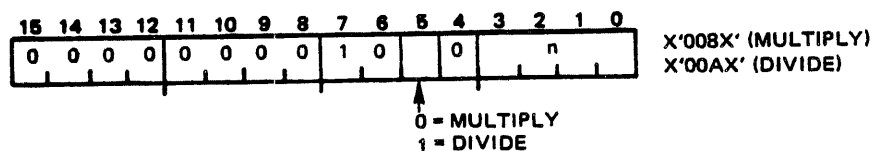
Memory mode in a GA-16/110 is set by means of the +64KM line, accessible via the remote line connector (J20 on a compact MIB, J27 on a jumbo MIB, or pin 90 on the GA-16/110 module itself). The mode setting is as follows:

64K mode	+64KM ungrounded
32K mode	+64KM to GND

Refer to Section 4.15.2.4 for description of the effects of memory mode on operation of the system.

4.16 HARDWARE MULTIPLY AND DIVIDE

Multiplication and division are performed by hardware circuitry. These instructions operate on fixed-point, positive numbers only. Hardware multiply and divide instructions are of the following format:



466-4-91

Indicators affected: Link
 Registers affected: B,C

The CAP-16 source-statement format for the hardware multiply and divide instructions is as follows:

Command	Parameters
MPY } DIV }	n

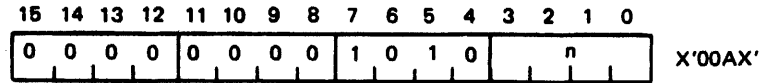
where:

n is a positive integer, contained in bits 3 to 0 as *n*. *n* determines the number of shift/add operations in multiplication or the number of shift/subtract operations in division. A CAP-16 source statement of "MPY 10" will assemble as X'8A'. The statement "DIV 6" will assemble as X'A6'.

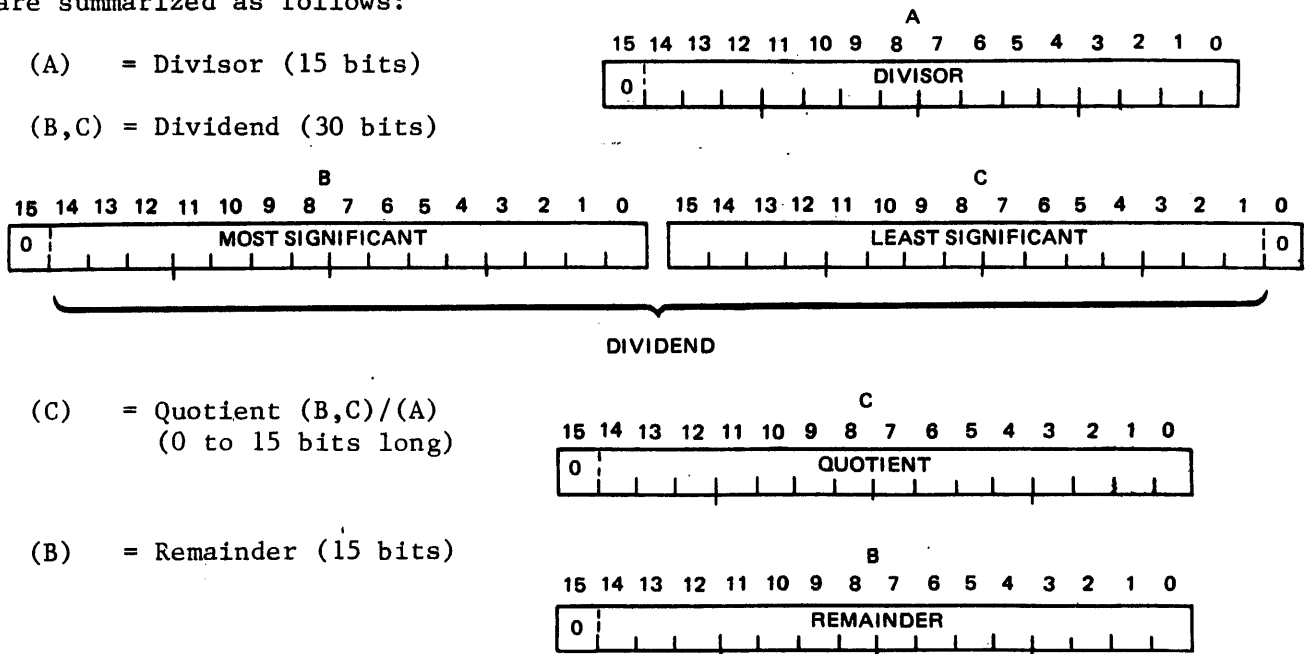
NOTE

If n is omitted, n=15 is assumed.

4.16.1 HARDWARE DIVIDE (DIV)



This instruction divides a double-word dividend taken as the contents of registers B and C by the contents of register A. The quotient is stored in register C, and register B holds the remainder. The number in register B before division must be less than the number in register A. The contents of the source/destination registers are summarized as follows:



243-3-60

All 15 bits of the divisor (register A) are used. Bit 15 of register A must be zero. During the divide operation, the bits of the dividend are shifted (left) as a unit; for example, after one shift, bit 15 of register C is moved left one place to bit 0 of register B. With each shift/subtract, one digit of the quotient is formed (at C₀). The number of quotient digits (n) must be large enough so that n shifts will remove all one bits of the dividend from register C. Bits 15 of register B and 0 of register C must both be zero prior to the divide operation. Upon completion of the division, the quotient is contained in register C and the remainder is in register B. Bit 15 of both registers will be zero.

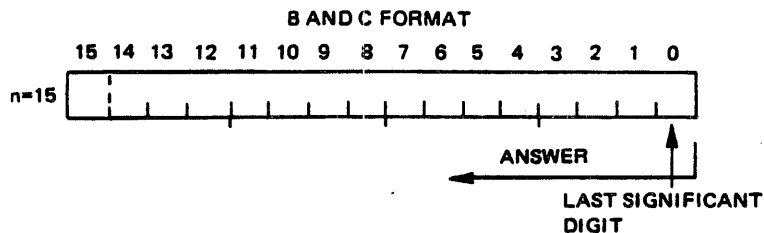
The remainder is formed such that after division, multiplying the quotient by the divisor and adding the remainder, with any carry bit out of C added to B, will result in the original dividend in B and C. The link indicator will be set to the status of bit (15-n) of register C at the end of the division.

The instruction time required for the divide operation is given by:

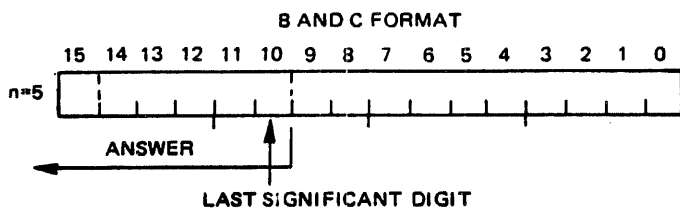
$3.50 \mu\text{s}$ plus $2.00 \mu\text{s}$ times number of bits (n) in the divisor

The 15-bit divisor can represent integers in the range 0 to X'7FFF' and may include 0 to 15 fractional digits. The 30-bit dividend can represent integers in the range 0 to X'3FFFFFFF' and may include 0 to 30 fractional digits. Locating the binary point in the answer depends upon two things:

1. The location of the binary points in the divisor and dividend. As in decimal division, the number of fractional digits in the quotient can be found by subtracting the number of fractional digits in the divisor from the number of fractional digits in the dividend, and the remainder has the same number of fractional digits as the dividend (e.g., decimal $0.0649 \div 0.03 = 2.16$, remainder of 0.001).
2. The value of n. This determines where the quotient and remainder lie within the 15 digits of the respective answer (registers C and B). With $n=15$, 15 digits of the quotient are considered significant, with the last significant digit of the quotient in bit 0 of register C and the last significant digit of the remainder in bit 0 of register B.



For each decrement of n, the last significant digit is to the left one bit.
For example:



243-3-83

The binary point in the quotient can be determined either as a count of the number of integer digits or the number of fractional digits.

The following formula calculates the binary point relative to the high-order bit of the quotient (bit 14 of register C):

$$BP_C = BP_{B,C} - BP_A - (n-15)$$

where:

$BP_{B,C}$ is the number of integer digits in the dividend.

BP_A is the number of integer digits in the divisor.

BP_C is the number of integer digits in the quotient. ($15 - BP_C$ would be the number of fractional digits.)

$(n-15)$ is the number of trailing zeros. (When $n=15$, this term is zero, indicating that the last significant digit of the result is bit 0 of register C.)

The binary point in the remainder is calculated as follows:

$$BP_B = BP_{B,C} - n$$

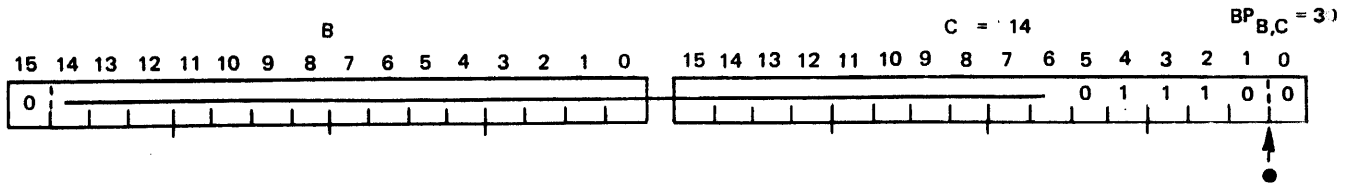
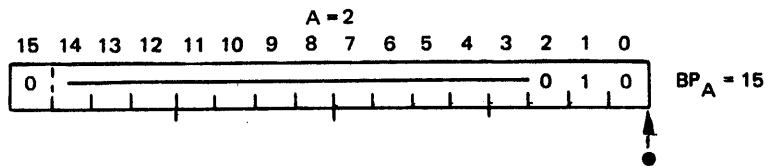
where:

$BP_{B,C}$ is the number of integer digits in the dividend.

BP_B is the number of integer digits in the remainder.

Examples:

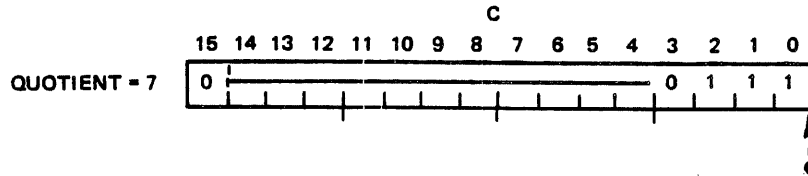
- Given the following bit configurations prior to division:



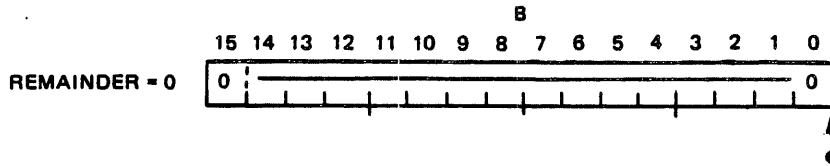
243-4-84

DIV 15 Divide the contents of registers B and C by the contents of register A, (or DIV) using all 15 bits of the dividend in register C.

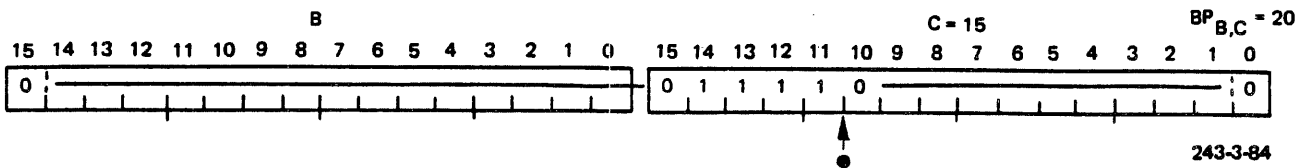
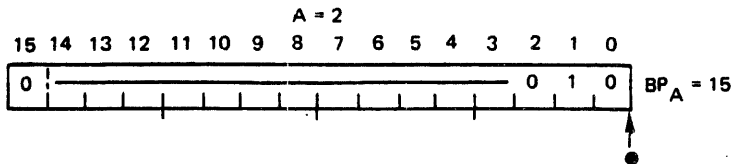
Result: $BP_C = BP_{B,C} - BP_A (n-15)$
 $= 30 - 15 - (15 - 15)$
 $= 15$



$BP_B = BP_{B,C} - n$
 $= 30 - 15$
 $= 15$

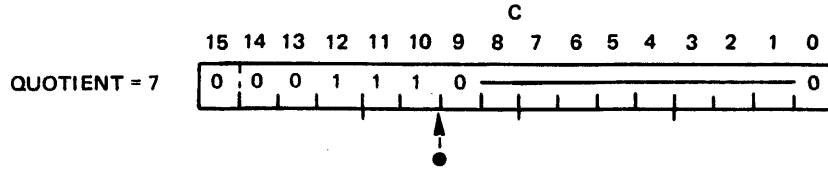


2. Given the following bit configurations prior to division:

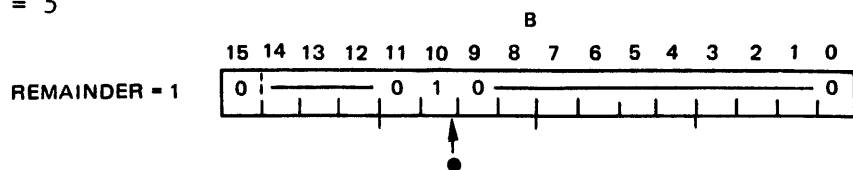


- a. DIV 15 Divide the contents of registers B and C by the contents of (or DIV) register A, using all 15 bits of the dividend in register C.

Result: $BP_C = BP_{B,C} - BP_A - (n-15)$
 $= 20-15-(15-15)$
 $= 5$

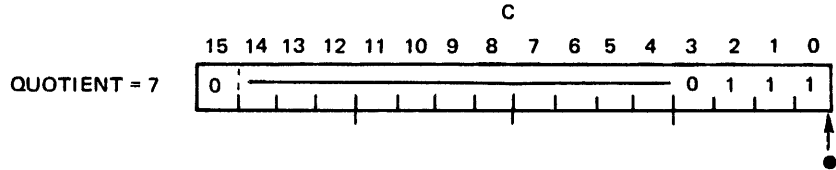


$BP_B = BP_{B,C} - n$
 $= 20-15$
 $= 5$

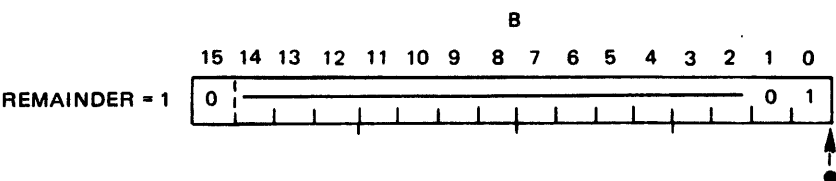


- b. DIV 5 Divide the contents of registers B and C by the contents of register A, using the five high-order bits of register C.

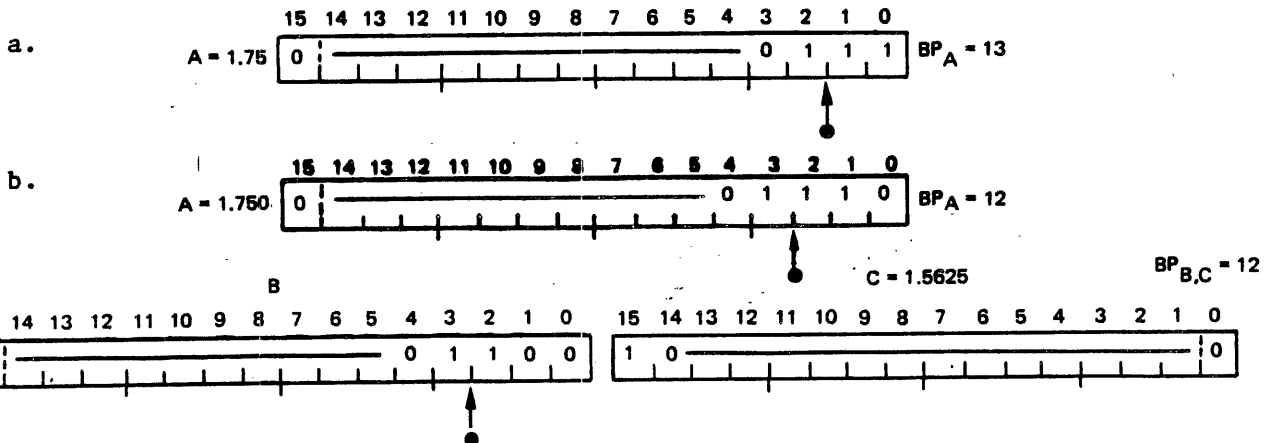
Result: $BP_C = BP_{B,C} - BP_A - (n-15)$
 $= 20-15-(5-15)$
 $= 15$



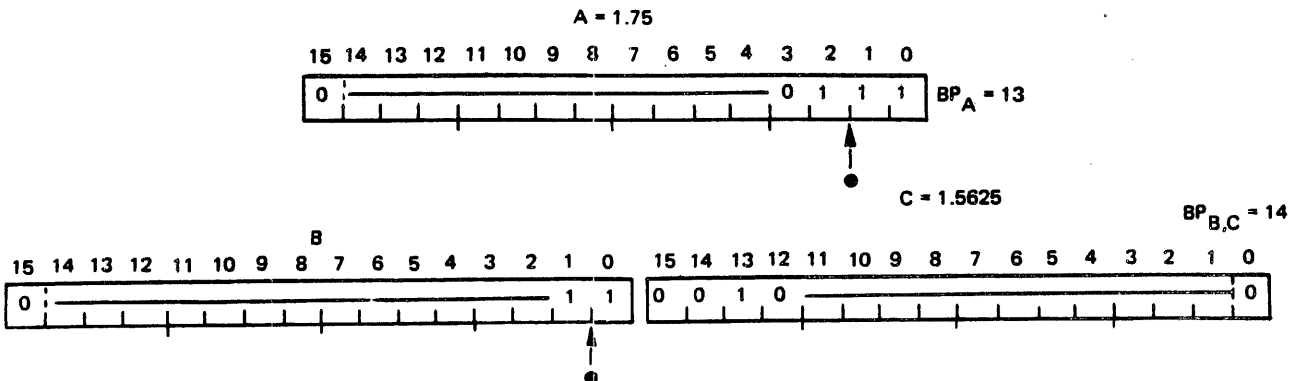
$BP_B = BP_{B,C} - n$
 $= 20-5$
 $= 15$



3. Given the following bit configuration prior to any attempted division, the result will be meaningless. The value in register B must be less than the value in register A prior to division (regardless of the values determined by the binary points). Attempting to perform the division with register B initially containing a value greater than the value contained in register A is equivalent to omitting the computation of the first significant digits of the answer. The way to avoid this problem is to place the decimal point in the divisor so that the binary value in register A is greater than that in B. This is illustrated by example b.

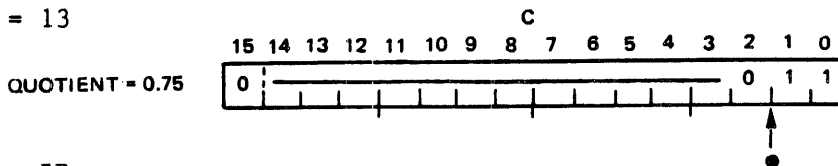


4. Given the following bit configuration prior to division:

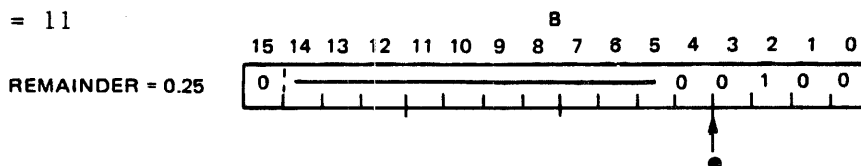


DIV 3 Divide the contents of registers B and C by the contents of register A, using the three high-order bits of register C.

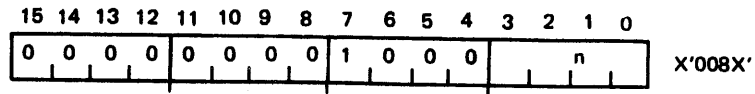
Result: $BP_C = BP_{B,C} - BP_A - (n-15)$
 $= 13$



$BP_B = BP_{B,C} - n$
 $= 11$

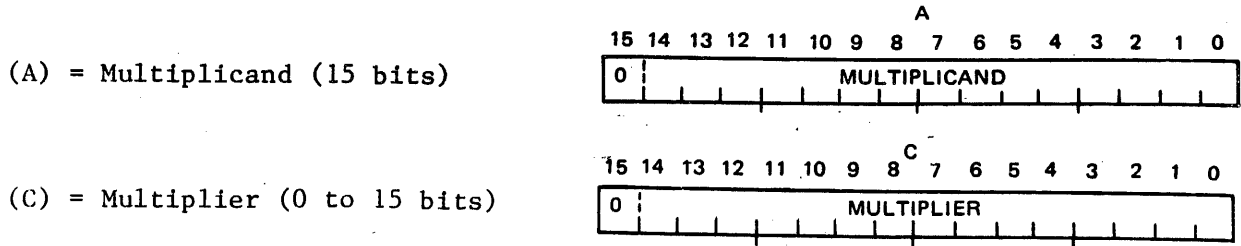


4.16.2 HARDWARE MULTIPLY (MPY)

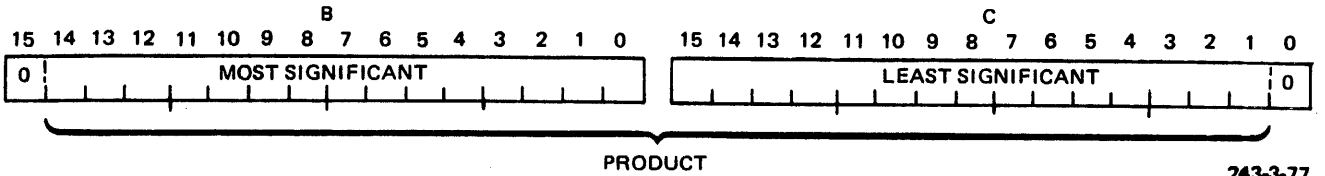


This instruction multiplies the contents of register A by the contents of register C and places the result as a double-word product in registers B and C.

The operand registers are summarized as follows:



(B,C) = Product (A) x (C), (0 to 30 bits)



243-3-77

All 15 bits of the multiplicand (register A) are used. Bit 15 of registers A and C must be zero. The answer will be contained in registers B and C, with the digit in bit 15 of register C considered as immediately following the digit in bit 0 of register B. Bit 0 of register C and bit 15 of register B will always be zero.

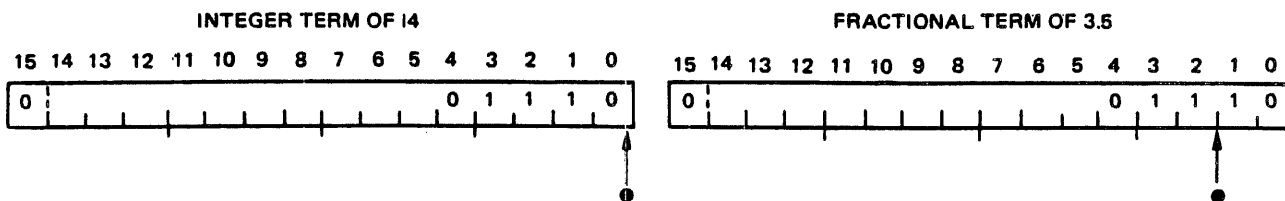
During a multiply operation, the multiplier bits are shifted (right) out of bit 0. Register C and product bits are shifted (right) into bit 15 of register C from register B. n must be large enough so that all one bits of the multiplier are shifted out of register C. After a multiply, the content of the link indicator is indeterminate.

The instruction execution time required for the multiply operation is given by:

$$1 \text{ bit MPY} = 3.50 \mu\text{s}; \text{ plus } 1.50 \mu\text{s times number of bits (n) in the multiplier plus } 0.50 \mu\text{s times number of one bits in register C}$$

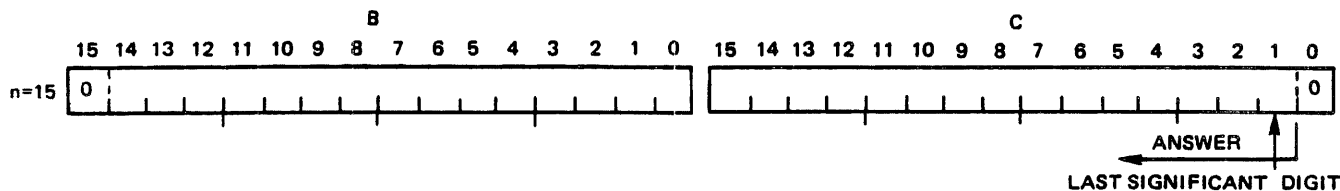
The 15-bit multiplicand and multiplier can each represent integers in the range 0 to X'7FFF'. By keeping track of a binary point, this range also includes up to 15 fractional digits, yielding a product containing from 0 to 30 fractional digits.

For an integer term, the binary point is to the right of bit 0. A term can also be considered as having a fractional part, with the binary point between the integer and fractional portions. For example:

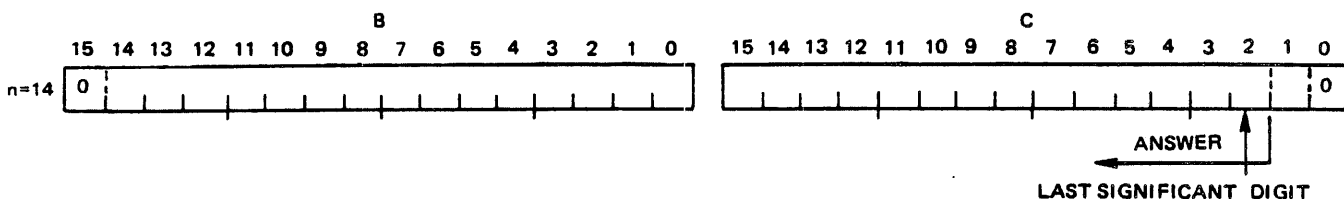


Locating the binary point in the answer depends upon two things:

1. The location of the binary points in the multiplicand and multiplier. This determines the relative location of the binary point among the product's significant digits. As in decimal multiplication, the number of fractional digits in the product is the sum of the number of fractional digits in the two terms (e.g., decimal $2.16 \times 0.03 = 0.0648$).
2. The value of n . This determines where the product lies within the 30 digits of the answer registers. With $n=15$, all leading zeros and all significant digits of the multiplier are processed, which shifts the answer to the far right:



For each decrement of n , the last significant digit is to the left one bit.
For example:



243-3-78

The binary point in the product can be located either as a certain number of bits from the high-order bit of the answer registers (bit 14 of B) or as a certain number of bits from the low-order bit of the answer registers (bit 1 of C). The high order count is the number of integer digits in the answer, including leading zero, and the low-order count is the number of fractional digits in the answer, including trailing zeros. The following formula calculates the binary point relative to the high-order bit of the answer registers:

$$BP_{B,C} = BP_A + BP_C - (15 - n)$$

where:

BP_A is the number of integer digits of the multiplicand.

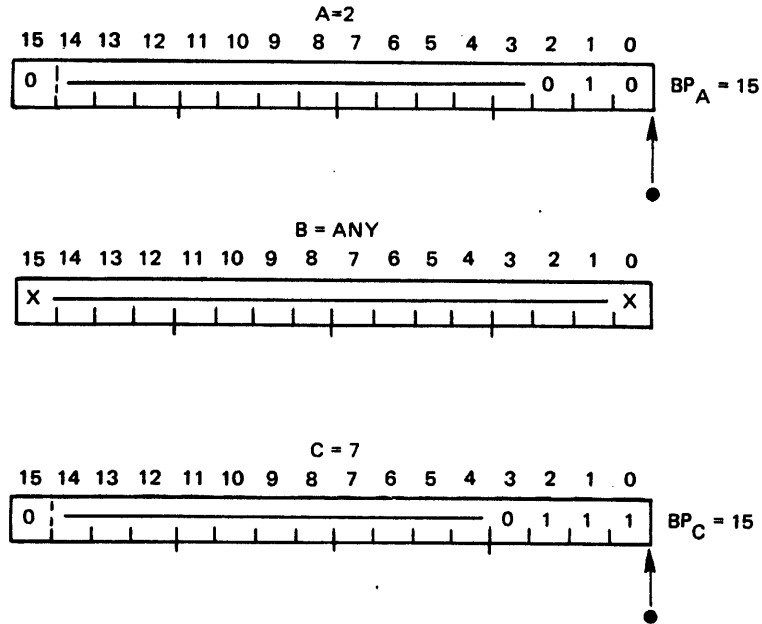
BP_C is the number of integer digits of the multiplier.

$BP_{B,C}$ is the number of integer digits in the answer ($30 - BP_{B,C}$ would be the number of fractional digits).

(15-n) is the number of trailing zeros. (When n=15, this term is zero, indicating that the last significant digit of the result is bit one of register C.)

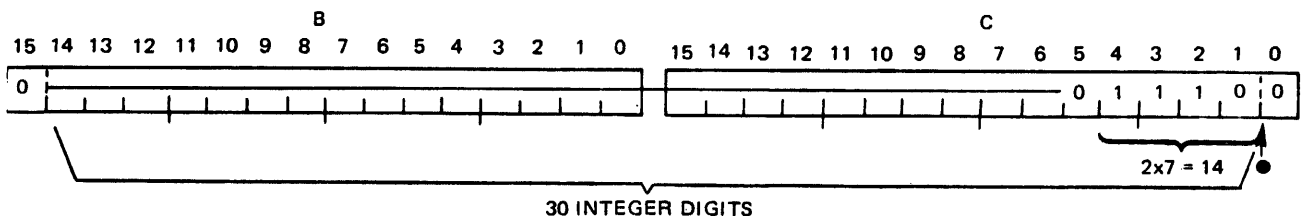
Examples:

- Given the following bit configuration for two integer terms prior to multiplication:



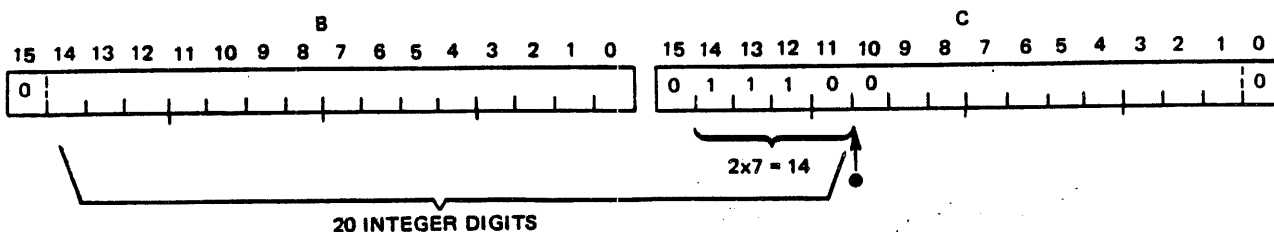
- MPY 15 Multiply the contents of register A by the low-order 15 bits of (or MPY) register C, with the answer in registers B and C.

$$\begin{aligned}
 \text{Result: } BP_{BC} &= BP_A + BP_C - (15 - n) \\
 &= 15 + 15 - 0 \\
 &= 30
 \end{aligned}$$



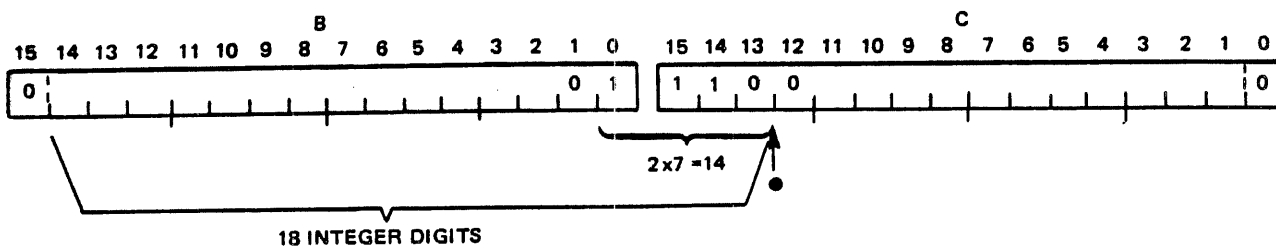
- b. MPY 5 Multiply the contents of register A by the low-order five bits of the C register, with the answer in registers B and C.

$$\begin{aligned} \text{Result: } BP_{B,C} &= BP_A + BP_C - (15-n) \\ &= 15 + 15 - (15-5) \\ &= 20 \end{aligned}$$

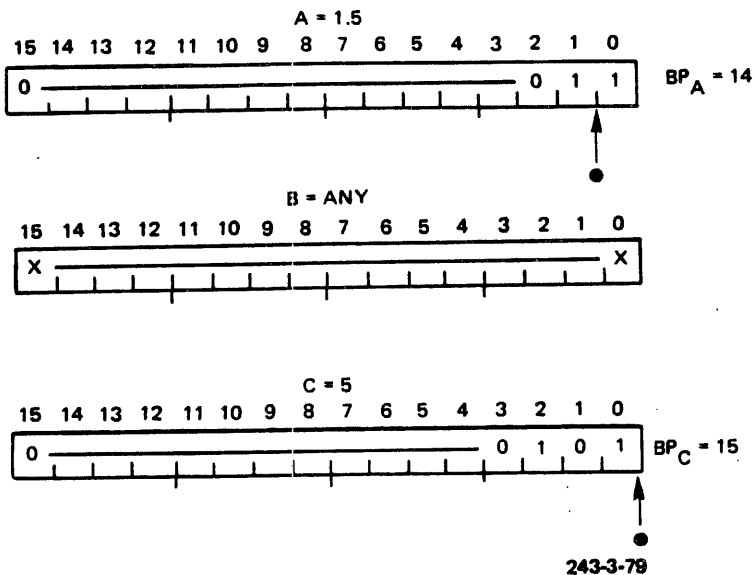


- c. MPY 3 Multiply the contents of register A by the low-order three bits of register C, with the answer in registers B and C.

$$\begin{aligned} \text{Result: } BP_{B,C} &= BP_A + BP_C - (15-n) \\ &= 15 + 15 - (15-3) \\ &= 18 \end{aligned}$$

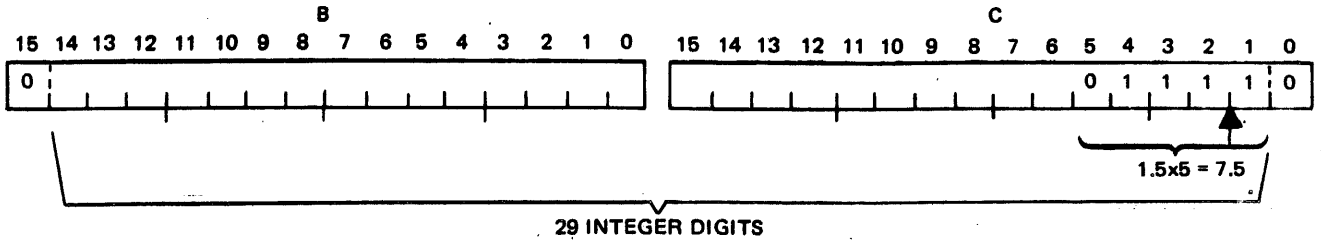


2. Given the following bit configuration for one integer term and one fractional term:



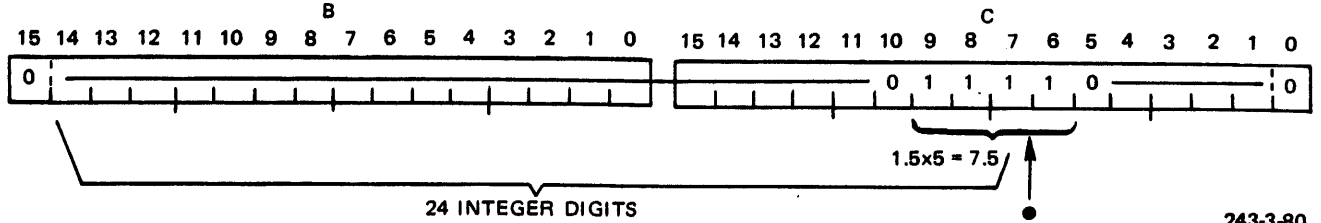
- a. MPY 15 Multiply the contents of register A by the low-order fifteen bits of (or MPY) register C, with the answer in registers B and C.

$$\begin{aligned} \text{Result: } BP_{B,C} &= BP_A + BP_C - (15-n) \\ &= 14 + 15 - (15-15) \\ &= 29 \end{aligned}$$



- b. MPY 10 Multiply the contents of register A by the low-order ten bits of register C, with the answer in registers B and C.

$$\begin{aligned} \text{Result: } BP_{B,C} &= BP_A + BP_C - (15-n) \\ &= 14 + 15 - (15-10) \\ &= 24 \end{aligned}$$



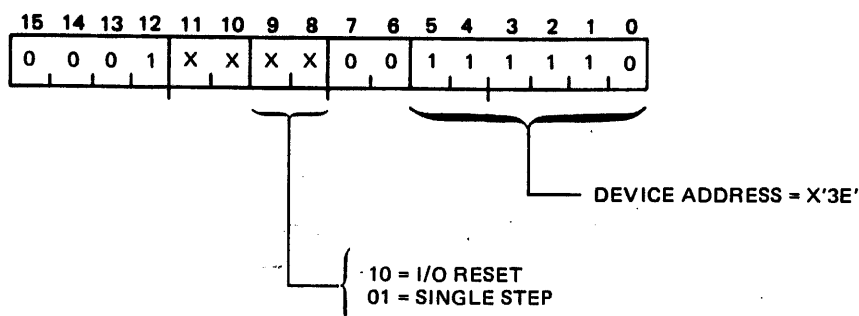
243-3-80

3. Negative (two's complement) numbers can be multiplied by using the hardware multiply instruction by multiplying their absolute values, then converting the (positive) result back to negative if the signs of the terms are different.

	ZERO	X		Bit 0 of X will be sign bit of answer.
	ANDVC	A,X'8000'	}	Get absolute value of multiplicand.
	SKZ	CREG		
	CMPL	A		
	INCR	A		
	INCR	X		
CREG	ANDVC	C,X'8000'	}	If negative, change sign of answer.
	SKZ	NEXT		
	CMPL	C		
	INCR	C		
	INCR	X		
NEXT	MPY			If negative, change sign of answer. Perform multiplication.
	SRLC	B,1	}	Position result in registers.
	SRCL	C,1		
	SRA	X,1		Put sign bit into link.
	SKR	DONE		If positive, done.
	CMPL	B	}	Otherwise, change B,C result to negative.
	CMPL	C		
	INCR	C		
	RLK	B		
	.			
DONE	.			
	.			

4.17 SPECIAL INSTRUCTIONS

Two special instructions are provided for the GA-16/220 computer. These are PIO control instructions addressed to device X'3E' which provide for (1) resetting I/O controllers and (2) executing a single instruction. These instructions are primarily designed to be used with the SCI console ROM program, and have the following format:

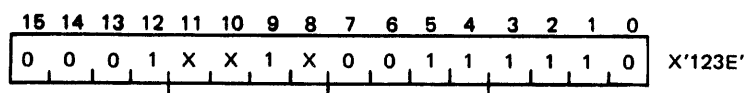


The CAP-16 Source Statements for these instructions are:

<u>Command</u>	<u>Parameter</u>
CTRL	2, X'3E' (for IORST)
CTRL	1, X'3E' (for SSTEP)

Detailed description is provided in the following sections.

4.17.1 I/O RESET (IORST)



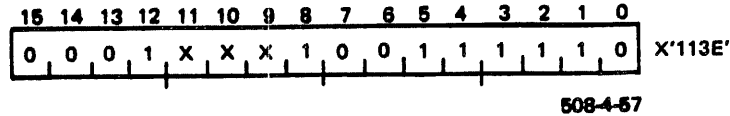
508-4-56

This instruction is a special-case PIO control instruction with bit 9 always on to device code X'3E'. It generates a reset signal on the I/O bus.

RESET → I/O BUS

Indicators affected: None
 Registers affected: None

4.17.2 ENABLE SINGLE STEP INTERRUPT (SSTEP)



This instruction is a special-case PIO control instruction with bit 8 always on to device code 3E. The interrupt via vector X'46' occurs immediately after execution of the first instruction outside the upper 1K of memory. You cannot single step through WAIT or XEC instructions.

4.18 SAMPLE GA-16/220 PROGRAM

Figure 4-8 is a CAP-16 assembly listing of a sample program which illustrates usage of instructions discussed in this section. The annotations added to this listing identify the paragraph where the instruction is described in detail.

NOTE

The NTRC, CARGM and NTRD instructions identified as "macro" are among the extended stack and argument transfer instructions of the GA-16/440, which are not implemented on the GA-16/110/220. Still, all extended instructions may be used as macros in writing programs for the GA-16/110/220. The CAP-16 Assembler (Rev. 11 or higher) simply responds to such macros by producing in-line object code for GA-16/110/220 software routines equivalent to the GA-16/440 hardware functions.

88A00508A-E

```

0001 *
0002 *           TITLE 'GA-16 TIME SUBROUTINE'
0003 *
0004 *   GENERAL AUTOMATION, INC. ALL RIGHTS RESERVED
0005 *****
0006 *
0007 *   PROGRAM NAME:  TIME SUBROUTINE
0008 *
0009 *   MODEL NUMBER:  6MX41
0010 *
0011 *   PURPOSE:      TIME$ WILL RETURN THE CURRENT TIME OF DAY
0012 *                IN HOURS, MINUTES, AND SECONDS TO THE
0013 *                CALLER IN ASCII.
0014 *                TIME$B WILL RETURN THE CURRENT TIME OF DAY
0015 *                IN HOURS, MINUTES, AND SECONDS IN INTEGER
0016 *                FORMAT.
0017 *                CALLING SEQUENCE:
0018 *                   CALL TIME$B (I,J,K)
0019 *                   CALL TIME$ (I,J,K)
0020 *                RETURNS I = HOURS, J = MINUTE, K = SECOND.
0021 *
0022 *
0023 *****          REVISION LIST          *****
0024 *
0025 *   RV DATE      SCO # BY   REASON FOR CHANGE
0026 *   -----
0027 *
0028 *   01 12/01/72  72289 JNO INITIAL RELEASE
0029 *   06 08/15/75  75056 JWN UPDATE FOR GA-16/440 COMPATIBILITY
0030 *                   CORRECT USE OF STATUS SAVE FOR FLAGS
0031 *   07 03/31/76  76161 DAR CORRECTED CARGM ERRORS
0032 *
0033 *****
0034 *****
0035 EJECT

```

	0036	DEF	TIME\$,TIME\$B		
	0037	REF	E\$SLO,E\$FST,SIM\$,E\$CPM		
	0038	DSECT	0		
	0039	DS	9	REGISTER SAVE	
0000	0040	ARG1	DS	1	
0000	0041	ARG2	DS	1	
0009	0042	ARG3	DS	1	
000A	0043	SAVE	DS	1	
000B	0044	DS	1		
000C	0045	STAT	DS	1	ASCII/BINARY FLAG
000D	0046	PSECT			
000E	0047	TIME\$	NTRC	0,15	
0000					
0001	F4C0				
0002	C0C0				
0003	0EB5				
0004	01B9				
0005	000F				
0006	DOA0				
0007	01B6				
0008	0006				
0009	05E1				
	38CE				
	0048	RBIT	6,STAT	SET ASCII FLAG.	4.5.7
	0049	T2	CARGM	FETCH ARGUMENTS	macro
000A	6445				
000B	0402				
000C	C4BF				
000D	0000 X				
000E	C45F				
000F	0000 X				
0010	05E1				
0011	403F				
0012	602C				
0013	D88C				
0014	A8CE				
0015	2406				
	0050	INH			4.13.4
	0051	LDR	C,*E\$SLO	GET MINUTES.	4.5.6
	0052	LDR	Y,*E\$FST	GET BST'S.	4.5.6
	0053	RISE	E		4.10.9
	0054	LDA	=60		4.4.3
	0055	JSR	TDIV	SEPARATE HOURS/MINUTES.	4.4.2
	0056	STR	B,SAVE	SAVE MINUTES.	4.5.11
	0057	TBIT	6,STAT		4.5.12
	0058	SKN	T4	RETURN BINARY.	4.6.2

Discussed in Paragraph No.

Assembler Directives

macro

508-4-57

Figure 4-8. Sample GA-16/220 Program (Sheet 1 of 2)

88A00508A-E

					<i>Discussed in Paragraph No.</i>
0016	403B	0059	LDA	=10	4.4.3
0017	6027	0060	JSR	TDIV	4.4.2
0018	06A4	0061	EXBY	C	4.10.5
0019	0C8D	0062	PR	C,B	4.7.5
001A	018D	0063	ORV	C,'00'	4.8.6
001B	B0B0				
001C	DCA9	0064	T4 STR	C,*ARG1	4.5.11
001D	C8AC	0065	LDR	C,SAVE	4.5.6
001E	A8CE	0066	TBIT	6,STAT	4.5.12
001F	2405	0067	SKN	T6	4.6.2
0020	601E	0068	JSR	TDIV	4.4.2
0021	06A4	0069	EXBY	C	4.10.5
0022	0C8D	0070	OR	C,B	4.7.5
0023	018D	0071	ORV	C,'00'	4.8.6
0024	B0B0				
0025	DCAA	0072	T6 STR	C,*ARG2	4.5.11
0026	402A	0073	LDA	=60	4.4.3
0027	0AB5	0074	RTR	C,Y	4.7.7
0028	2614	0075	SKM	T20	4.6.1
0029	6429	0076	JSR	SIM\$	
002A	008F	0077	MPY	15	4.16.2
002B	C41F	0078	LDR	A,*E\$CPM	4.5.6
002C	0000 X				
002D	6425	0079	JSR	SIM\$	
002E	00AF	0080	DIV	15	(COUNTS*SEC/MIN)/(COUNTS/MINUTE) 4.16.1
002F	38CE	0081	RBIT	6,STAT	4.5.7
0030	2406	0082	SKN	T12	4.6.2
0031	4020	0083	LDA	=10	4.4.3
0032	600C	0084	JSR	TDIV	4.4.2
0033	06A4	0085	EXBY	C	4.10.5
0034	0C8D	0086	OR	C,B	4.7.5
0035	018D	0087	T10 ORV	C,'00'	4.8.6
0036	B0B0				
0037	DCAB	0088	T12 STR	C,*ARG3	4.5.11
		0089	RTRN1 RTND	0	macro
0038	0402				
0039	D0C0				
003A	F880				
003B	0815				
003C	05E3				
003D	06A0	0090	T20 ZERO	C	4.10.15
003E	73F6	0091	JMP	T10	4.4.1
003F	0F35	0092	TDIV RTR	X,E	4.7.7
0040	0680	0093	ZERO	B	4.10.15
0041	0DB9	0094	ADD	C,C	4.7.1
0042	6410	0095	JSR	SIM\$	
0043	00AF	0096	DIV	15	4.16.1
0044	0523	0097	RTRN	X	4.10.11
		0098	TIME\$B NTRC	0,15	macro
0045	F4C0				
0046	C0C0				
0047	0EB5				
0048	01B9				
0049	000F				
004A	D0A0				
004B	01B6				
004C	0006				
004D	05E1				
004E	B8CE	0099	SBIT	6,STAT	4.5.9
004F	73BA	0100	JMP	T2	4.4.1
		0101	END		Assembler Directive
0050	0000 X				
0051	003C				
0052	000A				
0053	0000 X				
	0000 ERRS.				

Figure 4-8. Sample GA-16/220 Program (Sheet 2 of 2)

memory parity protect (MPP) 5

The GA-16/110 and GA-16/220 Memory Parity Protect Option (MPP), Model 1622-0050 provides both memory parity and write protection. Its features include:

- Independent enabling of features.
- Parity error, write protect error, or program timeout are signaled by a non-inhabitable (NI) interrupt.
- Separate program and DMA write protect maps.
- Each map contains sixty-four, 1K segments.
- DMA and program write protection are individually enabled.
- Switch or program enabling of memory parity error detection.
- Switch enabling to stall the CPU if a parity error occurs.

Hardware for MPP requires the use of memory modules with 18-bit words and the inclusion of the MPP board. The use of MPP is the same for a GA-16/110 and GA-16/220 with the exception of DMA write protect. DMA is not available on a GA-16/110.

5.1 MEMORY PARITY

During all memory write operations, odd parity is generated and written into memory for both bytes of words being written. Memory write operations include DMA inputs and instructions which write into memory. These are DECM, DTIM, INCM, RBIT, SARS, SBIT, STA, STBY, and STR.

During all memory read operations, both bytes of the data word are checked for odd parity. If either byte does not have odd parity, a parity error bit is set in the MPP status register to indicate a program or DMA parity error. At the same time, the upper and lower parity bits are copied into the MPP status register. Thus, the program can do a pair of DTIR/M instructions to input both the data word and the MPP status word in order to monitor the bits associated with the error. Memory read operations which check parity include: DMA output, LDA, LDR, CMR, LDBY, STBY, LARS, INCM, DECM, SBIT, RBIT, TBIT, DTOM, Literal operation, and the access of an instruction. The 18-bit memory modules, which are designed for use with MPP, have switches which enable and disable the parity protect for the module. Assuming the switches are set to enable parity, the parity check feature may then be enabled by program control. A parity error stall switch is provided on the MPP module which will enable a memory management stall (MSTAL) condition if an error is detected. In this condition, the CPU is in a run mode and is also busy, that is, cannot execute instructions. The only way to exit this condition is by pressing RESET (Table 3-3, (16)).

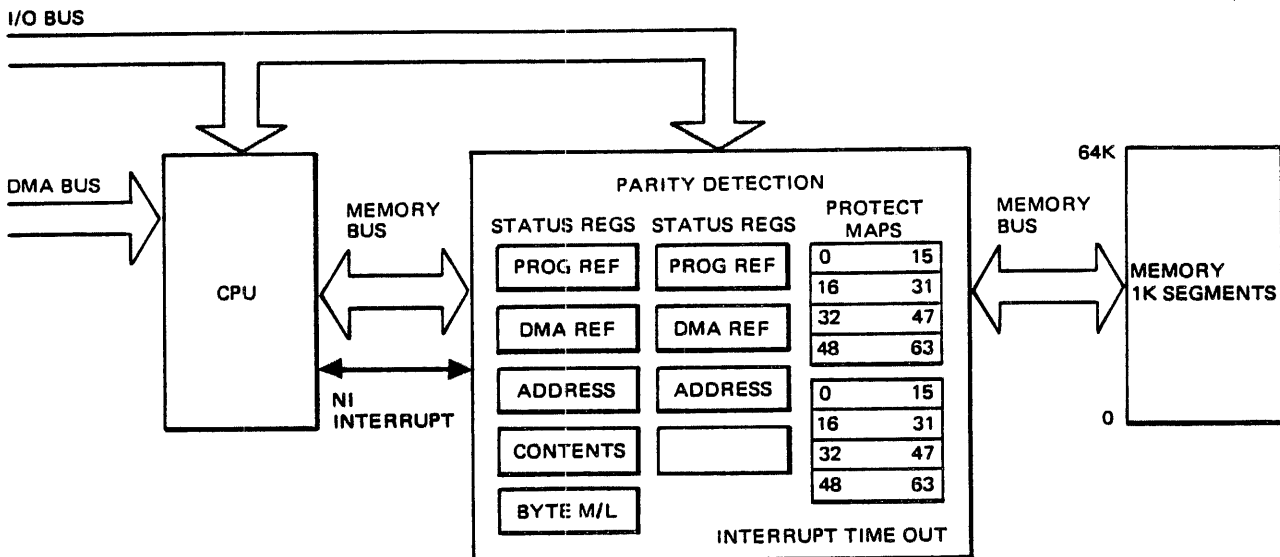
NOTE

The parity error stall switch is accessible only when the MPP module is out of the chassis (Figure 5-1).

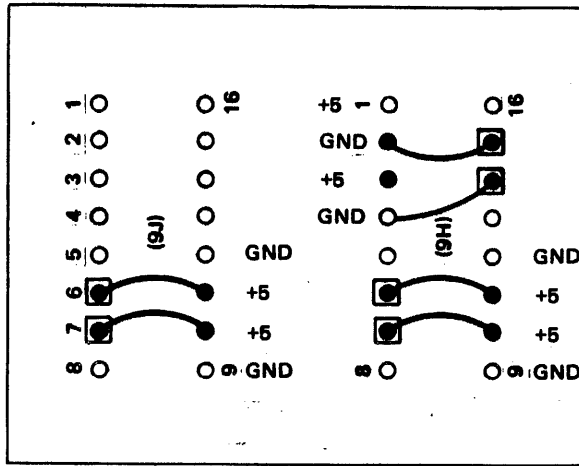
If the parity check is enabled only by program control, a parity error will generate a non-inhabitable interrupt (through vector X'41'). (See Sections 2.6 and 4.9 for discussion of NI processing.) The program routine, processing the NI interrupt, may determine the cause of the error and reset the error indicator by means of I/O instructions to the MPP. The routine may also provide the means to recover from the fault (if possible).

5.2 WRITE PROTECT FEATURES

The MPP module contains a 128-bit memory which is loaded under program control (by programmed I/O). Each bit represents the protect bit for one segment of 1024 words as illustrated below. One 64-bit half of the memory is for program protect and the other half is for DMA protect. When the block protect is enabled by outputting (DTOR/M) an output command word, any attempt to write into any word of a protected block will be inhibited and an NI interrupt will be generated. In addition to the protect maps the MPP module contains status information which may be read for diagnostic purposes.



JUMPERS SHOWN FOR 960 - 970 μ S



DETAIL A

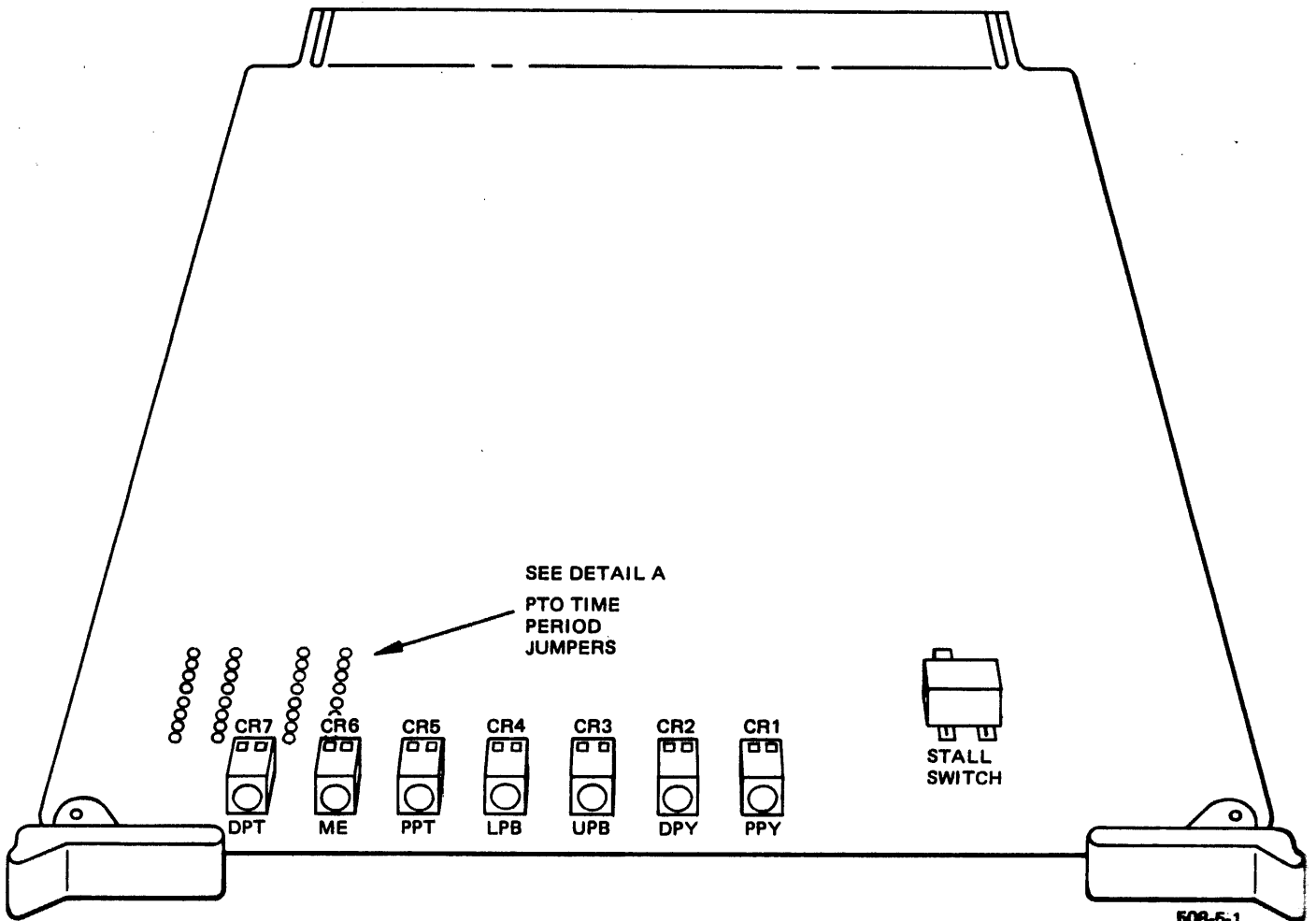


Figure 5-1. GA-16/220 Memory Parity Protect Module

508-5-1

5.3 PROGRAM TIMEOUT (PTO)

MPP provides a program timeout function which monitors two conditions: the state of ISE and the execution of non-interruptable instructions. ISE must be set for the duration of at least two consecutive interruptable instructions in order to reset the PTO timing function. If ISE is on, the timing function is reset each time an interruptable instruction is executed, thus preventing endless JMP and JSR loops. The standard timing period is 960 to 970 microseconds. Other timing periods down to 20 to 25 microseconds can be obtained by changing jumpers on the MPP circuit cards. See Table 5-1 for PTO time periods and Figure 5-1 for jumper locations. When PTO detection is enabled by output of a command word, any sequence of instructions which inhibits the interrupt system beyond the timeout period will cause an NI interrupt. The interrupt routine can then determine that PTO has occurred by input of the status word, and can determine the address the last instruction executed by input of the memory address word.

Table 5-1. PTO Time Period Jumpers

Jumper Zone	Delay On:		Delay Off:		Time Period (μ s)
	Hardwired To Ground	Logic	Hardwired To Ground	Logic	
(9J)	9	7	10		20-25
(9J)	12	6	11		40-45
(9H)	12	7	10		80-85
(9H)	9	6	11		160-165
(9H)	4	14	3		320-325
(9H)	2	15	1		640-645

Jumpers for 960-970 μ sec

NOTES:

1. Jumper zone refers to the location of jumpers on module (Figure 5-1); jumpers connect between numbered board points in the zone, either LOGIC to GROUND (for enabling DLY) or LOGIC to +5 VOLTS (for disabling DLY).
2. Time periods shown are enabled by jumpering LOGIC to GROUND; all other LOGIC points are jumpered to +5 VOLTS.
3. More than one LOGIC point may be connected to GROUND to provide additive time period (e.g., (9J) 12 to 6 and (9H) 9 to 6 to provide 200-210 μ s time) and remaining jumpers to +5 VOLT (e.g., (9J) 7 to 10, (9H) 7 to 10, (9H) 14 to 3, and (9H) 15 to 1).
4. Normal jumper connections (actually etched on board) are for 960-970 μ s as illustrated in Figure 5-1.

NOTE

CPU interrupts stop the timer (but do not reset it) from counting. To start the timer counting again (from where it left off) it must be re-enabled (Section 5.6.1.2).

5.4 MPP CONTROLS AND INDICATORS

The controls and indicators for the MPP module are described in Section 3, Table 3-3, key numbers (24) through (31).

5.5 NON-INHIBITABLE MPP INTERRUPT

The interrupt generated by MPP is not affected by the state of the interrupt system enable (ISE). The interrupt vector for MPP is location X'42'. The register P and ISE status storage locations are X'7A' and X'7B', respectively. For a more detailed discussion of interrupts, refer to Section 2.6.

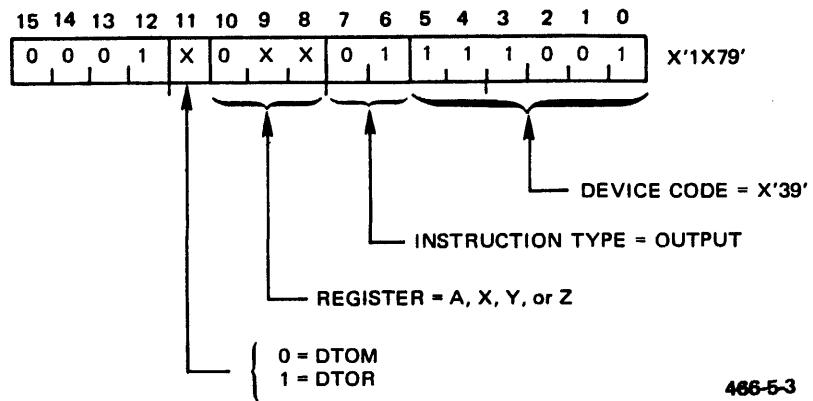
5.6 MPP COMMANDS

5.6.1 OUTPUT WORDS

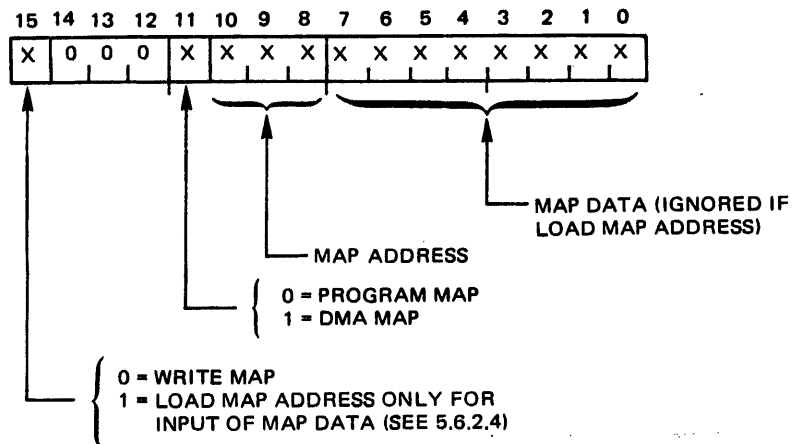
MPP recognizes two data output instructions. It decodes the REG field of the output instructions to determine how to interpret the data output from the register or from a memory location specified by the register contents.

5.6.1.1 Output Map Address or Map Data

INSTRUCTION FORMAT



DATA FORMAT



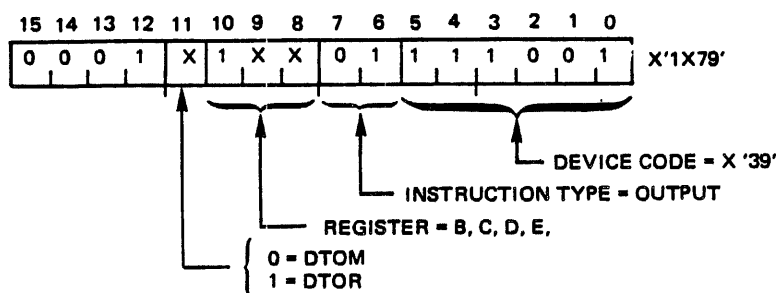
The MPP module contains two write protect maps for (1) program protection and (2) for direct memory access protection. Each map is organized as eight words of eight bits each. The maps are either loaded or their contents accessed by output of one to eight words in the data format shown.

Map selection is specified by bit 11, and each word in the map is specified by bits 10, 9, and 8. A map word is loaded with bits 7 to 0 if bit 15=0. The word is accessed for subsequent input when bit 15=1. Refer to Section 5.6.2.4 to input map word.

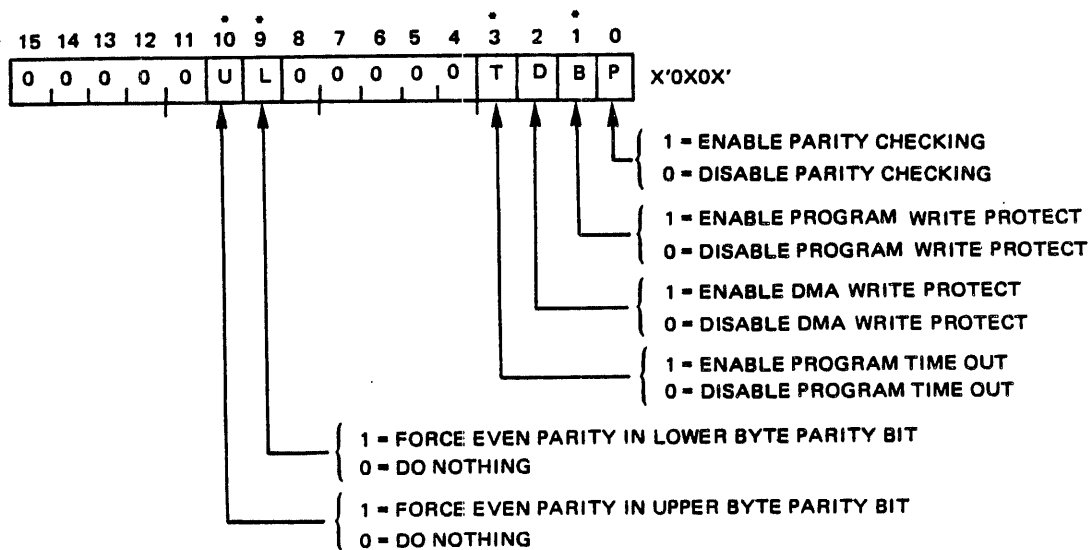
Figure 5-2 shows a typical data table containing all map data. As each word is output, the data in bits 7 to 0 will set the map to protect the corresponding 1K segment in core. Figure 5-3 illustrates the way in which a typical program map word and DMA map word protect memory provided write protect has been enabled by output of a command word.

5.6.1.2 Output Command Word (Resets Status)

INSTRUCTION FORMAT

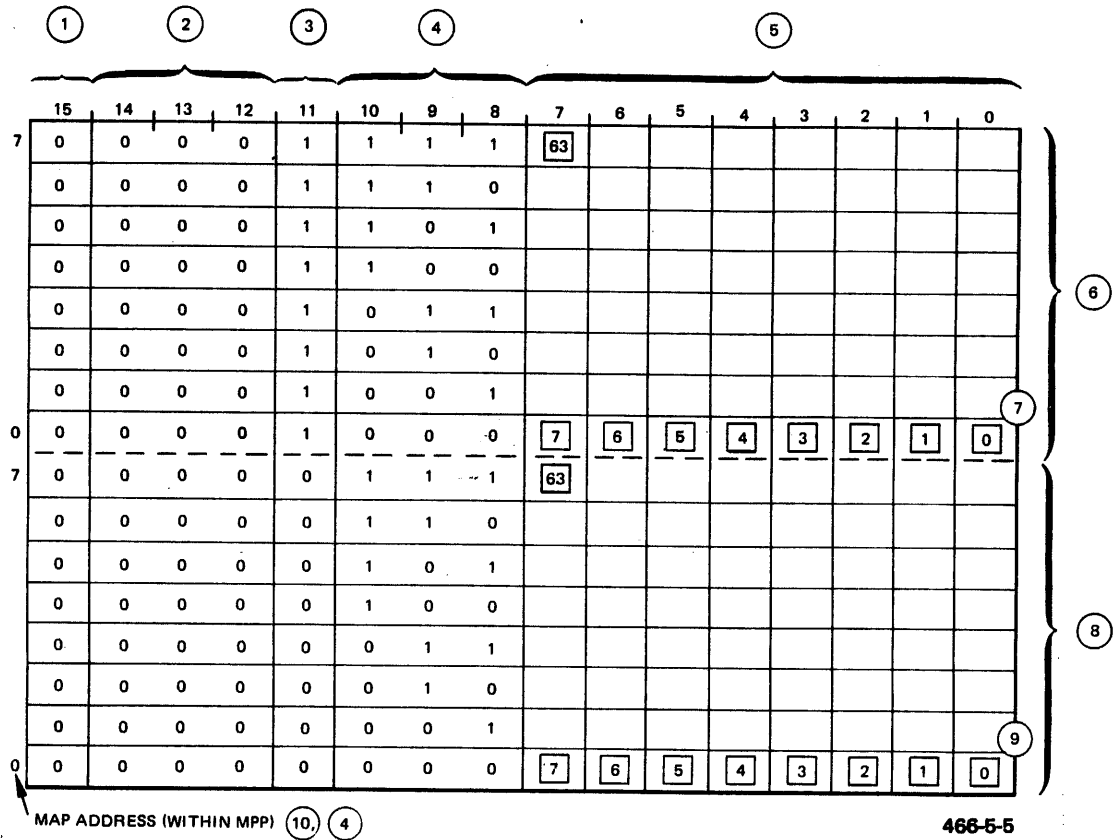


COMMAND WORD DATA FORMAT



*FUNCTIONS OF BITS 1, 3, 9, AND 10 ARE RESET AFTER ANY MPP INTERRUPT

88A00508A-E



NOTES:

- ① Bit 15=0 to load map data (bits 0 to 7) into MPP map
Bit 15=1 to access map word
- ② Bits 14,13,12 always = 0
- ③ Bit 11=0, program protect map
Bit 11=1, DMA protect map
- ④ Map address (within MPP)
- ⑤ ⑥ Boundaries of DMA protect map as it exists in MPP module
- ⑦

n	indicates 1024 word segment protect bit	}	=1, protected
			=0, unprotected

(e.g., if 7 =1, memory segment from 7168 to 8191 is DMA protected)
- ⑤ ⑧ Boundaries of program protect map
- ⑨ n Same as 7 but for program protect map
- ⑩ Map data is accessed by output of map address word with bit 15=1, followed by input of protect map data, Section 5.6.2.4.

Figure 5-2. MPP Protect Maps

Control bits U and L are used to test MPP parity circuits. By outputting the above data word, the various features of MPP are enabled and disabled. Program protection would then enable or disable those 1K segments previously sent to the MPP protection map. PTO and PE checking apply blanket coverage for all memory; no map is needed. Bits 10 and 9 (upper and lower byte) force the next data read to cause an even parity error on the byte specified for testing the MPP parity circuits. This command also resets the MPP status register.

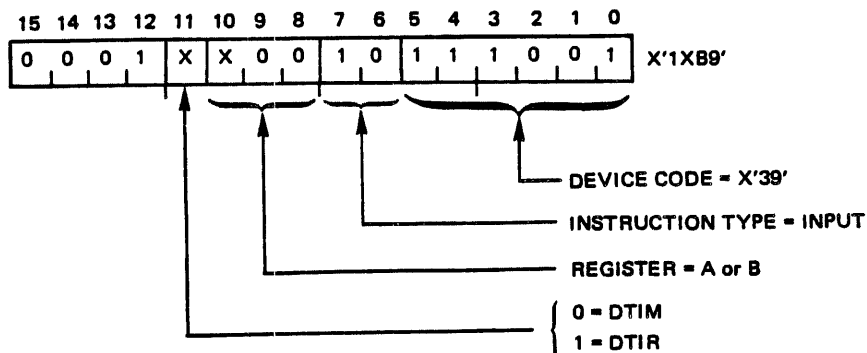
5.6.2 INPUT WORDS

MPP recognizes four data input instructions. It decodes the REG field of the instruction to determine which input word is to be sent to the register or to the memory location specified by the register contents.

5.6.2.1 Input Status Word

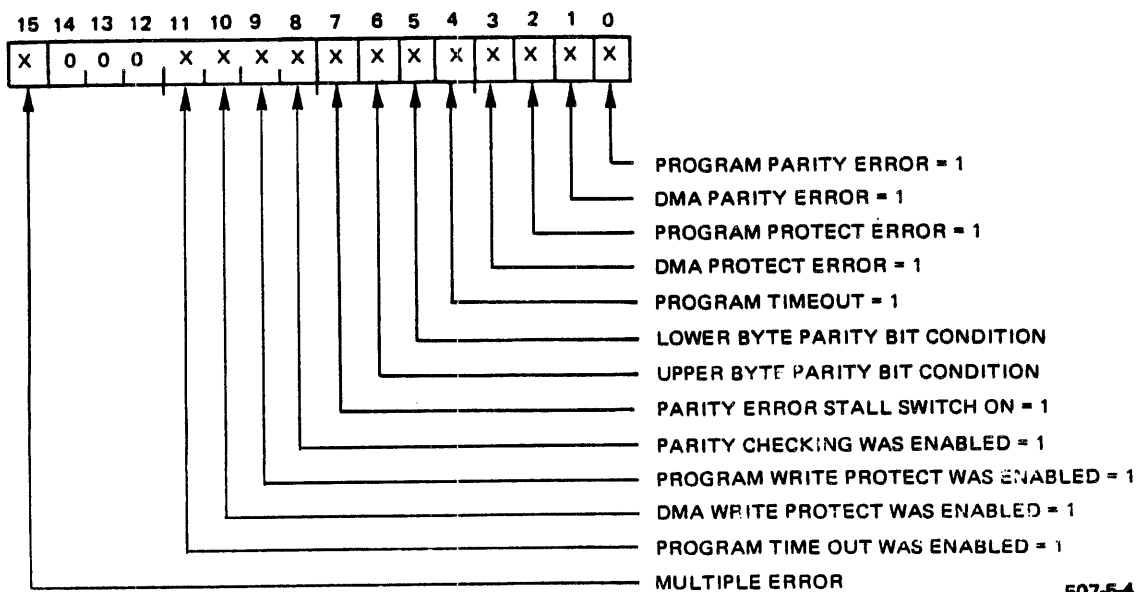
The input status word operation inputs to the CPU the six error bits, two parity bits, the position of the parity stall switch, and indicates which MPP features are currently enabled.

INSTRUCTION FORMAT

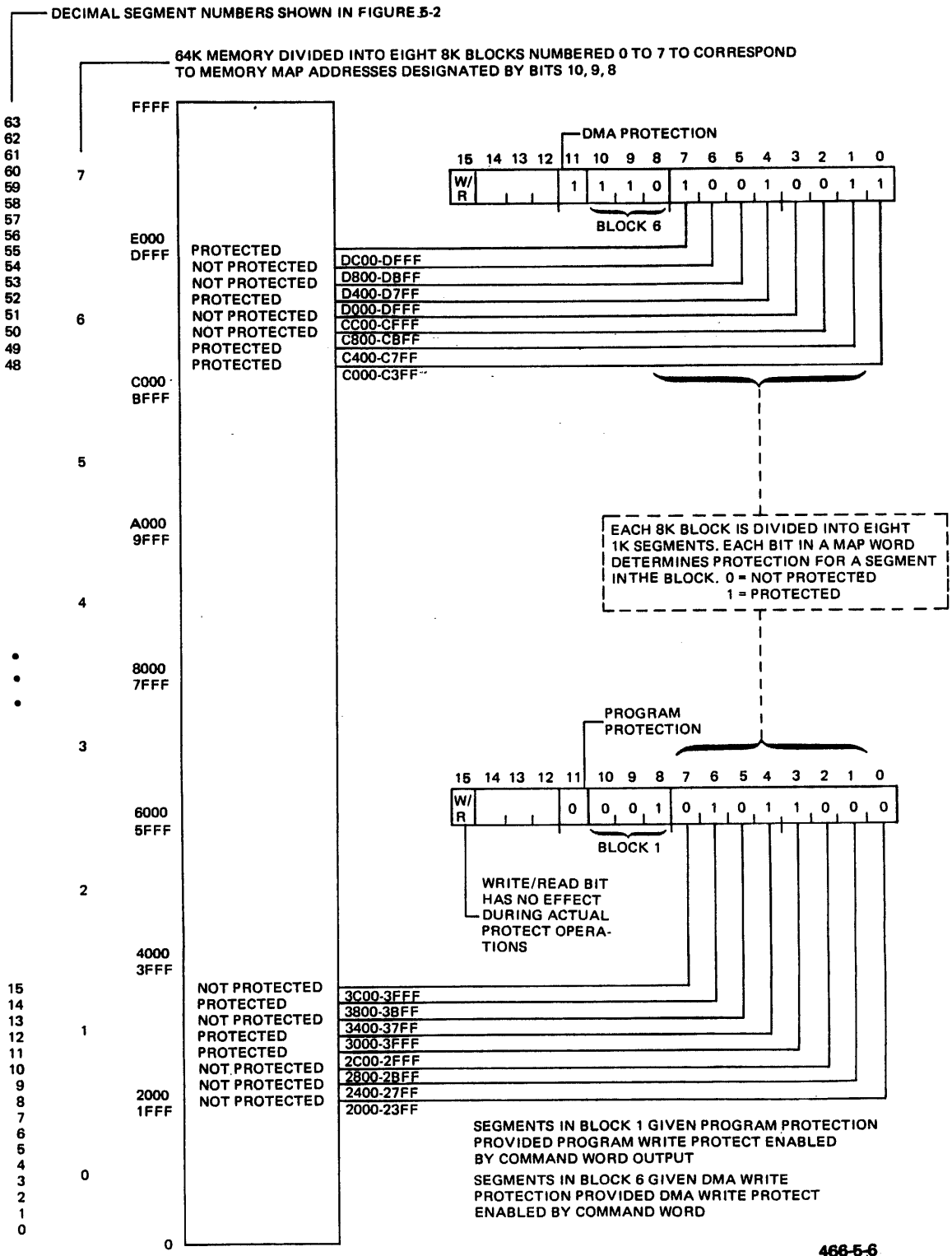


MPP STATUS WORD DATA INPUT WORD FORMAT

466-5-7



507-5-4



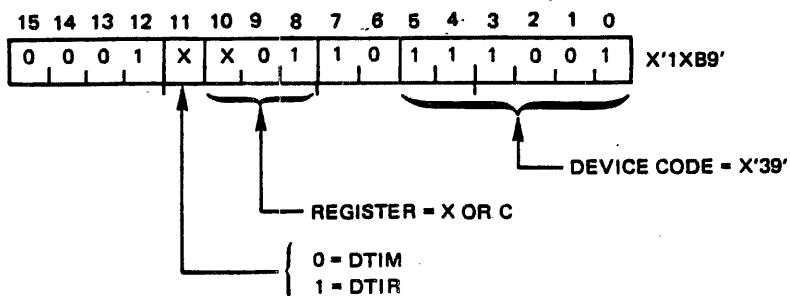
468-5-6

Figure 5-3. Illustration of Memory Protection

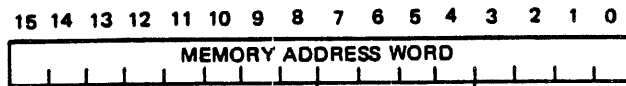
Bits 1 and 0 of the Status Word tell which type of memory access had the parity error. Bits 3 and 2 tell if a program or DMA store operation caused a protect error. Bit 4 indicates that a program timeout occurred. Bits 6 and 5 (the lower and upper byte parity bits) are meaningful only if a parity error bit is set. Bit 7 provides the program a means of sensing the state of the parity error stall switch. Bits 8, 9, 10 and 11 have the state of the command register of the MPP at the time of the last interrupt. These bits may be used to restore the MPP to its original status after the interrupt has been processed. The MPP status register is reset by outputting a new command (Section 5.6.1.2).

5.6.2.2 Input Memory Address Word

INSTRUCTION FORMAT



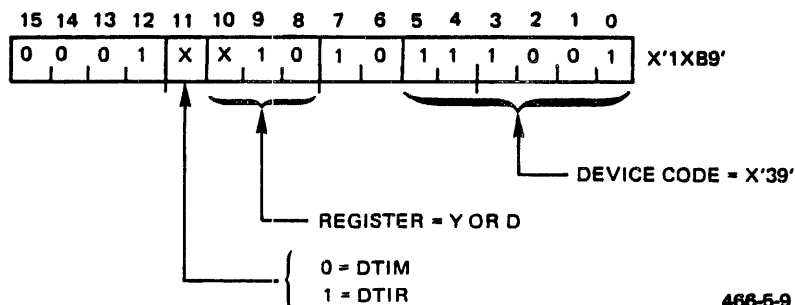
MEMORY ADDRESS WORD DATA INPUT FORMAT



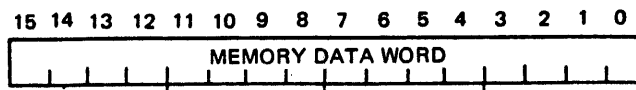
Interpretation of this word depends on the contents of the status word. The memory address word contains the address of the location being accessed which caused a parity or protect error. In the case of a program timeout, the contents will be the address of the last instruction executed.

5.6.2.3 Input Memory Data Word

INSTRUCTION FORMAT



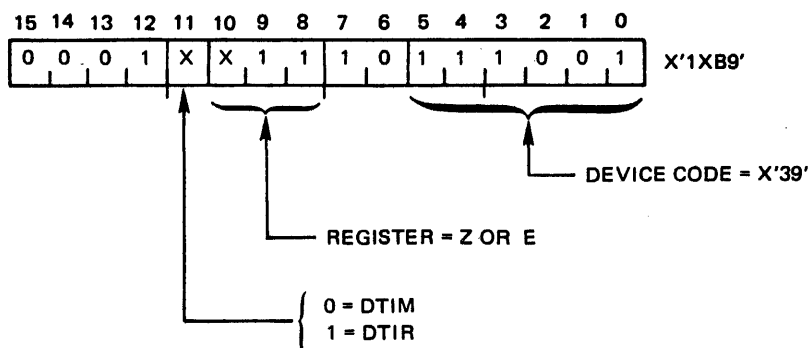
DATA INPUT WORD FORMAT



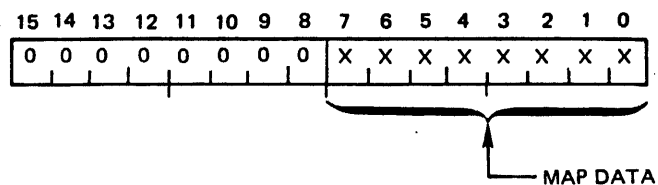
The memory data word is used in diagnosing memory parity errors. It contains the 16 data bits which, along with the two parity bits contained in the status word, make up the 18-bit memory word which includes parity.

5.6.2.4 Input Protect Map Data

INSTRUCTION FORMAT



DATA FORMAT



466-5-10

The input protect map data function is used to input the MPP protect memory data. This function is included as a diagnostic tool. The address for the protection map is determined by the output map address function. (See Section 5.6.1.1 with bit 15 on in Data Format.)

5.7 MPP OPERATION

INITIAL CONDITIONS

The following conditions will exist on power-up or after system reset:

- Parity, Program Timeout, and protect disabled.
- All error bits reset.

5.8 INSTRUCTION AND DATA SUMMARY

Table 5-2 lists the machine code and assembler directives for the instructions described in the preceding sections.

Figure 5-4 illustrates the output data formats and also shows the format of the program and DMA map within the MPP module. Figure 5-5 illustrates the input data formats. Figure 5-6 is a listing of an MPP demonstration program.

NOTE

The MPP demonstration program is presented for illustration of instruction sequences to MPP. It will run only on a GA-16/220 equipped with an SCI, TTY, and disk drive. As shown, the DSPL instructions will cause no observable effect. This program could be modified by using an SCI subroutine as follows:

For each DSPL instruction, substitute:

<i>RTR</i>	<i>B,D</i>	<i>Put status pointer into B</i>
<i>LDV</i>	<i>D,X'FE00'</i>	<i>Set up ROM DATA address</i>
<i>JSR</i>	<i>X'FC61'</i>	<i>Call SCI subroutine to output characters on TTY</i>
<i>RTR</i>	<i>D,B</i>	<i>Restore pointer</i>

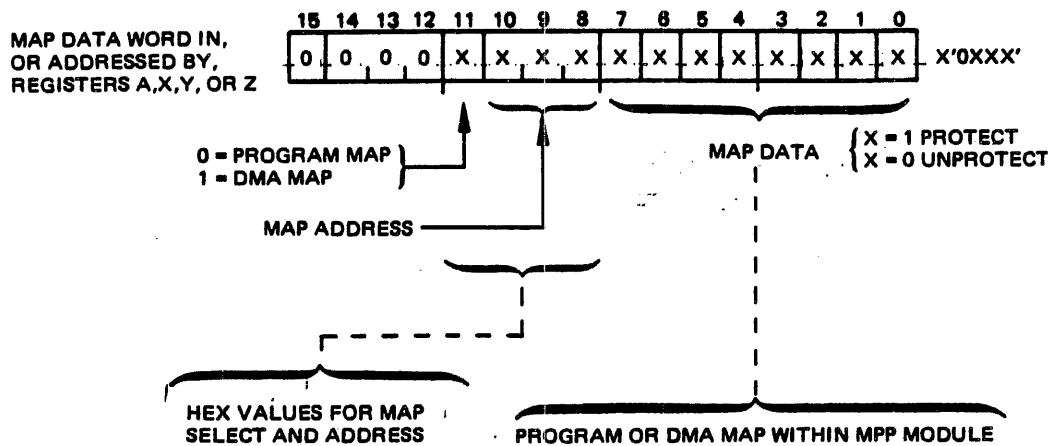
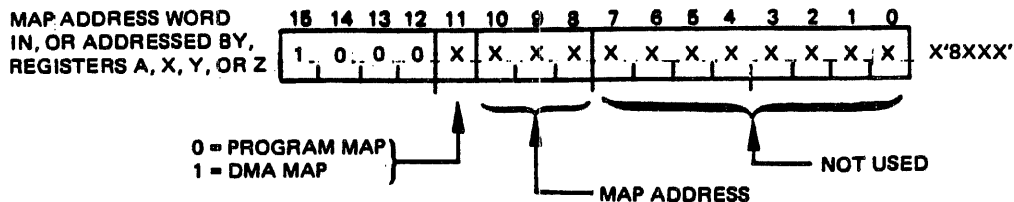
Table 5-2. Instruction Summary for MPP Module

Device Select Code: X'39'
 Interrupt Vector: X'42' (non-inhibitable)
 Data Channel Locations: N.A.
 ① P-Storage: X'7A'; ISE: X'7B'

Instruction		Function
Machine Code	Source Code	
1 $\begin{Bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{Bmatrix}$ 79	DTOM $\begin{Bmatrix} A \\ X \\ Y \\ Z \end{Bmatrix}$, X'39'	Output map address word or map data from memory
1 $\begin{Bmatrix} 8 \\ 9 \\ A \\ B \end{Bmatrix}$ 79	DTOR $\begin{Bmatrix} A \\ X \\ Y \\ Z \end{Bmatrix}$, X'39'	Output map address word or map data from register
1 $\begin{Bmatrix} 4 \\ 5 \\ 6 \\ 7 \end{Bmatrix}$ 79	DTOM $\begin{Bmatrix} B \\ C \\ D \\ E \end{Bmatrix}$, X'39'	Output command word (resets status) from memory
1 $\begin{Bmatrix} C \\ D \\ E \\ F \end{Bmatrix}$ 79	DTOR $\begin{Bmatrix} B \\ C \\ D \\ E \end{Bmatrix}$, X'39'	Output command word (resets status) from register
1 $\begin{Bmatrix} 0 \\ 4 \end{Bmatrix}$ B9	DTIM $\begin{Bmatrix} A \\ B \end{Bmatrix}$, X'39'	Input MPP status word to memory
1 $\begin{Bmatrix} 8 \\ C \end{Bmatrix}$ B9	DTIR $\begin{Bmatrix} A \\ B \end{Bmatrix}$, X'39'	Input MPP status word to register
1 $\begin{Bmatrix} 1 \\ 5 \end{Bmatrix}$ B9	DTIM $\begin{Bmatrix} X \\ C \end{Bmatrix}$, X'39'	Input memory address word to memory
1 $\begin{Bmatrix} 9 \\ D \end{Bmatrix}$ B9	DTIR $\begin{Bmatrix} X \\ C \end{Bmatrix}$, X'39'	Input memory address word to register
1 $\begin{Bmatrix} 2 \\ 6 \end{Bmatrix}$ B9	DTIM $\begin{Bmatrix} Y \\ D \end{Bmatrix}$, X'39'	Input memory data word to memory
1 $\begin{Bmatrix} A \\ E \end{Bmatrix}$ B9	DTIR $\begin{Bmatrix} Y \\ D \end{Bmatrix}$, X'39'	Input memory data word to register
1 $\begin{Bmatrix} 3 \\ 7 \end{Bmatrix}$ B9	DTIM $\begin{Bmatrix} Z \\ E \end{Bmatrix}$, X'39'	② Input protect map data to memory
1 $\begin{Bmatrix} B \\ F \end{Bmatrix}$ B9	DTIR $\begin{Bmatrix} Z \\ E \end{Bmatrix}$, X'39'	② Input protect map data to register

① Non-inhibitable interrupts cause storage of the contents of register P+1 and ISE in dedicated locations before transfer of control via interrupt vector.

② Must be preceded by output map address word.



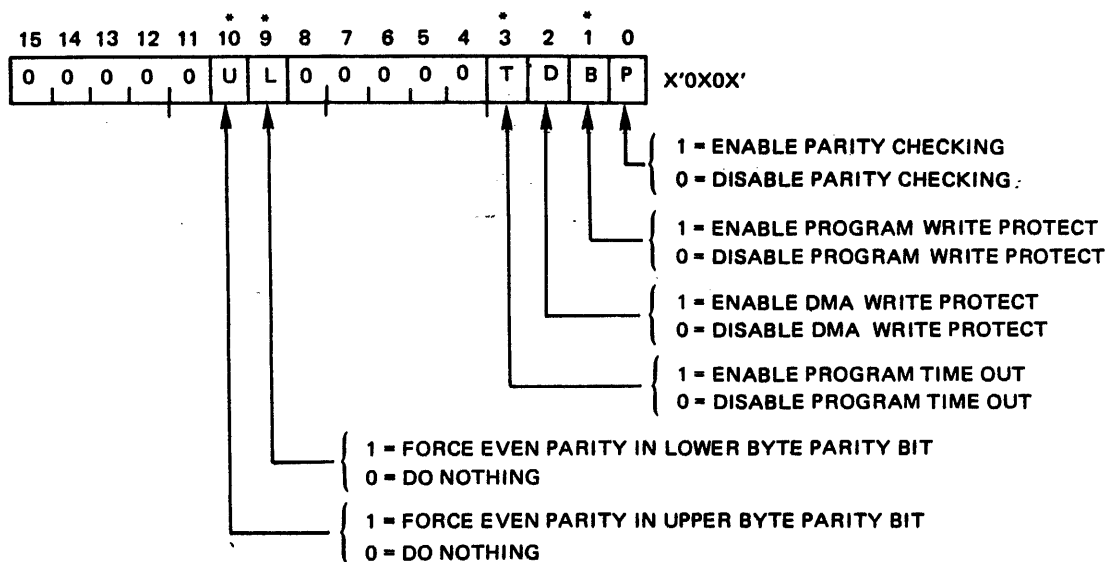
PROGRAM	DMA	PROGRAM OR DMA MAP WITHIN MPP MODULE							
		7	6	5	4	3	2	1	0
7	F	63	62	61	60	59	58	57	56
6	E	55	54	53	52	51	50	49	48
5	D	47	46	45	44	43	42	41	40
4	C	39	38	37	36	35	34	33	32
3	B	31	30	29	28	27	26	25	24
2	A	23	22	21	20	19	18	17	16
1	9	15	14	13	12	11	10	9	8
0	8	7	6	5	4	3	2	1	0

MULTIPLY NUMBERS IN SQUARES BY 1024 TO OBTAIN THE LOWER BOUNDARY OF 1024 WORD MEMORY SEGMENT WHICH IS PROTECTED WHEN MAP BIT = 1, AND PROTECT IS ENABLED BY COMMAND WORD (B OR P BITS)

508-6-2

Figure 5-4. Output Data to MPP (Sheet 1 of 2)

Command word in, or addressed by, registers B, C, D or E.

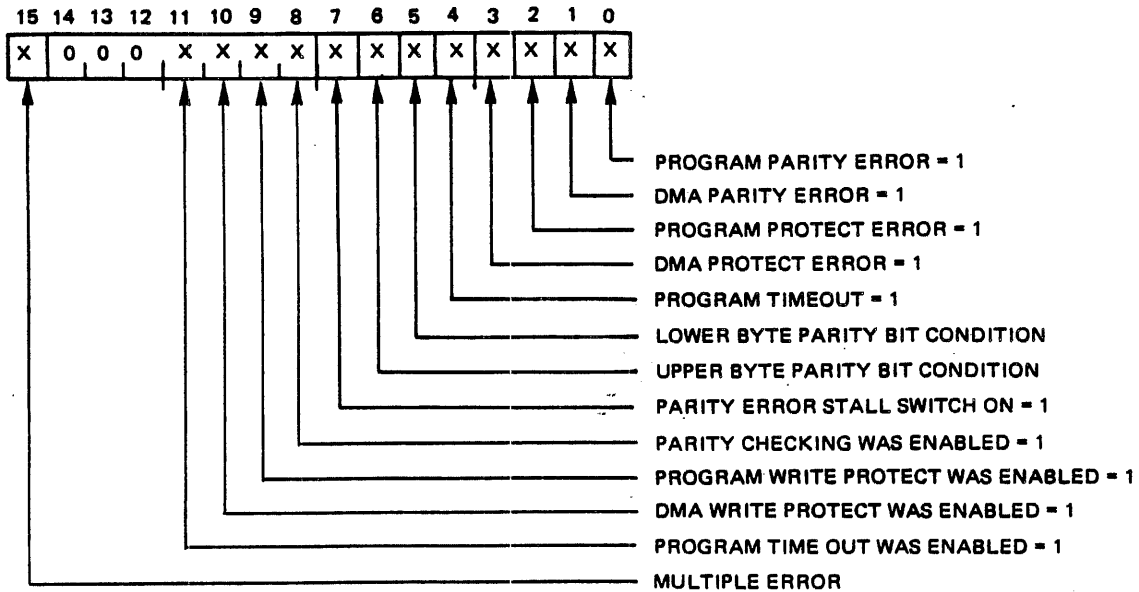


*FUNCTIONS OF BITS 1, 3, 9 AND 10 ARE RESET AFTER ANY MPP INTERRUPT.

508-5-2

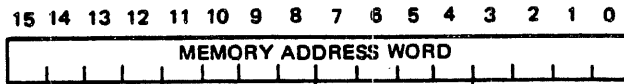
Figure 5-4. Output Data to MPP (Sheet 2 of 2)

MPP status word in or addressed by registers A or B:

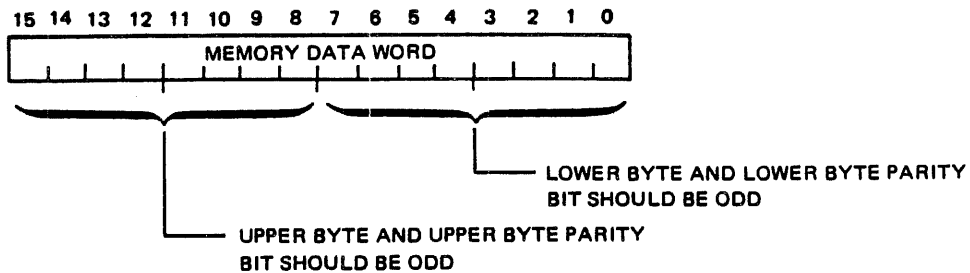


Memory address word in or addressed by registers X or C:

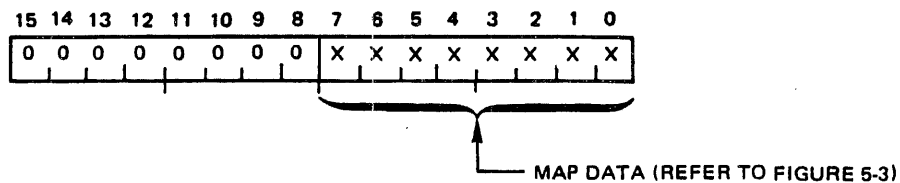
507-5-4



Memory data word in or addressed by registers Y or D:



Protect Map (previously addressed with output map address word) in or addressed by registers Z or E:



466-5-9

Figure 5-5. Input Data from MPP

88A00508A-E

```

0001 *****
0002 *
0003 *      MPP DEMONSTRATION PROGRAM
0004 *
0005 *      DATE: 02/26/76
0006 *
0007 *      ALL RIGHTS RESERVED
0008 *
0009 *****
0010 *
      0039 MPP EQU X'39'
0001 0001 0012 CTR SET 1
3000 0013 PSECT X'3000'
      0402 0014 START INH
3001 0115 0015 LDV A,MPPERR INITIALIZE INTERRUPT VECTOR
3002 3089
      D01F 0016 STR A,X'42'
3004 0042
      0115 0017 LDV A,X'AAAA' INITIALIZE STATUS TABLE TO AAAA
3005 0115
      AAAA 0018 LDV X,TBLSIZ
3006 AAAA
      0135 0019 STRTBL STR A,TABLE,X
3007 0135
      0068 0020 DECR X
3008 0068
      D11F 0021 SKP STRTBL
3009 D11F
      4000 0022 LDV D,TABLE INITIALIZE TABLE POINTER
300A 4000
      0722 0023 ZERO A DISABLE INTERNAL INTERRUPTS.
300B 0722
      2FFC 0024 DTOR A,X'3E'
300C 2FFC
      01D5 0025 *
300D 01D5
      4000 0026 * TURN ON FORCE ERROR IN RIGHT BYTE. NO ERROR SHOULD OCCUR.
300E 4000
      0600 0027 *
300F 0600
      187E 0028 LDV B,1+1@9
3010 187E
      0195 0029 STR B,MPPCMD SAVE MPP COMMAND WORD
3011 0195
      0201 0030 DTOR B,MPP
3012 0201
      D09F 0031 ADDV D,8 UPDATE STATUS POINTER
3013 D09F
      3085 0032 DSPL D DISPLAY IT
3014 3085
      1C79 0033 CTR SET CTR+1
3015 1C79
      01D9 0034 *
3016 01D9
      0008 0035 * WRITE INTO MEMORY, NO ERROR SHOULD OCCUR.
3017 0008
      05C4 0036 LDV A,X'FFFF'
3018 05C4
      0002 0037 STR A,PERRRB
      0115 0038 ADDV D,8 UPDATE STATUS POINTER
3019 0115
      FFFF 0039 DSPL D DISPLAY IT
301A FFFF
      D01F 0040 CTR SET CTR+1
301B D01F
      3086 0041 * TURN ON FORCE ERROR IN LEFT BYTE. NO ERROR SHOULD OCCUR.
301C 3086
      01D9 0042 *
301D 01D9
      0008 0043 LDV B,1+1@10
301E 0008
      05C4 0044 STR B,MPPCMD SAVE MPP COMMAND WORD
301F 05C4
      0003 0045 DTOR B,MPP
      0195 0046 ADDV D,8 UPDATE STATUS POINTER
3020 0195
      0401 0047 DSPL D DISPLAY IT
3021 0401
      D09F 0048 CTR SET CTR+1
3022 D09F
      3085 0049 * WRITE INTO MEMORY, NO ERROR SHOULD OCCUR.
3023 3085
      1C79 0050 *
3024 1C79
      01D9 0051 STR A,PERRLB
3025 01D9
      0008 0052 ADDV D,8 UPDATE STATUS POINTER
3026 0008
      05C4 0053 DSPL D DISPLAY IT
3027 05C4
      0004 0054 CTR SET CTR+1
3028 D01F
3029 3087
302A 01D9
302B 0008
302C 05C4
      0005

```

466-5-14

Figure 5-6. MPP Demonstration Program (Sheet 1 of 4)

88A00508A-E

```

0055 * TURN OFF FORCE, WRITE INTO MEMORY.
0056 * NO ERROR SHOULD OCCUR.
0057 *
302D 0195 0058 LDV B,1
302E 0001
302F D09F 0059 STR B,MPPCMD SAVE MPP COMMAND WORD
3030 3085
3031 1C79 0060 DTOR B,MPP
3032 D01F 0061 STR A,NOERR
3033 3088
3034 01D9 0062 ADDV D,8 UPDATE STATUS POINTER
3035 0008
3036 05C4 0063 DSPL D DISPLAY IT
0006 0064 CTR SET CTR+1
0065 * READ FROM PERRRB, ERROR SHOULD OCCUR.
0066 *
3037 C01F 0067 LDR A,PERRRB
3038 3086
3039 01D9 0068 ADDV D,8 UPDATE STATUS POINTER
303A 0008
303B 05C4 0069 DSPL D DISPLAY IT
0007 0070 CTR SET CTR+1
0071 *
0072 *
0073 * READ FROM PERRLB, ERROR SHOULD OCCUR.
0074 *
303C C01F 0075 LDR A,PERRLB
303D 3087
303E 01D9 0076 ADDV D,8 UPDATE STATUS POINTER
303F 0008
3040 05C4 0077 DSPL D DISPLAY IT
0008 0078 CTR SET CTR+1
0079 * READ FROM NOERR, NO ERROR SHOULD OCCUR.
0080 *
3041 C01F 0081 LDR A,NOERR
3042 3088
3043 01D9 0082 ADDV D,8 UPDATE STATUS POINTER
3044 0008
3045 05C4 0083 DSPL D DISPLAY IT
0009 0084 CTR SET CTR+1
0085 *
0086 * TURN ON PGM AND DMA WRITE PROTECT FOR 4K-8K BLOCK.
0087 * NO ERROR SHOULD OCCUR.
0088 *
3046 0115 0089 LDV A,X'FO'
3047 0050
3048 1879 0090 DTOR A,MPP
3049 0119 0091 ADDV A,1@11
304A 0800
304B 1879 0092 DTOR A,Mpp
304C 0195 0093 LDV B,6
304D 0006
304E D09F 0094 STR B,MPPCMD SAVE MPP COMMAND WORD
304F 3085
3050 1C79 0095 DTOR B,MPP
3051 01D9 0096 ADDV D,8 UPDATE STATUS POINTER
3052 0008
3053 05C4 0097 DSPL D DISPLAY IT
000A 0098 CTR SET CTR+1
0099 * WRITE INTO PROTECTED CORE, ERROR SHOULD OCCUR.
0100 *
3054 D01F 0101 STR A,X'1000'
3055 1000
3056 01D9 0102 ADDV D,8 UPDATE STATUS POINTER
3057 0008
3058 05C4 0103 DSPL D DISPLAY IT
000B 0104 CTR SET CTR+1

```

Figure 5-5. MPP Demonstration Program (Sheet 2 of 4)

88A00508A-E

```

0105 * DMA INTO PROTECTED CORE, ERROR SHOULD OCCUR.
0106 *
3059 0115 0107 LDV A,1
305A 0001
305B D01F 0108 STR A,X'22'
305C 0022
305D 0115 0109 LDV A,X'1000'
305E 1000
305F D01F - 0110 STR A,X'23' DISK CAR
3060 0023
3061 188E 0111 DSTAT1 DTIR A,X'E'
3062 0117 0112 ANDV A,X'60'
3063 0060
3064 25FC 0113 SKN DSTAT1
3065 0115 0114 LDV A,X'4000'
3066 4000
3067 184E 0115 DTOR A,X'E'
3068 188E 0116 DSTAT2 DTIR A,X'E'
3069 0117 0117 ANDV A,X'60'
306A 0060
306B 25FC 0118 SKN DSTAT2
306C 01D9 0119 ADDV D,8 UPDATE STATUS POINTER
306D 0008
306E 05C4 0120 DSPL D DISPLAY IT
000C 0121 CTR SET CTR+1
0122 *
0123 * TURN OFF WRITE PROTECT.
0124 *
306F 0600 0125 ZERO A
3070 1879 0126 DTOR A,MPP
3071 0119 0127 ADDV A,1@11
3072 0800
3073 1879 0128 DTOR A,MPP
0129 *
0130 * TURN ON PROGRAM TIME OUT, WAIT FOR ERROR.
0131 * ERROR SHOULD OCCUR.
0132 *
3074 0133 LAST EQU $
3074 0195 0134 LDV B,8
3075 0008
3076 D09F 0135 STR B,MPPCMD SAVE MPP COMMAND WORD
3077 3085
3078 1C79 0136 DTOR B,MPP
3079 0600 0137 ZERO A
307A 0702 0138 DECR A
307B 25FE 0139 SKN $-1
307C 01D9 0140 LSTRTN EQU $ UPDATE STATUS POINTER
307D 0008 0141 ADDV D,8
307E 05C4 0142 DSPL D DISPLAY IT
000D 0143 CTR SET CTR+1
0144 *
0145 * TURN OFF PTO, JUMP TO BUS AT 3A00.
307F 0680 0146 ZERO B
3080 D09F 0147 STR B,MPPCMD SAVE MPP COMMAND WORD
3081 3085
3082 1C79 0148 DTOR B,MPP
3083 7400 0149 JMP **+1
3084 3A00 0150 DC X'3A00'
3085 0151 MPPCMD DS 1
0152 *
0153 *
0154 * ERROR LOCATIONS
3086 0155 PERRRB DS 1 PARITY ERROR, RIGHT BYTE
3087 0156 PERRLB DS 1 PARITY ERROR, LEFT BYTE
3088 0157 NOERR DS 1 NO PARITY ERROR
0158 EJECT

```

Figure 5-6. MPP Demonstration Program (Sheet 3 of 4)

88A00508A-E

```

0159 * MPP INTERRUPT ROUTINE.
0160 *
0161 MPPERP EQU *
0162 SARS SAVE
3089 F0DF
308A 30A4
308B 0620 0163 ZERO X
308C D900 0164 STR A,O,X,1
308D 072E 0165 INCR X
308E C01F 0166 LDR A,X'7A' RETURN ADDRESS
308F 007A
3090 D900 0167 STR A,O,X,1
3091 072E 0168 INCR X
3092 1CB9 0169 DTIR B,MPP STATUS
3093 D980 0170 STR B,O,X,1
3094 072E 0171 INCR X
3095 1DB9 0172 DTIR C,MPP MEMORY ADDRESS
3096 D9A0 0173 STR C,O,X,1
3097 072E 0174 INCR X
3098 1AB9 0175 DTIR Y,MPP MEMORY DATA
3099 D940 0176 STR Y,O,X,1
309A 0106 0177 SUBVC A,LAST
309B 3074
309C 2FDF 0178 SKP LSTRTN
0179 * RE-ENABLE MPP & CLEAR STATUS
0180 LDR B,MPPCMD
309D C09F
309E 3085
309F 1C79 0181 DTOR B,MPP
30A0 F09F 0182 LARS SAVE
30A1 30A4
30A2 0112 0183 RTNIV X'7A'
30A3 007A
30A4 0184 SAVE DS 9
4000 0185 ORG X'4000'
4000 0186 TABLE DS 8*CTR
0068 0187 TBLSIZ EQU *-TABLE
3000 0188 END START
0000 ERRS

```

Figure 5-6. MPP Demonstration Program (Sheet 4 of 4)

input /output operations

6

There are two basic types of I/O operations on a GA-16/110 computer:

1. Programmed I/O (PIO)
2. Interrupt-Driven Programmed I/O

Two additional basic types are added for a GA-16/220:

1. Direct Memory Access via Data Channel I/O (DCIO)
2. Direct Memory Access (DMA) Direct I/O

A serial I/O controller for TTY/CRT is also added on a GA-16/220 which utilizes PIO (with or without interrupts).

They may work separately or in conjunction with one another. However, DCIO operations are always initiated by a PIO instruction (Section 4.14). The purpose of these operations is to transfer data, status, and commands between the CPU and a peripheral device. In some cases, another CPU may be the peripheral device. These types of operations will be discussed separately in the following sections.

6.1 PIO OPERATIONS

Programmed I/O derives its name from the fact that it is part of a program. Each PIO operation is derived from an instruction in a program stream. These PIO instructions are sometimes referred to in a general sense as execute I/O (XIO) instructions. They really mean the same thing. To be consistent in this manual, they will always be referred to as PIO. The PIO implies that a program "knows" when these instructions will occur, since they are part of a program. This is not true for interrupt-driven programmed I/O, DCIO, or DMA.

All peripheral units associated with a GA-16/110/220 communicate with the CPU via the I/O Bus. Each PIO instruction specifies the controller to which it refers by a 6-bit device select code. Each controller interfaced to a particular GA-16/110/220 must have a unique select code and a selection network to recognize that select code when it appears on the I/O Bus. When an instruction is executed, only the controller whose select code is specified will respond to the indicated operation. A number of device select codes have been assigned to standard GA controllers; these assignments are listed in Table 6-1. The six select code bits are the least significant bits of the PIO instruction. To simplify logic and programming, the interrupt vector associated with a controller is usually equal to the device select code plus forty.

Table 6-1. Standard Device Select Codes and Interrupt Assignments
(Sheet 1 of 2)

Hexadecimal Device Select Code	Controller	Hexadecimal Interrupt Vector Location
00 } 01 } 02 } 03 } 04 } 05 } 06 }	Floating Point Processor	NA
07	GAARD	
08	Paper Tape Reader	048
09	Paper Tape Punch	049
0A	¹ Floppy Disk or Cassette	04A
0B	¹ Card Reader	04B
0C	¹ Line Printer	04C
0D	¹ Card Punch	04D
0E	¹ Disk	04E
0F	¹ Magnetic Tape	04F
10	¹ Selectric	050
11	Plotter	051
12	¹ Fixed-Head Disk	052
13	¹ 2nd Fixed-Head Disk	053
14		
15		
16		
17		
18	2nd Paper Tape Reader	058
19	2nd Paper Tape Punch	059
1A	¹ 2nd Cassette	05A
1B	¹ 2nd Card Reader	05B
1C	¹ 2nd Line Printer	05C
1D	¹ 2nd Card Punch	05D
1E	¹ 2nd Disk	05E
1F	¹ 2nd Magnetic Tape	05F
20 } 21 } 22 } 23 }	² } 1590 Communication MUX or. 1561 or 1581 Asynchronous Controller 201 Synchronous Controller	060 061 062 063

¹Commonly used DMA devices (GA-16/220 only) (see Table 6-3) for data channel, scan control register (SCR), and channel address register (CAR) assignments. To use these devices on a GA-16/110, a DMT controller must be specified.

²Two consecutive codes for Process I/O AC Input/Output, Variable Threshold Digital input, Digital input, or Digital output control up to 3F, depending on availability of channels. No interrupt vectors used.

Table 6-1. Standard Device Select Codes and Interrupt Assignments
(Sheet 2 of 2)

Hexadecimal Device Select Code	Controller	Hexadecimal Interrupt Vector Location
24	2nd 1590 Communication MUX or 1561 or 1581 Asynchronous Controller	064
25		065
26		066
27		067
28	2nd 201 Synchronous Controller	068
29		069
2A		
2B		06B
2C	3rd 1590 Communication MUX or 1561 or 1581 Asynchronous Controller	06C
2D		06D
2E		06E
2F		06F
30	4th 201 Synchronous Controller	070
31		071
32		072
33		
34	5th 1590 Communication MUX or 1561 or 1581 Asynchronous Controller	074
35		075
36		076
37		
38	High-Speed Data Channel SCR Input	}
39	MPP	
3A	High-Speed,	
3B	Signed	
3C	Multiply/	}
3D	Divide	
	⁵ Memory Mode Mask } Output ⁶ Interrupt Mask } ⁶ Console Switches - Input ⁶ Real-Time-Clock Interrupt ⁶ Console Interrupt ⁶ TTY	NA
3E		043
		047
3F		045

³Range of select codes used for Process I/O Analog-to-Digital controllers.

⁴Range of select codes used for Process I/O Digital-to-Analog controllers.

⁵32K/64K memory mode (Section 4.15.2) (GA-16/220 only).

⁶Interrupts enabled by setting interrupt mask (Section 4.15.2). On a GA-16/220, mask is internal. On a GA-16/110, mask is contained in 1582 Controller, but only TTY interrupt bit is used.

PIO instructions provide four types of operations:

- Data Input (DTIR/M)
- Data Output (DTOR/M)
- Control (CTRL)
- Test (TEST)

When the CPU executes an I/O instruction, it generates a sequence of signals on the I/O Bus and responds to signals generated by the selected controller. A controller must recognize and respond to the signals generated by the CPU and thereby perform the operation dictated by an instruction. Often only a subset of the I/O instructions apply to a particular controller; in these cases, the controller must ignore irrelevant instructions. In all cases, it is the responsibility of the programmer to know what effect, if any, a particular I/O instruction will have on a particular controller.

The basic PIO interface signals associated with the PIO instruction mnemonics are listed in Table 6-2. These signals (and others required for timing, interrupts, etc.) will be described in greater detail beginning with Section 6.9 to aid in designing an I/O controller. The description that follows is primarily to familiarize the reader with the interactions which occur when the various PIO instructions are executed. Additional descriptions of PIO instructions are contained in Section 4.14.

During any PIO instruction, FAP (Function/Address Pulse) occurs to signal controllers to decode the six address bits. If a controller detects its own address combination, it is to respond to the function bits in the proper mode, READ, WRIT, CNTL, or TEST. In a READ or WRIT mode, one word of data is transferred to or from the CPU on the In-Bus (INB00-15) or Out-Bus (OTB00-15). A Data Transfer Pulse (DTP) is used to synchronize the data transfer. It occurs when the output data is available for the controller or to signal the controller that the CPU has the input data and that data may be removed from the In-Bus. DTP also occurs after a TEST or CNTL instruction. In those cases, it has no specific function but may be used to indicate that the TEST or CNTL operation is finished.

In a TEST operation, the addressed controller's status is interrogated. Bits 8, 9, and 10 of the PIO instruction are used to specify a register for a READ or WRIT operation; however, since no register need be specified during a TEST operation, these bits are used to specify which one of eight (maximum) controller conditions are to be tested. These conditions might be such signals as BUSY, READY, ERROR, etc. If the specified condition tested is true, the CPU advances to P+2. If the condition is false, the program continues at the next location, P+1. The contents of P+1 would typically contain a jump back to continue the test operation until the condition was satisfied.

A particular controller may not need a full eight test lines or it may need more than eight if it is controlling a relatively complex device. In the latter case, a DTIR or DTIM may be used to input a full 16-bit status word and the burden is put on the program to interpret the word.

Table 6-2. I/O Signal Summary

Description of I/O Operation	Instruction Mnemonic	Hardware Interface Signal	Controller Action
Control	CTRL	FAP- → CNTL- → OTB00 - OTB15 →	Device Address Selection Control Function (Decodes Instruction)†
Test	TEST	FAP- → OTB00 - OTB15 → TEST- ←	Device Address Selection Test Function (Decodes Instruction)†
Data Input	DTIR or DTIM	FAP- → OTB00 - OTB15 → READ- → INB00 - INB15 ← DTP- ←	Device Address Selection Phase (Decodes Instruction)† Data Transfer Phase
Data Output	DTOR or DTOM	FAP- → OTB00 - OTB15 → WRIT- → OTB00 - OTB15 → DTP- ←	Device Address Selection Phase (Decodes Instruction) Data Transfer Phase

→ To Controller

← To CPU

† OTB00 - OTB05 Device Address

OTB08 - OTB10 Function (on CTRL & Test)

(entire instruction is available for use by controller
but normally a controller will only use above signals)

In a CTRL operation, one of eight control functions is decoded by the addressed controller. These functions are typically used to issue commands to a controller, such as Transmit, Read Card, or Enable Interrupt.) Again, the instruction bits 8, 9, and 10 are not used to specify a register, but instead are used to specify which one control function of eight is to be decoded.

If a particular controller has the ability to perform more than eight commands, a DTOR/M instruction can be used to output a full 16-bit command word. The burden is put on controller logic to interpret the command word.

A simple example of how PIO instructions might be used with a standard paper tape reader is shown on the following page. The program is not complete in itself, but merely shows how typical PIO instructions might be used.

<u>Location</u>	<u>Instruction</u>			
0202	1108	CTRL	1,X'08'	Disable Interrupt
0203	11C8	TEST	1,X'08'	Skip to 0205 if Ready
0204	73FE	JMP	\$-1	Back to 0203 if Not Ready
0205	1308	CTRL	3,X'08'	Step reader to next frame
0206	10C8	TEST	0,X'08'	Skip to 0208 if operation complete
0207	73FE	JMP	\$-1	Not complete, back to TEST
0208	1888	DTIR	A,X'08'	Read Character to A register
0209	D100	STR	A,0,X	Store A indexed

The program is Test loop driven, thus the interrupt on the controller is disabled by outputting control function one, to device address 08, the paper tape reader. This turns off the interrupt mask in the controller. The TEST instruction in location 203 checks test function number one, Reader Ready. If it is not true, the TEST-line is false and the next instruction is fetched, which jumps back to the TEST instruction. If the Ready line is true, the TEST-line is true and the program skips the next instruction and goes to P+2 location 0205. Here it issues control function three, which the controller logic recognizes as a step command. In location 0206, test function zero, operation complete, is interrogated. Until the reader advances to the next frame, the TEST-line will be false and the program will loop between 0207 and 0206. When operation complete comes true, the JMP at 0207 is skipped. In location 0208, one frame of data is input from the Reader to the A register. The STR instruction in 0209 stores Register A, wherever the index register specifies.

In general, the I/O program steps are as shown in Figure 6-1.

Most data transfer operations involve more than one character or number. Program steps will then be required to keep track of how many characters have been transferred and where in memory the characters must be written/read.

Another example of test loop-driven, programmed I/O for serial I/O (TTY) is given in Section 4.15.3.

Test loop-driven programmed I/O can be an effective method of conducting data transfers if there is only one peripheral unit attached to the system or in cases where there are two units and one uses Direct Memory Access. However, test loop-driven programmed I/O requires the full attention of the CPU while the program is looping; during this time, the CPU cannot perform any other tasks.

There are two situations in which test loop-driven programmed I/O is insufficient:

1. Where the peripheral unit has to initiate the data transfer; there is no way in which a program can anticipate a peripheral unit's requests to transmit data at irregular intervals. This situation calls for the use of interrupt-driven programmed I/O.
2. When very high-speed data rates are needed, every character that is transmitted via programmed I/O requires execution of between 5 and 20 instructions (which may take as long as 50 microseconds). This limits data transfer rates to 20,000 characters per second, which is too slow for disk and tape units and may not be fast enough for some high-speed instruments. For high-speed data transfers,

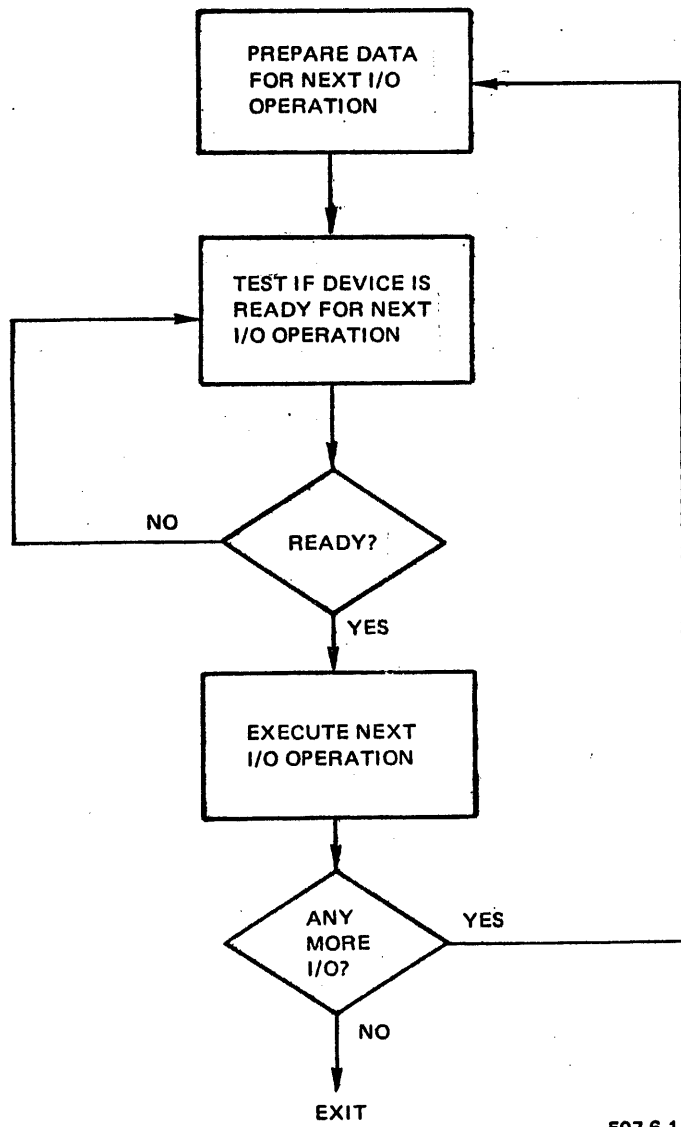


Figure 6-1. Typical Programmed I/O Sequence

Direct Memory Access (DMA), which allows a block of data to be transferred under control of the peripheral controller, is required.

6.2 INTERRUPT-DRIVEN PROGRAMMED I/O

Unlike test loop-driven programmed I/O, interrupt-driven programmed I/O permits the CPU to perform other tasks between the transfer of data items.

With the interrupt mask enabled (usually by a Control command), the computer can continue processing other programs and yet be able to service I/O requests from peripheral units on demand. When the peripheral is ready to transmit data, it forces a programm interrupt. In response to the interrupt, the CPU temporarily stops executing the current program and branches to a program (called the "interrupt service routine") that services interrupts from that particular unit.

The GA-16/110/220 system can accommodate up to 64 addressable units. Each of these units has associated with it a dedicated memory location, called an "interrupt vector", which contains the address of that unit's interrupt service routine. When an interrupt signal enters the computer, the system will automatically cause execution to jump indirectly (through the interrupt vector) to the interrupt service routine.

Interrupt-driven programmed I/O can be initiated by programming a jump to an I/O initialization routine designed to initiate I/O with the selected Unit. Along with the jump-to-subroutine (JSR) instruction, the programmer must also specify the number of data items to be transferred (words or characters) and the memory location to/from which these items are to be transferred. Depending on the peripheral unit, the programmer may also be required to specify the drive selected and, for multi-function units, the operation to be performed. In the examples given in this section, the calling sequence looks like this:

JSR	DEVIN	Jump to I/O initialization routine dedicated to the required peripheral device.
DC	M	Define M number of data items to be transferred.
DC	BUFFER	Define starting address of data buffer in memory, usually some relocatable address.
BYTE	N,P	Select drive N, specify operation P is to be performed, if required.
DC	Q	Define additional parameter (e.g., disk sector number) if required.

The initialization routine (DEVIN above) must perform the following functions:

1. Save the current contents of working registers and status register, usually in nine consecutive memory locations.
2. Increment the dynamic storage pointer by nine to address the next free storage word if system storage is being used.

3. Set up D register to fetch parameters from the calling program.
4. Enable master interrupt (ISE).
5. Fetch word count and buffer address and store in memory.
6. Set up interrupt vector to point to interrupt service routine.
7. Decode drive and operation indexes if applicable.
8. See if peripheral unit is ready to transfer data.
9. Initiate transfer.
10. Restore the saved contents of working registers and status register.
11. Increment E register, which contains return address, to location past parameter data items following JSR.
12. Return to calling program.

When control has been returned, execution of the calling program continues until the peripheral unit is ready to send/receive another data item. At that time, the unit interrupts the CPU, which causes program control to be transferred to the interrupt service routine. The interrupt service routine must perform the following functions:

1. Save the current contents of working register in nine consecutive memory locations.
2. Perform operation required by interrupt.
3. Restore the saved contents of working registers and status register.
4. Return control to interrupted program.

The interrupt system is described in Section 2.6.

Like test loop-driven programmed I/O, interrupt-driven programmed I/O may prove too slow to accommodate data transfer from some high-speed units. For this reason, the controllers for high-speed peripheral units are designed to use Direct Memory Access.

6.3 GA-16/220 DIRECT MEMORY ACCESS VIA DATA CHANNEL I/O (DCIO)

Data Channel I/O operations transfer data via the Direct Memory Access (DMA) and are usually reserved for high-speed, block or record-oriented devices such as disk or magnetic tape units. The bookkeeping a program must do to effect the orderly transfer of a large block of data may consume a prohibitive amount of time. In other words the time taken by a PIO operation to test the status of a device, input a word of data, store the data in memory, increment the storage address and, finally, jump back to the start of the loop may take more time than the data transfer rate allows. In that case

some data words would be lost. A faster means of transferring data to/from a device is to inform the controller where the data is and how much data to transfer. The controller then relieves the bookkeeping burden of the program and lets the hardware effect the data transfer to or from memory.

Before a program initiates a DCIO block transfer operation, it must first store the size of the block to be transferred in the appropriate Scan Control Register (SCR) location and the starting address minus one in a Channel Address Register (CAR) location. The starting address refers to where in memory the data transfer is to take place. The SCR and CAR for various data channel devices are listed in Table 6-3. After these registers are initialized, the program can test the device with a TEST instruction (or a DTIR/M which inputs a status word, depending on controller design) to see if it is ready to accept a command. If it is, a CTRL instruction (or a DTOR/M which outputs a command word, depending on controller design) is issued to start the DCIO block transfer. The actual block transfer is then controlled by hardware located in the device, the device controller, the data channel module, and CPU logic. The standard GA data channel module is called a Multiple High-Speed Data Channel Controller (MHSDC).

When the initialize command is issued, the device controller responds by telling the MHSDC to get the contents of its SCR and CAR locations out of dedicated memory and load them into appropriate registers on the MHSDC. The device controller then tells the device to start transferring data. Every time the device is ready with a word of data, a request for transfer (DREQ) is issued and the CPU responds with an acknowledge (DACK). The CAR register on the MHSDC connects to the DMA In-Bus to specify the location of the transfer. At that time, the transfer takes place and a cycle is "stolen" by the data channel, between CPU instruction cycles. The CPU instruction stream (Program) is not broken or interrupted, it may be postponed while the transfer takes place. Every time one word is transferred, the MHSDC increments the CAR by one to point to the next consecutive location in the data table, and the SCR is decremented by one to indicate the remaining size of the block to be transferred. The operation continues until the SCR is decremented to zero, to indicate the final word transfer. When this occurs, the MHSDC issues a Data Channel Block Complete (DCBC) signal that is passed to the device controller. Typically, the device controller then terminates the command to the device and turns on an 'Operation Complete' signal. This signal can be monitored by the CPU using a TEST instruction (or a DTIR/M which inputs a status word, depending on controller design), or if interrupts have been enabled (Mask and ISE on), waiting for an interrupt.

6.4 INTERFACING TO THE DATA CHANNEL BUS, MHSDC

As mentioned previously, General Automation implements most of its GA-16/220 DMA operations via an MHSDC controller containing eight high-speed channels on a single module, as opposed to putting a single channel (address and block-count registers and associated control logic) on each controller requiring direct memory transfers. This technique eliminates a considerable amount of logic on those controllers requiring data-channel operation and results in a much more efficient system in the large majority of applications. With each data channel is associated a pair of memory cells dedicated to block-count and starting-address information (see Data Channel assignments, Table 6-3).

Table 6-3. Data Channel Assignments for Standard Peripheral Units

Channel	¹ SCR (Hex)	² CAR (Hex)	Peripheral Unit
0	20	21	3342 Head-per-Track Device
1	22	23	3341, 43, 46, 47 Disk
2	24	25	3331, 32, 33 Magnetic Tape
3	26	27	Card Reader or Reader Portion of Card Reader/Line Printer
4	28	29	"Floppy" Disk or Cassettes
5	2A	2B	Card Punch
6	2C	2D	Not Assigned
7	2E	2F	Line Printer
8	30	31	Not Assigned
9	32	33	Not Assigned
10	34	35	Not Assigned
11	36	37	Not Assigned
12	38	39	Not Assigned
13	3A	3B	Not Assigned
14	3C	3D	Not Assigned
15	3E	3F	Not Assigned

} Requires installation of 2nd MHSDC. SCR/CAR assignments for peripheral units in order shown above.

¹SCR contains block count, see text.

²CAR contains starting address, see text.

For graphic representation of SCR and CAR locations in memory, see Dedicated Memory Map, Table 2-4.

Figure 6-2 illustrates an example block-transfer memory map. It should be noted that the Scan Control Register (SCR) contains the block-count, chaining and interrupt information, and that the Channel Address Register (CAR) contains the starting address (minus one) of the data. "Chaining refers to the linking together of different data blocks that will appear to the controller as a continuous stream of data. The example shown in Figure 6-2 shows two blocks of data chained together. To indicate that a given block of data is to chain to another block, its SCR word must have bit 15 set. The program also has the alternative of creating an interrupt after any given block. If bit 14 is set, then an interrupt will not be generated by the device at the end of that block and conversely so. It should also be noted that it takes a small amount of logic capability by the controller to chain or to interrupt at the end of a block. Not all General Automation controllers implement both these features

To start a peripheral controller block transfer, a user's program must initialize the dedicated memory SCR and CAR (and any subsequent next starting addresses and SCRs if

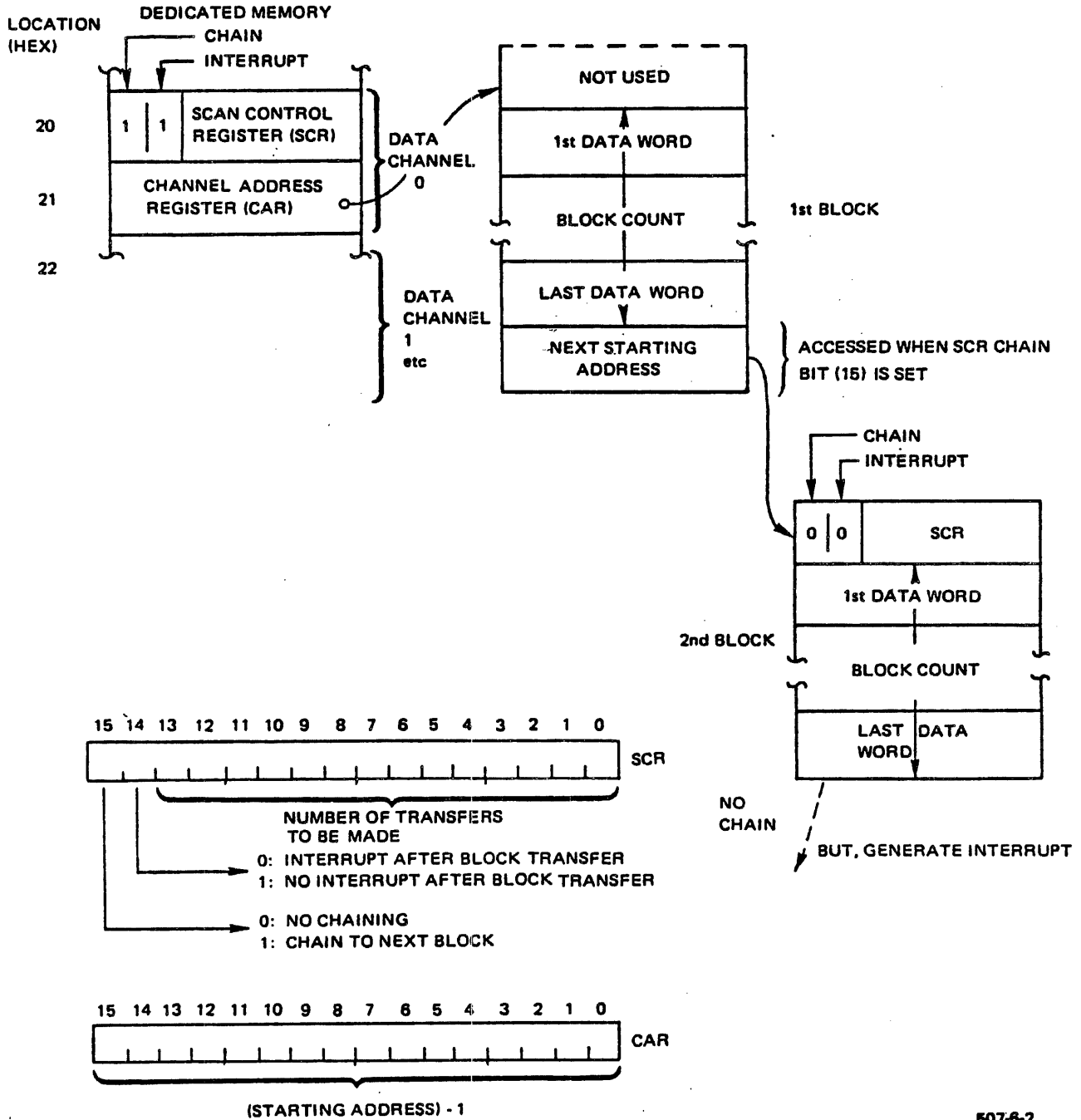


Figure 6-2. Example Block Transfer Memory Map (GA-16/220)

chaining is used), then execute the I/O control instruction that is assigned to that controller to initiate a Read or Write operation. From this point, the rest of the operation is automatic until an interrupt occurs requiring program intervention.

NOTE

A "zero-word" block transfer does not exist logically, and an SCR word containing all zeros is prohibited because the decrement occurs before the chain and interrupt bits are sensed; thereby indicating a false chain. That is, a decrement from 0000 produces FFFF; this indicates a truly huge block transfer.

6.5 GA-16/220 DIRECT MEMORY ACCESS (DMA) DIRECT I/O

Direct memory access directly to a controller functions in much the same manner as DCIO except that the MHSDC module is not used. A DMA device has its own logic for keeping track of block size and addressing memory. It can be a block or word-oriented device and connects directly to the I/O Bus.

Dedicated memory locations for CAR and SCR are not used. Instead, the CAR and SCR data are loaded into the controller with a DTOR or DTOM instruction before the CTRL instruction (or a DTOR/M which outputs a command word, depending on controller design) is issued to transfer block data via DMA. TEST instructions (or a DTIR/M which inputs a status word, depending on controller design) may be used to determine the status of the device after DMA transfer is complete.

A direct DMA I/O controller might be custom-built by the user when only one or two devices use DMA and the user wishes to eliminate the need for an MHSDC controller. Some high-speed data link controllers also use DMA direct I/O.

6.6 DELETED

6.7 I/O SUMMARY

The figures and tables which follow are provided as a quick reference to the I/O instructions and data for commonly used I/O controllers. Those controllers labeled "(MHSDC)" are usable on a GA-16/220 with MHSDC controller. They do not operate on a GA-16/110. The instruction summaries do not apply to direct DMA controllers.

1. High-Speed Paper Tape Reader, Model 3321.
2. High-Speed Paper Tape Punch, 3322.
3. Floppy Disk Controller, Model 3349 (MHSDC).
4. Controller (Card Reader Section), Model 3356 (MHSDC).
5. Controller (Line Printer Section), Model 3356 (MHSDC).
6. Disk Storage System Controller, Model 3341/3342 (MHSDC).
7. Disk Storage System, Model 3345 (MHSDC).
8. Disk Storage System Controller, Model 3346/3347 (MHSDC).
9. Magnetic Tape Unit Controller, Model 3331, 3332, 3333 (MHSDC).
10. Head-Per-Track Disk Controller, Model 3342 (MHSDC).
11. GA-16/220 Console and Internal Functions (part of system).
12. GA-16/220 TTY Controller (part of system), but also applies to external 1582 TTY Controller.

Normally, a system user will implement I/O by taking advantage of system software. This includes Series-16 Input/Output System (IOS) and the Series-16 processors which call IOS, such as File Management System (FMS); the READ and WRITE instructions provided by the FORTRAN, COBOL, or BASIC compilers; and the I/O macros provided by the CAP-16 Assembler.

Programming of I/O instructions at the level shown in this section is customarily employed for testing and verification (T&V) routines, training of I/O fundamentals, special-purpose I/O utilities, and specialized applications which cannot use standard Series-16 IOS software.

A series of controller manuals are available and provided for controllers ordered as part of the system integration package. Details on hardware functions and appropriate instruction sequences are provided in these manuals. Further information on the I/O instructions and examples for the built-in controller for a teletype, for the console display switches, and for 16/220 console and internal functions are provided in Section 4.15.

NOTE

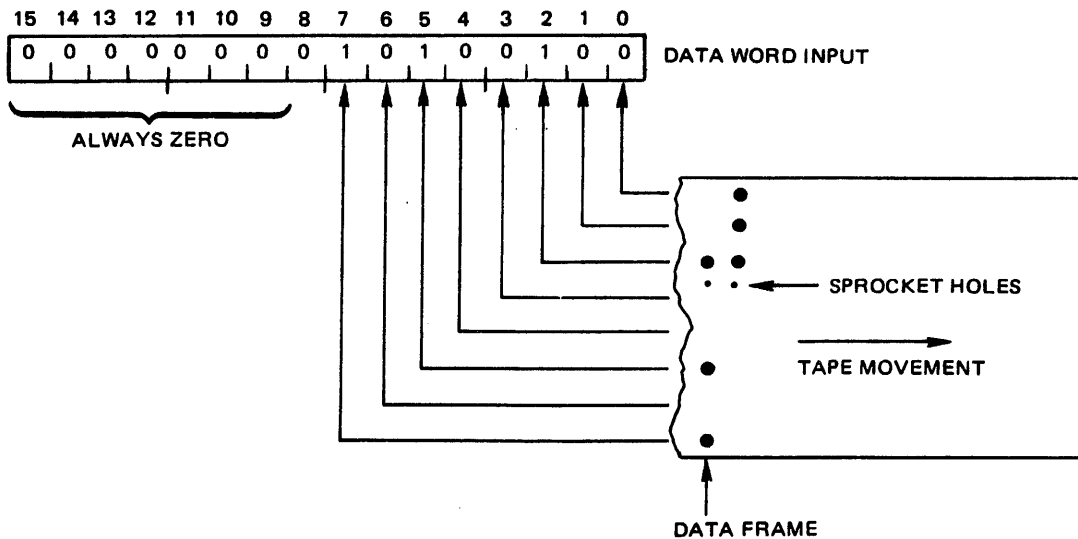
Although not strictly classified as I/O controllers, because they are not interfaced to external devices, the following hardware also uses I/O instruction sequences: Memory Parity Protect (MPP), Signed Multiply/Divide Hardware option, and Floating-Point Hardware option.

Table 6-4. Instruction Summary for High-Speed Paper Tape Reader (HSPTR) Controller (Also reference Controller Manual No. 88A00401A)

Device Select Code: X'8'
 Interrupt Vector: X'48'
 Data Channel Locations: N.A.

Instruction	Function
TEST 0,X'8'	Skip if OPERATION COMPLETE is true
TEST 1,X'8'	Skip if READY is true
TEST 3,X'8'	Skip if COMMAND REJECT is false
CTRL 0,X'8'	ENABLE INTERRUPT from HSPTR
CTRL 1,X'8'	DISABLE INTERRUPT from HSPTR
CTRL 2,X'8'	RESET STATUS indicators on device
CTRL 3,X'8'	Initiate STEP to next data frame
DTIR R,X'8'	INPUT DATA from current data frame into register (R) or into memory
DTIM R,X'8'	

R = A,X,Y,Z,B,C,D or E



243-43

Figure 6-3. Input Data From HSPTR

Table 6-5. Instruction Summary for High-Speed Paper Tape Punch (HSPTP) Controller (also reference Controller Manual No. 88A00401A)

Device Select Code: X'9'
 Interrupt Vector: X'49'
 Data Channel Locations: N.A.

Instruction	Function
TEST 0,X'9'	Skip if OPERATION COMPLETE is true
TEST 1,X'9'	Skip if READY is true
TEST 3,X'9'	Skip if COMMAND REJECT is false
CTRL 0,X'9'	ENABLE INTERRUPT from HSPTP
CTRL 1,X'9'	DISABLE INTERRUPT from HSPTP
CTRL 2,X'9'	RESET STATUS indicators on device
DTOR R,X'9'	OUTPUT data from register (R) or
or	from memory to HSPTP controller;
DTOM R,X'9'	advance tape; punch character

R = A,X,Y,Z,B,C,D or E

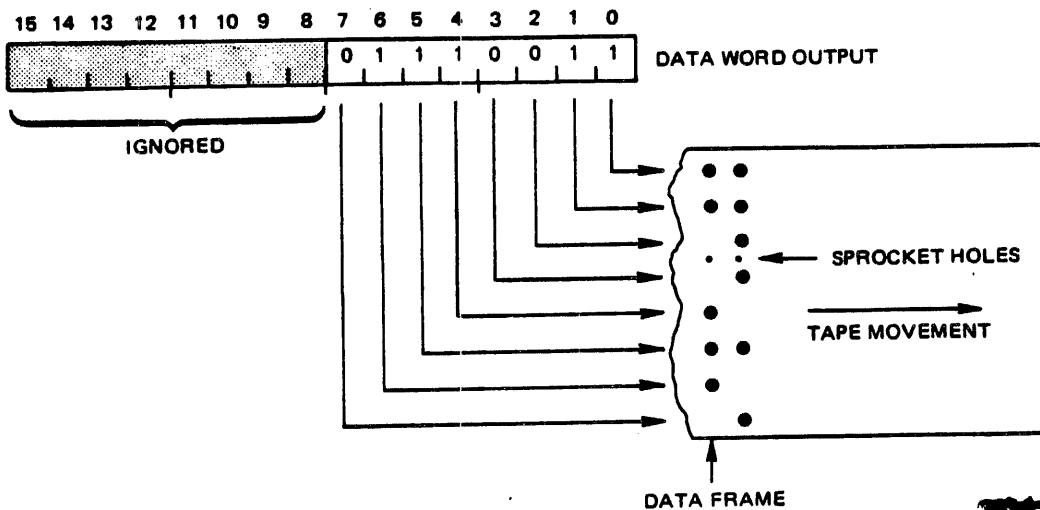


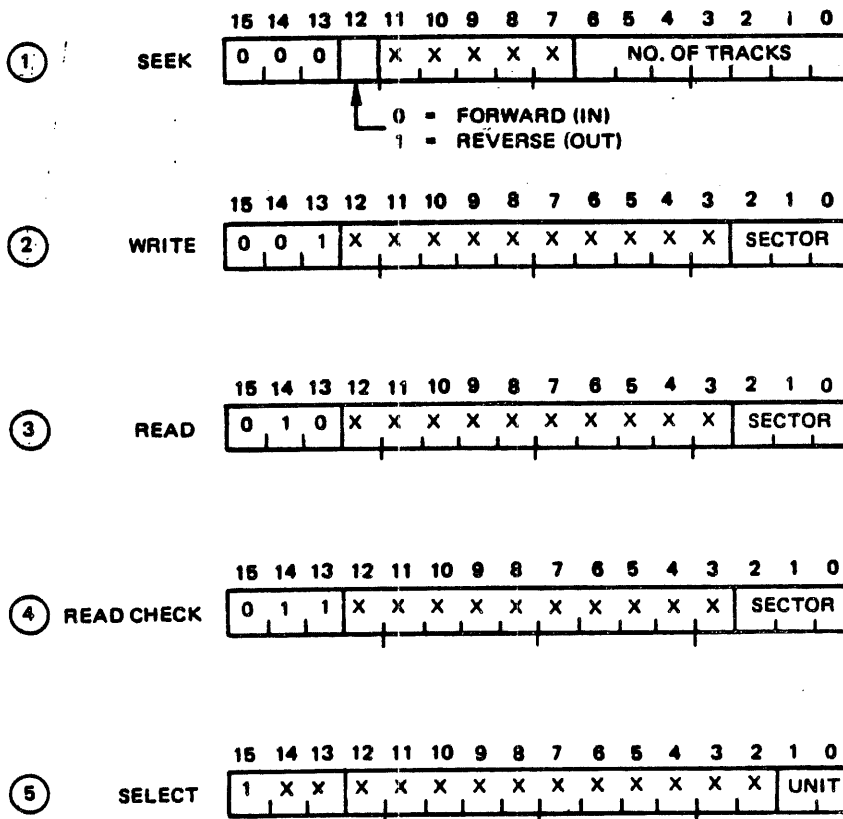
Figure 6-4. Output Data to HSPTP

Table 6-6. Instruction Summary for Model 3349 for Floppy Disk Controller
(Also reference Controller manual 88A00411A)

Device Select Code: X'A'
 Interrupt Vector: X'4A'
 Data Channel Locations: SCR=X'28', CAR=X'29'

Instruction		Function
CTRL	0,X'A'	ENABLE INTERRUPT from disk controller
CTRL	1,X'A'	DISABLE INTERRUPT from disk controller
CTRL	2,X'A'	RESET STATUS indicators (operation complete, command reflect, and data error)
CTRL	3,X'A'	RESET STATUS indicator (file unsafe)
DTOR	R,X'A'	OUTPUT object word from register (R) or from memory to disk controller
or		
DTOM	R,X'A'	
DTIR	R,X'A'	INPUT status word to register (R) or to memory from disk controller
or		
DTIM	R,X'A'	

R = A,X,Y,Z,B,C,D or E



411-4-4

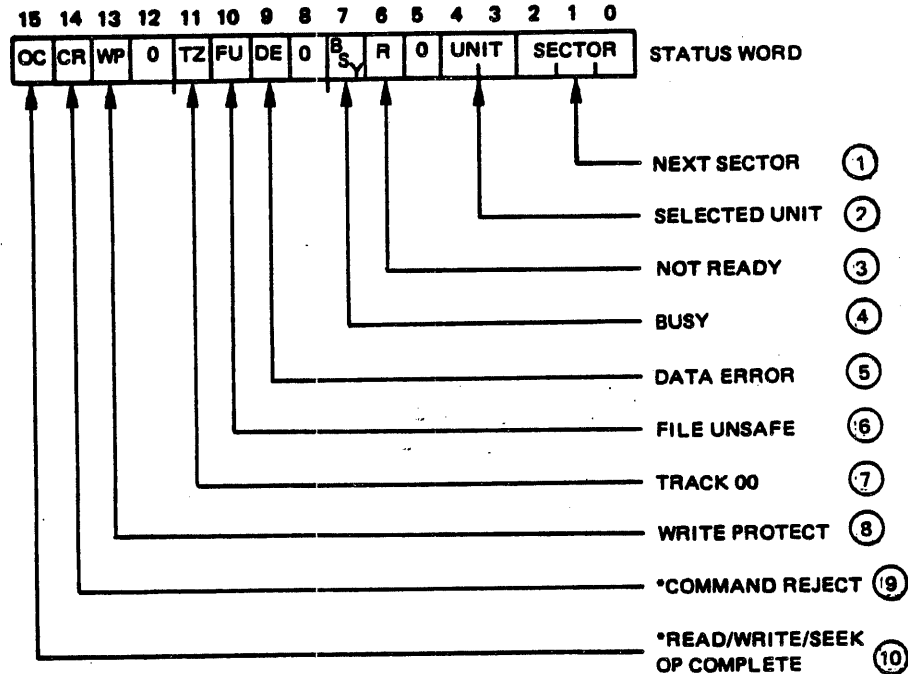
Figure 6-5. Object Word Format to Floppy Disk Controller (Sheet 1 of 2)

- ① The SEEK object word is used to move the read/write head some number of tracks. The number of tracks to be moved may be set from 0 to 127, although there are only 64 tracks available. Bit 12 specifies if the move is to be in (forward) or out (reverse). Normally, when a disk unit is selected for the first time, the head will be positioned at some unknown position. Even though the disk unit will not itself position the head below track 0 it is possible to have the head positioned below track 0 after servicing. The head positioning stepper motor may be manually turned when power to the drive is OFF. In order to ensure positioning of the head at track 0, the following procedure is recommended upon initial power up, power recovery and after a disk has been changed.
- a. Seek forward (in) 16 tracks.
 - b. After operation complete seek reverse (out) 70 tracks.
 - c. After operation complete head should be at track 0.

After the above procedure has been performed all that need be done to return to track 0 from anywhere on the disk is to issue a seek reverse (out) 70 tracks.

- ② The WRITE object word is used to transfer data from the CPU onto the disk. Data will be transferred to the sector specified by the object word, at the track location specified by the last seek and to the disk unit last selected. Data previously in the sector will be lost. The number of words specified must be in the range 1 to 289 (X'1' to X'121'). If a word count less than 289 is specified the controller will write that number of words then fill the balance of the sector with zeroes. There are 289 (X'121') words available on each sector. If a word count greater than 289 is specified, the controller will write only 289 words and set the data error status bit when the operation is complete.
- ③ The READ object word is used to transfer data from the disk unit into the CPU. Data will be transferred from the sector specified by the object word, at the track location specified by the last seek and from the disk unit last selected. The number of words specified must be in the range 1 to 289 (X'1' to X'121'). If a word count greater than 289 is specified, the controller will transfer 289 words and set the data error status bit when the operation is complete.
- ④ The READ CHECK object word is used to check the data in a particular sector against the check character also contained in that sector. The effect is similar to the READ object word except that no data are transferred into the CPU.
- ⑤ The SELECT object word selects one of the four possible disk units connected to the GA-3349 Disk Controller. A SELECT should be executed upon initial start up and power recovery. A particular unit, once selected, remains selected until a SELECT command is executed. SELECT commands are ignored if executed when the disk is busy or not ready. System Reset will select unit 0.

Figure 6-5. Object Word Format to Floppy Disk Controller (Sheet 2 of 2)



411-4-5

- ① **NEXT SECTOR** (bits 0 through 2): These three bits identify the next sector to become available for reading or writing. They are derived from the currently selected disk unit and are updated as the disk spins.
- ② **SELECTED UNIT** (bits 3 and 4): These bits are provided directly from the controllers unit address register and indicate the currently selected disk unit.
- ③ **NOT READY** (bit 6): This bit is true when the selected disk unit does not have power applied or has the cartridge access door open or does not have a disk cartridge installed. It will also be true if a File Unsafe condition exists in the selected disk unit.
- ④ **BUSY** (bit 7): This bit is true when the controller is Busy. While Busy, the controller rejects the execution of input/output commands and ignores select commands. It does, however, correctly respond to sense status commands.
- ⑤ **DATA ERROR** (bit 9): Data Error is set true at termination of a Write or Read command if the word count specified is greater than 289 (X'121'), or if slow data channel access causes lost data (data overrun).
 - Data Error is set true at termination of a Read or Read Check operator if a parity error is detected or if the read operation is not completed when the next sector boundary occurs.

Figure 6-6. Status Word Format for Floppy Disk Controller (Sheet 1 of 2)

- Data Error is reset by a Reset Status Control Command (CTRL 2,X'A') or by the system reset or when an acceptable command is issued to the controller via an execute input/output command (Section 4.2.2).
- ⑥ FILE UNSAFE (bit 10): Certain electronic conditions are monitored in the disk unit to prevent loss of data. If any of these conditions occur, this status condition will be indicated. File Unsafe will also cause a Not Ready status (bit 6). When the File Unsafe indicator is set or ON the write driver circuits are disabled. The conditions that may cause the File Unsafe indicator to be set are:
- Write enable signal and no write current.
 - Write current and no write enable signal.
 - Write enable signal and head not loaded.
 - Write enable signal and no write data.
- (If any of these conditions persist, refer to the Memorex 651 Flexible Disk File Maintenance Manual 2466.002.)
- File unsafe indicator is reset by control command CTRL 3,X'A'.
- ⑦ TRACK 00 (bit 11): This bit indicates that the currently selected unit has the head positioned at track 00.
- ⑧ WRITE PROTECT (bit 13): This bit indicates that the currently selected unit has a disk cartridge installed which cannot be written onto, i.e., is write protected.
- ⑨ COMMAND REJECT (bit 14): Command Reject status when set, indicates abnormal termination of an operation by the controller. Command Reject will be set when any operation is attempted either with a Not Ready storage unit or when the controller is busy with a previous operation. Command Reject status is also set if a write operation is attempted on a write protected disk cartridge.
- Command Reject causes Operation Complete status and interrupt.
 - Command Reject is reset by Reset Status Control Command (CTRL 2,X'A'), by the System Reset, or when an acceptable input/output command is issued to the controller.
- ⑩ OP COMPLETE (bit 15): Op Complete status/interrupt occurs when the controller terminates an accepted read/write or seek operation. This condition also occurs if command reject status is detected. Op Complete is reset by a Reset Status Command (CTRL 2,X'A'), by System Reset, or when an acceptable input/output command is issued to the controller:

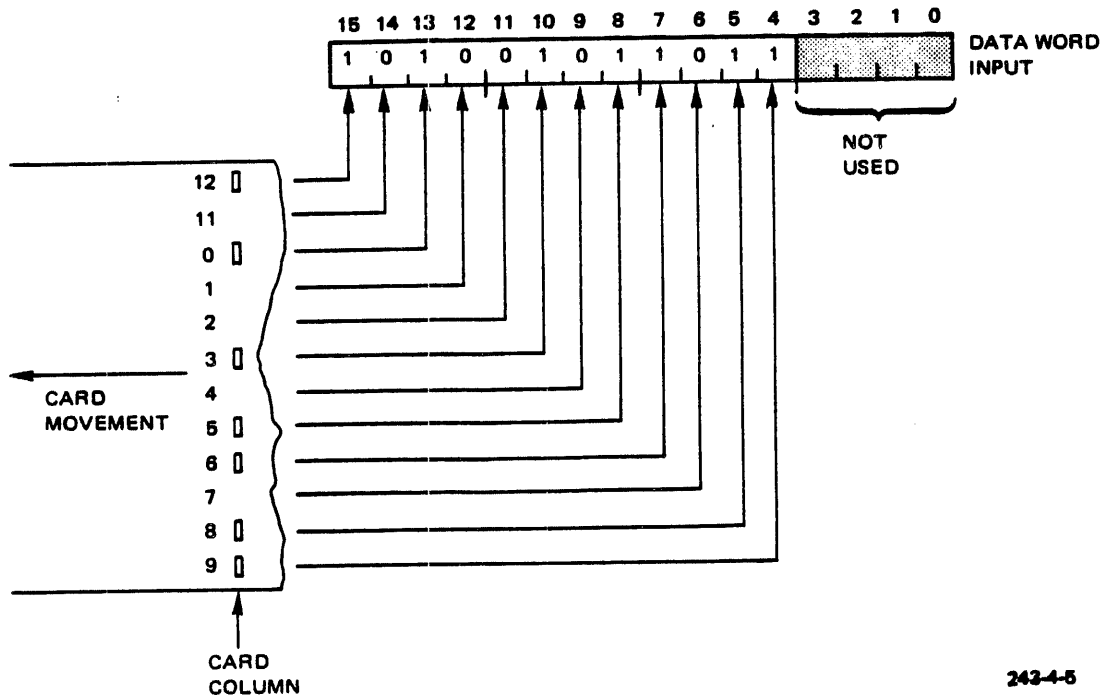
Figure 6-6. Status Word Format for Floppy Disk Controller (Sheet 2 of 2)

Table 6-7. Instruction Summary for the Model 3356 Controller (Card Reader System)
(also reference Controller manual 88A00404A)

Device Select Code: X'B'
 Interrupt Vector: X'4B'
 Data Channel Locations: SCR=X'26'; CAR=X'27'

Instruction	Function
TEST 0,X'B'	Skip if OPERATION COMPLETE is true
TEST 1,X'B'	Skip if READY is true
TEST 2,X'B'	Skip if ERROR is false (i.e., no ERROR)
TEST 3,X'B'	Skip if COMMAND REJECT is false (i.e., no REJECT)
TEST 5,X'B'	Skip if NOT BUSY is true
CTRL 0,X'B'	ENABLE INTERRUPT from CR
CTRL 1,X'B'	DISABLE INTERRUPT from CR
CTRL 2,X'B'	RESET STATUS bits (OPERATION COMPLETE & COMMAND REJECT)
CTRL 3,X'B'	READ card (start card moving and initiate data transfer)

R = A,X,Y,Z,B,C,D,E



243-4-5

Figure 6-7. Read Card Input Format (via DCIO)

Table 6-8. Character Codes for Card Reader

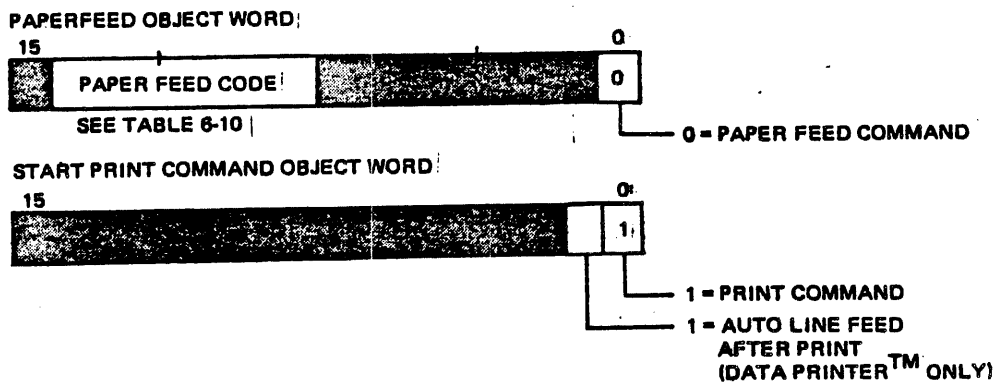
Graphic	Punches	Hexadecimal Memory Contents	Graphic	Punches	Hexadecimal Memory Contents
A	12-1	9000	'	5-8	0120
B	12-2	8800	(12-5-8	8120
C	12-3	8400)	11-5-8	4120
D	12-4	8200	*	11-4-8	4220
E	12-5	8100	+	12-6-8	80A0
F	12-6	8080	,	0-3-8	2420
G	12-7	8040	-	11	0800
H	12-8	8020	.	12-3-8	8420
I	12-9	8010	/	0-1	3000
J	11-1	5000	:	2-8	0820
K	11-2	4800	;	11-6-8	40A0
L	11-3	4400	<	12-4-8	8220
M	11-4	4200	=	6-8	00A0
N	11-5	4100	>	0-6-8	20A0
O	11-6	4080	?	0-7-8	2060
P	11-7	4040	[12-8-2	8820
Q	11-8	4020	\	0-8-2	2820
R	11-9	4010]	0-8-5	2120
S	0-2	2800	↑	7-8-11	4060
T	0-3	2400	←	12-7-8	8060
U	0-4	2200	@	4-8	0220
V	0-5	2100	blank	No Punch	0000
W	0-6	2080	0	0	2000
X	0-7	2040	1	1	1000
Y	0-8	2020	2	2	0800
Z	0-9	2010	3	3	0400
!	11-2-8	4820	4	4	0200
"	7-8	0060	5	5	0100
#	3-8	0420	6	6	0080
\$	11-3-8	4420	7	7	0040
%	0-4-8	2220	8	8	0020
&	12	8000	9	9	0010

Table 6-9. Instruction Summary for the Model 3356 Controller (Line Printer Section)
(also see Controller manual 88A0404A)

Device Select Code: X'C'
 Interrupt Vector: X'4C'
 Data Channel Locations: SCR=X'2E';CAR=X'2F'

Instruction		Function
TEST	0,X'C'	Skip if OPERATION COMPLETE is true.
TEST	1,X'C'	Skip if READY is true.
TEST	2,X'C'	Skip if BOTTOM OF FORM is true.
TEST	3,X'C'	Skip if COMMAND REJECT is false (i.e., NO REJECT).
TEST	4,X'C'	Skip if TRANSFER COMPLETE is true.
TEST	5,X'C'	Skip if BUSY is false (i.e., NOT BUSY).
CTRL	0,X'C'	ENABLE INTERRUPT from line printer.
CTRL	1,X'C'	DISABLE INTERRUPT from line printer.
CTRL	2,X'C'	RESET STATUS bits (COMMAND REJECT & TRANSFER COMPLETE)
CTRL	3,X'C'	RESET OPERATION COMPLETE bit.
DTOR or DTOM	R,X'C'	Output object word from register (R) or from memory to line printer controller; object word specifies a PRINT or PAPER FEED command.

R = A,X,Y,Z,B,C,D, or E

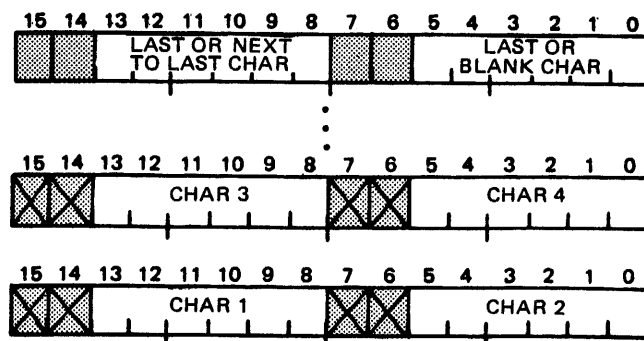


607-6-3

Figure 6-8. Object Word to Line Printer

Table 6-10. Paper Feed Codes for Printer

Bit:	14	13	12	11	10	9	8	Action
0	0	0	0	0	0	0	0	NO LINE FEED
0	0	0	0	0	0	0	0	SKIP 1 LINE THRU SKIP 63 LINES
0	1	1	1	1	1	1	1	DATA TM PRINTER ONLY
1	X	X	0	0	0	0	X	SKIP TO CHANNEL 1 of VFU Tape
1	X	X	0	0	1	0	0	SKIP TO CHANNEL 2 of VFU Tape
1	X	X	0	0	1	1	1	SKIP TO CHANNEL 3 of VFU Tape
1	X	X	0	1	0	0	0	SKIP TO CHANNEL 4 of VFU Tape
1	X	X	0	1	0	1	1	SKIP TO CHANNEL 5 of VFU Tape
1	X	X	0	1	1	0	0	SKIP TO CHANNEL 6 of VFU Tape
1	X	X	0	1	1	1	1	SKIP TO CHANNEL 7 of VFU Tape
1	X	X	1	0	0	0	0	SKIP TO CHANNEL 8 of VFU Tape
1	X	X	1	0	0	1	1	SKIP 1 LINE
1	X	X	1	0	1	0	0	SKIP 1 LINE
1	X	X	1	0	1	1	1	SKIP 1 LINE
1	X	X	1	1	0	0	0	SKIP 1 LINE
1	X	X	1	1	0	1	1	SKIP 1 LINE
1	X	X	1	1	1	0	0	SKIP 1 LINE
1	X	X	1	1	1	1	1	SKIP 1 LINE



466-7-4

NOTE

Each data word includes two characters, as shown above. The controller will always transfer an even number of characters to the line printer. Therefore, if an odd number of characters are to be printed, the even character (bits 5-0) of the last data word must be ASCII "blank" code (i.e., X'20'). The character set is ASCII, 64 characters. Bits 13 and 5 are the most significant bits.

Figure 6-9. Data Words to Line Printer (via DCIO)

Table 6-11. Character Codes for the Line Printer

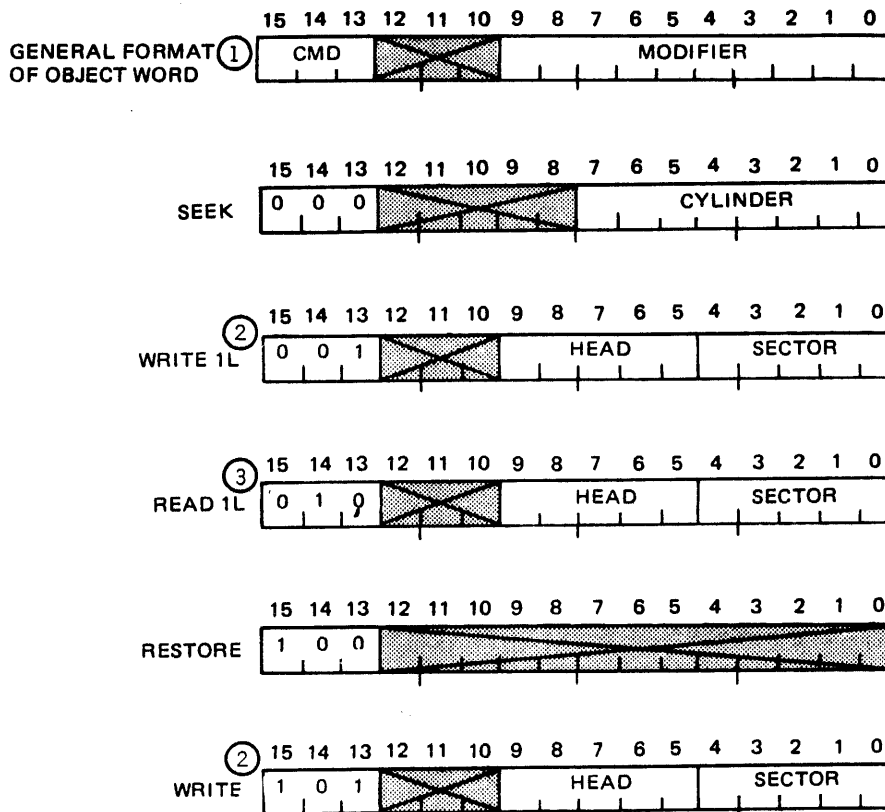
Character	Octal Code	Hex Code	Binary Code	Character	Octal Code	Hex Code	Binary Code
@	00	00	XX000000	Space	40	20	XX100000
A	01	01	XX000001	!	41	21	XX100001
B	02	02	XX000010	"	42	22	XX100010
C	03	03	XX000011	#	43	23	XX100011
D	04	04	XX000100	\$	44	24	XX100100
E	05	05	XX000101	%	45	25	XX100101
F	06	06	XX000110	&	46	26	XX100110
G	07	07	XX000111	'	47	27	XX100111
H	10	08	XX001000	(50	28	XX101000
I	11	09	XX001001)	51	29	XX101001
J	12	0A	XX001010	*	52	2A	XX101010
K	13	0B	XX001011	+	53	2B	XX101011
L	14	0C	XX001100	,	54	2C	XX101100
M	15	0D	XX001101	-	55	2D	XX101101
N	16	0E	XX001110	.	56	2E	XX101110
O	17	0F	XX001111	/	57	2F	XX101111
P	20	10	XX010000	0	60	30	XX110000
Q	21	11	XX010001	1	61	31	XX110001
R	22	12	XX010010	2	62	32	XX110010
S	23	13	XX010011	3	63	33	XX110011
T	24	14	XX010100	4	64	34	XX110100
U	25	15	XX010101	5	65	35	XX110101
V	26	16	XX010110	6	66	36	XX110110
W	27	17	XX010111	7	67	37	XX110111
X	30	18	XX011000	8	70	38	XX111000
Y	31	19	XX011001	9	71	39	XX111001
Z	32	1A	XX011010	:	72	3A	XX111010
[33	1B	XX011011	;	73	3B	XX111011
\	34	1C	XX011100	<	74	3C	XX111100
]	35	1D	XX011101	=	75	3D	XX111101
^	36	1E	XX011110	>	76	3E	XX111110
_	37	1F	XX011111	?	77	3F	XX111111

Table 6-12. Instruction Summary for the 3341/3343 Disk Storage System
(also reference Controller manual 88A00408A)

Device Select Code: X'E'
 Interrupt Vector: X'4E'
 Data Channel Locations: SCR = X'22'; CAR = X'23'

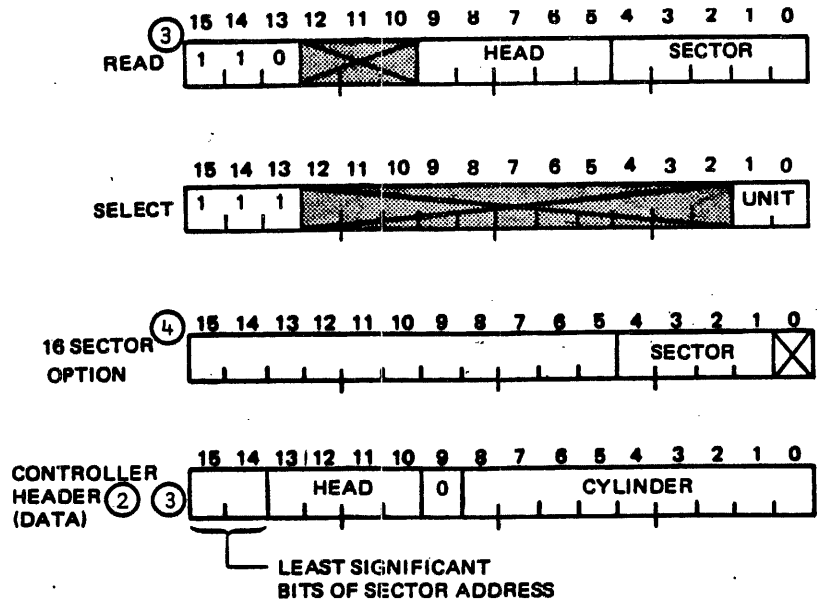
Instruction		Function
CTRL	0,X'E'	ENABLE INTERRUPT from Disk Storage System.
CTRL	1,X'E'	DISABLE INTERRUPT from Disk Storage System.
CTRL	3,X'E'	ABORT (initialize controller)
DTOR	R,X'E'	OUTPUT object word from register (R) or from memory to disk controller.
or		
DTOM	R,X'E'	INPUT status word to register (R) or to memory from disk controller.
DTIR	R,X'E'	
or		
DTIM	R,X'E'	

R = A,X,Y,Z,B,C,D or E



243-4-8

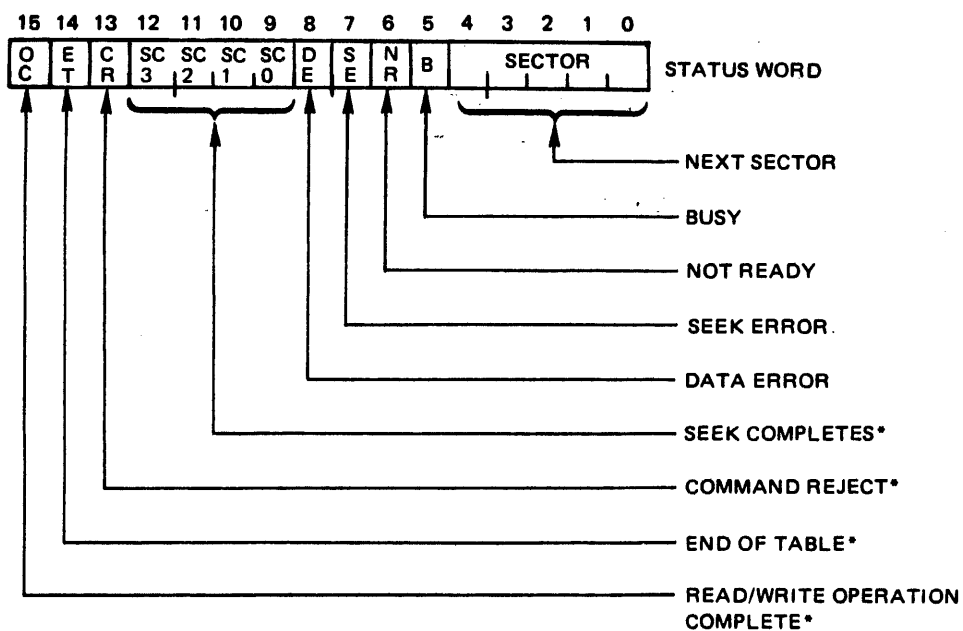
Figure 6-10. Object Word Format to 3341/3343 Disk Controller (Sheet 1 of 2)



400-7-5

- ① General format of data output instruction used with disk storage system. 3-bit command (CMD) field determines the interpretation of modifier bits, however, CMD = 011 is undefined, and is rejected.
- ② Diagram shows header generated by controller when WRITE command is issued. When WRITEL command is issued, the header will originate in first word of data area, and is user defined.
- ③ Header written with a WRITE or WRITEL command is read into first word of data area when READL command is issued. When READ command is issued, header is ignored and first word of data area contains data.
- ④ Bit 0 of Sector field is not used when 3343 sixteen-sector optional configuration is used.

Figure 6-10. Object Word Format to 3341/3343 Disk Controller (Sheet 2 of 2)



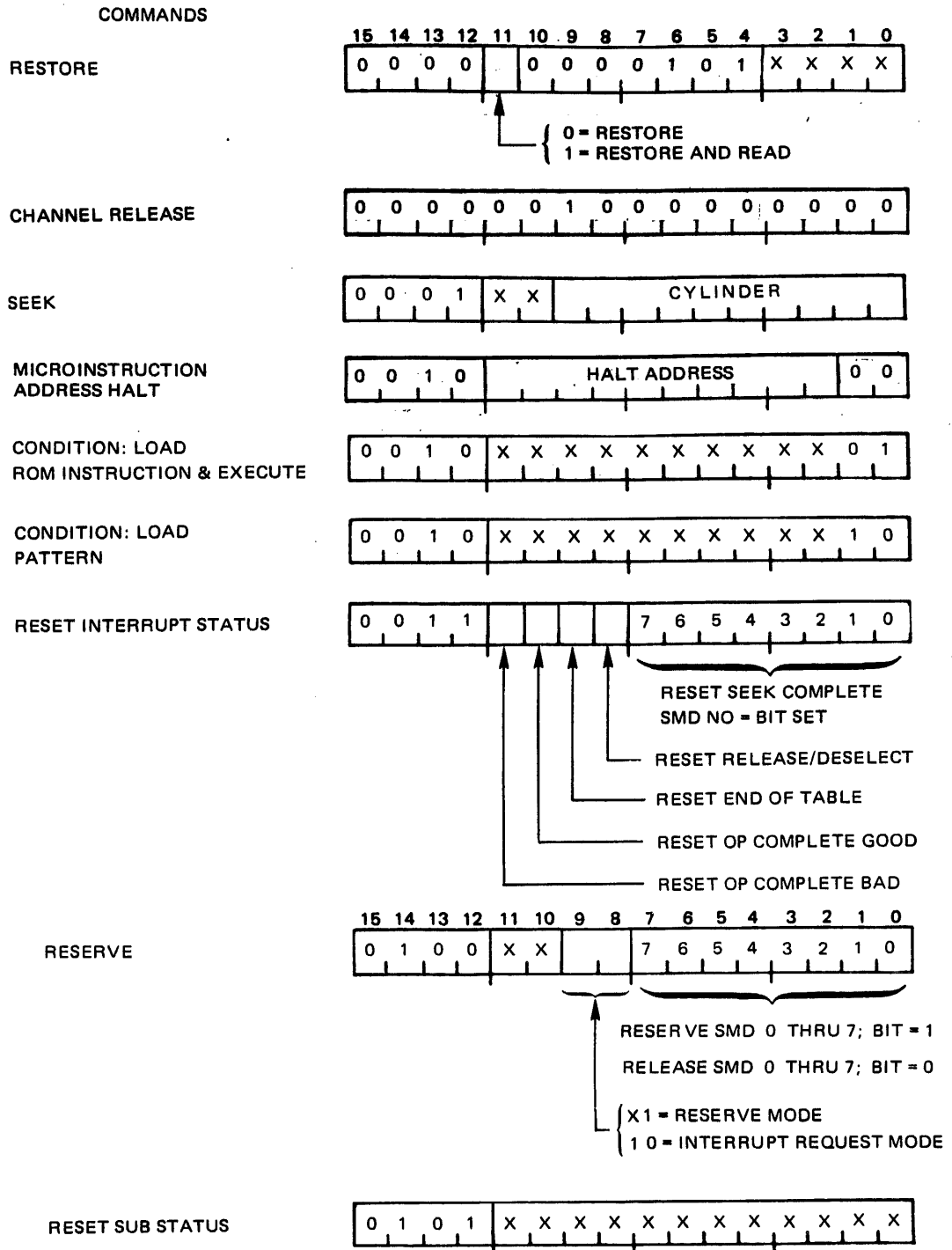
243-4-10

Figure 6-11. Status Word for the 3341/3343 Disk Controller

Table 6-13. Instruction Summary for the 3345 Disk Controller
(also reference Controller manual 88A00522A)

Device Select Code: X'E'
 Interrupt Vector: X'4E'
 Data Channel Locations: SCR = X'22'; CAR = X'23'

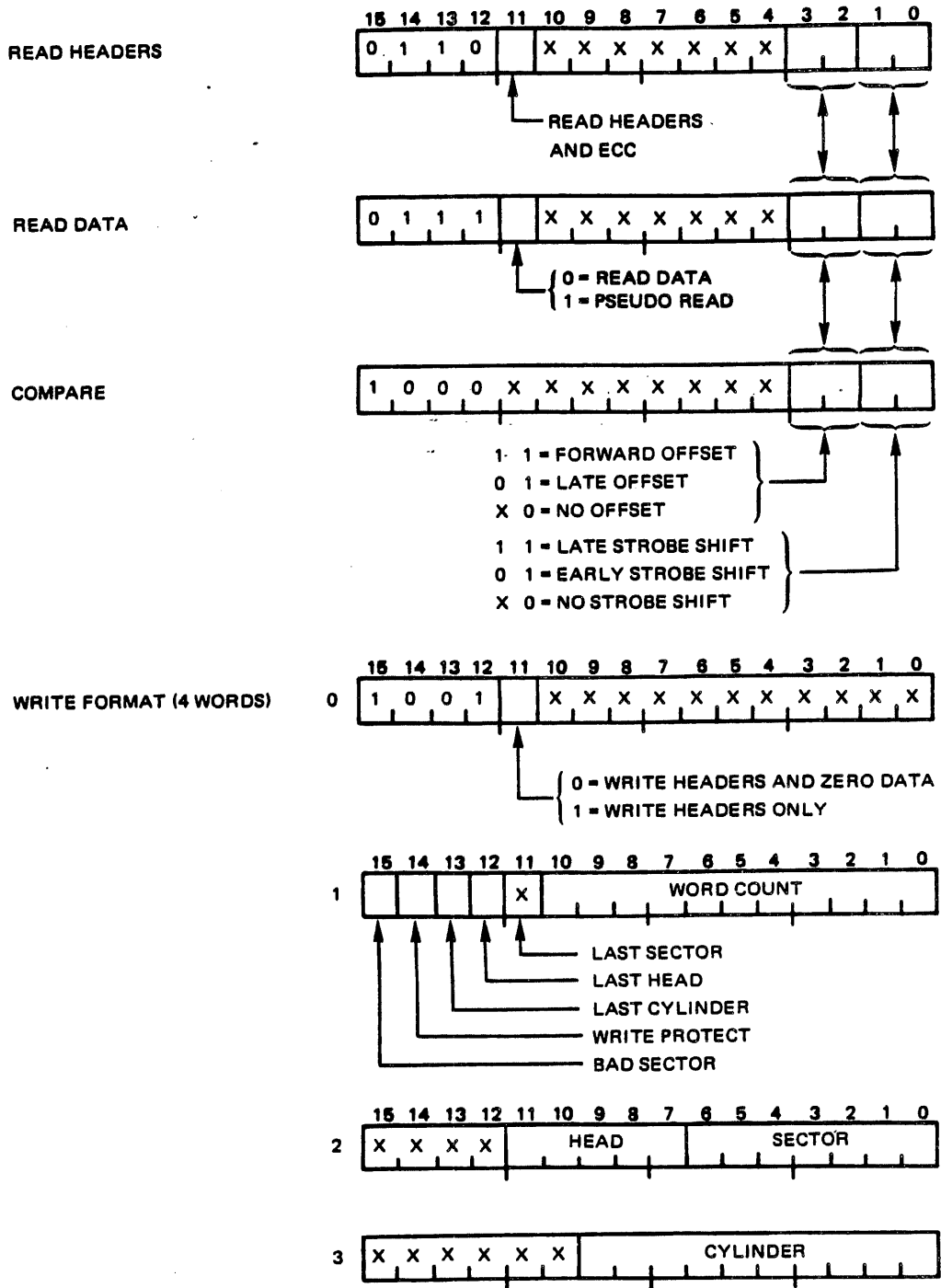
Instruction	Function
TEST 0,X'E'	Skip if partner processor has control
TEST 1,X'E'	Skip if formatter NOT BUSY is true
TEST 2,X'E'	Skip if channel released
CTRL 0,X'E'	Enable interrupt from disk controller
1,X'E'	Disable interrupt from disk controller
2,X'E'	Transfer sub-status from formatter to interface module
3,X'E'	Transfer reserve status from formatter to interface module
4,X'E'	Transfer register F from formatter to interface module
5,X'E'	Abort partner processor
6,X'E'	Execute ROM override register
DTOR or DTOM R,X'E' }	OUTPUT object word from register or from memory to disk controller
DTIR or DTIM R,X'E' }	INPUT status word to register or to memory from disk controller



① SMD: STORAGE MODULE DRIVE

Figure 6-12. Object Words to 3345 Disk Controller (Sheet 1 of 3)

88A00508A-E



② ECC: ERROR CORRECTION CODE

507-8-4

Figure 6-12. Object Words to 3345 Disk Controller (Sheet 2 of 3)

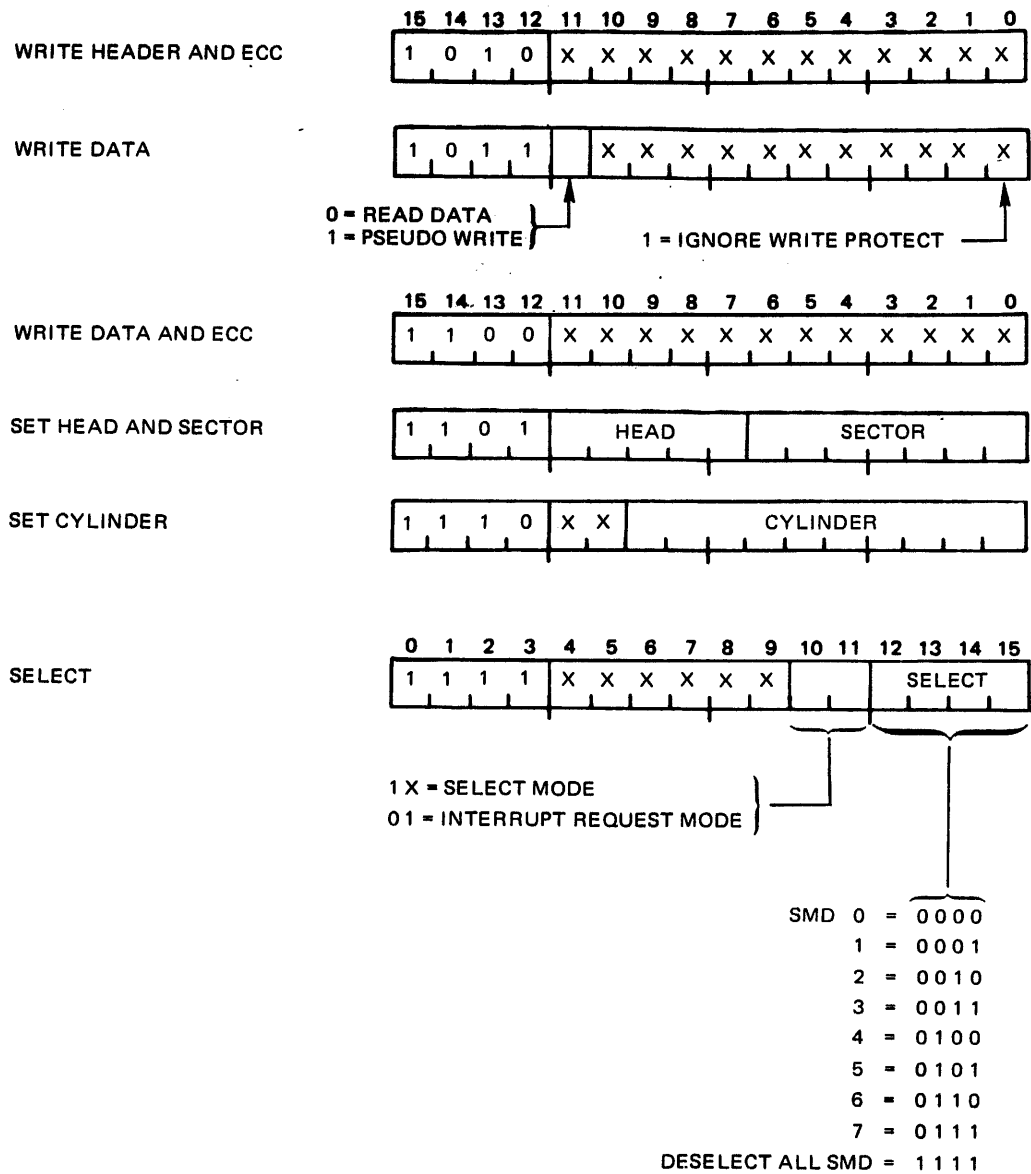
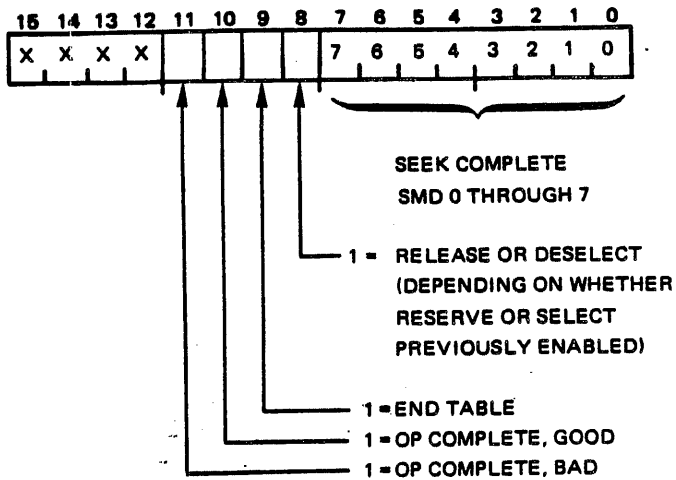
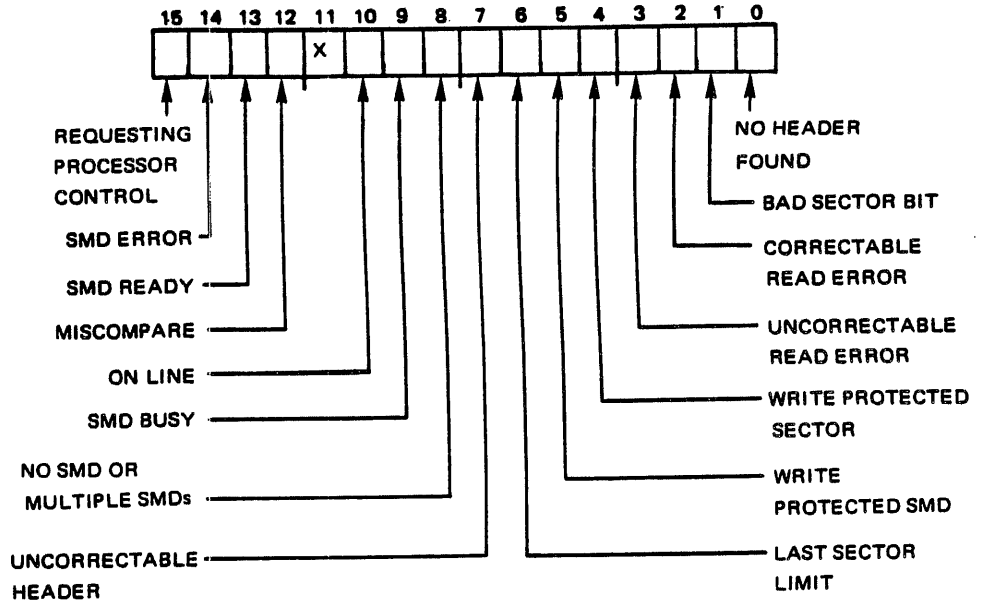


Figure 6-12. Object Words to 3345 Disk Controller (Sheet 3 of 3)

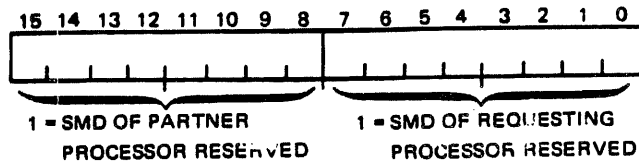
MAJOR STATUS



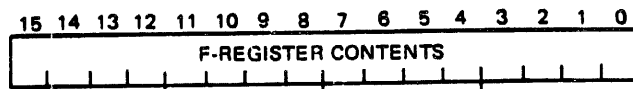
SUB-STATUS
(MUST BE PRECEDED
BY CTRL 2, 'E')



RESERVED STATUS
(MUST BE PRECEDED
BY CTRL 3, 'E')



F-REGISTER
STATUS
(MUST BE PRECEDED
BY CTRL 4, 'E')



522-4-2

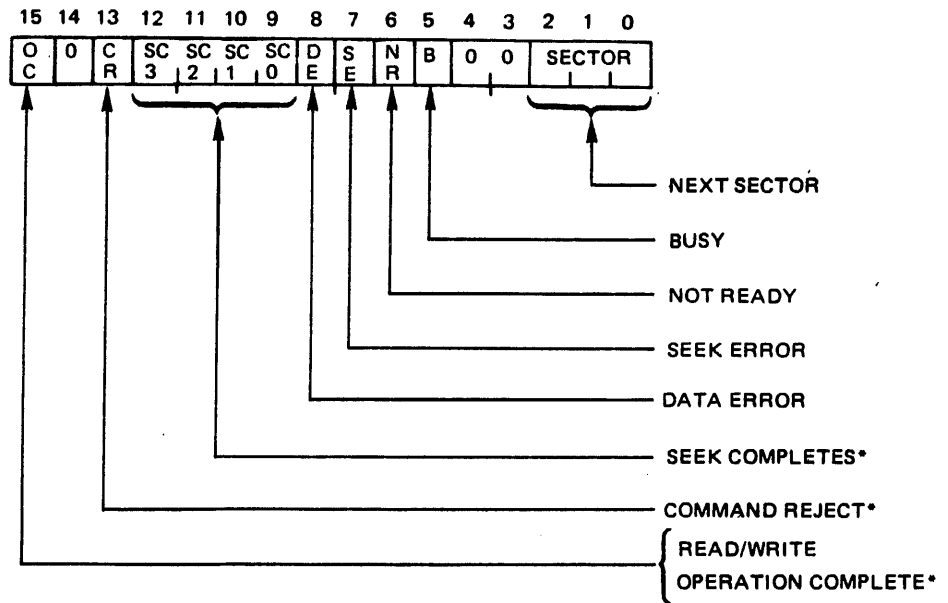
Figure 6-13. Status Words From 3345 Disk Controller

Table 6-14. Instruction Summary for the 3346/3347 Disk Storage System Controller
(also reference Controller manual 88A00403A)

Device Select Code: X'E'
 Interrupt Vector: X'4E'
 Data Channel Locations: SCR = X'22'; CAR = X'23'

Instruction		Function
CTRL	0, X'E'	ENABLE INTERRUPT from disk storage system.
CTRL	1, X'E'	DISABLE INTERRUPT from disk storage system.
CTRL	3, X'E'	ABORT (initialize controller)
DTOR	R, X'E'	OUTPUT object word from register (R) or from memory to disk controller.
or		
DTOM	R, X'E'	
DTIR	R, X'E'	INPUT status word to register (R) or to memory from disk controller
or		
DTIM	R, X'E'	

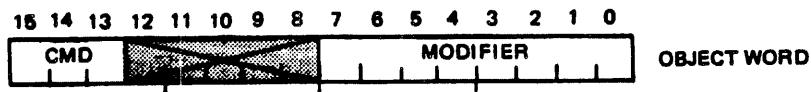
R = A, X, Y, Z, B, C, D or E



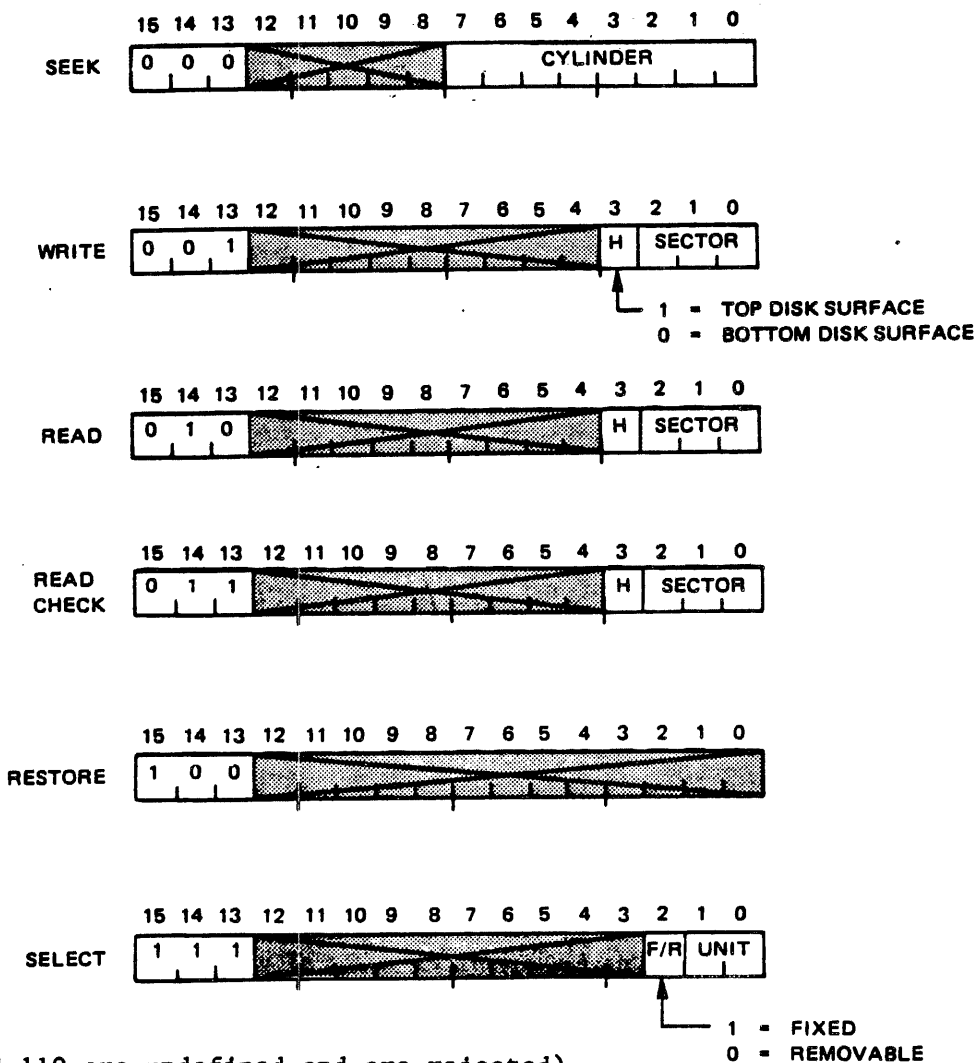
* = INTERRUPT CONDITIONS

Figure 6-14. Status Word from 3346/3347 Disk Storage System Controller

One data output instruction is used with the disk storage system. This instruction can cause one of six different operations to be performed depending on the format of the object word it transmits to the controller. The general format of the object word is shown in Figure 6-15.



The 3-bit CMD field (bits 15-13) determines the interpretation of the modifier:



(CMD = 101 and 110 are undefined and are rejected)

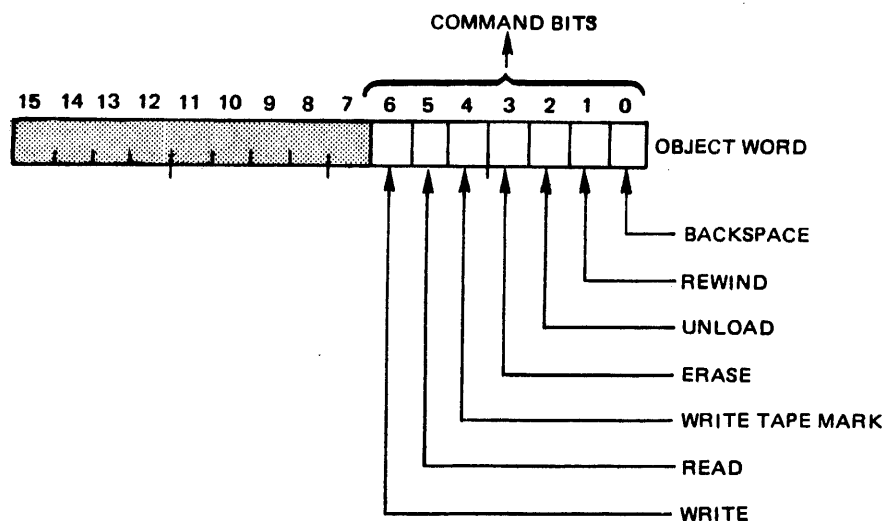
Figure 6-15. Object Word for 3346/3347 Disk Storage System Controller

Table 6-15. Instruction Summary for Models 3331, 3332, 3333 Magnetic Tape Unit Controller (also reference Controller manual 88A00043A)

Device Select Code: X'F'
 Interrupt Vector: X'4F'
 Data Channel Locations: SCR = X'24'; CAR = X'25'

Instruction		Function
CTRL	0,X'F'	ENABLE INTERRUPT from Magnetic Tape Unit
CTRL	1,X'F'	DISABLE INTERRUPT from Magnetic Tape Unit
CTRL	3,X'F'	RESET controller
CTRL	4,X'F'	SELECT DRIVE 0
CTRL	5,X'F'	SELECT DRIVE 1
CTRL	6,X'F'	SELECT DRIVE 2
CTRL	7,X'F'	SELECT DRIVE 3
DTOR	R,X'F'	OUTPUT object word from register (R) or from memory to MTU controller.
or		
DTOM	R,X'F'	INPUT status word to register (R) or to memory from MTU controller.
DTIR	R,X'F'	
or		
DTIM	R,X'F'	

R = A,X,Y,Z,B,C,D or E

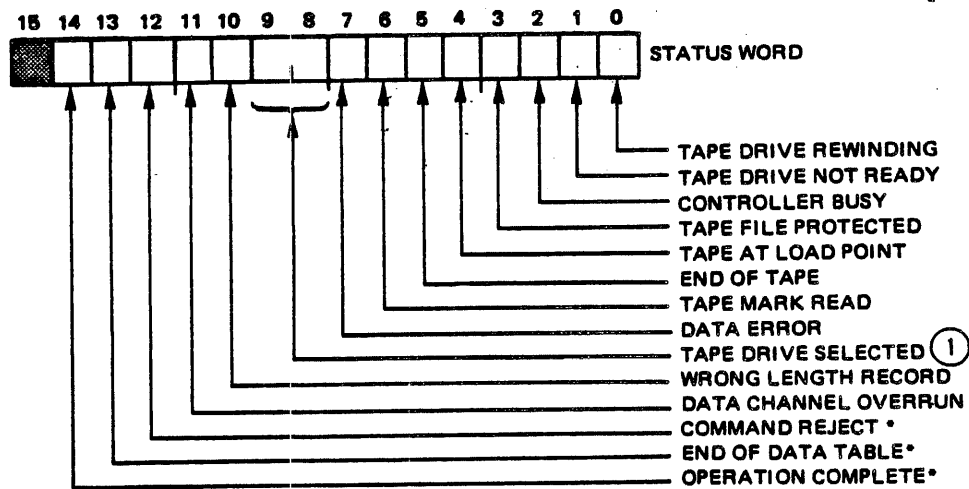


243-4-15

NOTE

Only one command bit may be set true at any one time, except REWIND and UNLOAD may be set simultaneously.

Figure 6-16. Object Word to Models 3331, 3332 and 3333 Magnetic Tape Unit Controller



243-4-15

* - INTERRUPT CONDITIONS

TAPE DRIVE SELECTED (bits 9 and 8) indicate the binary number of the tape drive selected. The coding is:

- ① 00 = Drive 0
- 01 = Drive 1
- 10 = Drive 2
- 11 = Drive 3

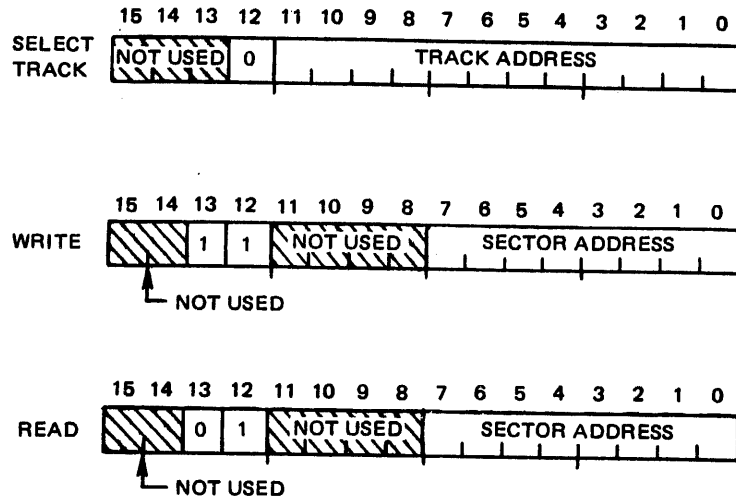
Figure 6-17. Status Word For Model 3331, 3332 and 3333 Magnetic Tape Controller

Table 6-16. Instruction Summary for Model 3342 Head-Per-Track Controller
(also reference Controller manual 88A00407A)

Device Select Code: X'12'
 Interrupt Vector: X'52'
 Data Channel Locations: SCR = X'20', CAR = X'21'

Instruction		Function
CTRL	0, X'12'	<u>Enable interrupt</u> from HPT controller
CTRL	1, X'12'	<u>Disable interrupt</u> from HPT controller
CTRL	2, X'12'	<u>Reset status</u> indicators
CTRL	3, X'12'	<u>Abort</u> (initialize controller)
DTOR or DTOM	R, X'12'	OUTPUT object word from register (R) or memory to HPT controller
	R, X'12'	
DTIR or DTIM	R, X'12'	INPUT status word to register (R) or memory from HPT controller.
	R, X'12'	

R = A, X, Y, Z, B, C, D or E

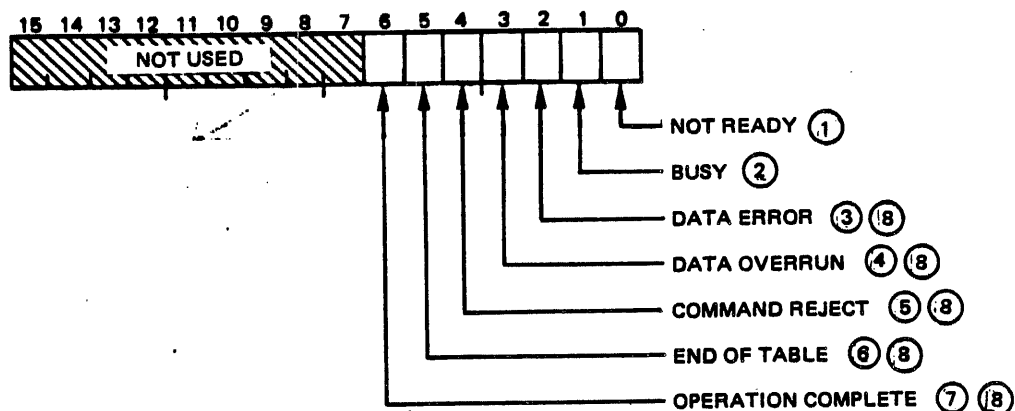


466-7-6

① The SELECT TRACK object word is output first. Then, either the WRITE or READ object word is output. Upon receipt of WRITE or READ object word, the data transfer will commence.

Figure 6-18. Object Word Formats to HPT Controller

88A00508A-E



486-7-7

- ① NOT READY (bit 0) becomes true when the disk unit is off-line.
- ② BUSY (bit 1) becomes true when the controller is performing a read/write operation.
- ③ DATA ERROR (bit 2) becomes true if a cyclic check error is detected during a read operation.
- ④ DATA OVERRUN (bit 3) becomes true if a slow data channel access causes a loss of data.
- ⑤ COMMAND REJECT (bit 4) becomes true if a Data Output instruction is sent to the controller while it is BUSY or NOT READY. The rejected command does not affect any present operations. If the controller's interrupt mask is enabled, COMMAND REJECT will cause an interrupt.
- ⑥ END OF TABLE (bit 5) becomes true during a read or write operation when the final word has been transferred to or from the data table in core, and bit 14 of the word count location (SCR) is zero. If the controller's interrupt mask is enabled, END OF TABLE will cause an interrupt.
- ⑦ OPERATION COMPLETE (bit 6) becomes true when the controller terminates an accepted read/write operation. OPERATION COMPLETE will cause an interrupt if the controller's interrupt mask is enabled.
- ⑧ DATA ERROR, DATA OVERRUN, COMMAND REJECT, END OF TABLE and OPERATION COMPLETE are all "resettable" status indicators. They will be cleared by issuing a RESET STATUS instruction, by issuing an ABORT instruction, by pressing the RESET switch on the CPU console, or when the controller becomes BUSY in response to a new Data Output instruction.

Figure 6-19. Status Word Format for HPT Controller

Table 6-17. Instruction Summary for 16/220 Internal Functions and Console
(also reference Sections 4.15.1 and 4.15.2)

		Device Select Code: X'3E'
		Interrupt Vector: N.A.
		Data Channel Location: N.A.
Instruction		Function
DTOR	R, X'3E'	OUTPUT mask word from register (R) or from memory
DTOM	R, X'3E'	INPUT console switches to register (R) or to memory
RCSR	R	Alternate to RCSR on GA-16/440 (invalid on GA-16/220; use RCSR)
RCSM	R	
RCSW	R	
DSPL	R	OUTPUT data to data bus (used for display on console on SPC-16 and GA-16/440; no display on GA-16/220)

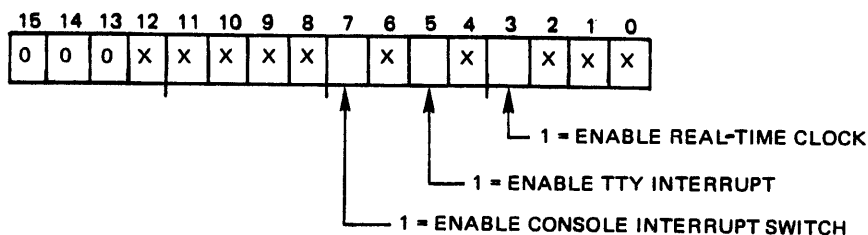
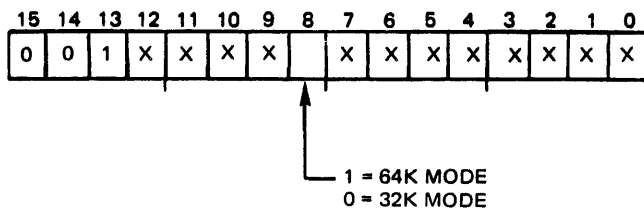


Figure 6-20. Mask Word for Internal Interrupts



507-6-5

Figure 6-21. Mask Word for Setting Memory Mode

Table 6-18. Instruction Summary for Serial I/O Controller (GA-16/220) or External Model 1582 Controller (GA-16/110) (also refer to Section 4.15.3 or Controller manual 88A00416A)

Device Select Code: X'3F'
 Interrupt Vector: X'45'
 Data Channel Locations: N.A.

Instruction		Function
TEST	0,X'3F'	Skip if TTY NOT BUSY is true
CTRL	0,X'3F'	TRANSMIT mode
CTRL	2,X'3F'	RECEIVE ONLY mode (initialized to this mode)
CTRL	4,X'3F'	RECEIVE & ECHO mode
CTRL	6,X'3F'	BREAK mode
DTIR	R,X'3F'	INPUT character byte to register (R)
DTIM	R,X'3F'	INPUT character byte to memory (location contained in R)
DTOR	R,X'3F'	OUTPUT character byte from register (R)
DTOM	R,X'3F'	OUTPUT character byte from memory (location contained in R)

R = A,X,Y,Z,B,C,D or E

NOTE

Interrupt is enabled by setting internal interrupt mask, bit 5, via DTOR or DTOM to device X'3E'. This applies to either built-in or external controller.

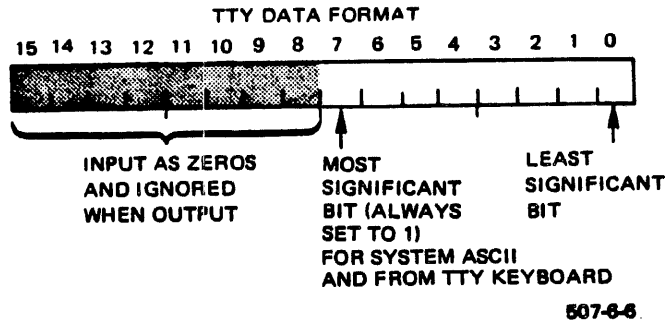


Figure 6-22. TTY Input or Output Data Format

Table 6-19. Detail Description of ASR 33 TTY Functions

Mode	Description	To Remove Service Request	Specific Rule
TRANSMIT Mode	Transmit to printer and/or punch.	<ol style="list-style-type: none"> 1. Output character. 2. Change mode. 	<ol style="list-style-type: none"> 1. New character start when DTOR or DTOM occurs. 2. Service request occurs when not busy.
RCV ONLY Mode	Teletype input to processor buffer only.	<ol style="list-style-type: none"> 1. Input a character. 2. Receive subsequent sync character. 	<ol style="list-style-type: none"> 1. New character starts when not busy and mark-to-space transition occurs. Mark= current flow = logical one. Space no current flow. = logical zero
RCV & ECHO Mode	Teletype input to processor buffer and to teletype printer and/or punch.	<ol style="list-style-type: none"> 3. Change from one receive mode to another receive mode. 	<ol style="list-style-type: none"> 2. To avoid data overrun, character must be input within 4 ms after service request. 3. Service request occurs when not busy and character has been received.
BREAK Mode	Transmit line to teletype is forced to quiescent (space=open=zero) condition. This mode remains active until system is reset or mode changes. Transmit mode must have been established in order to have BREAK mode effective.		

6.8 I/O HARDWARE INTERFACE

Figure 6-23 shows the I/O interfaces between the CPU section and the I/O section. Signals for programmed I/O and programmed I/O with interrupts interface to the 110 module. Signals required for DMA and the DMA-related MHSDC originate in the 220 module. Serial TTY signal lines also originate from the 220 module. (From this it follows that DMA or serial TTY interfaces are not features of the 110.)

The data-channel-bus signals appear in the I/O section and originate in the MHSDC controller for distribution to the peripheral controllers (up to eight controllers) in the I/O section.

6.8.1 CONTROLLER DESIGN CONSIDERATIONS

Programmed I/O may be implemented in controllers that are designed to generate interrupts or in controllers that have no interrupt capability, depending upon the application. Several considerations are important in deciding whether or not to provide a controller with an interrupt capability. A controller designed to use programmed I/O without interrupts is simpler from the hardware standpoint and provides transfers with much lower programming overhead, but the CPU's attention is required from the beginning to the end of the transfer. This is not necessarily a detriment; in applications requiring only one peripheral unit or in an application with little or no need for concurrent I/O operations by a number of peripheral units, the system can often perform most effectively without using interrupts.

The interrupt capability requires the CPU's participation only when the item is ready to be transferred. That is, the controller initiates an "interrupt" signal sequence only when it requires service (e.g., when it has a character ready to input to the CPU), freeing the CPU to perform other processing in the interim. Interrupts are useful, then, when several I/O operations are to be running concurrently; the CPU can service each controller when it requires service. In a multiprocessing application of this sort, programmed I/O without interrupts could require a large amount of "bookkeeping" by the running program; the program would be forced to monitor the state of each operation. Further, any application in which it would be desirable for a peripheral unit to initiate an I/O operation would require the use of interrupts. Consider, for example, a peripheral unit that requires service at random intervals, perhaps only once every few hours. Programmed I/O without interrupts would necessitate periodic polling to detect a service request with the vast amount of polling in vain; interrupt logic, however, could detect a request and initiate service as soon as the need arose, without any polling required.

The decision whether to incorporate an interrupt capability into the design of a particular controller is, then, based on the expected usage of the controller. As a rule, though, controllers that use programmed I/O should be designed with interrupt logic included unless the designer is absolutely certain that an interrupt capability will never be required.

Controllers that use the interrupt capability must have an interrupt request network and a network to determine priority relative to other controllers on the I/O Bus that may also interrupt. On the I/O Bus, the priority is determined by physical placement in the I/O chassis. The closer a module is to the CPU side of the I/O

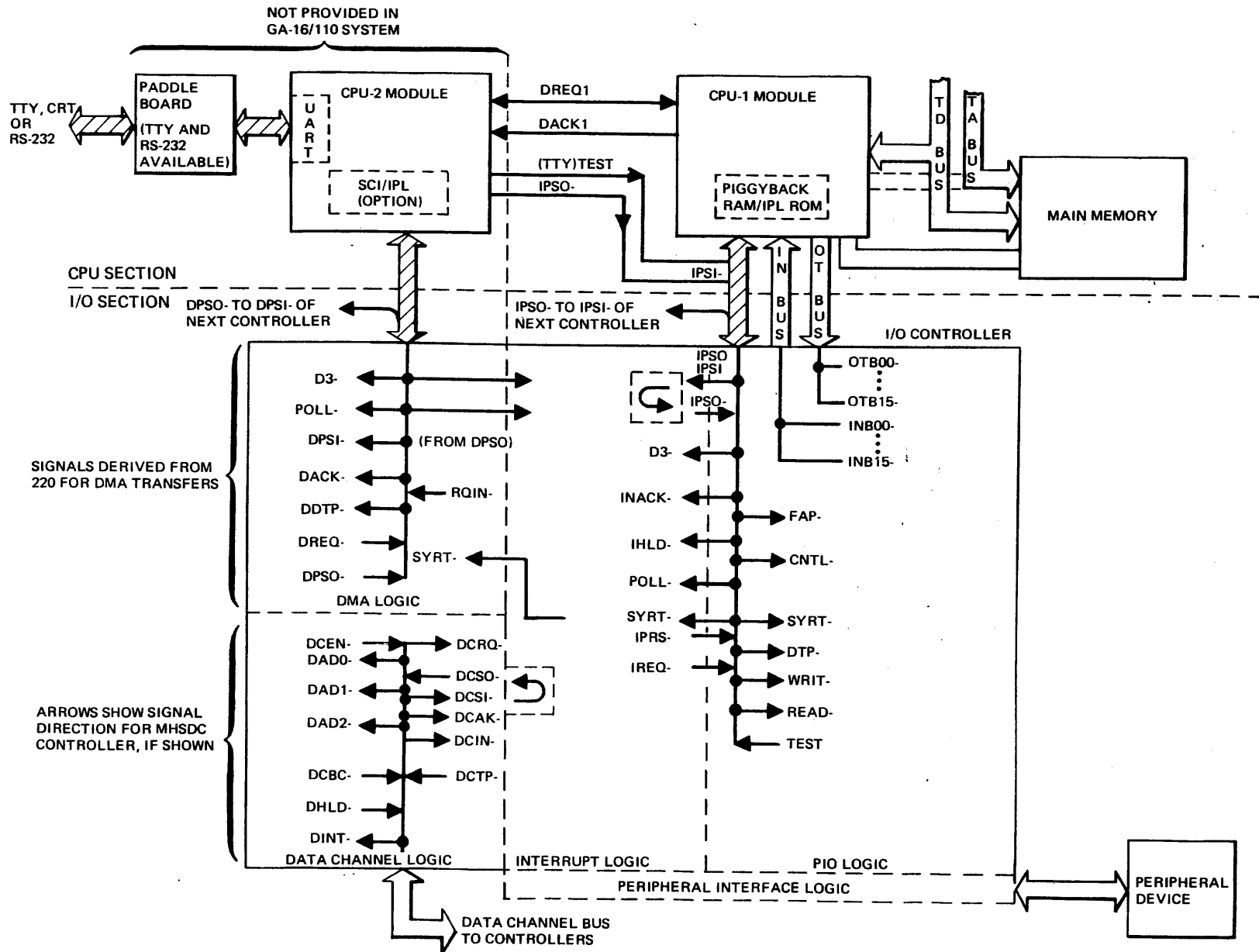


Figure 6-23. I/O Interfaces

chassis, the higher its priority. This is accomplished by a "daisy chain" signal that loops in and out of every module slot. An Interrupt Priority Status Out (IPSO) comes from the CPU and goes to the first slot in the I/O chassis where it connects to that module's Interrupt Priority Status In (IPSI). If that module has no interrupt request pending, it passes the signal (IPSO) onto the next slot and so on down the line.

IPSO is required by any standard controller to enable that controller's Interrupt Requests (IREQ). If a particular controller has a request pending, it kills the IPSO that connects to the following controllers, thereby disabling their requests.

When the CPU acknowledges an interrupt, the highest priority controller that is requesting an interrupt must place an address on the In Bus. This is the address of a memory location dedicated to that controller's priority level. The location contains the address (interrupt vector) of the interrupt routine responsible for servicing that controller's interrupt. The CPU executes a JSR* (Jump-to-Subroutine Indirect) through the address supplied by the interrupting controller. Generally, each controller has a unique vector location associated with it; therefore, by virtue of arriving at the interrupt routine (when the JSR* instruction is simulated by the CPU), the program has identified the interrupting controller. When more than one condition "status bit" associated with the single controller can initiate an interrupt request, the program must 'sense' the status of the controller (usually via Test Instructions) to determine the interrupting condition. A number of interrupt vector locations have been assigned to standard GA controllers; these assignments are summarized in Table 6-1.

There are two situations in which programmed I/O may prove insufficient:

1. When the peripheral unit is a block or a record-oriented device (e.g., disk magnetic tape) and/or,
2. When the peripheral unit is a high-speed device.

In the first case, programmed I/O may prove insufficient because too much CPU time is devoted to transferring a large block of words, one word at a time. The CPU program time could be better spent than to do a lot of mundane bookkeeping.

In the second case, a programmed I/O routine may be too slow to accept data from a high-speed peripheral unit. For instance, a drum memory or fixed-head disk may transfer data at a rate of 10^7 bits/second, which means that the CPU would have to be capable of accepting a new data word every two cycles. Obviously, this is an impossibility using programmed I/O.

The solution to either of the above situations on a GA-16/220 is to use a controller that is designed to perform Direct Memory Access (DMA) transfers. A block transfer of data may be set up by the program via I/O instructions to the DMA peripheral controller. The transfer then proceeds under control of the controller without program intervention. The DMA controller supplies an address to memory which gains command of memory transfer any time CPU is not on the memory bus fetching instructions or transferring data (to or from a register). Memory access may therefore be interleaved between the CPU and DMA with DMA having highest priority. CPU execution of instructions is never halted since it does not participate in DMA transfers. In some cases the CPU execution of instructions which require access to memory may be delayed by one or more microinstruction cycles. In most cases, interleaved DMA and CPU instruction execution proceed simultaneously with no effect on execution time. When a specified number of data items has been transferred, the controller normally requests a program interrupt to signal the program that the block transfer operation is complete.

6.8.2 PHYSICAL DESCRIPTION OF I/O SYSTEM

Figure 6-24 shows a typical I/O system. The diagram illustrates the physical relationship between the CPU, I/O Bus, and Data Channel Bus.

The External I/O Enclosure, which is optional, accommodates up to nineteen modules; one module must be a Cable Interface Translator card; therefore, up to eighteen controllers can be accommodated. If any of the controllers housed in the enclosure require the Data Channel Bus, one module slot must be devoted to a Data Channel Module card (MHSDC). The Data Channel Bus is constrained to its own physical enclosure; i.e., its signals do not go to additional I/O enclosures that might be on the system.

Vacant slots between controller boards must be filled with GA-supplied "Shorter Boards". These boards maintain the continuity of both interrupt priority and DMA priority signals, which daisy-chain down the I/O Bus.

The I/O chassis shown is the standard configuration. It conforms to the I/O Bus; Data Channel Bus signal descriptions and pin numbers are shown later in this section. There are two other special Master Interconnect Boards (MIBs) for installation in external I/O enclosures for Communications Controllers and Process Control Interfacing. They are designed for unique applications and controllers designed for them will not operate in the standard I/O chassis and vice versa.

As illustrated, some controllers require one module, others require two or more; in some cases, two peripheral controllers are implemented on a single module. (The location of controller cards on the diagram is arbitrary.)

DMT controllers do not occupy the I/O section or expansion chassis. The DMT controllers (one for 220 system, two for 110 system) must be designed to plug into a memory slot in the CPU section or to interface via the rear connectors. DMT controllers are built per the customer's requirements by GA.

CAUTION

I/O controllers must be inserted correctly (component side left) in I/O section of jumbo MIB chassis and in external I/O chassis, if used. If controller is installed upside down, severe damage to system will result.

6.8.3 I/O BUS DESCRIPTION

The Input/Output Bus is the communication path between the GA-16/220 computer and any peripheral equipment except DMT controllers. The bus consists of sixteen data output lines, sixteen data input lines, nineteen control lines to the controllers and nine control lines from the controllers.

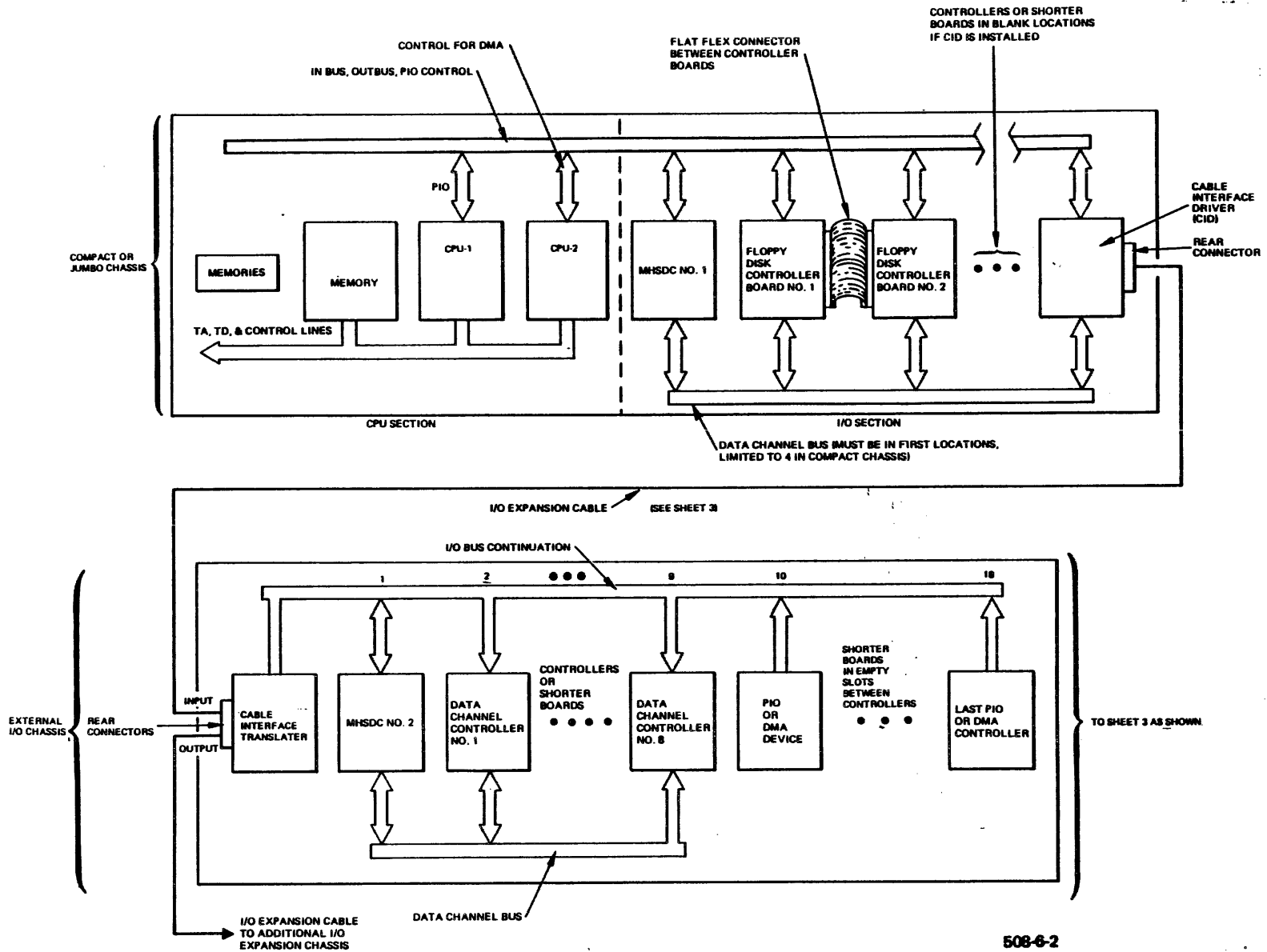
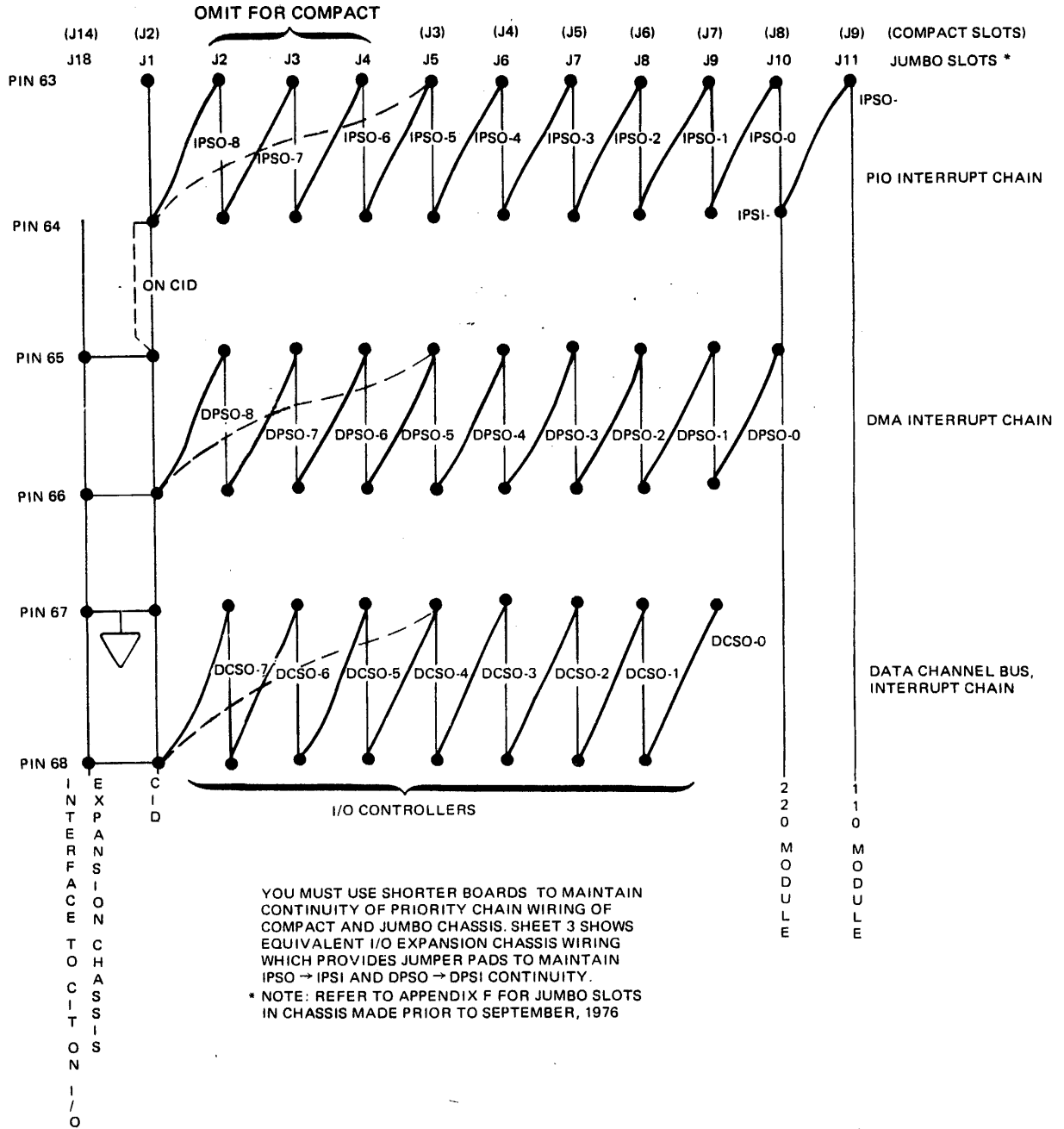


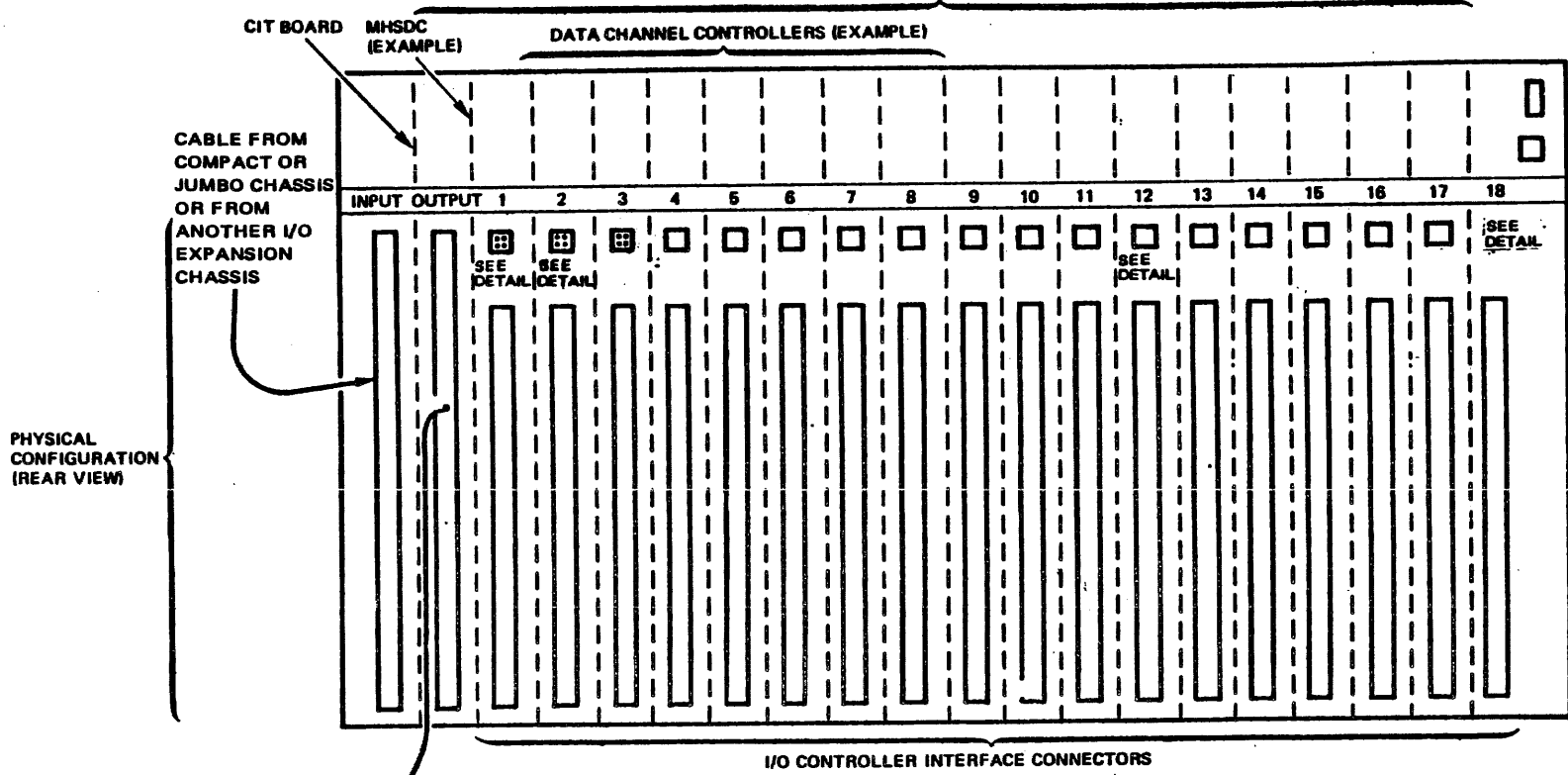
Figure 6-24. Typical I/O System (Sheet 1 of 3)



508-6-3

Figure 6-24. Typical I/O System (Sheet 2 of 3)

I/O CONTROLLERS SLOT CONNECTORS



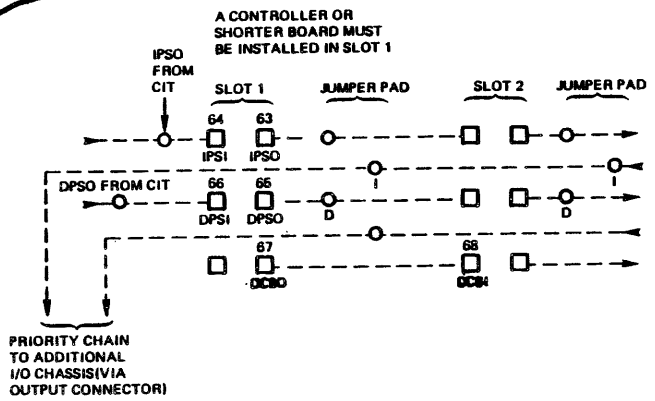
PHYSICAL CONFIGURATION (REAR VIEW)

88A00508A-E

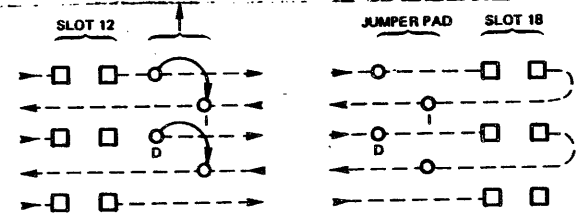
Figure 6-24. Typical I/O System (Sheet 3 of 3)

6-50

CABLE TO SECOND EXTERNAL I/O CAGE



IF 12 IS THE LAST CONTROLLER INSTALLED AND THE CHASSIS IS NOT FILLED WITH SHORTER BOARDS
SOLDER JUMPERS TO I AND D TO MAINTAIN CONTINUITY OF PRIORITY CHAIN



508-6-4

Reference Table 6-20 for I/O Bus signal descriptions.

All bus signals are binary. A minus sign following a signal mnemonic indicates that the signal is between 0 and 0.4 volt when 'true' (i.e., 'energized'). A plus sign following a signal mnemonic indicates that the signal is true when between 2 and 5.0 volts. An "I" following a signal mnemonic indicates that the signal is available on the cable to an external I/O chassis.

Table 6-20. I/O Bus Descriptions (Sheet 1 of 6)

Signal	I/O Pin No.	Direction	Definition
INB00-I to INB15-I	5 to 20	To CPU	<p>IN-Bus. These sixteen data input lines are used to transfer data from any peripheral unit (other than DMT peripherals) to the CPU.</p> <p>For programmed input, the CPU generates READ- to place data from the selected controllers onto the IN-Bus.</p> <p>For servicing interrupt requests, the CPU generates IACK- to place the indirect address (i.e., the memory address of the interrupt vector respective to the interrupting controllers) onto the IN-Bus (Bits 0-15).</p> <p>For DMA transfers, the CPU generates DACK- to place the address of the memory location to/from which data is to be transferred onto the IN-Bus. For DMA data input, the controller places its data onto the IN-Bus.</p>
OTB00-I to OTB15-I	21 to 36	To controller	<p>OUT-Bus. These sixteen data output lines are used to transfer data and control information from the CPU to any peripheral unit.</p> <p>For programmed input, output, control and test operations, function and device select code data are placed onto the OUT-Bus by the CPU, telling the controller whose select code is on bits 0-5 to accept the function data (bits 8-10) and to respond to subsequent signals that may occur during the execution of the operator. For programmed output, the data are placed onto these lines and WRIT- and DTP- are generated, telling the selected controller to accept the data item.</p> <p>For DMA outputs, the data is placed onto these lines and DDTP- is generated telling the controller with priority to accept the data.</p>

Table 6-20. I/O Bus Descriptions (Sheet 2 of 6)

Signal	I/O Pin No.	Direction	Description
FAP-I	37	To controller	Function-and-Address Pulse. This pulse is generated by the CPU during a programmed I/O instruction and serves to signal controllers that the Out-Bus contains function and device select code data. A controller is selected if its device select code is on the OUT-Bus when FAP- occurs. The controller must 'remember' it is selected until the next I/O instruction is executed; it must also 'remember' any function information required during the I/O instruction.
READ-I	51	To controller	Read Signal. This signal is used to time the selected controller when placing data onto the In-Bus during the execution of an input instruction. FAP- occurs before READ-; thus, a particular controller has already been selected when READ- occurs.
WRIT-I	49	To controller	Write Signal. This signal is used in conjunction with DTP- to allow the selected controller to accept data from the OUT-Bus during the execution of an output instruction. FAP- occurs before WRIT-; thus, a particular controller has already been selected when WRIT- occurs.
DTP-I	38	To controller	Data Transfer Pulse. This signal is used in conjunction with WRIT- to time the selected controller's acceptance of data from the OUT-Bus. This pulse occurs on all I/O instructions and can be used for various timing operations, such as termination of certain input signals.
CNTL-I	48	To controller	Control Signal. This signal is used by the selected controller to perform a specified control function. This signal occurs only during the execution of a control instruction. CNTL- is energized before FAP-, and FAP- is used to "time" the control pulse.
TEST-I	53	To CPU	Common Test Line. This line is used by the selected controller to return the result of a previously specified test function. When FAP- occurs, the selected controller accepts and

Table 6-20. I/O Bus Descriptions (Sheet 3 of 6)

Signal	I/O Pin No.	Direction	Description
TEST-I (cont'd)			stores bits 8-10 from the OUT-Bus. These function bits specify one of eight test conditions that is gated onto TEST-. Note that the selected device performs a test for all I/O instructions; however, the CPU only samples the TEST- line for a test instruction.
D3-I	41	To controller	Clock pulse running with the CPU. It always occurs with a fixed skew with respect to POLL-.
POLL-I	40	To controllers	Periodic pulse with a fixed skew with respect to D3-. POLL- is used to synchronize interrupt and DMA requests to the CPU. Interrupt and DMA requests should be energized at the leading edge of POLL-.
IREQ-I	54	To CPU	Interrupt Request. This common line to the CPU is used by any controller to request an interrupt. This signal should be energized on the leading edge of POLL-. It remains energized until the controller is serviced.
IHLD-I	47	To controller	Interrupt Hold. This common signal is used by controller interrupt networks to freeze any further interrupt requests.
IPSO-I, IPSI-I	63, 64	To controller	Interrupt Priority Status Out/In. This serially transmitted signal indicates that no controller of higher priority is requesting an interrupt. If the controller receiving IPSO- does not wish to request an interrupt, it must generate IPSO- for transmission to the next controller input, IPSI-. (The signal IPSI- does not exist on the CPU I/O Bus, but only on a controller, i.e., controller receives its series priority on its IPSI-line and transmits it on the IPSO- line.) If a controller is requesting an interrupt, it uses the input signal to generate IPRS-I and does not transmit the signal to the next controller in the series.
IPRS-I	59	To CPU	Interrupt Priority Return Status. This common signal to the CPU is enabled by IHLD- and IPSO-

Table 6-20. I/O Bus Descriptions (Sheet 4 of 6)

Signal	I/O Pin No.	Direction	Description
IPRS-I (cont'd)			to indicate to the CPU that the interrupt priority has been established and that the CPU can proceed with the interrupt. Only the highest priority controller requesting an interrupt will energize IPRS-.
IACK-I	55	To controller	Interrupt Acknowledge. This timing signal is used by the highest priority controller requesting an interrupt to place the memory address of its interrupt vector onto the IN-Bus (bits 0-15). Although this signal is common to all controllers, it is used only by the controller that has accepted IPSO- (i.e., the highest priority controller requesting an interrupt).
DREQ-I	52	To CPU	DMA Request. This common line is used by a controller that has DMA priority to request a DMA transfer. This signal should be energized on the leading edge of POLL-. This signal is generated on the MHSDC from one of eight data channel requests (DCRQ) input to the MHSDC from a data channel controller.
DPSO-I, DPSI-I	65, 66	To controller	<p>DMA Priority Status Out/In. This serially transmitted signal indicates that no higher priority controller is requesting a DMA transfer. The series priority scheme for a DMA is similar to the interrupt priority scheme; i.e., DPSO- and IPSO- perform a similar function. DPSI- exists on each DMA controller but not on the CPU. Each controller receives on DPSI- and transmits on DPSO-. DPSO- is wire energized at the CPU.</p> <p>In GA's scheme of data channel implementation, 8 high-speed data channels are implemented on one Data Channel Module. If there is more than one Data Channel Module, the priority among these distinct modules is determined by DPSO- (i.e., DPSO and DPSI exist only on and between the data channel modules themselves). A different (physically) series (DCSO- and DCSI-) is implemented between controllers associated with a given Data Channel Module (see Table 6-23). This technique allows a much higher</p>

Table 6-20. I/O Bus Descriptions (Sheet 5 of 6)

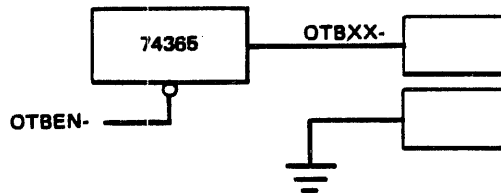
Signal	I/O Pin No.	Direction	Description
DPSO-I, DPSI-I (cont'd)			number of data channel units to operate on a priority system due to the elimination of continuous priority network (and, hence, the continuous addition of priority circuit logic delays) among all units.
DACK-I	50	To controller	DMA Acknowledge. This signal is used by the controller with DMA priority to place the address of the memory location to be accessed onto the IN-Bus. Typically, this signal goes from the CPU to the MHSDC. It in turn sends DCAK to the appropriate controller.
DDTP-I	39	To controller	DMA Data Transfer Pulse. This signal is used by the cycle-stealing controller to accept data on the OUT-Bus or to indicate completion of its cycle in other modes.
RQIN-I	58	To CPU	Request Data-Input Mode. This common line to the CPU indicates the direction of a DMA transfer; RQIN- energized indicates data input to the CPU. If it is not energized, it indicates data output. RQIN- is only energized by the controller or MHSDC that has priority.
DCHN-	61	Between Data Channel Modules or DMA Controllers	DMA Chain. This common signal between Data Channel Modules is energized by a Data Channel Module to indicate to other modules that it is in either an "initiate" or "chaining" mode and that other modules cannot initiate a cycle steal (regardless of their relative priority). In an "initiate" or "chaining" mode that Data Channel Module must access two locations containing Channel Address (CAR) and Block Size/Chaining (SCR) information. The controller is expecting this information at a specific time; any other sequence of events would yield erroneous information to the controller.
SYRT-I	46	To controller	System Reset. This signal is used to initiate the I/O system. SYRT- occurs during power on/off, when the RESET switch (16) in Table 3-3 is pressed, or when the IPL switch is pressed.

Table 6-20. I/O Bus Descriptions (Sheet 6 of 6)

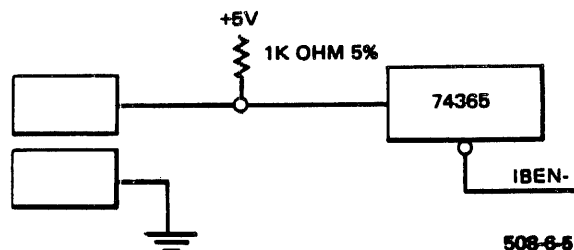
Signal	I/O Pin No.	Direction	Description
SFEC-I	43	To controller	System Safe. This signal is energized when the CPU is in the "safe" mode; i.e., when the Operations Monitor Alarm has not been activated and timed out.
SYNC-I	42	To controller	Synchronization Pulse. This pulse is generated by the CPU during the 'SYNC' instruction. Sync allows an oscilloscope to be synchronized to software events.
CLDS-I	62	To CPU	Cold Start. This signal is used to change the Auto-Restart Interrupt Vector to a ROM for cold start operation. When CLDS-I is energized, a CPU reset or power-on sequence will force program execution to the Console ROM or IPL ROM depending on the position of the console mode switch.
DMFO-I	56	To MPP	Direct Memory Access Function. This signal is used to force the MPP option to use the DMA MAP MEMORY PROTECT DATA on DMA transfers. The GA Floating-Point Processor makes use of this line.

6.8.4 I/O BUS DRIVERS/RECEIVERS

All output drivers from the processor are implemented using 74365 tri-state TTL drivers capable of sinking 40 ma to ground with a saturation voltage of 0.4 volt.



All IN-Bus lines are interfaced as follows:



508-6-5

When a cable termination is required, each signal must have a twisted-pair line of approximately 100 ohm characteristic impedance with the returns grounded at both ends. The drivers should be terminated with the 220/330 divider at the end of the cable (this places a termination at both ends of the receiving line).

It is General Automation's policy to present one 2-ma load maximum to any one I/P Bus signal with any peripheral controller.

6.8.5 I/O CONNECTOR PIN ASSIGNMENTS

Table 6-21 shows connection pin assignments for the External I/O Bus cable. Pins 65 through 136 provide the communication between the CIT-16 board and the I/O cable. An "I" following a signal mnemonic indicates the signal is on the External I/O Bus cable. Further description of external I/O chassis and connectors is contained in Section 7.

Table 6-22 summarizes signal pin assignments for the I/O controllers. Pins 67 through 80 are active only if the enclosure contains a Data Channel Module card. Pins 81 through 136 provide communication between a peripheral unit and its controller.

Table 6-21. I/O Cable and CIT-16 Connector Pin Assignments

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	+5V	36	OTB15-	71	INB02-I	106	DTP-I
2	-15	37	FAP--	72	INB03-I	107	GND
3	GND	38	DTP--	73	INB04-I	108	DDTP-I
4	GND	39	DDTP-	74	INB05-I	109	POLL-I
5	INB00-	40	POLL-	75	GND	110	D3-I
6	INB01-	41	D3-	76	INB06-I	111	SYNC-I
7	INB02-	42	SYNC-	77	INB07-I	112	SFEC-I
8	INB03-	43	SFEC-	78	INB08-I	*113	
9	INB04-	*44		79	INB09-I	*114	
10	INB05-	*45		80	INB10-I	115	GND
11	INB06-	46	SYRT-	81	INB11-I	116	SYRT-I
12	INB07-	47	IHLD-	82	INB12-I	117	IHLD-I
13	INB08-	48	CNTL-	83	GND	118	CNTL-I
14	INB09-	49	WRIT-	84	INB13-I	119	WRIT-I
15	INB10-	50	DACK-	85	INB14-I	120	DACK-I
16	INB11-	51	READ-	86	INB15-I	121	READ-I
17	INB12-	52	DREQ-	87	OTB00-I	122	DREQ-I
18	INB13-	53	TEST-	88	OTB01-I	123	GND
19	INB14-	54	IREQ-	89	OTB02-I	124	TEST-I
20	INB15-	55	LACK-	90	OTB03-I	125	IREQ-I
21	OTB00-	56	DMFO-	91	GND	126	LACK-I
22	OTB01-	57	DMF1-	92	OTB04-I	127	DMFO-I
23	OTB02-	58	RQIN-	93	OTB05-I	128	DMF1-I
24	OTB03-	59	IPRS-	94	OTB06-I	129	RQIN-I
25	OTB04-	*60		95	OTB07-I	130	IPRS-I
26	OTB06-	61	DCHN-	96	OTB08-I	131	GND
27	OTB06-	62	CLDS-	97	OTB09-I	*132	
28	OTB07-	63	DPSO-	98	OTB10-I	133	DCHN-I
29	OTB08-	64	IPSO-	99	GND	134	CLDS-I
30	OTB09-	65	IPSO-I	100	OTB11-I	*135	
31	OTB10-	66	DPSO-I	101	OTB12-I	*136	
32	OTB11-	67	GND	102	OTB13-I	137	GND
33	OTB12-	*68		103	OTB14-I	138	GND
34	OTB13-	69	INB00-I	104	OTB15-I	139	+15V
35	OTB14-	70	INB01-I	105	FAP-I	140	+5V

*Not applicable to the GA-16/220

Table 6-22. I/O Signal Pin Assignments

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	+5	36	OTB15-	71	DHLD-	106	*
2	-15	37	FPA-	72	DADO-	107	*
3	GND	38	DTP-	73	DAD1-	108	*
4	GND	39	DDTP-	74	DAD2-	109	*
5	INB00-	40	POLL-	75	DINT-	110	*
6	INB01-	41	D3-	76	DCAK-	111	*
7	INB02-	42	SYNC	77	DCBC-	112	*
8	INB03-	43	SFEC-	78	DCIN-	113	*
9	INB04-	44	Unused	79	DCTP-	114	*
10	INB05-	45	Unused	80	DCEN-	115	*
11	INB06-	46	SYRT-	81	*	116	*
12	INB07-	47	IHLD-	82	*	117	*
13	INB08-	48	CNTL-	83	*	118	*
14	INB09-	49	WRIT-	84	*	119	*
15	INB10-	50	DACK-	85	*	120	*
16	INB11-	51	READ-	86	*	121	*
17	INB12-	52	DREQ-	87	*	122	*
18	INB13-	53	TEST-	88	*	123	*
19	INB14-	54	IREQ-	89	*	124	*
20	INB15-	55	IACK-	90	*	125	*
21	OTB00-	56	DMFO-	91	*	126	*
22	OTB01-	57	Unused	92	*	127	*
23	OTB02-	58	ROIN-	93	*	128	*
24	OTB03-	59	IPRS-	94	*	129	*
25	OTB04-	60	Unused	95	*	130	*
26	OTB05-	61	DCHN-	96	*	131	*
27	OTB06-	62	CLDS-	97	*	132	*
28	OTB07-	63	IPSO-	98	*	133	*
29	OTB08-	64	IPSI-	99	*	134	*
30	OTB09-	65	DPSO-	100	*	135	*
31	OTB10-	66	DPSI-	101	*	136	*
32	OTB11-	67	DCSO-	102	*	137	GND
33	OTB12-	68	DCSI-	103	*	138	GND
34	OTB13-	69	Unused	104	*	139	+15
35	PTB14-	70	DCRQ-	105	*	140	+5

*Controller card/peripheral cable interface

6.9 I/O BUS TIMING AND INTERFACING

This section describes the operations that take place on the I/O Bus. Relationships among the various bus signals and the signals internal to controllers interfaced to the I/O Bus are shown in a series of timing diagrams. The basic interface network required for the implementation of the various I/O functions are shown in accompanying logic diagrams.

The execution of any instruction in the Programmed Input/Output group requires two phases. With respect to a controller, the first phase is termed the "device select phase" and the second phase is termed the "data phase".

During the device select phase of all I/O instructions, the device select code (bits 0-5 of the instruction) and the function code (bits 8-10 of the instruction) are placed onto the OUT-Bus (OTB00- to OTB05- and OTB08- to OTB10-, respectively). The processor generates the signal FAP- to indicate to all controllers that this information is on the OUT-Bus. The controller whose select code is on the OUT-Bus when FAP- occurs is "selected"; the selected controller must store the function code as well as the fact that it has been selected.

I/O instructions provide four types of operations:

- TEST
- CONTROL
- DATA OUTPUT
- DATA INPUT

Logic and timing will be shown for the above operation. The logic is typical to that mechanized on standard GA controllers.

Figure 6-25 is a flow chart of the signal sequence that occurs for I/O instructions.

6.9.1 TEST AND CONTROL LOGIC TIMING

Figure 6-26 includes timing diagrams for the execution of a TEST instruction and a CTRL (Control) instruction. Figure 6-27 illustrates a typical scheme for implementing the device selection network and the test function and control function on any peripheral controller.

An eight input NAND gate performs the device select decoding; this gate continuously monitors that state of OTB00 to OTB05. The 'jumper pad' wiring determines the select code that will activate the gate (i.e., bring IOADD+ true). The diagram illustrates the select code X'2A'.

When FAP-I occurs, IOADD+ will be true only on the controller that is to be selected. The negative-going edge of FAP-, together with IOADD+, set the device select flip-flop. In addition, FAP- is used by all controllers, whether selected or not, to pulse the function code flip-flops, which store the state of OTB08+ to OTB10+. (The state of function code flip-flops is irrelevant on a controller which has not been selected.)

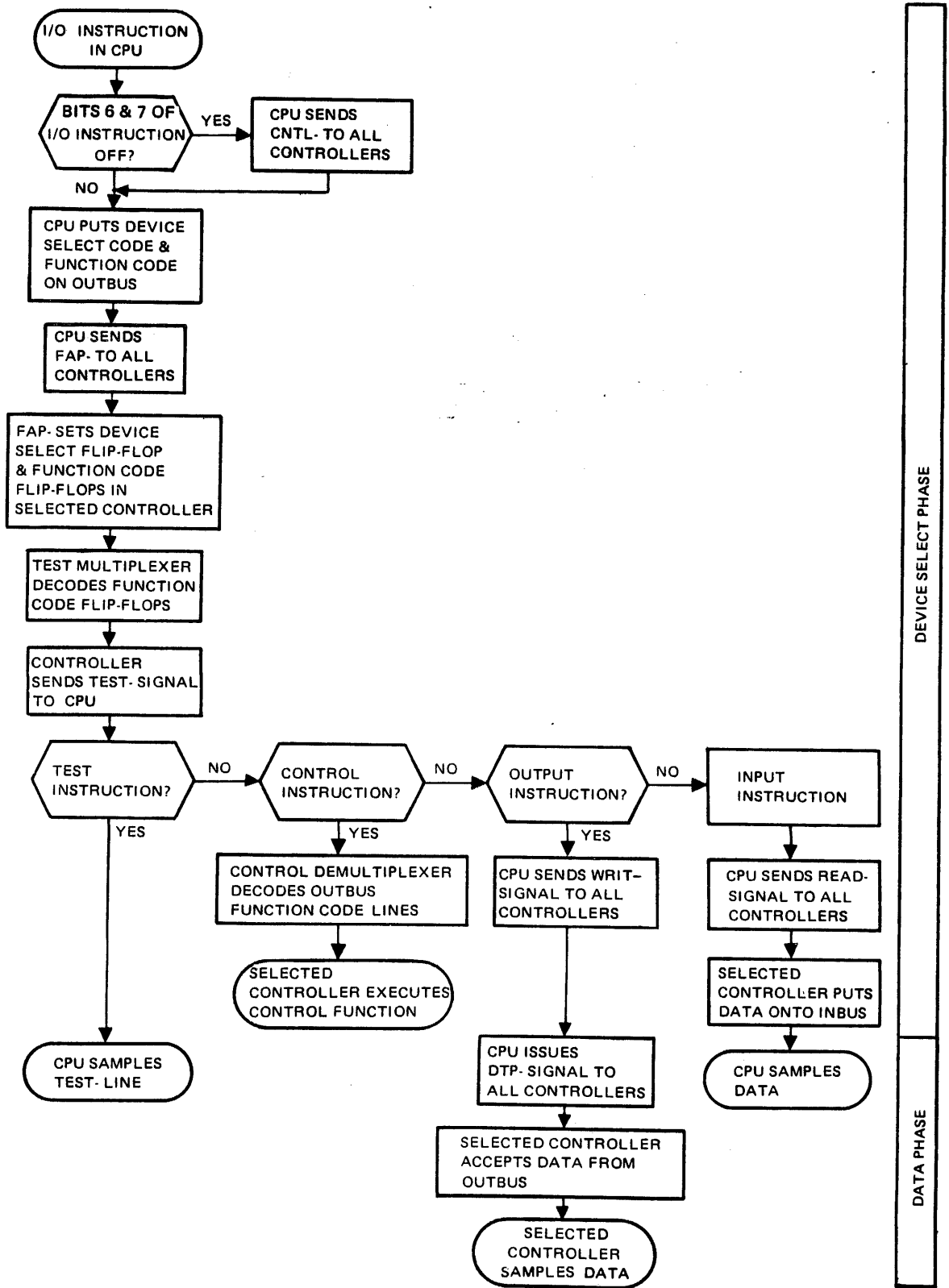


Figure 6-25. Flowchart of I/O Instructions

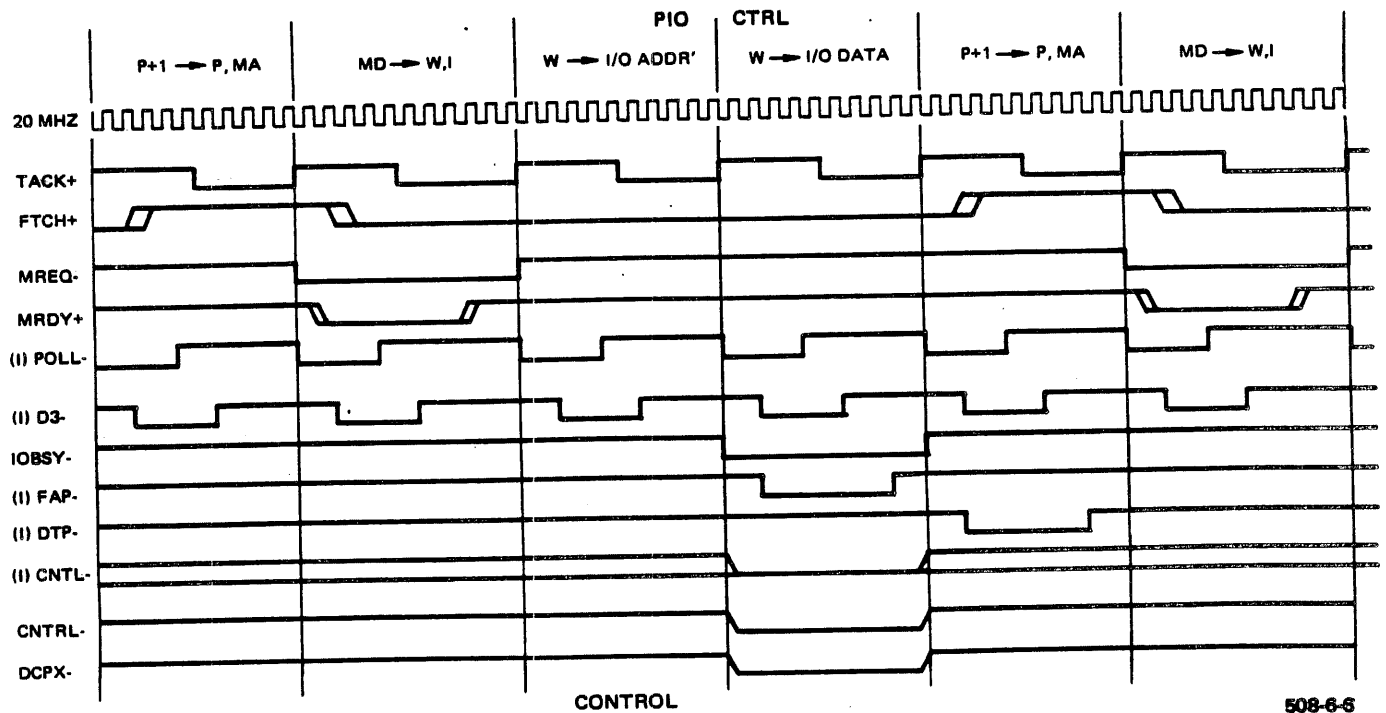
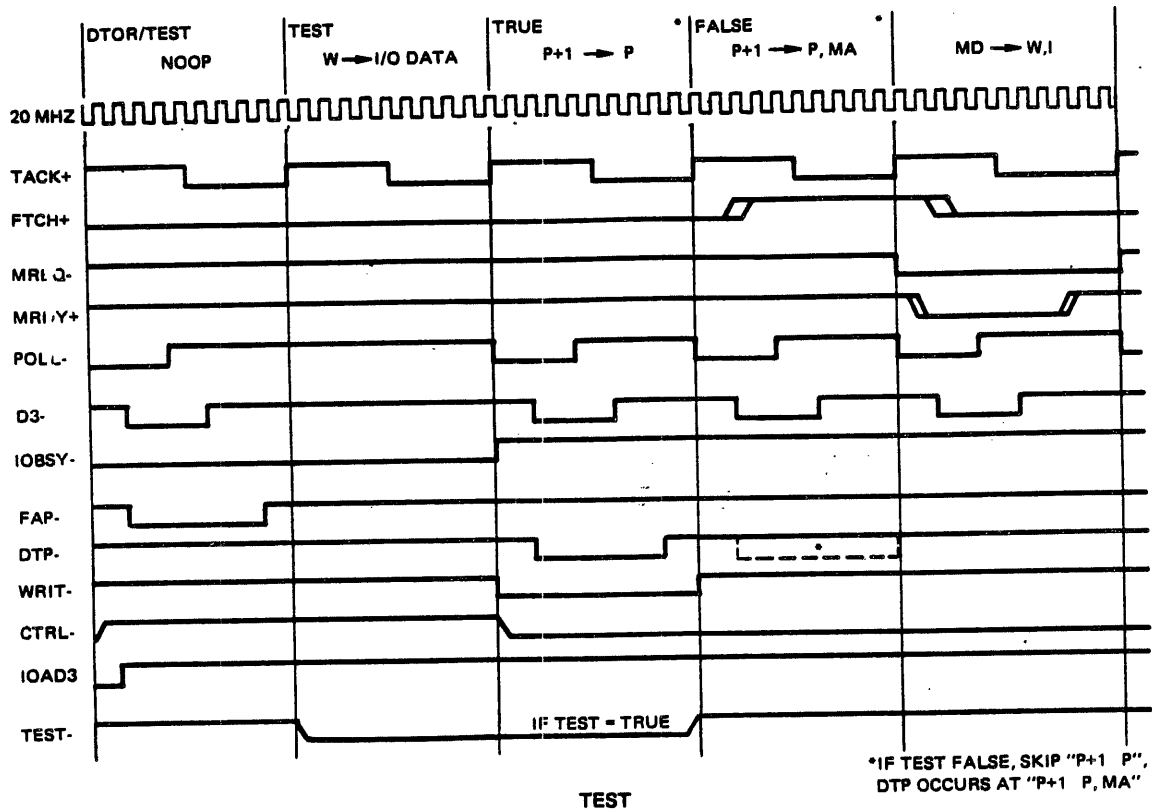


Figure 6-26. Timing for I/O TEST and CONTROL Instructions

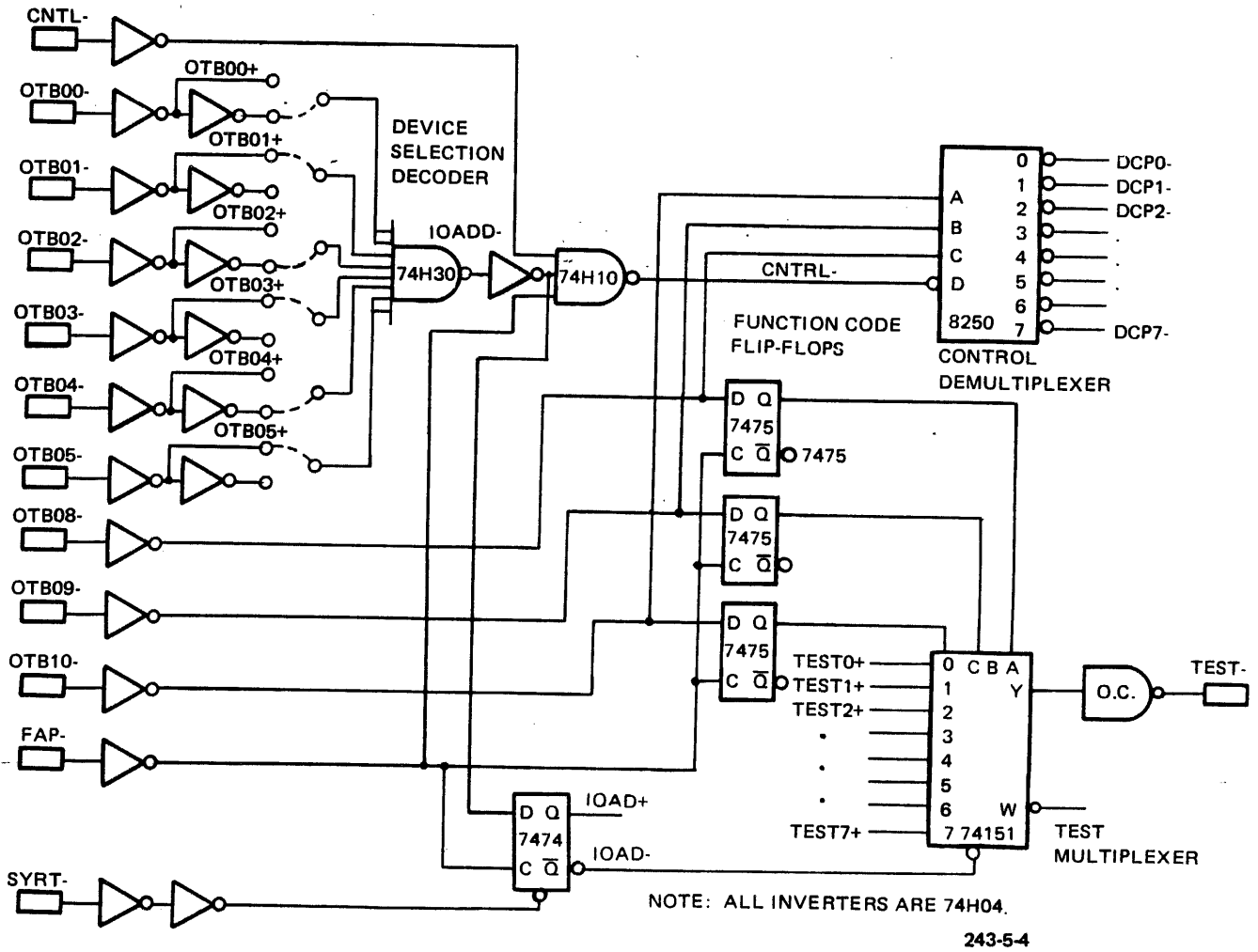


Figure 6-27. Logic for I/O TEST and CTRL Instructions

The test multiplexer selects the test input line indicated by the contents of the function code flip-flop. If the controller has been selected (i.e., IOAD- is low), the multiplexer gates the state of the specified test input line (i.e., one of the lines TEST0+ to TEST7+) onto the TEST-I line. A test input line will, typically, be connected to the output of a status flip-flop in the controller; thus, as many as eight status conditions in a particular controller may be tested. Unused test input lines may be left open.

When any I/O instruction is executed, the selected controller will indicate the result of the test, specified by the function code lines, on TEST-. However, the CPU only samples the TEST- signal when a Test instruction is executed.

The control demultiplexer continuously decodes lines OTB08+ to OTB10+ (not the output of the function code flip-flops, since the state of these flip-flops may change when FAP occurs). During the execution of a Control instruction, the CPU energizes CNTL-. IOADD- goes high on the selected device; before FAP occurs, CNTL- goes low and enables the control demultiplexer. Consequently, the selected control output line (one of the lines DCP0- to DCP7-) is pulsed for the duration of FAP.

The control output lines may be used to initiate any control functions in the controller. A few typical applications are discussed in subsequent sections.

Although D3-, POLL-, and DTP- are not used in the logic diagram, the timing of these pulses is illustrated in the timing diagram since they occur during the execution of all I/O instructions.

6.9.2 DATA OUT LOGIC AND TIMING

Figure 6-28 includes a timing diagram for the execution of a DATA OUTPUT instruction. The logic diagram, also in Figure 6-28, illustrates the logic required on a controller to implement the data output function.

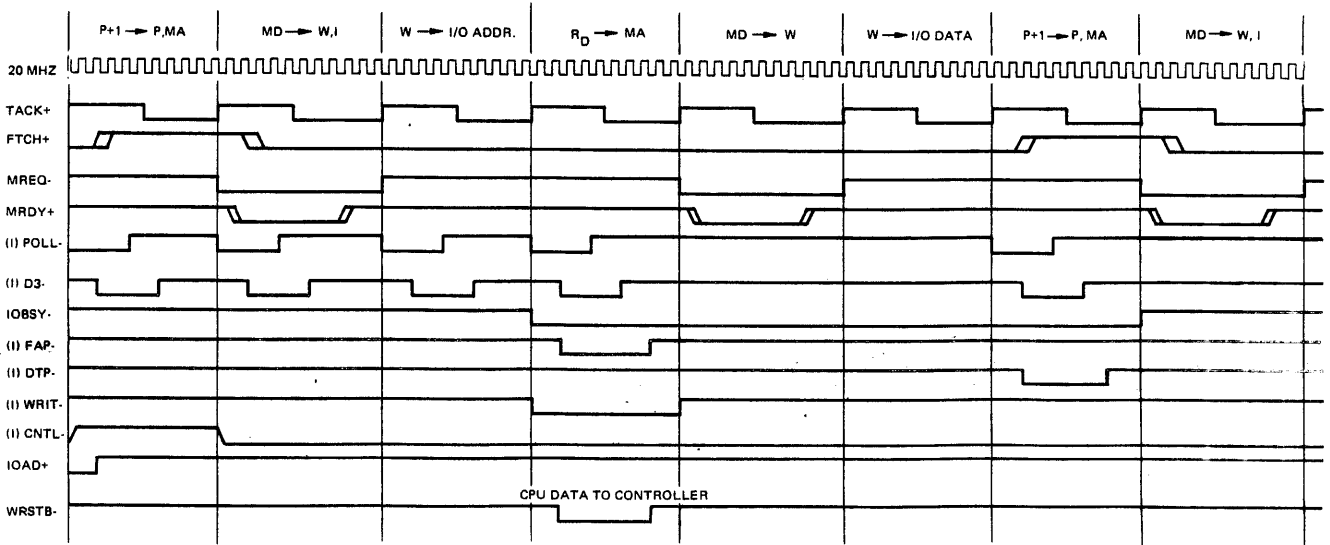
The select phase for a Data Output instruction is identical to the select phase for a Test instruction. When the controller is selected, IOAD+ goes high. The CPU energizes WRIT- only during the execution of a Data Output instruction. If both IOAD+ and WRIT- are true when DTP- is generated by the CPU during the Data phase, the controller stores the state of the signals on the OUT-Bus in the Data Output Buffer flip-flops. SYRT can be used to clear the buffer register by connecting it to the direct clear input.

Many controllers do not use all sixteen data bits; only the OUT-Bus lines relevant to a controller need be sampled and stored.

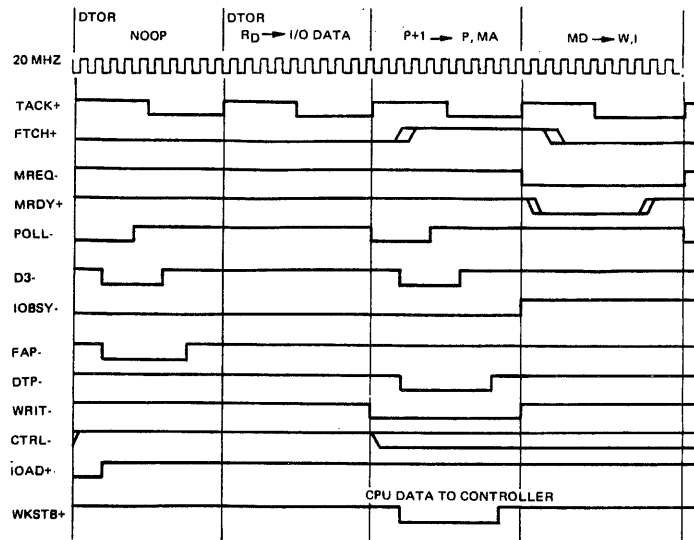
6.9.3 DATA IN LOGIC AND TIMING

Figure 6-29 includes a timing diagram for the execution of a DATA INPUT instruction. The logic diagram, also in Figure 6-29, illustrates the logic required to implement the data input function.

88A00508A-E



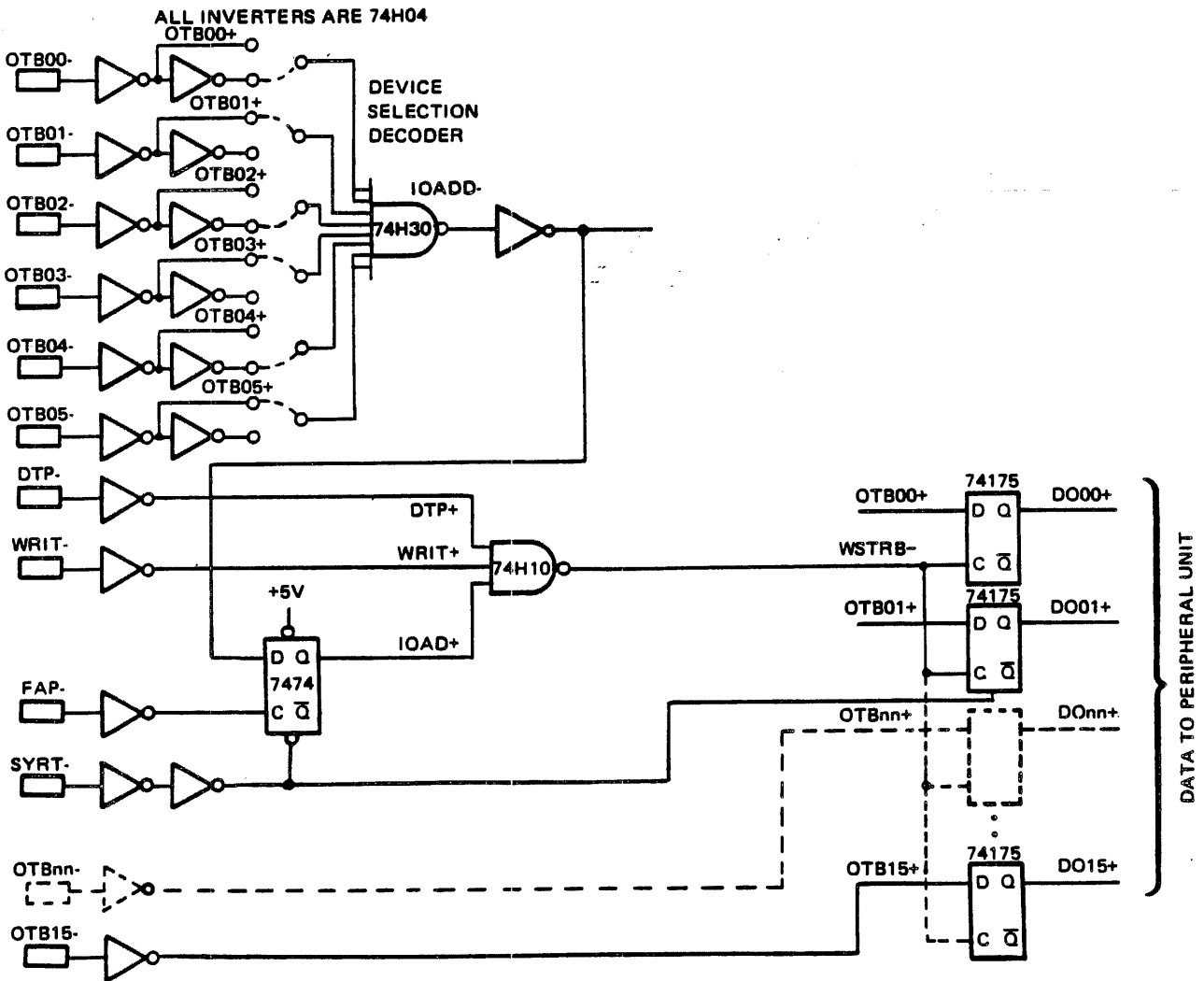
DTOR



DTOM

508-6-7

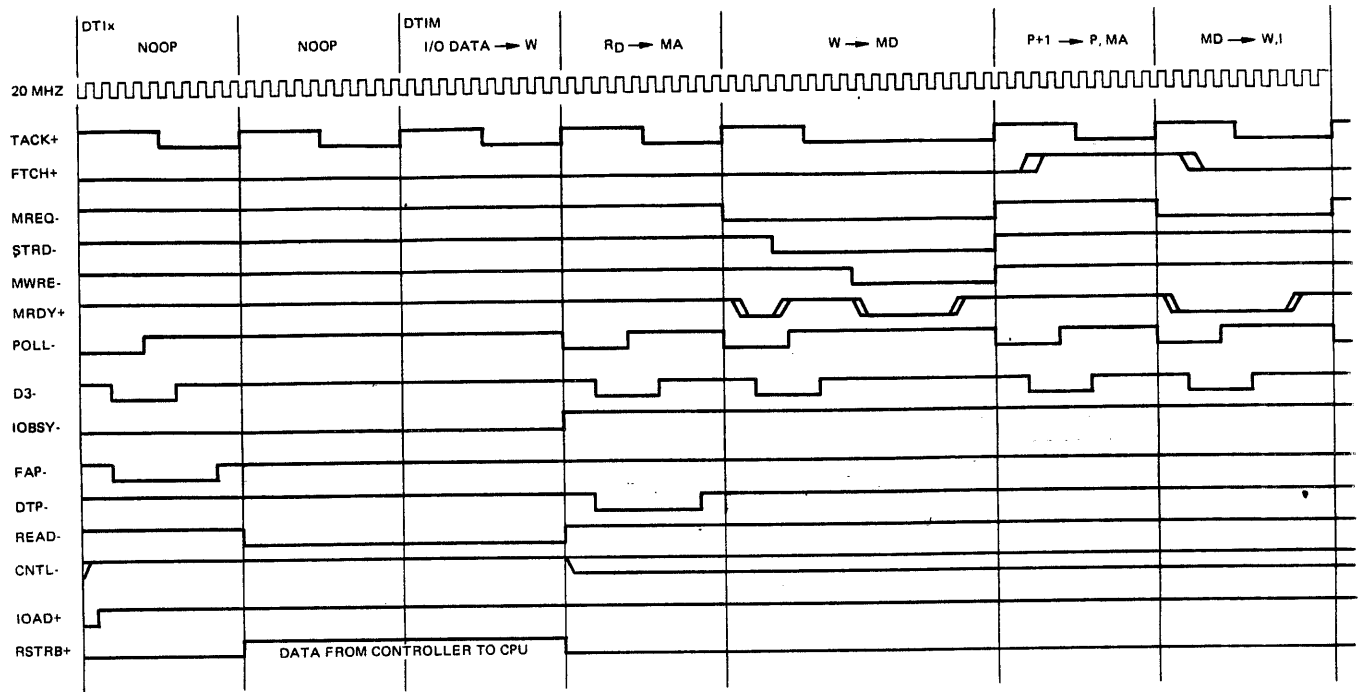
Figure 6-28. Timing and Logic for Data Output Function (Sheet 1 of 2)



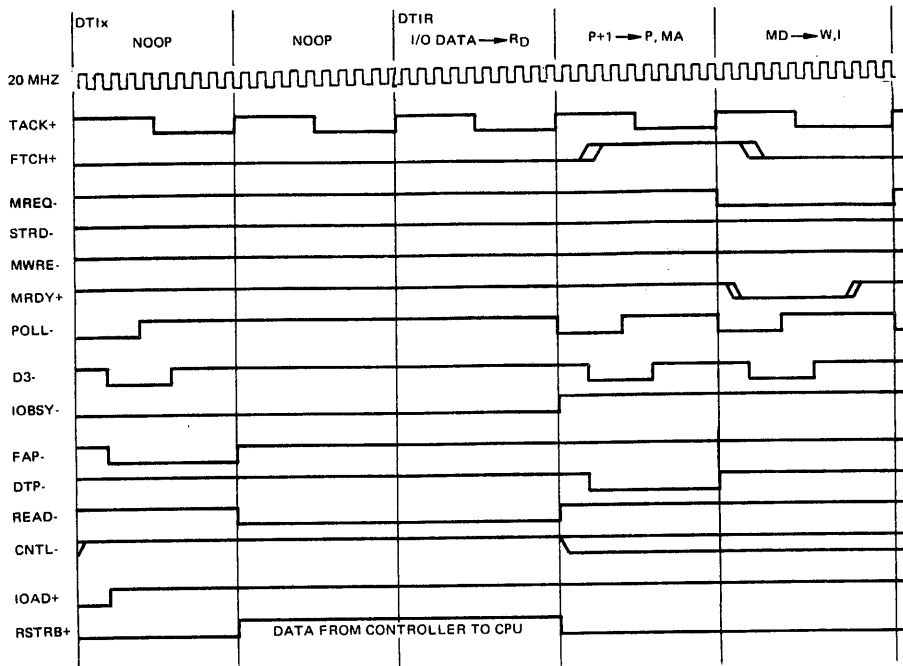
508-6-6

Figure 6-28. Timing and Logic for Data Output Function (Sheet 2 of 2)

88A00508A-E



DTIM



DTIR

508-6-8

Figure 6-29. Timing and Logic for Data Input Function (Sheet 1 of 2)

The select phase for a Data Input instruction is identical to the select phase for a Test instruction. READ- and IOAD+ generate RSTRB+, which is used to strobe the data in the Data Input Buffer onto the IN-Bus. This data must be settled on the IN-Bus when the CPU samples it (see timing diagram).

6.9.4 INTERRUPT LOGIC AND TIMING

There are basically three phases that occur in an interrupt sequence:

- Request Phase
- Priority Determination Phase
- Acknowledge Phase

Although the interrupt sequence is initiated by the controllers and is, therefore, independent of program instructions, control pulses that govern the phases of an interrupt sequence are synchronized with memory cycles.

Figure 6-30 includes a timing diagram for an interrupt sequence. The logic diagram, also in Figure 6-30, illustrates a typical scheme for implementing the interrupt logic on a controller.

6.9.4.1 Request Phase

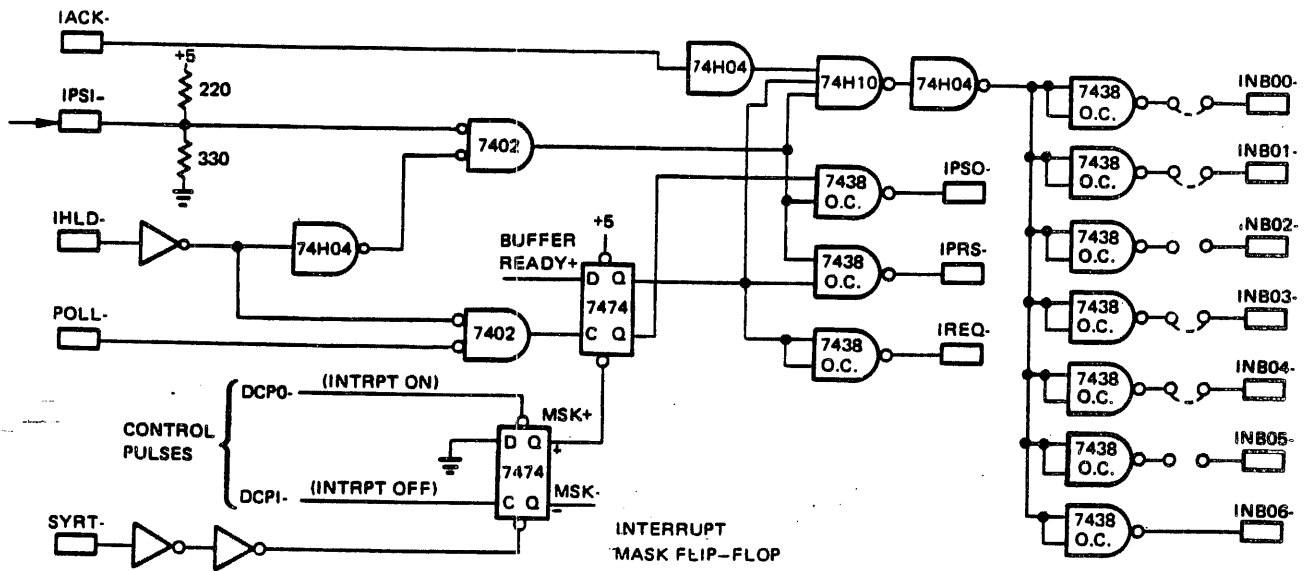
The request phase initiates an interrupt sequence and may occur in any microcycle in which IHLD- is not energized. When a controller requires service, it sets BUFFER READY+ true. Provided the controller's interrupt mask flip-flop is set (described below), the negative-going edge of POLL- will set the controller's interrupt flip-flop. The interrupt flip-flop energizes IREQ, which signals the CPU that one or more controllers require service.

6.9.4.2 Priority Determination Phase

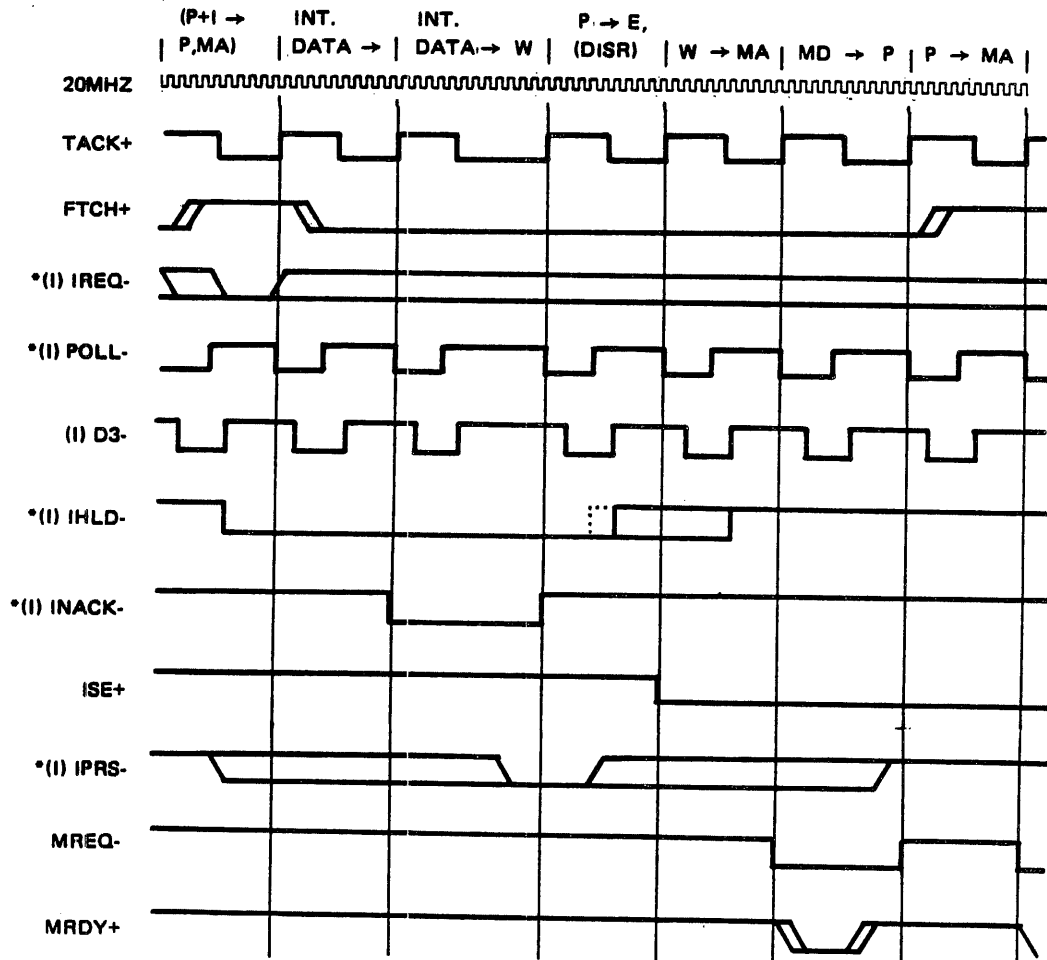
Several controllers may request an interrupt at the same time. The priority determination phase effectively selects only the highest priority controller that is requesting an interrupt. When the processor senses an interrupt request (IREQ- is energized by a controller), the processor energizes IHLD-. IHLD- is common to all controllers and, therefore, arrives almost simultaneously at all controllers. IPSO-, however, is a serial signal; it actually goes only to the highest priority controller (the controller closest to the CPU on the I/O Bus). The highest priority controller receives IPSO- at its IPSI- input; if this controller is not requesting an interrupt, it must transmit an IPSO signal to the controller of the next highest priority, and so on down the line. The highest priority controller that is requesting an interrupt must trap the IPSO signal (it does not transmit the signal to the next controller) and use this signal in conjunction with IHLD- to energize IPRS-.

Although IPSO is not explicitly shown in the timing diagram, its effect is shown in the delay between IHLD- and IPRS-. The CPU will wait for IPRS- signal. The IPRS- signal indicates to the processor that the priority determination phase is complete.

88A00508A-E



466-7-21



508-6-9

Figure 6-30. Logic and Timing for Program Interrupt Sequence

6.9.4.3 Acknowledge Phase

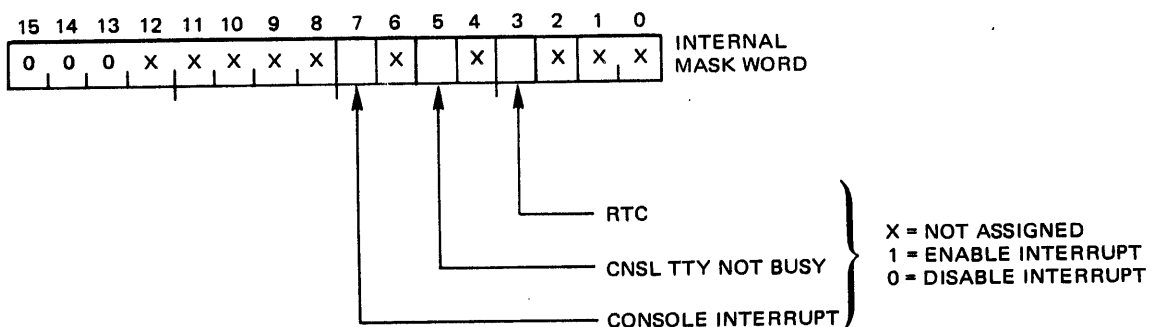
In the last microcycle of an instruction following the one in which IPRS- was received, the processor generates IACK- (unless interrupts are inhibited by the running program during this cycle). This signal is used in conjunction with IPSO, which is present at the highest priority controller seeking service, by the controller to gate the address of its interrupt vector onto the IN-Bus.

At the beginning of the next microcycle, the CPU samples the IN-Bus and simulates a JSR indirect instruction using the address on the IN-Bus as the indirect address. The logic diagram in Figure 6-30 illustrates the "jumper pad" wiring for the address X'5B'. Thus, memory location X'5B' must contain the memory address of the routine responsible for handling an interrupt from this controller.

The forced JSR indirect instruction, caused by the interrupt, disables further response to other controllers that may be requesting an interrupt (i.e., ISE→0). There may be more than one condition on a particular controller that can cause it to seek an interrupt (i.e., cause BUFFER READY+ to go high). In this case, the interrupt routine must determine which condition is relevant to the interrupt at hand; the interrupt service routine may contain Test instructions that interrogate controller conditions. When the routine services the controller, say it inputs a data item from the controller, the controller must remove its interrupting condition (i.e., BUFFER READY+ must be brought low). At the next occurrence of POLL, the controller's Interrupt request flip-flop will then be cleared.

GA's standard peripheral controllers include an interrupt mask flip-flop. When this flip-flop is set, the controller may request an interrupt; when it is reset, the controller may not request an interrupt. The SYRT- signal will reset this mask. Function codes 0 and 1 of the Control instruction for a device are normally assigned to setting and resetting, respectively, its interrupt mask flip-flop.

As an alternative to setting or resetting the interrupt mask flip-flop individually for each controller, a "mask-word" philosophy may be used.



NOTE: IF BIT 13 OF THE ABOVE WORD IS A ONE (1),
BIT 8 IS INTERPRETED AS FOLLOWS:
1 = SET 64K MEMORY MODE, 0 = SET 32K MEMORY MODE.
ALL OTHER BITS MUST BE ZERO. A RTNIV INSTRUCTION
MUST BE EXECUTED TO AFFECT THE MODE CHANGE

A controller may be designed to recognize the device select code X'3E' and to sample one of the assigned bits when a write operation to this controller occurs. The bit sampled would determine the setting of the controller's interrupt mask flip-flop.

6.9.5 GA-16/220 DMA TRANSFERS

A DMA Transfer sequence may be initiated by a DMA peripheral controller if no other DMA controller of higher priority is requesting a transfer. DPSO- goes to the DPSI- input of the highest priority DMA controller. If the highest priority DMA controller is not requesting a DMA interrupt, this signal is transmitted via the controller's DPSO- output to the next controller's DPSI- input. When a controller makes a DMA request, it must not transmit DPSO to the next controller in the DMA series. The DPSO- signal at the Processor is energized when the processor is ready to honor a DMA request.

The timing diagram in Figure 6-31 illustrates the signal sequence of a DMA data output. The logic diagram, also in Figure 6-31, illustrates a typical scheme for implementing the DMA data output function on any controller. Figures 6-32 and 6-33 illustrate the timing of consecutive DMA cycles for the DMA data input/output functions.

A DMA controller must supply the CPU with a memory address for the data items to be transferred. This address is normally contained in an address register (CAR) in the controller; the address register is initially loaded under program control and incremented at each data item transfer to provide sequential location of the data block. In addition, the controller normally contains a block count register (SCR) that specifies the number of data items to be transferred; the block count register is initially loaded under program control and decremented as each data item is transferred. When the block count reaches zero, the controller requests an interrupt to notify the program that the DMA transfer is completed and another may be initiated.

When a controller requires a DMA transfer, TRANSFER REQUEST+ (TREQ+) on the controller becomes true. At the next occurrence of POLL-, the controller's DMA Request flip-flop (DMAR) is set. The DREQ- line is energized and the controller's DPSO- output line is disabled (i.e., brought high), thereby preventing lower priority controllers, which may also be requesting a DMA transfer, from responding to balance of the DMA cycle sequence.

The CPU generates DACK- in response to DREQ-; only the highest priority controller currently energizing DREQ- uses this signal. The controller must gate the contents of the current address register (CAR) onto the IN-Bus; it should also remove the TRANSFER REQUEST+ signal so that, at the next occurrence of POLL-, its DMA request flip-flop will be reset. The positive-going edge of DACK- should be used by the selected controller to set its DMA transfer flip-flop (i.e., DCAD+ becomes true) as well as to increment its current address register and decrement its block count register.

If DCAD+ is true when DDTP- occurs, the controller strobes (via WDT+) the data from the OUT-Bus into a buffer register in the controller. If BLOCK COMPLETED- is also true (i.e., the block count register when last decremented reached zero), DDTP- may be used to set an interrupt request flag in the controller (i.e., BCF- becomes true; this signal corresponds to BUFFER READY+ in the logic portion of Figure 6-30).

The DMA input request and acknowledge sequence (Figure 6-34) is identical to the DMA output request and acknowledge sequence. DCAD+ in the controller becomes true with the positive-going edge of DACK-; RESET TRANSFER REQUEST pulses the input operation flip-flop, thereby bringing RDT+ true. If DATA DIRECTION IN+ is true, RDT+ energizes the following signals:

- STROBE - which strobes the data from the controller's data input buffer into the IN-Bus;
- RQIN - which signals the CPU that the DMA transfer is a data input operation.

The negative-going edge of DDTP-I resets the input operation flip-flop (RDT+ becomes false), and the positive-going edge resets the DMA transfer flip-flop (i.e., DCAD+ becomes false).

6.10 INTERFACE TO MHSDC

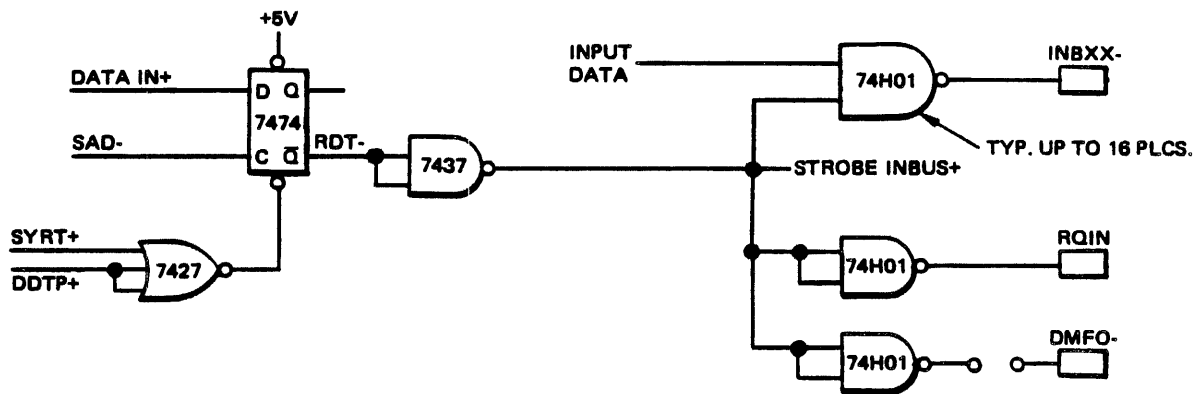
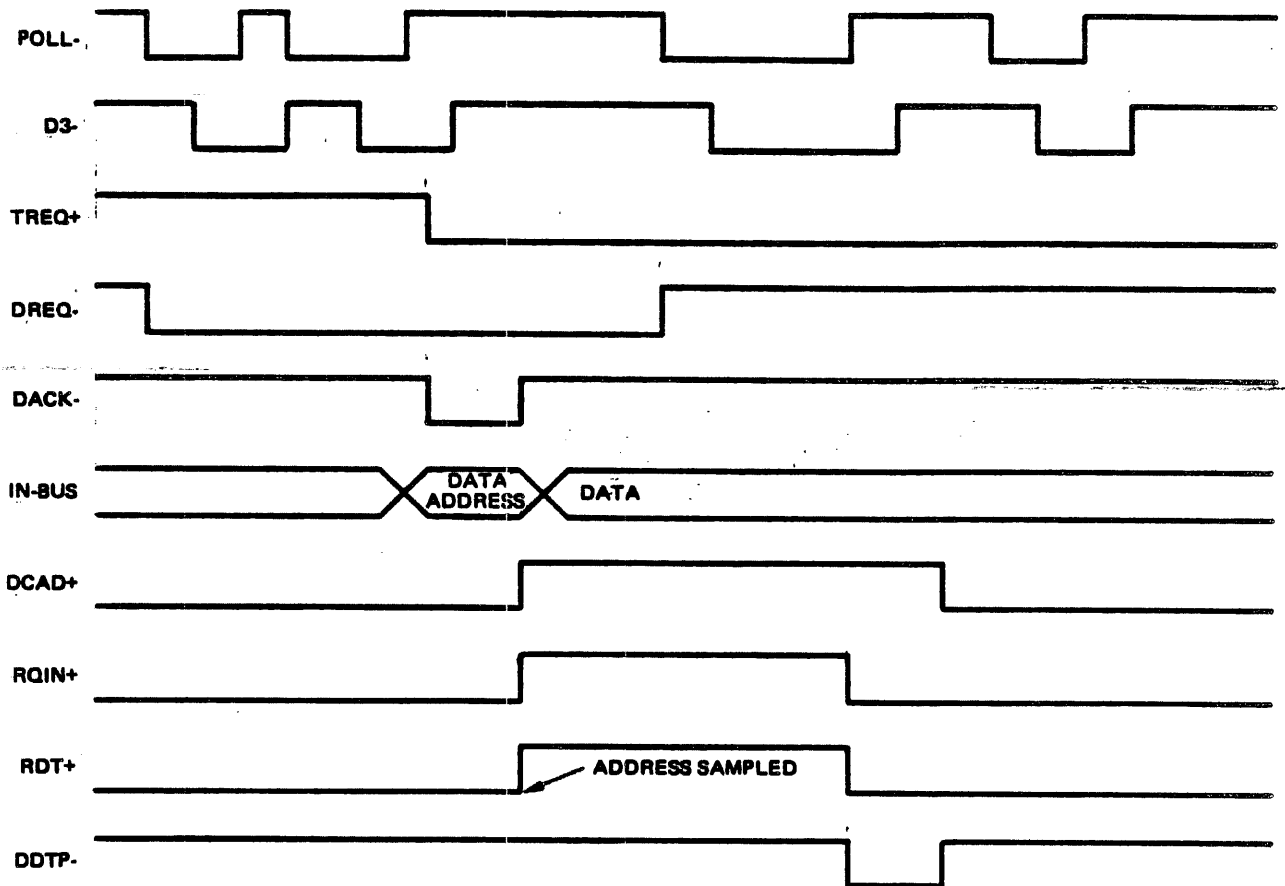
6.10.1 GA-16/220 DATA CHANNEL BUS DESCRIPTION (MHSDC)

A controller may contain all the logic required to implement a DMA function. In this case, it communicates directly with the I/O Bus during each DMA transfer it performs. The controller must contain an address register and a block count register that may be loaded under program control (normally two device select codes are assigned to such a controller). The controller must also have a network to request service, determine its priority (if there are other controllers that use DMA), supply the memory address to the CPU, and transfer data items to/from the I/O Bus. GA controllers for DMA application do not normally use this approach.

As an alternative to implementing all the necessary DMA logic separately on each controller that uses DMA, a Data Channel Module that incorporates a large share of the logic required for performing a DMA transfer is available from General Automation.

This data channel module is the MHSDC controller. A single Data Channel Module provides eight high-speed channels and will service up to eight peripheral units, maintaining a block count register and address register for each active channel. The MHSDC performs memory request and supplies the appropriate data transfer address to the I/O Bus. A Data Channel Module communicates with the CPU via the I/O Bus, but it communicates with controllers associated with its data channels via the Data Channel Bus. If a controller is connected to a data channel, it must have a network to request service from the Data Channel Module, a network to determine its priority relative to other devices that may be connected to that Data Channel Module, and a network to identify itself to the Data Channel Module. For the actual transfer of data, however, the controller uses the I/O Bus. Standard GA-supplied controllers that use DMA employ the Data Channel Module approach.

Just as IPSI/IPS0 determined interrupt priorities on the I/O Bus, DCSI/DCS0 (Data Channel Status In/Out) determine the priority of the different data channel device controllers that connect to the MHSDC's Data Channel Bus. The controller physically closest to the MHSDC will have the highest DMA priority.



508-6-13

Figure 6-34. Timing and Logic for DMA Input

In the GA-16/220, the MHSDC may be placed in any slot to the right of the CPU-2 module. In the event that an external I/O chassis is used to house data channel controllers (or an MHSDC), the 220 chassis must either have all I/O slots utilized or contain shorter boards in all unused I/O slots to maintain signal continuity.

If a second external I/O chassis, housing additional data channel controllers or another MHSDC, is to be cabled to the system signal, continuity must again be maintained. The Data Channel Priority Status In/Out (DPSI/DPSO in this case) signals may be propagated in three ways:

1. Filling all I/O slots in the first I/O cage with controller boards,
2. Placing GA shorter boards in all unused I/O connectors, or
3. Hardware jumpering pins 65 and 66 of the last data channel controller in the first external I/O chassis, to the corresponding pin locations of the CIT connector for that chassis.

The Data Channel Bus Signals are summarized in Table 6-23. For further information regarding the MHSDC module, reference GA Publication Number 88A00415A, GA 1615-0208/1615-0209 Multiple High-Speed Data Channel - 8 Module.

6.10.2 DATA CHANNEL TIMING AND INTERFACING

A Data Channel Module can operate in three modes with respect to each of its eight high-speed data channels. These are termed the initiate mode, the data transfer mode, and the chaining mode.

When a peripheral controller wishes to initiate a DMA block transfer of data (usually in response to a PIO instruction issued by the running program to that controller), it requests service from the Data Channel Module and indicates the initiate mode via the DINT- line. The Data Channel retrieves the Channel Address Register (CAR) and the Scan Control Register (SCR) words from the memory locations dedicated to that controller's data channel by sending DREQ- to the CPU. If the controller is to support chaining, it must store the two high-order bits of the SCR when the SCR is on the OUT-Bus.

Once the initialization phase is complete, the controller may request data transfers at will. It requests service from the Data Channel Module when it is ready to send or receive data (the Data Channel Module "knows" that it is in the data transfer mode with respect to this controller, since the controller does not energize DINT-). The Data Channel Module increments the CAR and decrements the SCR for the controller in question at each data channel transfer. When the block count is exhausted, the Data Channel Module signals the controller via DCBC-.

If chaining is indicated, the Data Channel Module automatically enters the chaining mode when it issues DCBC-. The operations performed in this mode are similar to the operations performed in the initiate mode; however, it retrieves the new CAR from the locations specified by the current CAR and the new SCR from the location specified by the new CAR. The controller performs the same operations it would in the initiate mode, except, of course, it does not energize the DINT- line.

Table 6-23. Data Channel Bus Signals
(Sheet 1 of 2)

Signal	MHSDC Pin No.	Direction	Definition
DCEN-	80	To controller	Data Channel Enable. This is a common line on the DC Bus that is energized when the Data Channel Module is present. This indicates to the controllers that operate on either Programmed I/O or DMA that they should enter the DMA mode.
DADO- to DAD2-	72 73 74	To Data Channel	Device Address Lines. These lines are used to return a 3-bit address to the Data Channel Module, identifying which controller is requesting a DMA.
DCBC-	77	To controller	Data Channel Block Complete. Common signal returned to the controllers from the Data Channel Module indicating completion of the current block transfer. This signal is also true during the data channel initialization sequence.
DHLD-	71	To controllers	Data Channel Hold. Common signal from the Data Channel Module to the controllers that use Data Channel logic to freeze any further Data Channel requests and to enable the Data Channel Address lines (DADO- through DAD2-) and other control signals.
DINT-	75	To Data Channel	Data Channel Initiate. Common line to the Data Channel Module from its associated controller that has priority indicating to the data channel that the controller is in an 'Initialize' mode, the data channel retrieves the starting address and block count data from the pair of dedicated memory locations and places it in the pair of associated hardware registers. DINT- is energized only by the controller that has priority. The enable for DINT- should be energized following the programmed I/O instruction that initializes the controller and the controller should request one redundant cycle, indicating to the data channel to retrieve the block count and starting address. DINT- should only be energized during this first redundant cycle.
DCRQ-	70	To Data Channel	Data Channel Request. Common signal from the controllers to the data channel requesting a memory access operation.

Table 6-23. Data Channel Bus Signals
(Sheet 2 of 2)

Signal	MHSDC Pin No.	Direction	Definition
DCSO- DCSI-	67 68	Between controllers	Data Channel Priority Status Out. Serially transmitted signal, among controllers associated with a given Data Channel Module indicating that no controller of higher priority is requesting a DMA cycle. Equivalent to DPSO and DPSI and using approximately the same scheme. Each controller receives on DCSI and transmits on DCSO.
DCAK-	76	To controller	Data Channel Acknowledge. Common acknowledge line to the controllers indicating to the controller that has priority that the channel address has been accepted.
DCIN-	78	To Data Channel	Data Channel In. Common data direction line to the Data Channel Module. DCIN- energized indicates a data input to the CPU and otherwise a data output.
DCTP-	79	To controller	Data Channel Transfer Pulse. Common signal to the controllers used to accept output data or to terminate their operation.

6.10.3 DATA CHANNEL INITIALIZATION SEQUENCE

Figure 6-35 illustrates the timing relationships for a controller initialization sequence. The initialization normally starts immediately following the programmed I/O (PIO) instruction that is designed to initialize the controller. The logic diagram in Figure 6-36 illustrates a typical data channel/controller interface logic needed to implement initialization as well as the data input function (with timing diagram also shown).

When a controller requires a DMA transfer, it brings TRANSFER REQUEST+ true. If the Data Channel Module is not energizing DHLD- (i.e., it is not servicing another controller's request flip-flop), REQ+ will be set. REQ+ energizes dcrq- to request service from the Data Channel Module.

As many as eight controllers may be associated with a particular Data Channel Module (one per channel), all of which may be energizing DCRQ- simultaneously. At the next occurrence of POLL-, the Data Channel Module energizes DHLD-, which freezes any further data channel requests from controllers associated with that module. The highest priority controller associated with the module receives DCSO- on its DCSI- input. If this controller is not requesting a data transfer, it retransmits the signal on its DCSO- output to the controller of second highest priority. The highest priority

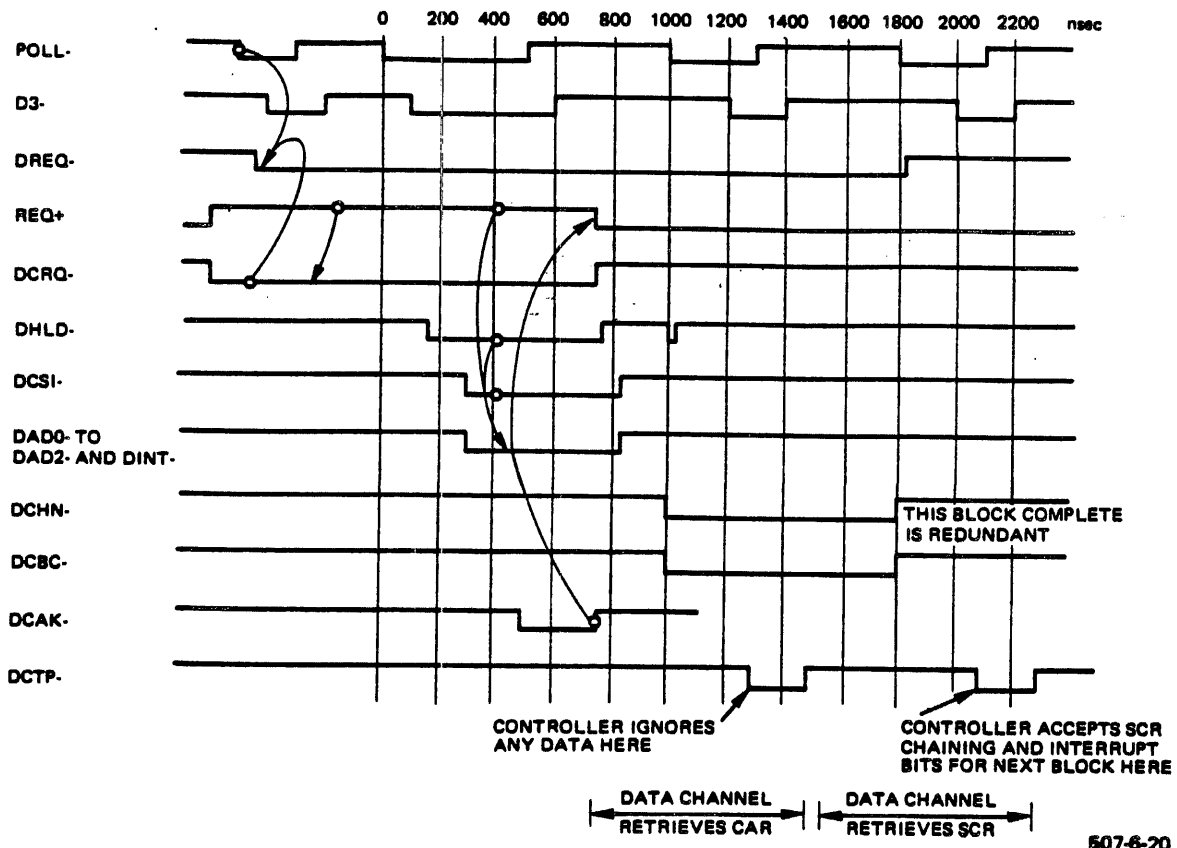


Figure 6-35. Timing for Data Channel Initialization Sequence

controller that is requesting service traps the DCSO signal, places its channel address onto lines DAD0 to DAD2, and energizes the initialization signal DINT-. The channel address enables the Data Channel Module to determine which of the eight controllers that may be connected to it is to be serviced.

The Data Channel Module then generates DCAK-. The selected controller should use the positive-going edge of DCAK- to reset the REQ+ flip-flop.

The Data Channel Module performs two consecutive Direct Memory Accesses to retrieve the CAR and SCR from the memory locations dedicated to the controller in question. The Data Channel Module energizes DCTP- for each of these cycles, as shown in the timing diagram. The controller ignores the data on the OUT-Bus during the first occurrence of DCTP-. At the second occurrence of DCTP-, the controller accepts the SCR interrupt and chaining bits for the next block transfer from OUT-Bus lines OTB14 and OTB15, respectively.

6.10.4 DATA CHANNEL INPUT BLOCK TRANSFER OPERATIONS

The timing diagram in Figure 6-36 illustrates the sequence of signals that occur during a transfer operation that causes a block of data to be input to main memory. The logic diagram, also in Figure 6-36, illustrates a typical data channel/controller interface logic. Figure 6-37 illustrates the timing of consecutive DMA cycles for the data input function.

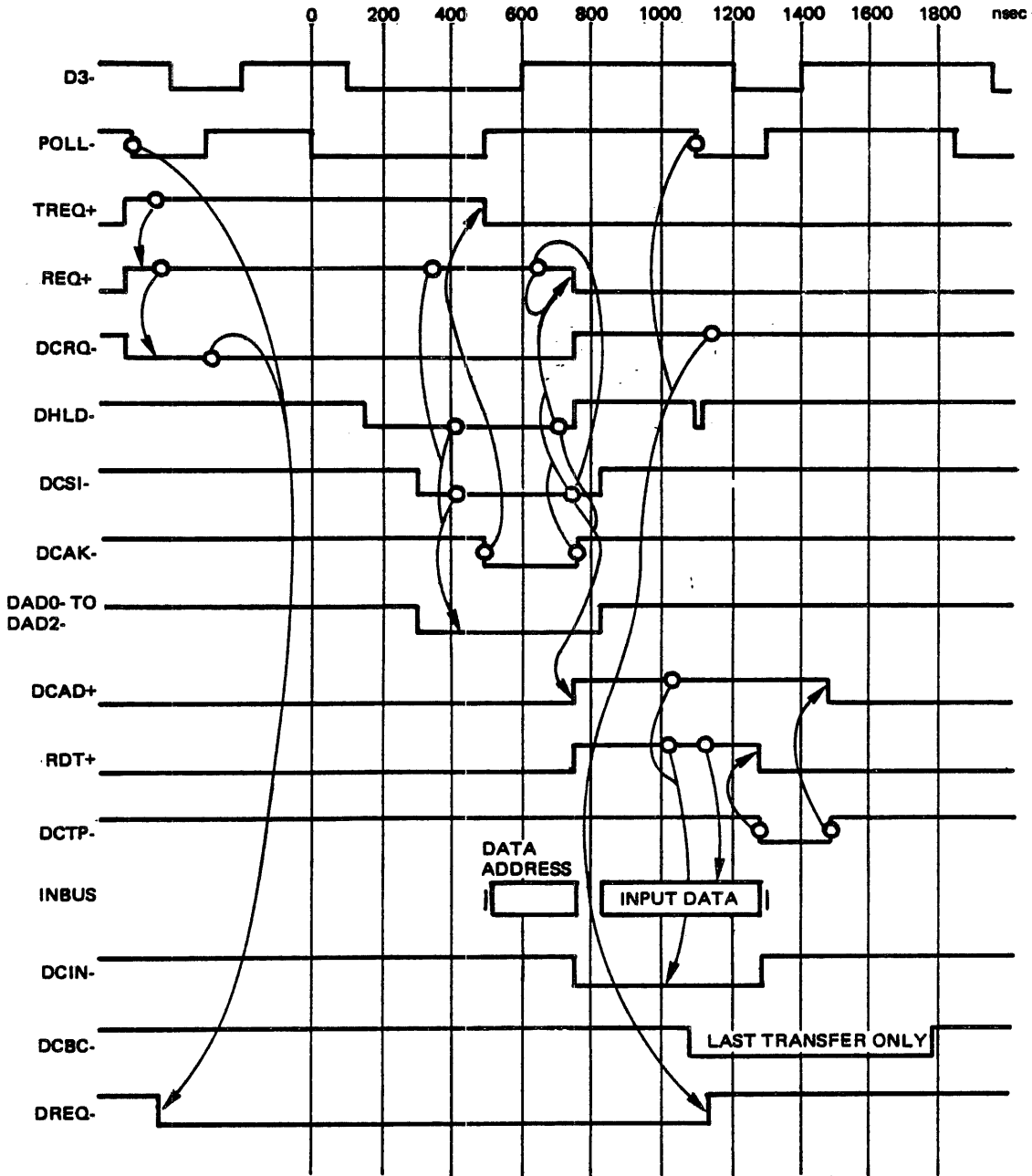
Following the initialization sequence, the controller again causes REQ+ to set. REQ+ energizes DCRQ- to request service from the Data Channel Module. As in the initialization sequence, priority is resolved and the Data Channel Module determines which of the eight controllers is to be serviced.

The Data Channel Module then generates DCAK-. The controller should use the negative-going edge of DCAK- to reset TRANSFER REQUEST and the positive-going edge of DCAK- to set its "select" flip-flop (DCAD+), reset the REQ+ flip-flop, and set RDT+ in the controller (via the RESET REQ+ signal).

The RDT+ signal is used to select the data onto the IN-Bus (INB00- to INB15-) and to energize DCIN-, which signals the Data Channel Module that the DMA transfer is a data input operation.

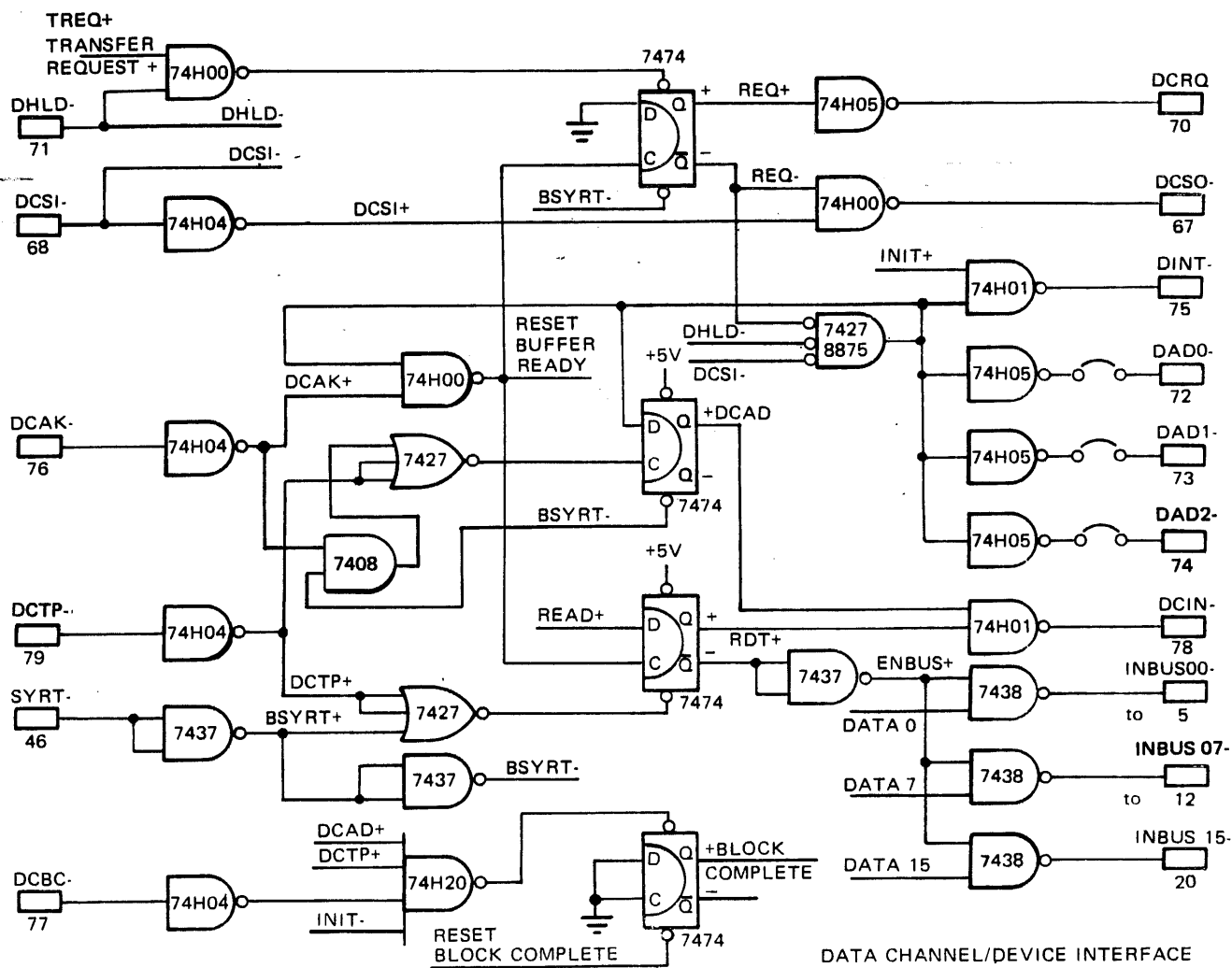
The Data Channel Module energizes DCTP-. On the negative-going edge of this signal, the controller clears RDT+, thereby disabling DCIN- and the selection of data onto the IN-Bus. The positive-going edge of DCTP- is used to reset DCAD+ in the controller.

The Data Channel Module increments the CAR and decrements the SCR for the selected controller with each data word that is transferred. If the least significant 14 bits of the SCR equals zero, the Data Channel Module energizes DCBC-, which signals the controller that the word count is exhausted (i.e., the transfer that the controller is currently performing is the last in that block). Upon receiving DCBC-, the controller will normally initiate a program interrupt sequence.



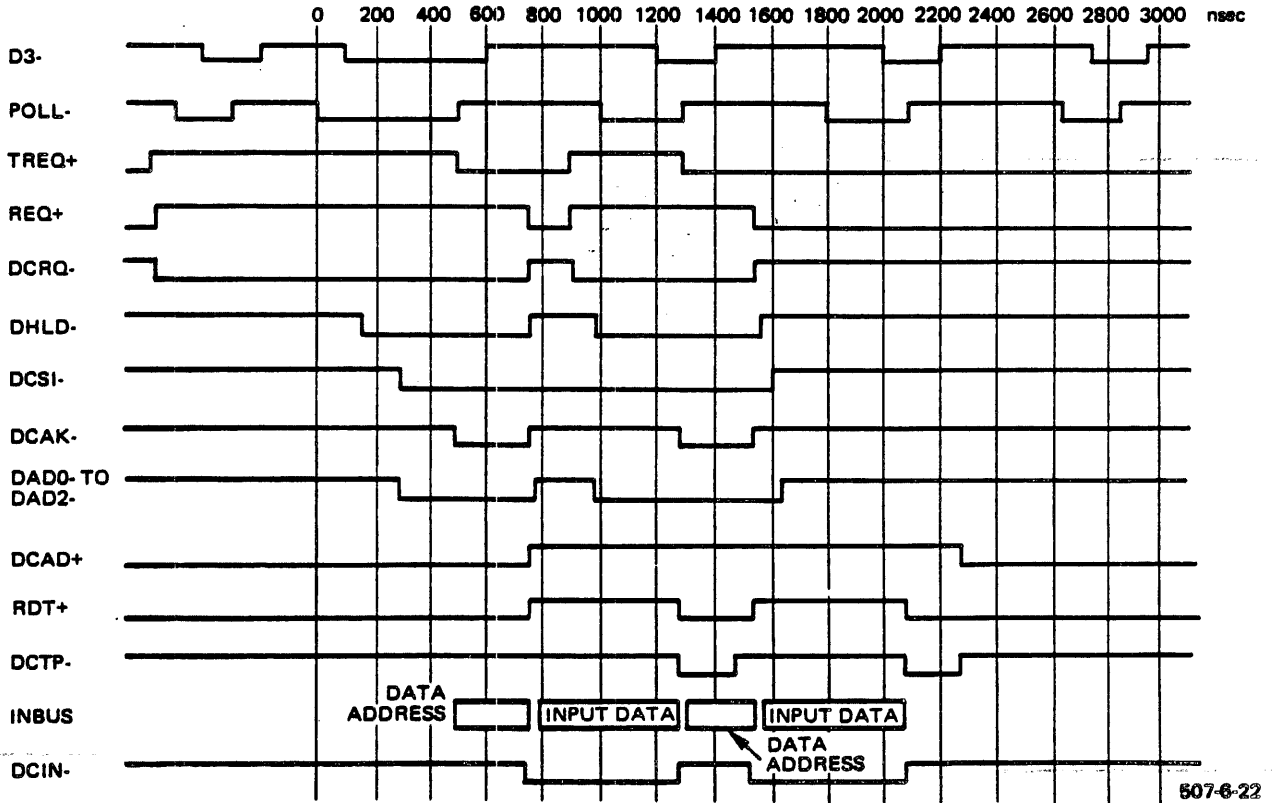
508-6-14

Figure 6-36. Data Channel Transfer (Input) (Sheet 1 of 2)



507-6-21

Figure 6-36. Data Channel Transfer (Inputs) (Sheet 2 of 2)



507-6-22

Figure 6-37. Timing for Consecutive Data Channel Transfers (Input)

6.10.5 DATA CHANNEL OUTPUT BLOCK TRANSFER OPERATIONS

The timing diagram in Figure 6-38 illustrates the sequence of signals that occur during a block transfer operation that causes a block of data to be output from main memory. The logic diagram, also in Figure 6-38, illustrates the logic required in addition to that shown in Figure 6-36 needed to implement the data output function. Figure 6-39 shows the timing of consecutive DMA cycles for the data output function.

Following the initialization sequence, the controller again causes REQ+ to set. REQ+ energizes DCRQ- to request service from the Data Channel Module. As in the initialization sequence, priority is resolved and the Data Channel Module determines which of the eight controllers is to be serviced.

The Data Channel Module then generates DCAK-. The controller should use the negative-going edge of DCAK- to reset TRANSFER REQUEST+ and the positive-going edge of DCAK- to set its "select" flip-flop (DCAD+) and reset the REQ+ flip-flop.

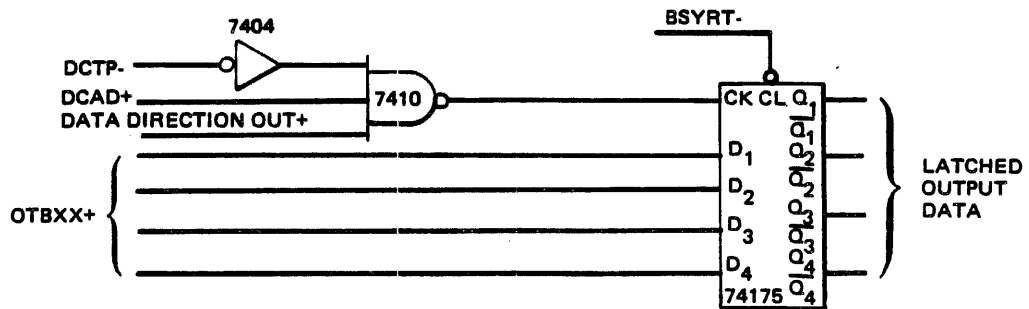
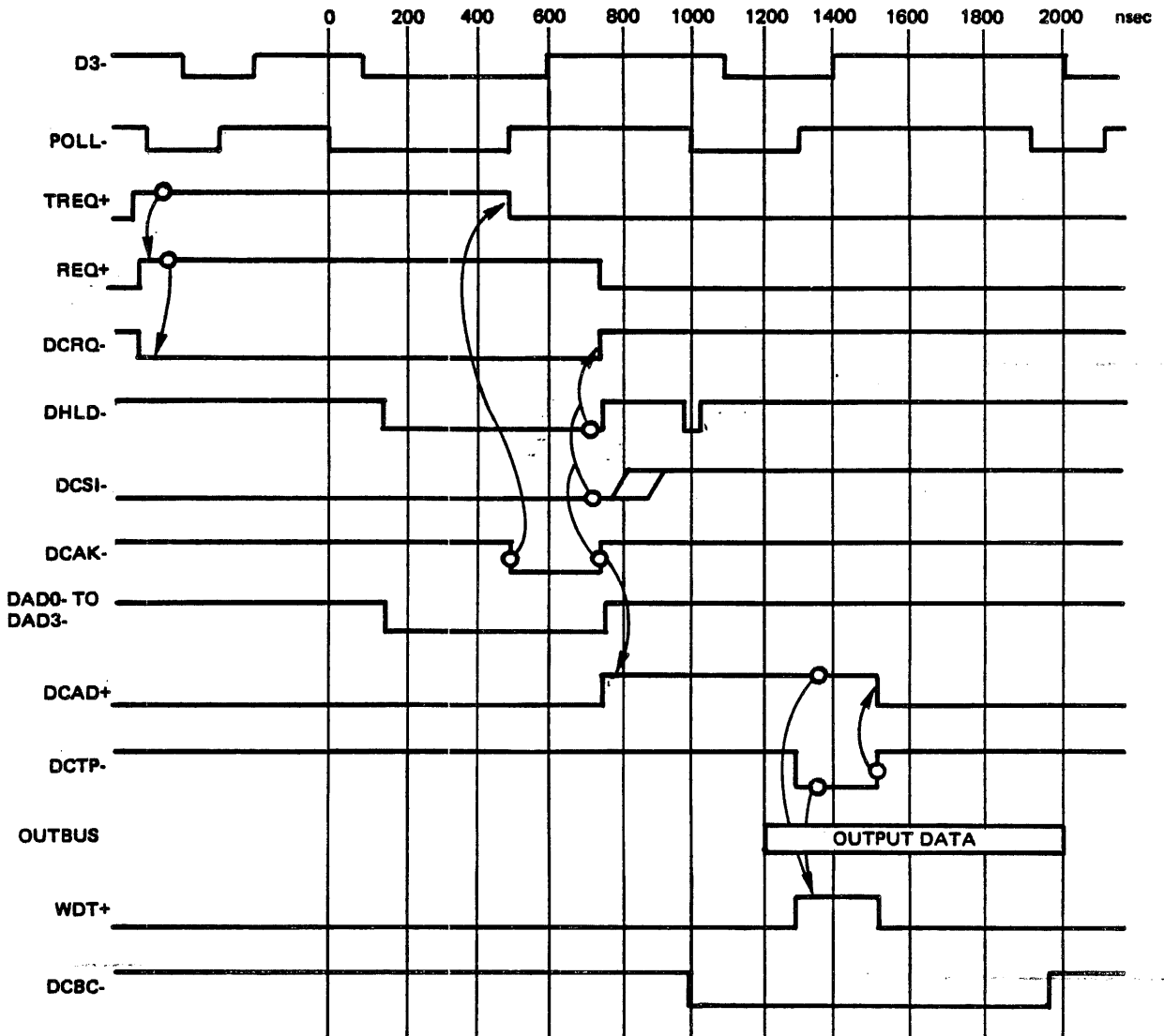
The Data Channel Module energizes DCTP-. On the negative-going edge of this signal, the controller generates WDT+ which strobes the data off the OUT-Bus. The positive-going edge of DCTP- is used to reset DCAD+ in the controller and to clear WDT+.

The Data Channel Module increments the CAR and decrements the SCR for the selected controller with each data word that is transferred. If the least significant 14 bits of the SCR become zero, the Data Channel Module energizes DCBC-, which signals the controller that the word count is exhausted (i.e., the transfer that the controller is performing is the last in that block). Upon receiving DCBC-, the controller will normally initiate a program interrupt sequence.

6.10.6 DATA CHANNEL/CONTROLLER CHAINING SEQUENCE

Figure 6-40 illustrates the timing relationships for a device chaining sequence. Figure 6-41 illustrates the data channel interface logic relative to all standard GA controllers. The Data Channel Module automatically enters this sequence when the block count for the respective controller is exhausted and chaining is indicated. This sequence is similar to the data channel initialization sequence. Since both sequences must retrieve the contents of two dedicated memory locations and load the respective SCR and CAR registers on MHSDC.

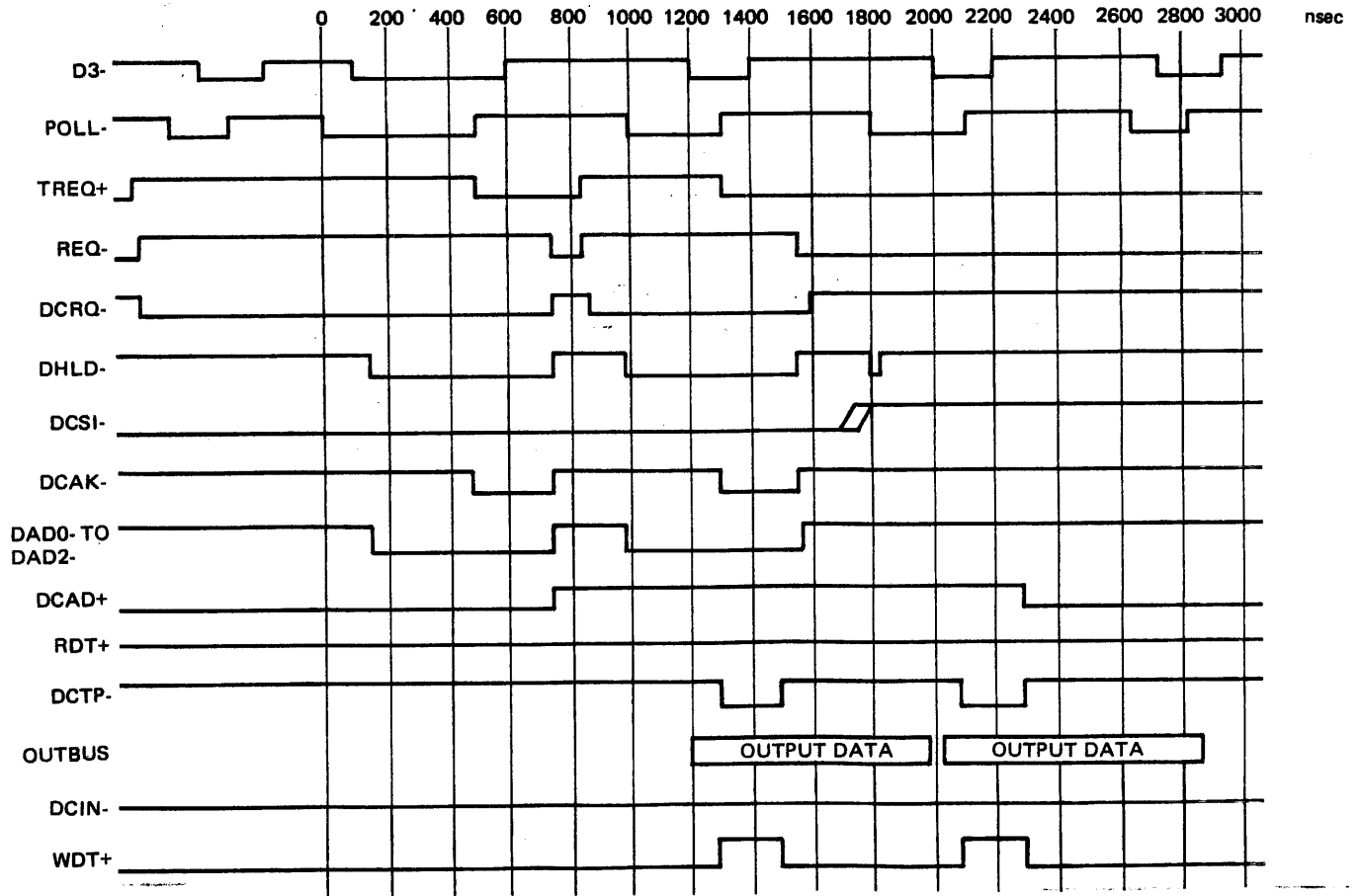
The left part of the timing diagram illustrates the signals which occur during the last data transfer from the present block. After the last item is transferred, the controller must "know" that the Data Channel Module has entered the chaining sequence since the controller must accept the new SCR chaining and interrupt bits from the OUT-Bus at the next occurrence of DCTP-. Also, if a program interrupt is indicated by the old SCR bit (stored by the controller during the last chaining or indicate operation), the controller must initiate this interrupt before it stores the new SCR chaining and interrupt bits.



508-6-15

Figure 6-38. Timing and Logic for Data Channel Data Transfer (Output)

88A00508A-E



508-6-16

Figure 6-39. Timing for Consecutive Data Channel Data Transfer (Output)

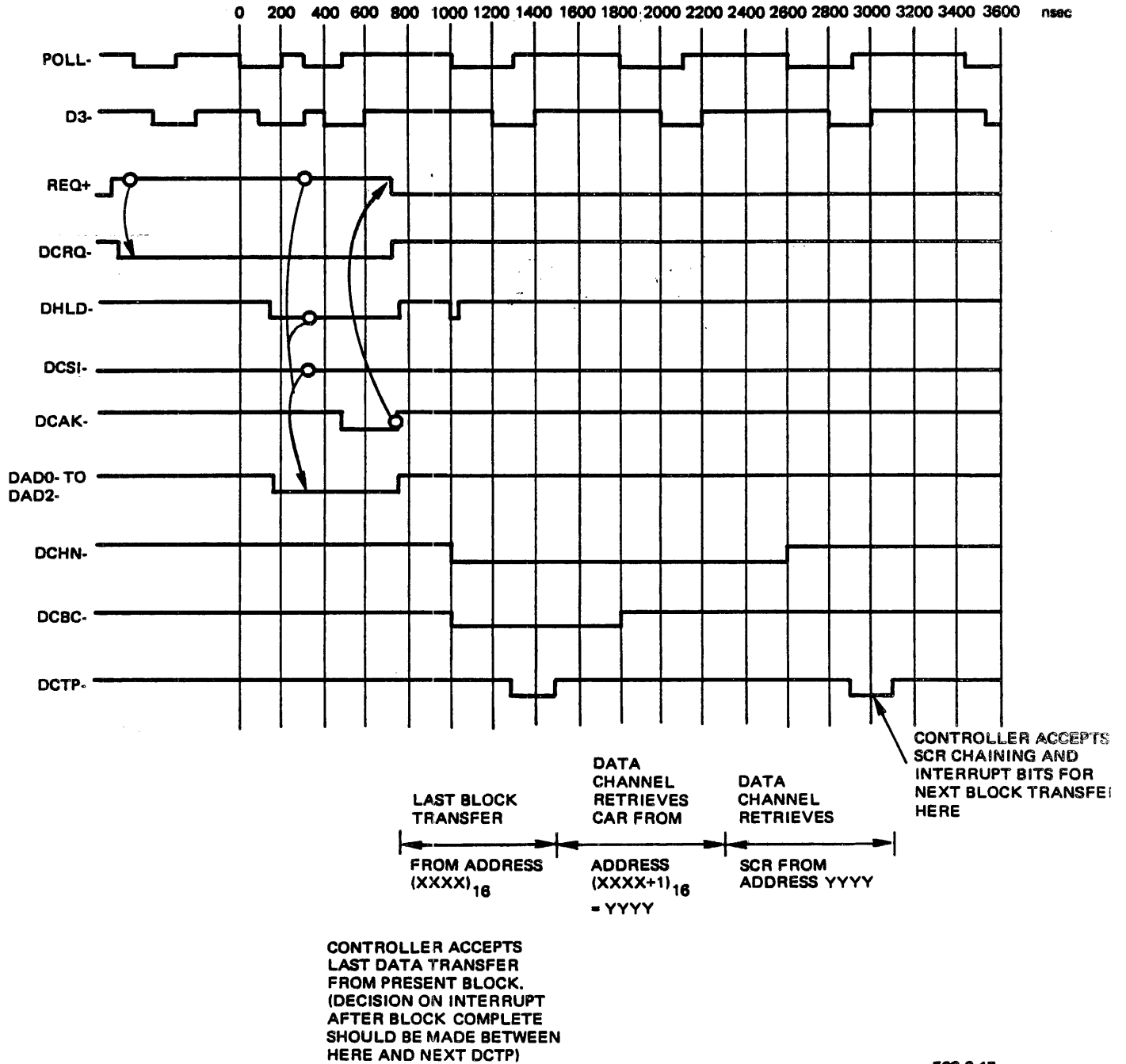


Figure 6-40. Data Channel/Controller Chaining Sequence

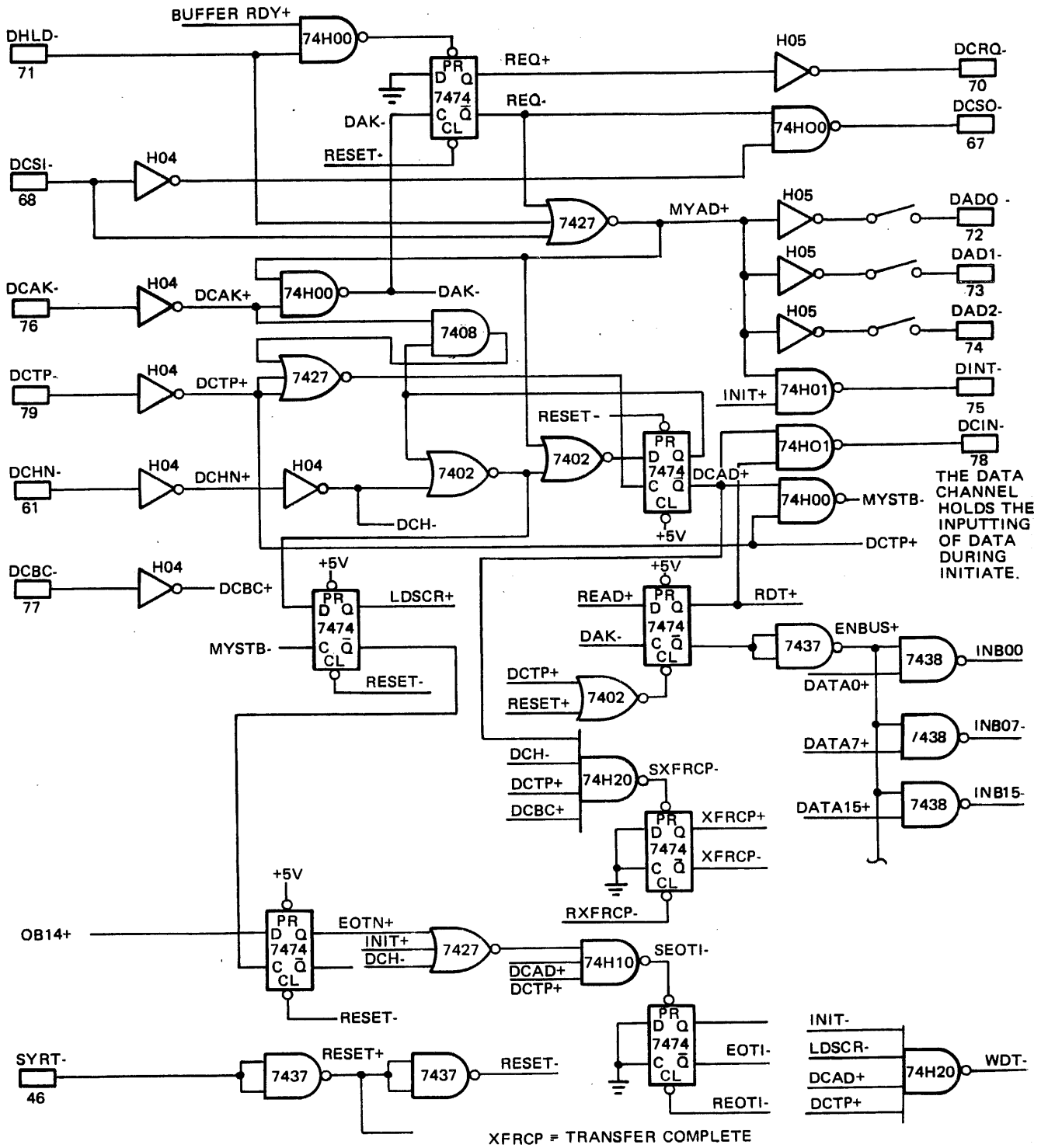


Figure 6-41. Standard Controller to Data Channel Interface Featuring Logic to Accommodate all Cases

6.10.7 LATENCY TIME

For a block transfer DMA cycle, the minimum latency time (time between TRANSFER REQUEST going true and the DMA cycle starting) is 50 nanoseconds and the maximum latency time is 2.5 microseconds.

During an initialization or chaining sequence, the active channel "locks out" all other channels until the sequence is completed. The DCHN signal is used to do this. Also, a channel cannot enter a cycle immediately following an initialization or chaining sequence or another channel on the same Data Channel Module. Therefore, the maximum latency time, when one channel could be initializing and other channel (of higher priority on the same Data Channel Module) is requesting, is 2.8 microseconds. Also, when one channel is chaining and another channel (of high priority on the same Data Channel) is requesting, the maximum latency time (for the higher priority channel) is 3.2 microseconds. If the two channels are on different Data Channel Modules, the latency time is reduced by one memory cycle.

One should design the system and assign priorities with these times in mind to meet the worst-case response times. It should be noted that to complete a data-out transfer, for example, takes one memory cycle in addition to the latency time.

6.11 INTERFACE EXAMPLES

This section presents several hypothetical controllers as examples illustrating the implementation of the various principles described earlier. Example Controllers 1 and 2 are both intended for a simple Console Panel device. They are essentially the same controllers with the exception that Controller 2 has interrupt capabilities while Controller 1 does not. Example Controllers 3 and 4 are both intended for a simple CRT device. Both controllers allow DMA output operations; Controller 3 does not use the Data Channel Module but Controller 4 does.

In all four examples, pin numbers have been assigned on the controller drawings; pin assignments for each signal are shown above the small rectangles. Pins 1 through 66 are reserved for specific I/O Bus signals; pins 67 through 80 are reserved for specific Data Channel Bus signals; and pins 81 through 136 have been arbitrarily assigned to lines between the peripheral and its controller. The reader should refer to Table 6-23 for a complete list of I/O signal pin assignments.

6.11.1 EXAMPLE CONTROLLERS 1 AND 2

Assume that a user wishes to design a controller that provides an interface between a simple Console Panel device and a GA-16/110 or the GA-16/220. This hypothetical Console Panel has 16 input and 16 output lines; assume that all 32 lines are buffered in the peripheral unit. The input buffer is loaded from 16 data switches on the Console Panel. The Console Panel also has two buttons that the operator can press to generate an "input data to CPU" request pulse or an "output data from CPU" request pulse (INPUT+ or OUTPUT+, respectively). There is also an indicator lamp on the panel that can be lit by a control pulse from the controller (DCP3-). In addition, there is a ready line (RDY+) from the Console Panel device; RDY+ remains true as long as the peripheral unit has power on. Table 6-24 summarizes the interface signals between the hypothetical peripheral unit and its controller.

Table 6-24. Interface Signals for Example Controllers 1 and 2

Mnemonic	Source	Description
RDY+	Peripheral Unit	Indicates that peripheral unit is powered on.
INPUT+	Peripheral Unit	Indicates that peripheral unit has data to be input to CPU.
OUTPUT+	Peripheral Unit	Indicates that peripheral unit is requesting data from CPU.
DI00+ to DI15+	Peripheral Unit	Data input lines to controller.
DO00+ to DO15+	Controller	Data output lines to peripheral.
DCP3-	Controller	Lights indicator lamp on Console Panel.

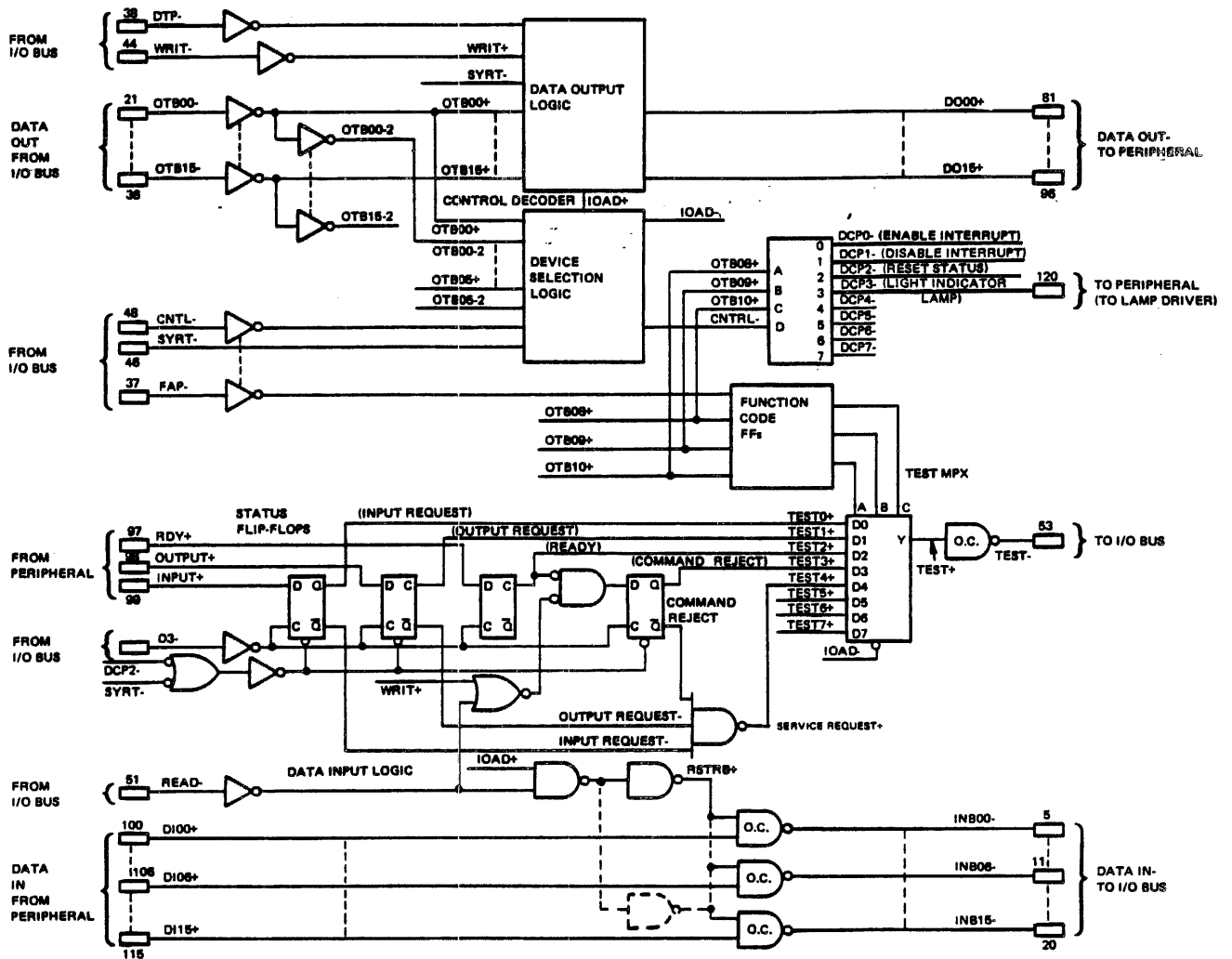
Both Controllers 1 and 2 (Figures 6-42 and 6-43, respectively), utilize the same type of buffered data output logic, device selection logic and function code flip-flops. Since the peripheral unit buffers data to be input to the CPU, both controllers utilize nonbuffered data input logic. An input or output transfer is initiated by a programmed I/O instruction.

Both controllers include four status flip-flops. The Input Request and Output Request flip-flops are set by the INPUT+ and OUTPUT+ pulses, respectively, from the peripheral unit. The Ready flip-flop is set or reset depending on the level of the RDY+ line from the peripheral unit. The Command Reject flip-flop is set is an input or output operation begins (WRIT+ or READ+ go true) while the Ready flip-flop is reset. All four flip-flops are clocked by D3- from the CPU and all but Ready are reset by the reset status control line (DCP2-) or system reset (SYRT-).

Both controllers include a control decoder and a test multiplexer. On Controller 1, only a third and fourth control decoder outputs (DCP1- and DCP3- are used. DCP2- resets all of the status flip-flops except Ready; DCP3- is available to the peripheral unit (via pin 120) to light the indicator lamp on the Console Panel. On Controller 2, two additional control outputs are used (DCP0- and DCP1-); DCP0- enables interrupts while DCP1- disables interrupts in the controller. Control outputs are activated as the result of a programmed I/O control instruction.

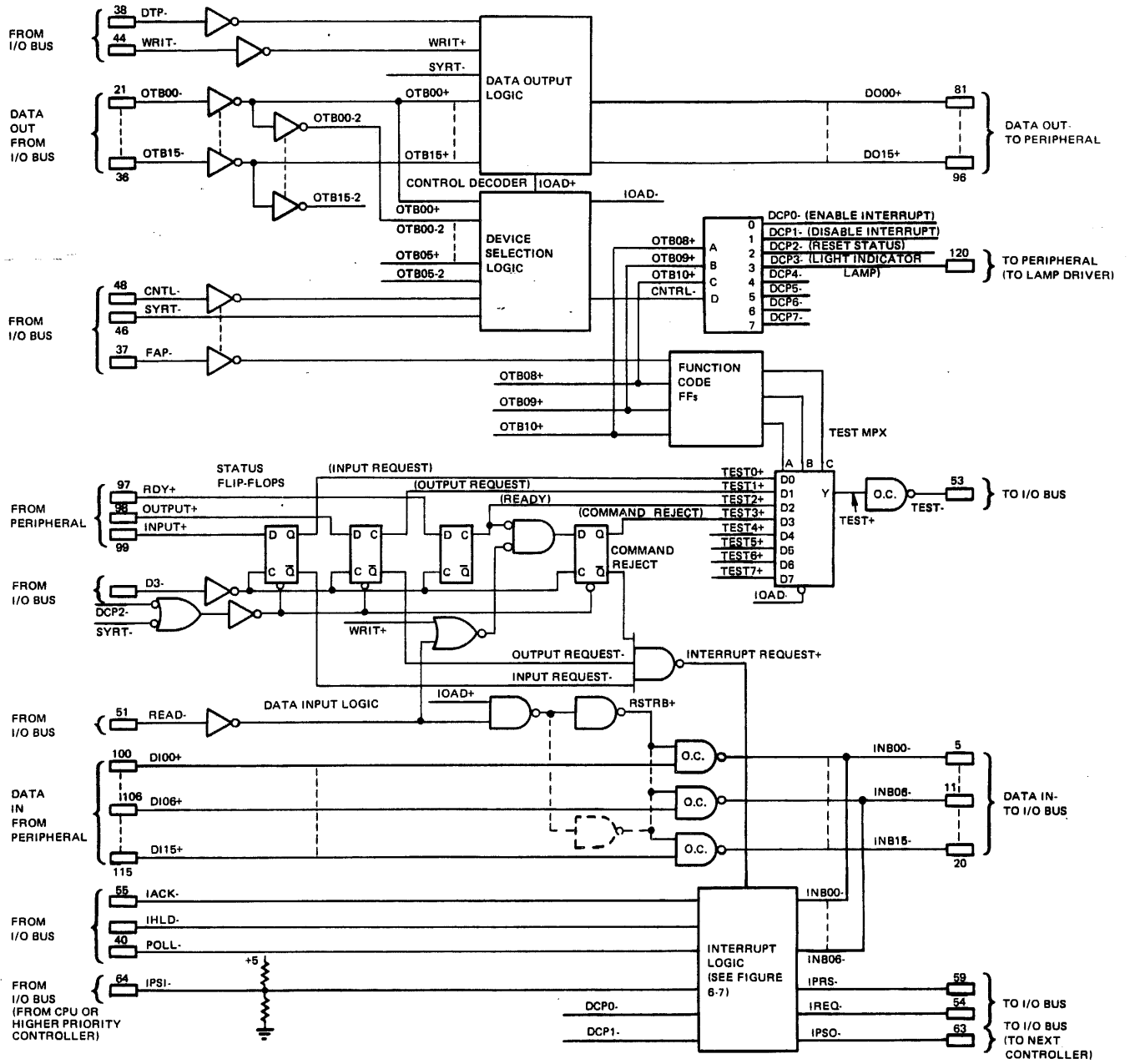
Both controllers allow the CPU to sample the state of the Input Request, Output Request, Ready, and Command Reject status flip-flops via the test multiplexer (lines TEST0+, TEST1+, TEST2+, and TEST3+, respectively). Controller 1 also provides an additional test input (TEST4+) which is in the logical OR of Input Request, Output Request and Command Reject; TEST4+ indicates that the peripheral unit requires service. This particular test input is not required on Controller 2 since Input Request, Output Request or Command Reject will activate the INTERRUPT REQUEST+ line. The CPU samples a particular test input as the result of a programmed I/O test instruction.

88A00508A-E



466-7-34

Figure 6-42. Example Controller 1



466-7-35

Figure 6-43. Example Controller 2

Only Controller 2 provides interrupt capabilities. The interrupt logic is as shown in Figure 6-30. The INTERRUPT REQUEST+ line is activated whenever the Input Request, Output Request, or Command Reject flip-flops are set.

6.11.2 EXAMPLE CONTROLLERS 3 AND 4

Assume that a user wishes to design a controller that provides an interface between a simple CRT device and the GA-16/220. This hypothetical CRT device is a block-oriented output device; that is, the CPU periodically refreshes the data on the CRT screen by outputting a block of data. Thus, Direct Memory Access (DMA) seems most appropriate. In addition to the 16-bit parallel data output path to the peripheral unit, there is an ACCEPT DATA+ line that notifies the peripheral unit when an output data word is available, and a block completed line (BCF+) that notifies the peripheral unit when the last word in a block of data has been transferred. The peripheral unit returns a DATA ACCEPTED- pulse after each data word is accepted. In addition, there is a ready line (RDY+) from the CRT. RDY+ goes true when the first word in a block of data is received and remains false until the block completed signal is received; RDY+ is false when the CRT is powered off. Table 6-25 summarizes the interface signals between the hypothetical peripheral unit and its controller.

Table 6-25. Interface Signals for Example Controllers 3 and 4

Mnemonic	Source	Description
D000+ to D015+	Controller	Data output lines to peripheral unit.
ACCEPT DATA+	Controller	Indicates that output data is available.
BCF+	Controller	Indicates that the last word in a block of data has been transferred.
DATA ACCEPTED-	Peripheral Unit	Indicates that the data word has been accepted.
RDY+	Peripheral Unit	Indicates that the CRT is powered on and not involved in a data output operation.

6.11.3 EXAMPLE CONTROLLER 3

Example Controller 3 (Figure 6-44) provides DMA capabilities but does not use the Data Channel Module. The test and control instruction logic is the same as shown in Figures 6-26 and 6-27 with the exception of the device selection logic. For test or control instructions, the controller is selected by a device select code of either X'16' or X'17' (e.g., CTRL 2,X'16' and CTRL 2,X'17' are equivalent instructions). There are two device select codes to allow loading of the channel address (CAR) and block count (SCR) parameters. A programmed output command directed to X'16' (e.g., DTOR R,X'16') enables loading of the current address register (CAR), whereas a programmed output command directed to X'17' (e.g., DTOR R,X'17') enables loading of the block count register (SCR). Consequently, a test or control instruction directed to either X'16' or X'17' is accepted by the controller (i.e., IOADD+ goes true for either device select code).

There are four control inputs (DCP0-, DCP1-, DCP2-, and DCP3-). DCP0- enables interrupts from the controller; DCP1- disables interrupts. DCP1- resets the Operation Complete and Command Reject status flip-flops and the BCF+ flip-flop; DCP3- initiates the DMA output operation. Control outputs are activated as the result of a programmed I/O control instruction.

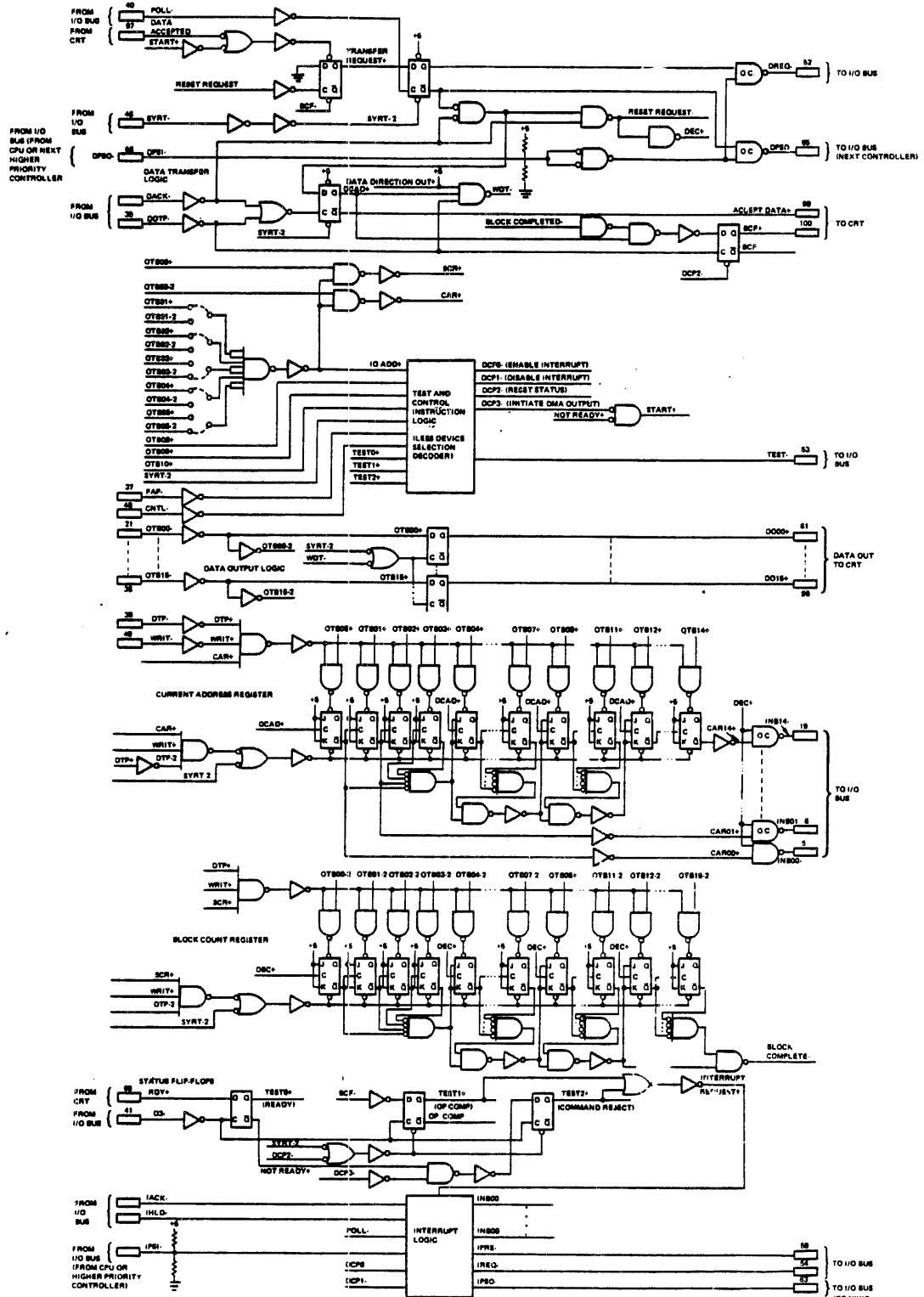
The CPU can sample the state of the Ready, Operation Complete, and Command Reject status flip-flops via the test multiplexer (lines TEST0+, TEST1+, and TEST2+, respectively). The CPU samples a particular test input as the result of a programmed I/O test instruction.

DMA operations require that a buffer address and block count be stored prior to initiating the first DMA transfer. The actual buffer address (*not the buffer address minus one as is done with the Data Channel Module*) is stored in the Channel Address Register (CAR) as the result of a programmed output instruction with a device code X'16' (e.g., DTOR R,X'16'). The CAR is cleared on the positive-going edge of the WRIT+ signal that precedes its being loaded. The block count is stored in the block count register (SCR) as the result of a programmed output instruction with a device code of X'17' (e.g., DTOR R,X'17'). The SCR is also cleared on the positive-going edge of WRIT+. Before each DMA transfer, the DEC+ signal is generated; DEC+ decrements the block count (i.e., increments the one's complement of the block count). The buffer address is gated onto the IN-Bus (lines INB00- to INB14-) prior to each transfer. The Channel Address Register (CAR) is then incremented by DCAD+. When the block count equals zero (i.e., when SCR contains all ones), BLOCK COMPLETE- is generated.

Controller 3 includes three status flip-flops. The Ready flip-flop is set or reset depending on the level of the RDY+ line from the peripheral unit. The Operation Complete flip-flop is set by the BCF- signal. The Command Reject flip-flop is set if the program attempts to initiate a DMA operation (i.e., if DCP3- is generated) while the Ready flip-flop is reset (i.e., NOT READY+ is true). All three flip-flops are clocked by D3- from the CPU, and all but Ready are reset by the reset status control line (DCP2-) or System Reset (SYRT-).

The DMA transfer logic is essentially the same as shown in Figure 6-31. After the program has loaded the Current Address Register (CAR) and the block count register (SCR), the program would activate the control pulse DCP3-, which would initially set

88A00508A-E



466-7-36

Figure 6-44. Example Controller 3

the TRANSFER REQUEST+ flip-flop (unless NOT READY+ were true). TRANSFER REQUEST+ generates DREQ- and sends it to the CPU unless a higher priority controller is also requesting a DMA transfer. When the controller is acknowledged (i.e., when DREQ has been generated and DACK is received from the CPU), the DEC+ signal is generated (DEC+ decrements the SCR), the TRANSFER REQUEST+ flip-flop is reset, the Channel Address (CAR00+ to CAR14+) is gated onto the IN-Bus, and the DCAD+ flip-flop is set (DCAD+ increments the current address register for the next transfer). The data is output to the CRT unit. When DCAD+ is reset on the positive-going edge of the transfer pulse (DDTP-) from the CPU, the ACCEPT DATA+ signal is generated and sent to the peripheral unit. After the CRT accepts the data, the CRT returns a DATA ACCEPTED+ signal to the controller; DATA ACCEPTED+ sets the TRANSFER REQUEST+ flip-flop and another DMA cycle follows. This process is repeated until the entire block of data has been transferred (i.e., until BCF- is generated).

When the Operation Complete or Command Reject flip-flop is set, an interrupt is requested (unless interrupts have been disabled).

6.11.4 EXAMPLE CONTROLLER 4

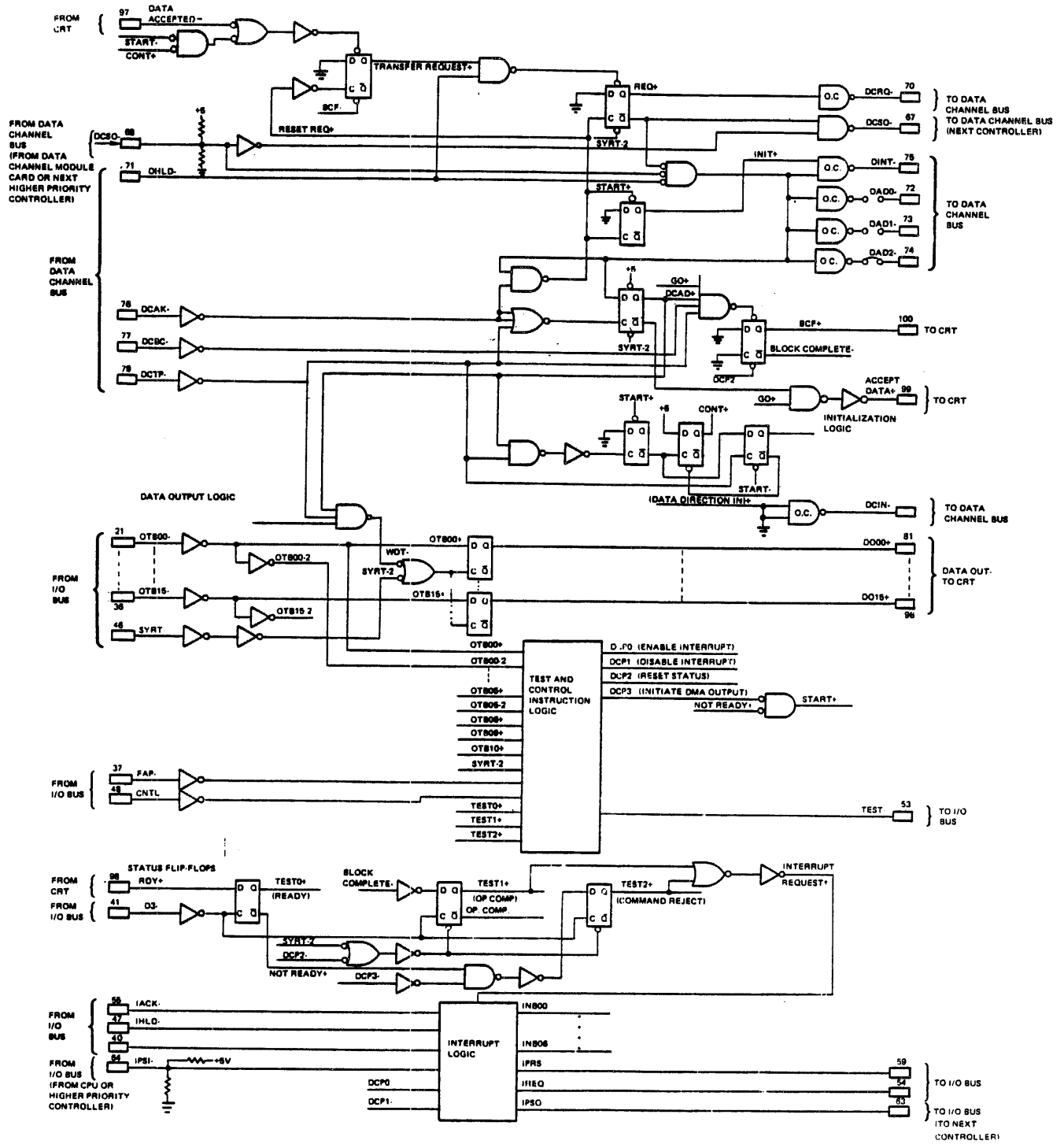
Example Controller 4 (Figure 6-45) is intended as an interface for the same CRT unit as was Controller 3. Controller 4, however, was designed for use with the Data Channel Module. Consequently, Controller 4 includes neither a Current Address Register (CAR) nor a block count register (SCR); these registers are provided by the Data Channel Module. The test and control instruction logic, the status flip-flops, the data output logic and the interrupt logic, however, are essentially the same as in Controller 3.

While the DMA transfer logic on Controller 3 communicates directly with the CPU in requesting a DMA cycle, the Data Channel transfer logic on Controller 4 issues its request to the Data Channel Module. The module resolves priority among the various controllers (up to 8) associated with it, then issues a DMA cycle request to the CPU on behalf of the highest priority controller that is requesting service.

Recall that with Controller 3, the program has to execute output instructions to load the Current Address Register (CAR) and the block count register (SCR) before initiating the first DMA transfer via a programmed I/O control instruction. This, however, is not necessary with Controller 4. When the program executes the programmed I/O Control instruction to initiate the data channel operation (i.e., when DCP3- is activated) the TRANSFER REQUEST+ is set and in turn sets the REQ+ flip-flop, which issues a request signal to the Data Channel Module (DCRQ-). At the same time that TRANSFER REQUEST+ is set, the INIT+ flip-flop is set and the initialization logic is returned to the start state. When the Data Channel Module finishes servicing all requests from higher priority controllers, the data channel address bits (DADO- to DAD2-) and the DINT- signal are gated onto the Data Channel Bus. The Data Channel Module performs two consecutive DMS to retrieve the CAR and SCR values from the memory locations dedicated to the controller in question.

The Data Channel Module energizes DCTP- for each of these cycles, as shown in Figures 6-39 and 6-40. The controller ignores the data on the OUT-Bus during both of these cycles. Controller 4 does not support chaining. The initialization logic counts both DCTP- pulses and then issues GO+. After the first DCTP, however, the CONT+ pulse is

88A00508A-E



466-7-37

Figure 6-45. Example Controller 4

88A00508A-E

generated; CONT+ again sets the TRANSFER REQUEST+ flip-flop. After the CAR and SCR values have been fetched, TRANSFER REQUEST+ causes a request to be issued for the transfer of the first data output word. GO+, which remains true, allows the data to be transferred to the peripheral unit along with the ACCEPT DATA+ signal. When the peripheral unit accepts the data word, the DATA ACCEPTED- pulse is returned to the controller; DATA ACCEPTED- again sets TRANSFER REQUEST+ and another data channel transfer cycle follows. This continues until all data words in the block have been transferred and DCBC- is received from the Data Channel Module. DCBC- enables setting of the block completed flip-flop (BCF+), which, in turn, sets the Operation Complete flip-flop.

An interrupt is requested when the Operation Complete or Command Reject flip-flop is set (unless interrupts have been disabled).

installation procedures **7**

7.1 GENERAL

The material presented in this section is intended to aid in the site preparation for, and physical installation of, the GA-16/110 or the GA-16/220 processor and supporting equipment.

The data supplied within this section will be somewhat generalized and is meant to supplement the Systems Integration Package (SIP). The SIP contains the logic and assembly drawings, physical configuration, cabling data, module location chart, programming and systems specifications for each individual system. A SIP is shipped with each non-OEM system and should be consulted for specific installation data.

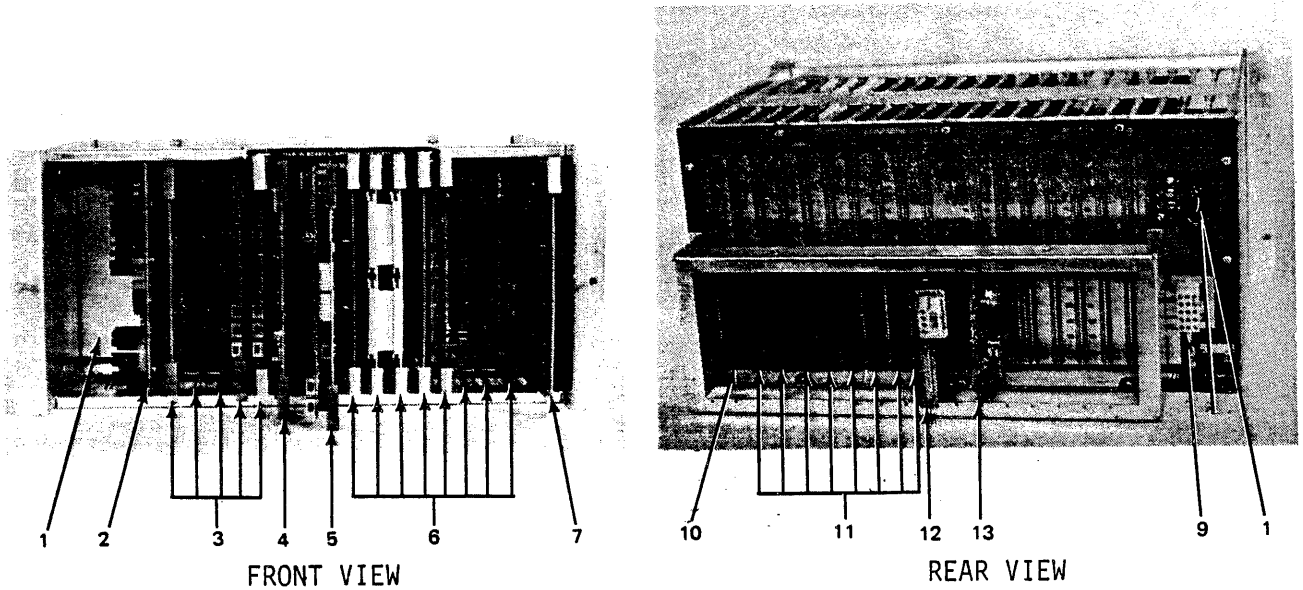
Section 7.2 contains a physical description of the CPU and power supply. Section 7.3 contains environmental and power requirements.

7.2 PHYSICAL DESCRIPTION

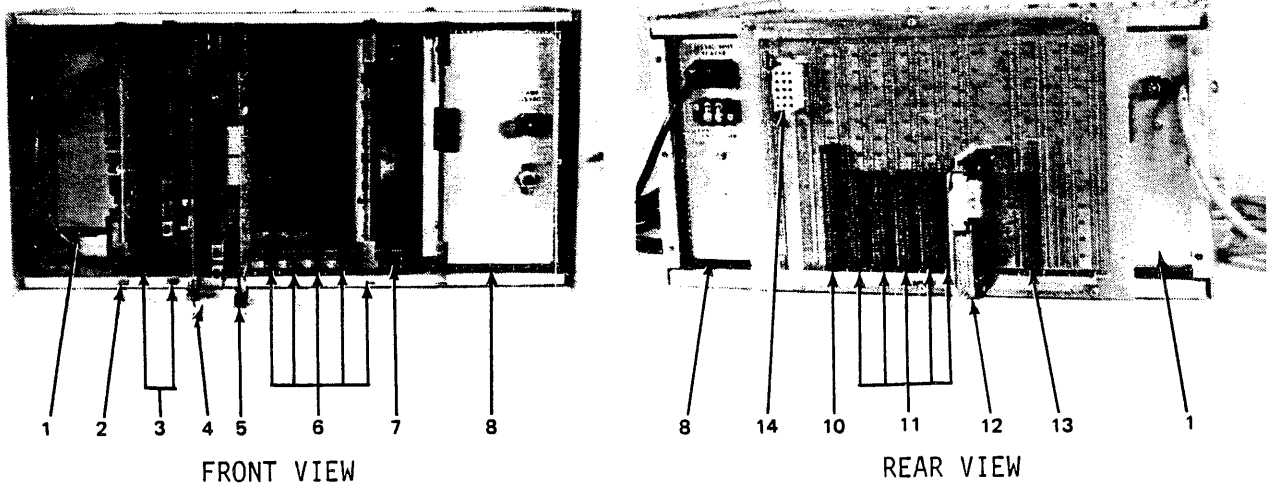
The GA-16/220 is composed of two basic CPU boards housed in a single cabinet measuring 19.0" x 8.69" x 21.25". There are two configurations, the Compact Chassis and the Jumbo Chassis.

The Compact Chassis accommodates:

- The main power supply.
- The Channel Interface Driver (CID) board, when the optional I/O expansion is installed.
- The Multiple High Speed Data Channel (MHSDC-8) controller for DMA capabilities (optional).
- Up to five I/O controllers (four I/O controllers if the MHSDC is installed).
- The CPU-2 module, with or without the Systems Console Interface (SCI) module. The SCI is piggyback mounted on the CPU-2 module.
- The CPU-1 module.
- A combination of 4K, 8K, and 16K memory boards up to a maximum of three (two maximum if the optional Memory Parity Protect option is installed); or a 32K/64K memory board.
- An optional Memory Parity Protect (MPP) module (occupies one memory slot).



JUMBO CHASSIS



COMPACT CHASSIS

Note: Paddleboard retainer cage not shown

- | | |
|---|--|
| <ol style="list-style-type: none"> 1. BATTERY BACK-UP POWER SUPPLY AND BATTERY PACK 2. MEMORY OR BATTERY BACK-UP REGULATOR BOARD 3. MEMORY (ONE SLOT MAY BE USED FOR MPP) 4. CPU-1 MODULE (GA-16/220), MEMORY (GA-16/110) 5. CPU-2 MODULE (GA-16/220), CPU-1 MODULE (GA-16/110) 6. I/O CONTROLLERS 7. I/O EXPANSION CABLE DRIVER (CID) | <ol style="list-style-type: none"> 8. COMPACT MAIN POWER SUPPLY 9. JUMBO POWER SUPPLY CONNECTOR 10. I/O EXPANSION CABLE CONNECTOR 11. I/O DEVICE CABLE CONNECTORS 12. TTY/CRT ADAPTER CONNECTOR 13. MEMORY SERVICE MODULE CONNECTOR 14. COMPACT CHASSIS EXTERNAL POWER SUPPLY CONNECTOR |
|---|--|

Figure 7-1. GA-16/220 CPU Configuration

- An optional error correction module if a 32K/64K memory board is used.
- An optional auxiliary power supply.
- The Memory Service Module (board) for dynamic RAM memories if no battery backup is installed.

The Jumbo Chassis accommodates the above mentioned components as well as:

- Internal slots for up to eight I/O controllers (with no MHSDC installed).
- Space for up to six memory modules (providing no MPP or Battery Backup control module is installed, as each option takes one memory slot).

For jack numbers of the above mentioned locations reference Table 3-1 and 3-2, Section 3.1.

Figure 7-2 is the installation drawing for the Model 1615-I/O Power Supply. Each power supply produces all voltages necessary to drive one I/O chassis.

Figure 7-3 is the installation drawing for the Model 1615-I/O enclosure having a capacity of up to 18 peripheral controller cards. Section 7.2.1 describes the I/O enclosure and connectors.

7.2.1 DESCRIPTION OF THE EXTERNAL I/O ENCLOSURE

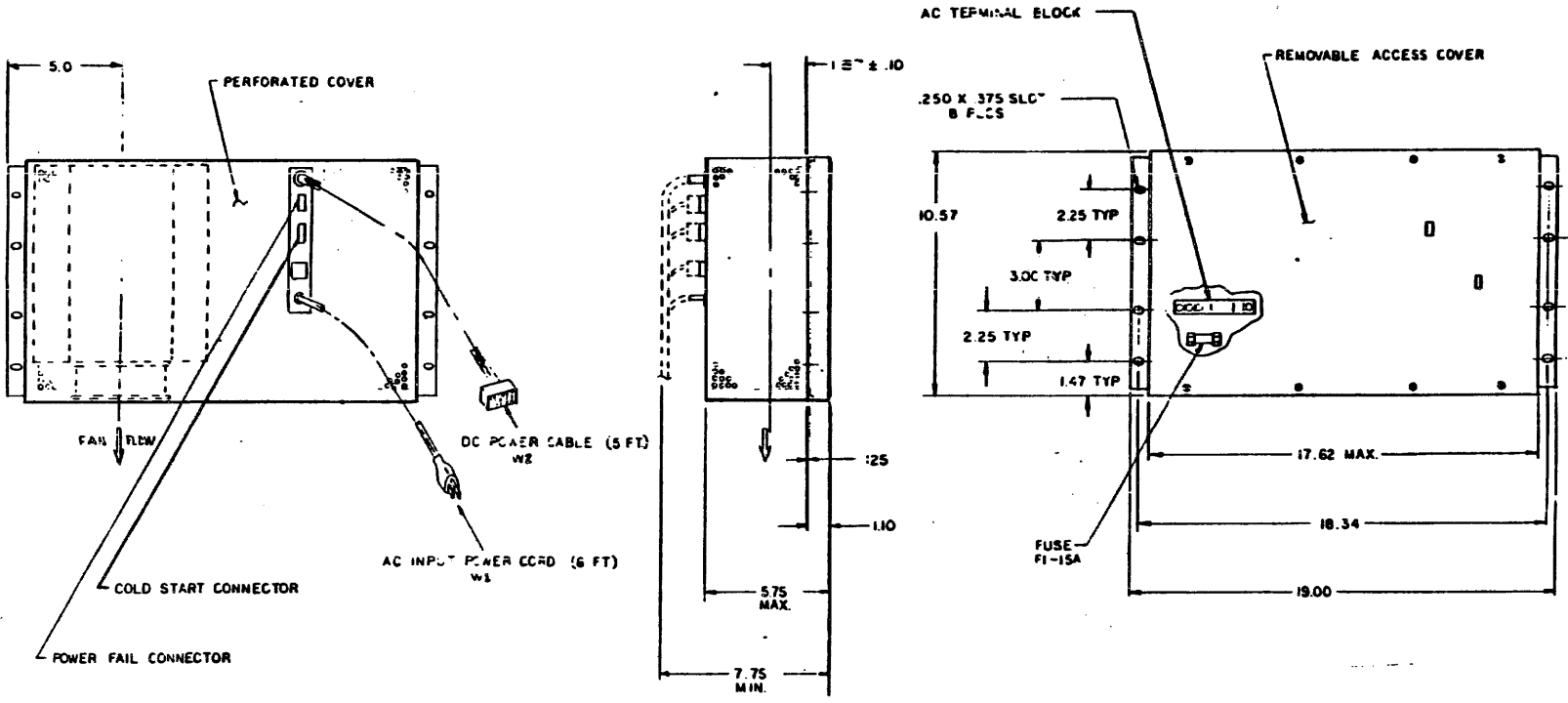
The I/O enclosure has nineteen card slot positions. One slot will be occupied by a Cable Interface Translator (CIT) module. The CIT module is primarily composed of Drivers and Receivers. Another slot will be occupied by a Multiple High-Speed Data Channel Controller (MHSDC) if one or more of the I/O devices operates on a data channel. This leaves seventeen or eighteen slots for device controllers. For systems requiring additional slots, more enclosures can be added. A rear view is shown in Figure 7-4.

In the I/O enclosure, controller boards plug into the 140-pin edge connectors on the front surface of the MIB board. Corresponding 56-pin edge connectors on the rear of the MIB board accept paddle boards on the peripheral cables. The I/O Bus and the Data Channel Bus lines are etched on the MIB. The I/O Bus cable enters the I/O enclosure via one 72-pin connector (and leaves the enclosure, if an additional enclosure is required, via another 72-pin connector) which is provided on the rear surface of the MIB.

Figure 7-5 illustrates the relationship between pins on the 140-pin connector that accepts the controller board and pins on the 56-pin connector that accepts the paddle board which is connected via the cable to the associate peripheral unit.

Figure 7-6 illustrates the relationship between pins on the 140-pin edge connector that accepts the Cable Interface Translator card (CIT-16) and the 72-pin edge connector that accepts the paddle board for the External I/O Bus cable. Figure 7-7 provides a summary of I/O signal connections in the external I/O enclosure.

Figure 7-2. Installation Drawing I/O Power Supply Model 1615



88A00508A-E

243-8-5

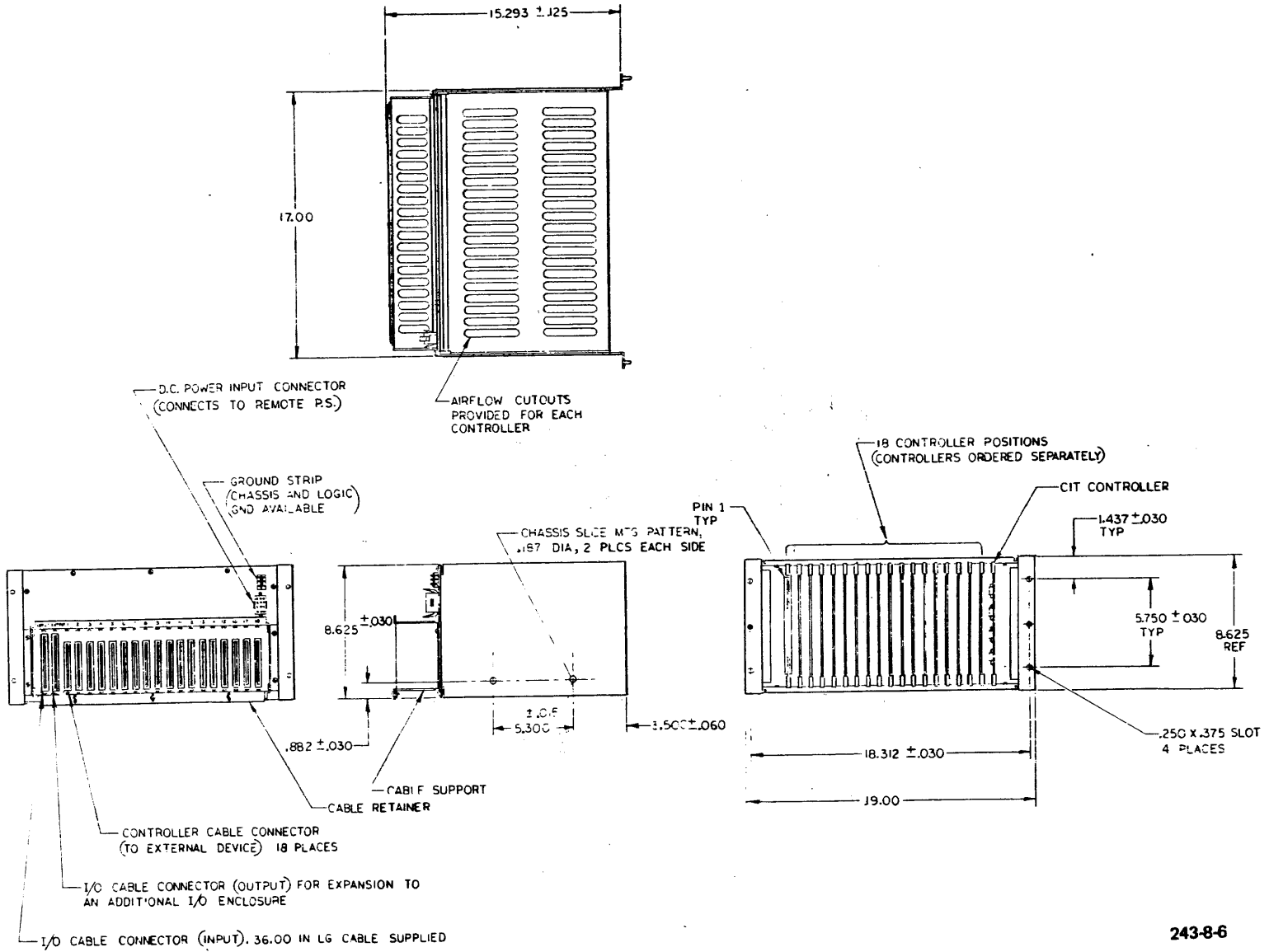
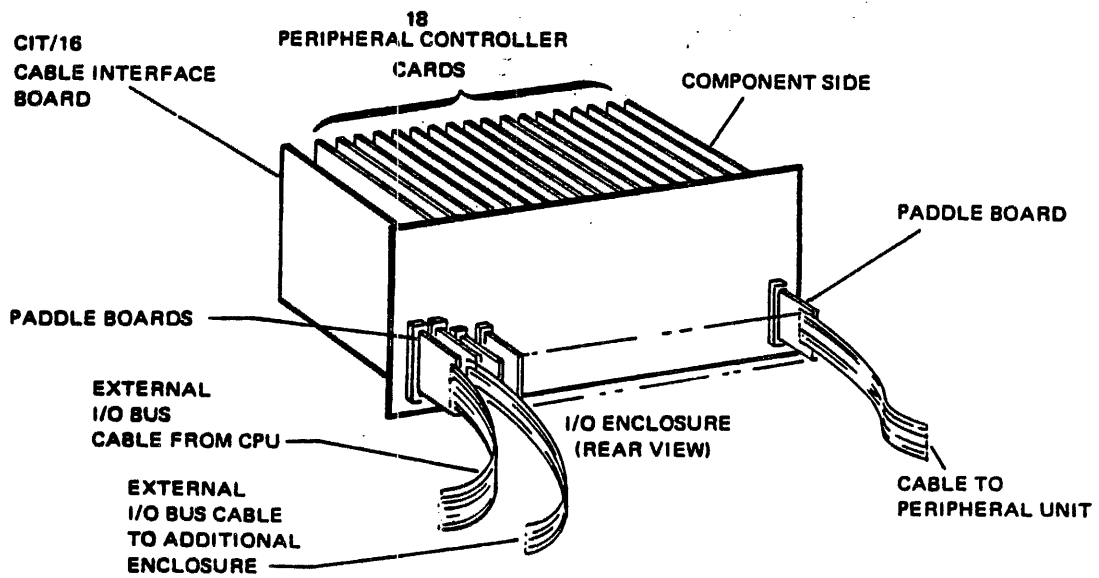


Figure 7-3. Installation Drawing, I/O Enclosure Model 1615-0001



507-7-1

Figure 7-4. Circuit Board Layout

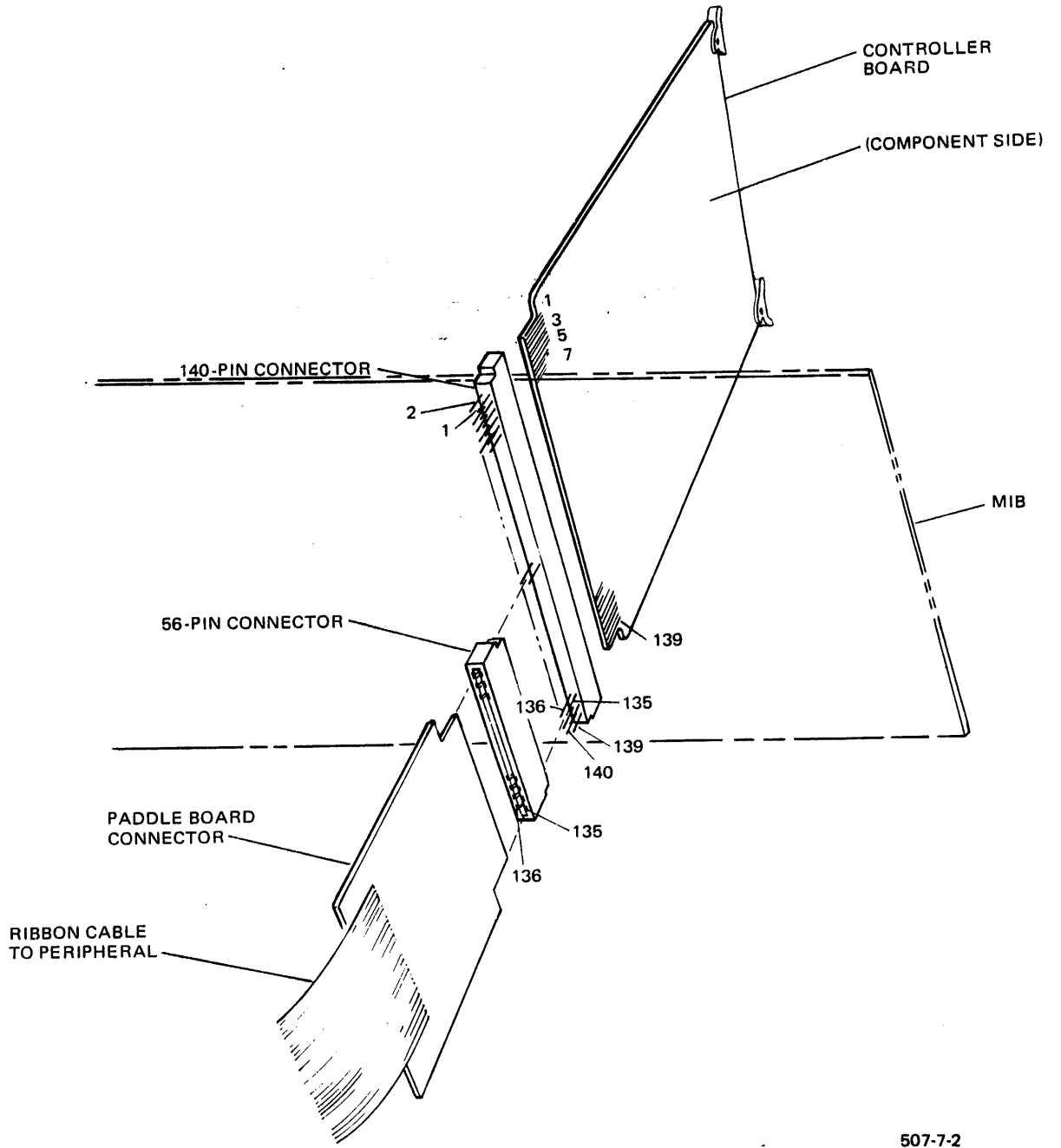
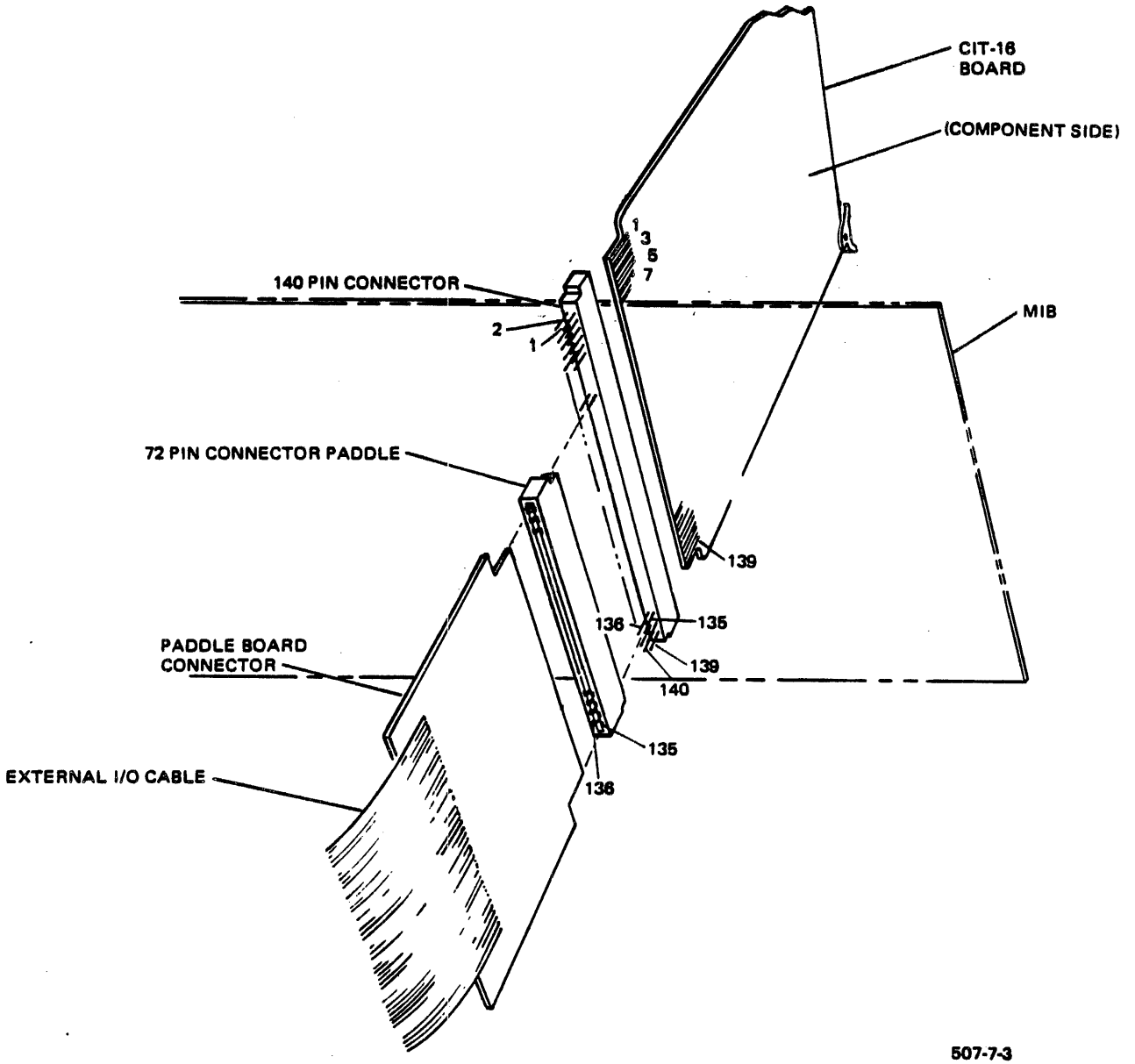
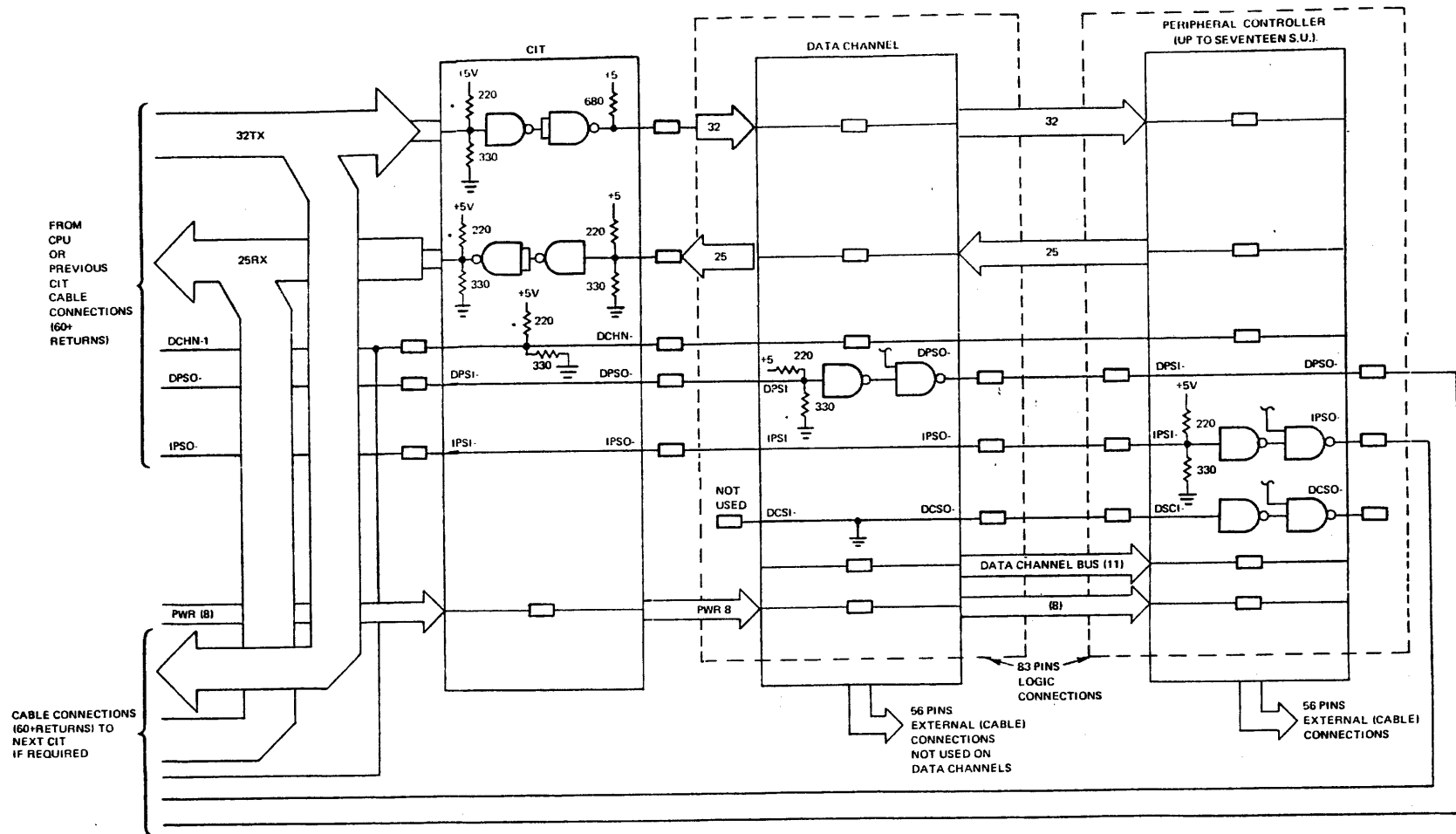


Figure 7-5. Pin Relationship Between Controller Card and Paddle Board



507-73

Figure 7-6. Pin Relationship Between CIT-16 Board and External I/O Bus Cable



FROM CPU OR PREVIOUS CIT CABLE CONNECTIONS (60+ RETURNS)

CABLE CONNECTIONS (60+ RETURNS) TO NEXT CIT IF REQUIRED

NOTE: *CABLE TERMINATIONS ARE EMPLOYED IN THE LAST CIT CARD ONLY. DCHN-1 IS TERMINATED AT FIRST AND LAST CIT.

88A00508A-E

Figure 7-7. I/O Signal Connections, External I/O Enclosure

7.3 SITE REQUIREMENTS

This section contains all environmental specifications and power requirements pertaining to the GA-16/220 processor, I/O system and associated power supplies.

7.3.1 ENVIRONMENTAL CONDITIONS

All models of the GA-16/220 are designed to operate within the temperature range 0°C thru 50°C (32°F thru 122°F) at 10-90% relative humidity without condensation.

All models of the GA-16/220 may be stored in environments within the temperature -20°C thru +70°C (-68°F thru +158°F).

The amount of cooling needed in the GA-16/220 systems operating environment equates to 3.4 BUT's per hour per watt of power specified.

The GA-16/110/220 and the I/O expansion chassis are designed for convective cooling if the following conditions are met:

1. There is adequate free space immediately above and below the chassis for unrestricted air circulation.
2. There is a general air circulation within the ambient environment such as that produced by a main exhaust fan in a rack.

NOTE

Reliability will always be greater if equipment is operated well below its maximum temperature ratings. GA strongly recommends the model 1995-1101 fan assembly for all GA-16/110/220 chassis and for I/O expansion chassis if heavily loaded with controllers.

7.3.2 POWER REQUIREMENTS

The power supplies for the GA-16/110/220 will operate under the ranges of voltages and frequencies following (power requirements approximate for maximum power supply loading).

1. External supply which may be used with compact or jumbo chassis:
90 - 250 volts (via transformer taps), single phase, 47 - 63 Hz , 500 watts.
2. Internal supply for compact chassis:
115 volt \pm 10%, single phase, 47 - 63 Hz, 230 watts.

Commonly supplied domestic AC power sources are:

- 115 volt, 60 \pm 3 Hz.
- 230 volt, 60 \pm 3 Hz.

88A00508A-E

Commonly supplied international AC power sources are:

100 volt, 50 \pm 3 Hz

200 volt, 50 \pm 3 Hz

220 volt, 50 \pm 3 Hz

Total power requirements are determined by GA-16/110/220 requirements plus additional requirements imposed by other equipment such as cooling fans, I/O expansion chassis, TTY or CRT, disk driver, etc.

GA-16/110/220 instruction summary **A**

This appendix contains tables which summarize the instructions for the GA-16/110/220. The tables provide a quick reference to all instructions and provide look-up tables of both CAP-16 assembly code and the corresponding binary and hexadecimal instruction format.

Table A-1 lists the basic types of instructions, the general format of the instructions, and provides a key reference number correlating to Table A-2.

Table A-2 is an alphabetical list of instructions, the corresponding hexadecimal representations, and a symbolic representation of the operations which result.

Table A-3 provides additional detail concerning memory reference instructions (Reference 1 in Tables A-1 and A-2.) This table shows addressing modes, range of displacements (or effective addresses), CAP-16 Assembler codes, and corresponding binary and hexadecimal instruction formats.

Table A-4 provides additional detail concerning memory referencing with indexing instructions (references 2, 3, and 4 of Tables A-1 and A-2).

Table A-5 provides detailed instruction formats for standard I/O. This includes the built-in teletype controller, internal mask words, console switch register, and display of data via console data display lights. Refer to Appendix B for the ASCII character set used for teletype.

Table A-1. General Instructions

INSTRUCTION TYPE GENERAL FORMAT CAP-16	TABLE A-2 REFERENCE	GENERAL FORMAT																																	
Memory Reference op-code [*]address,[m ₁]	1	<table border="1"> <tr> <td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td colspan="10">OP CODE</td> <td>m</td><td>*</td><td colspan="5">DISP</td> </tr> </table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	OP CODE										m	*	DISP				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																				
OP CODE										m	*	DISP																							
Memory Reference with Indexing op-code R,[*]address[,i][,m ₂]	2	<table border="1"> <tr> <td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td colspan="10">OP CODE</td> <td>m</td><td>*</td><td>i</td><td>R</td><td colspan="3">DISP</td> </tr> </table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	OP CODE										m	*	i	R	DISP		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																				
OP CODE										m	*	i	R	DISP																					
op-code [*]address[,i][,m ₂]	3	<table border="1"> <tr> <td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td colspan="10">OP CODE</td> <td>m</td><td>*</td><td>i</td><td>EXT</td><td colspan="3">DISP</td> </tr> </table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	OP CODE										m	*	i	EXT	DISP		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																				
OP CODE										m	*	i	EXT	DISP																					
op-code b,address[,i][,m ₂]	4	<table border="1"> <tr> <td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td colspan="10">OP CODE</td> <td>m</td><td>*</td><td>i</td><td>b</td><td colspan="3">DISP</td> </tr> </table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	OP CODE										m	*	i	b	DISP		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																				
OP CODE										m	*	i	b	DISP																					
	2 WORD FORMAT	<table border="1"> <tr> <td colspan="10">ONE OF ABOVE</td> <td>1</td><td>1</td><td>1</td><td>1</td><td>1</td> </tr> <tr> <td colspan="16">ADDRESS</td> </tr> </table>	ONE OF ABOVE										1	1	1	1	1	ADDRESS																	
ONE OF ABOVE										1	1	1	1	1																					
ADDRESS																																			
Skip (Extended Displacement) op-code address	5	<table border="1"> <tr> <td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td colspan="5">OP CODE</td> <td>S</td> <td colspan="5">CODE</td> <td colspan="5">DISP</td> </tr> </table>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	OP CODE					S	CODE					DISP					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																				
OP CODE					S	CODE					DISP																								
Register Operate op-code Rd,Rs	6	<table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td colspan="3">Rs</td><td colspan="3">Rd</td><td>1</td><td colspan="3">OPERATION</td> </tr> </table>	0	0	0	0	1	Rs			Rd			1	OPERATION																				
0	0	0	0	1	Rs			Rd			1	OPERATION																							
Register Operate Compare op-code Rd,Rs	7	<table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td colspan="3">Rs</td><td colspan="3">Rd</td><td>0</td><td colspan="3">OPERATION</td> </tr> </table>	0	0	0	0	1	Rs			Rd			0	OPERATION																				
0	0	0	0	1	Rs			Rd			0	OPERATION																							
Register Operate Literal op-code R,value	8	<table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td colspan="3">R</td><td>1</td><td colspan="3">OPERATION</td> </tr> <tr> <td colspan="16">VALUE</td> </tr> </table>	0	0	0	0	0	0	0	1	R			1	OPERATION			VALUE																	
0	0	0	0	0	0	0	1	R			1	OPERATION																							
VALUE																																			
Register Operate Literal Compare op-code R,value	9	<table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td colspan="3">R</td><td>0</td><td colspan="3">OPERATION</td> </tr> <tr> <td colspan="16">VALUE</td> </tr> </table>	0	0	0	0	0	0	0	1	R			0	OPERATION			VALUE																	
0	0	0	0	0	0	0	1	R			0	OPERATION																							
VALUE																																			
Subroutine Return Indirect op-code address	10	<table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td> </tr> <tr> <td colspan="16">EFFECTIVE ADDRESS</td> </tr> </table>	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	EFFECTIVE ADDRESS																
0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0																				
EFFECTIVE ADDRESS																																			
Register Change op-code R	11	<table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>X</td><td>X</td><td colspan="3">R</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td> </tr> </table>	0	0	0	0	0	1	X	X	R			X	X	X	X	X																	
0	0	0	0	0	1	X	X	R			X	X	X	X	X																				
Shift Left op-code R	12	<table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>←</td><td colspan="3">R</td><td>0</td><td>0</td><td>←</td><td>←</td><td>←</td> </tr> </table>	0	0	0	0	0	1	1	←	R			0	0	←	←	←																	
0	0	0	0	0	1	1	←	R			0	0	←	←	←																				
Shift Right op-code R,count	13	<table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>→</td><td colspan="3">R</td><td>→</td><td colspan="3">COUNT-1</td> </tr> </table>	0	0	0	0	0	0	1	→	R			→	COUNT-1																				
0	0	0	0	0	0	1	→	R			→	COUNT-1																							
Control op-code	14	<table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td colspan="12">DECODE HEX PATTERNS IN TABLE A-2</td> </tr> </table>	0	0	0	0	DECODE HEX PATTERNS IN TABLE A-2																												
0	0	0	0	DECODE HEX PATTERNS IN TABLE A-2																															
Input/Output op-code R,dev-addr	15	<table border="1"> <tr> <td>IN</td> <td>0</td><td>0</td><td>0</td><td>1</td> <td>M</td><td colspan="3">R</td><td>1</td><td>0</td><td colspan="4">DEV ADDR</td> </tr> <tr> <td>OUT</td> <td>0</td><td>0</td><td>0</td><td>1</td> <td>M</td><td colspan="3">R</td><td>0</td><td>1</td><td colspan="4">DEV ADDR</td> </tr> </table>	IN	0	0	0	1	M	R			1	0	DEV ADDR				OUT	0	0	0	1	M	R			0	1	DEV ADDR						
IN	0	0	0	1	M	R			1	0	DEV ADDR																								
OUT	0	0	0	1	M	R			0	1	DEV ADDR																								
Control/Test op-code Fun,dev-addr	16	<table border="1"> <tr> <td>CTRL</td> <td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td colspan="3">FUN</td><td>0</td><td>0</td><td colspan="4">DEV ADDR</td> </tr> <tr> <td>TEST</td> <td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td colspan="3">FUN</td><td>1</td><td>1</td><td colspan="4">DEV ADDR</td> </tr> </table>	CTRL	0	0	0	1	0	FUN			0	0	DEV ADDR				TEST	0	0	0	1	0	FUN			1	1	DEV ADDR						
CTRL	0	0	0	1	0	FUN			0	0	DEV ADDR																								
TEST	0	0	0	1	0	FUN			1	1	DEV ADDR																								
Read Console Switches op-code R,X'3E'	17	<table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>1</td><td>M</td><td colspan="3">R</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td> </tr> </table>	0	0	0	1	M	R			1	0	1	1	1	1	1	0																	
0	0	0	1	M	R			1	0	1	1	1	1	1	0																				
Multiply/Divide op-code [count]	18	<table border="1"> <tr> <td>MPY</td> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td colspan="3">COUNT</td> </tr> <tr> <td>DIV</td> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td colspan="3">COUNT</td> </tr> </table>	MPY	0	0	0	0	0	0	0	0	1	0	0	0	COUNT			DIV	0	0	0	0	0	0	0	0	1	0	1	0	COUNT			
MPY	0	0	0	0	0	0	0	0	1	0	0	0	COUNT																						
DIV	0	0	0	0	0	0	0	0	1	0	1	0	COUNT																						

Notes to Table A-1

① MODE CODE IN CAP-16 STATEMENT

CAP-16	ADDRESSING	INSTRUCTION FORMAT
m_1	0 Program-relative	1 word M=0
m_1	1 Base-relative	1 word m=1
m_2	0 Absolute	1 word m=0 disp \leq 11110
m_2	1 Base-relative	1 word m=1 disp \leq 11110
m_2	2 Absolute	2 word m=0; disp=11111
m_2	3 Base-relative	2 word m=1; disp=11111

[*] indicates indirect addressing, causes *field in instruction to be set=1.

Table A-2. Summary of General Instructions

COMMAND	TABLE A-1 REF.	HEX CODE				OPERATION
ADD	6	0	8+Rs	2Rd+1	9	$R_s+R_d \rightarrow R_d; S_Z, S_P, S_0, S_L$
ADDC	7	0	8+Rs	2Rd	9	$R_s+R_d \rightarrow DBUS; S_Z, S_P, S_0, S_L$
ADDS	11	0	7	2R	8	$R+S_{Shift} \rightarrow R; S_Z, S_P, S_0, S_L$
ADDV	8	0	1	2R+1	9	$R+(P+1) \rightarrow R; S_Z, S_P, S_0, S_L$
ADDVC	9	0	1	2R	9	$R+(P+1) \rightarrow DBUS; S_Z, S_P, S_0, S_L$
AND	6	0	8+Rs	2Rd+1	7	$R_d \wedge R_s \rightarrow R_d; S_Z, S_P$
ANDC	7	0	8+Rs	2Rd	7	$R_d \wedge R_s \rightarrow DBUS; S_Z, S_P$
ANDV	8	0	1	2R+1	7	$R \wedge (P+1) \rightarrow R; S_Z, S_P$
ANDVC	9	0	1	2R	7	$R \wedge (P+1) \rightarrow DBUS; S_Z, S_P$
BMS	14	0	4	0	8	$0 \rightarrow S_F$ (Use Background registers)
CMR	2	E	X	$\begin{matrix} 2R \\ 2R+1 \end{matrix}$	X^4	$R-(EA) \rightarrow DBUS; S_Z, S_P, S_0, S_L$
CMPL	11	0	7	2R	0	Complement $R \rightarrow R; 0 \rightarrow S_L$ [R initially = 0] $1 \rightarrow S_L$ [R initially $\neq 0$]
CTRL	16	1	fun	dev-addr		Controller carries out function
DECM	3	F	X	$\begin{matrix} 4 \\ 5 \end{matrix}$	X^4	$(EA)-1 \rightarrow (EA); S_Z, S_P, S_L$
DECR	11	0	7	2R	2	$R-1 \rightarrow R; S_Z, S_P, S_L$
DIV	18	0	0	A	count	$BC \div A \rightarrow C$; Remainder $\rightarrow B; S_L$
DSPL	11	0	5	2R	4	(No observable function on GA-16/110/220)
DTIM	15	1	R	$80+\text{dev-addr}$		Controller $\rightarrow R$
DTIR	15	1	8+R	$80+\text{dev-addr}$		Controller $\rightarrow R$
DTOM	15	1	R	$40+\text{dev-addr}$		$*R \rightarrow$ Controller
DTOR	15	1	8+R	$40+\text{dev-addr}$		$R \rightarrow$ Controller
EXBY	11	0	6	2R	4	$R_{7-0} \leftrightarrow R_{15-8}$
EXIT	11	0	5	2R	2	$R_{14-0} \leftrightarrow P_{14-0} / R \rightarrow P$
FMS	14	0	4	0	C	$1 \rightarrow S_F$ (Use Foreground registers)
INCM	3	F	X	$\begin{matrix} 0 \\ 1 \end{matrix}$	X^4	$(EA)+1 \rightarrow (EA); S_Z, S_P, S_L$
INCR	11	0	7	2R	E	$R+1 \rightarrow R; S_Z, S_P, S_L$
INE	14	0	4	0	3	$1 \rightarrow ISE$

88A00508A-E

Table A-2. Summary of General Instructions (Cont'd.)

COMMAND	TABLE A-1 REF.	HEX CODE				OPERATION
INH	14	0	4	0	2	0→ISE
JMP	1	7	X	X	X	EA ₁₄₋₀ ^P ₁₄₋₀ / EA ₁₅₋₀ ^P ₁₅₋₀
JSR	1	6	X	X	X	P+1→E ₁₄₋₀ ; ISE→E ₁₅ ; EA→P ₁₄₋₀ ; 0→ISE/P+1→E; ISE+S _{ISE} ; EA+P; 0→ISE
LARS	3	F	X	$\frac{8}{9}$	X ⁴	(EA)···(EA+7)→A,X,Y,Z,B,C,D,E; (EA+8) ₁₅ ^S _{ISE} ; (EA+8) ₁₃₋₀ ^S ₁₃₋₀
LDA	1	4	X	X	X	(EA)→A
LDBY	2	8	X	$\frac{2R}{2R+1}$	X ⁴	(EA) ₁₅₋₈ ^R ₇₋₀ [i even] (EA) ₇₋₀ ^R ₇₋₀ [i odd]
LDR	2	C	X	$\frac{2R}{2R+1}$	X ⁴	(EA)→R
LDV	3	0	1	2R+1	5	(P+1)→R; S _Z , S _P
LKR	14	0	4	2	0	0→S _L
LKS	14	0	4	3	0	1→S _L
MPY	18	0	0	8	count	A+C+B,C; S _L
OR	6	0	8+Rs	2Rd+1	D	RdVRS→Rd; S _Z , S _P
ORC	7	0	8+Rs	2Rd	D	RdVRS→DBUS; S _Z , S _P
ORV	8	0	1	2R+1	D	RV(P+1)→R; S _Z , S _P
ORVC	9	0	1	2R	D	RV(P+1)→DBUS; S _Z , S _P
PMA	14	0	4	4	0	0→PMA; Reset OMA timer, (must be issued every 200 ms±60ms)
RBIT	4	3	X	$\frac{2b}{2b+1}$	X ⁴	(EA) _{b+8} ^S _Z ; 0+(EA) _{b+8} [i even] (EA) _b ^S _Z ; 0+(EA) _b [i odd]
RCSM	17	1	R	B	E	CSW → *R
RCSR	17	1	8+R	B	E	CSW → R
RCSW	11	0	6	2R+1	0	(Cannot be used on GA-16/110/220; use RCSR instead)
RISE	11	0	5	2R	1	R ₁₅ ^{ISE} /S _{ISE} →ISE
RLK	11	0	7	2R	1	R+S _L →R; S _Z , S _P , S ₀ , S _L
RTNIV	10	0	1	1	2	((EA)) ₁₄₋₀ ^P ₁₄₋₀ ; ((EA)+1) ₁₅ ^{ISE} /((EA))→P; ((EA)+1) ₁₅ ^{ISE}
RTR	6	0	8+Rs	2Rd+1	5	Rs→Rd; S _Z , S _P
RTRN	11	0	5	2R	3	R ₁₄₋₀ ^P ₁₄₋₀ ; R ₁₅ ^{ISE} / R→P; S _{ISE} →ISE
SARS	3	F	X	$\frac{C}{D}$	X ⁴	A,X,Y,Z,B,C,D,E,S+(EA)···(EA+8)
SBIT	4	B	X	$\frac{2b}{2b+1}$	X ⁴	(EA) _{b+8} ^S _Z ; 1-(EA) _{b+8} [i even] (EA) _b ^S _Z ; 1+(EA) _b [i odd]

88A00508A-E

Table A-2. Summary of General Instructions (Cont'd.)

COMMAND	TABLE A-1 REF.	HEX CODE				OPERATION
		2	6 7	X	X	
SKM	5	2	6 7	X	X	EA+P [S _{p=0}] P+1+P[S _{p=1}]
SKN	5	2	4 5	X	X	EA+P [S _{Z=0}] P+1+P[S _{Z=1}]
SKOF	5	2	0 1	X	X	EA+P [S ₀₌₀] P+1+P[S ₀₌₁]; S ₀
SKOT	5	2	8 9	X	X	EA+P [S ₀₌₁]; S ₀ P+1+P[S ₀₌₀]
SKP	5	2	E F	X	X	EA+P [S _{p=1}] P+1+P[S _{p=0}]
SKR	5	2	2 3	X	X	EA+P [S _{L=0}] P+1+P[S _{L=1}]
SKS	5	2	A B	X	X	EA+P [S _{L=1}] P+1+P[S _{L=0}]
SKZ	5	2	C D	X	X	EA+P [S _{Z=1}] P+1+P[S _{Z=0}]
SLC	12	0	6	2R	3	R _b +R _{b+1} ; R ₁₅ +R ₀ ; R ₁₅ +S _L ; S _Z , S _p , S _L
SLCL	12	0	7	2R	3	R _b +R _{b+1} ; S _L +R ₀ ; R ₁₅ +S _L ; S _Z , S _p , S _L
SLIO	12	0	7	2R	5	R ₁₅ +S _L ; R _b +R _{b+1} ; 1-R ₀ ; S _Z , S _p , S _L
SLIZ	12	0	7	2R	4	R ₁₅ +S _L ; R _b +R _{b+1} ; 0-R ₀ ; S _Z , S _p , S _L
SRA	13	0	2	2R+1	count-1	R ₁₅ -R ₁₅ ; R _b -R _{b-1} ...; R ₀ -S _L [count]; S _Z , S _p , S _L
SRC	13	0	3	2R	count-1	R ₀ +R ₁₅ ; R _b +R _{b-1} ...; R ₀ -S _L [count]; S _Z , S _p , S _L
SRCL	13	0	3	2R+1	count-1	S _L +R ₁₅ ; R ₀ -L; R _b -R _{b-1} [count]; S _Z , S _p , S _L
SRLC	13	0	2	2R	count-1	0-R ₁₅ ; R _b -R _{b-1} ...; R ₀ -S _L [count]; S _Z , S _p , S _L , S _{Shift} [shift stops when S _L =1; count+S _{Shift}]
STA	1	5	X	X	X	A-(EA)
STBY	2	9	X	2R 2R+1	X ⁴	R ₇₋₀ -(EA) ₁₅₋₈ [i even] R ₇₋₀ -(EA) ₇₋₀ [i odd]
STR	2	D	X	2R 2R+1	X ⁴	R-(EA)
SUB	6	0	8+Rs	2Rd+1	6	Rd-Rs-Rd; S _Z , S _p , S ₀ , S _L
SUBC	7	0	8+Rs	2Rd	6	Rd-Rs-DBUS; S _Z , S _p , S ₀ , S _L
SUBV	8	0	1	2R+1	6	R-(P+1)+R; S _Z , S _p , S ₀ , S _L
SUBVC	9	0	1	2R	6	R-(P+1)-DBUS; S _Z , S _p , S ₀ , S _L
SYNC	14	0	4	8	0	PULSE-SYNC TEST POINT
TBIT	4	A	X	2 ^h 2 ^{b+1}	X ⁴	(E _A) _{b+8} -S _Z [i even] (i _A) _b +S _Z [i odd]
TEST	16	1	fun	CO+dev-addr		Function True P+2+P; Function False P+1+P
TRAP	14	0	0	1	X	ISE-(7D) ₁₅ ; I ₁₄₋₀ -(7D) ₁₄₋₀ ; P-(7C); (44)+P; 0-ISE

Table A-2. Summary of General Instructions (Cont'd.)

COMMAND	TABLE A-1 REF.	HEX CODE				OPERATION
TRS	11	0	5	2R	8	$R_{15} \rightarrow S_{15}; R_{13-9} \rightarrow S_{13-9}; R_8 \rightarrow S_7; R_7 \rightarrow S_2; R_6 \rightarrow S_p;$ $R_5 \rightarrow S_0; R_4 \rightarrow S_L; R_{3-0} \rightarrow S_{SHIFT}$
TSR	11	0	6	2R	8	$S_{15} \rightarrow R_{15}; S_{MODE} \rightarrow R_{14}; S_{13-9} \rightarrow R_{13-9}; S_F \rightarrow R_8; S_Z \rightarrow R_7;$ $S_p \rightarrow R_6; S_0 \rightarrow R_5; S_L \rightarrow R_4; S_{Shift} \rightarrow R_{3-0}$
WAIT	14	0	0	0	X	P→P; Must use console ROM to step past P
XEC	11	0	5	2R+1	0	R→I
XOR	6	0	8+Rs	2Rd+1	8	$Rd \oplus Rs \rightarrow Rd; S_Z, S_p$
XORC	7	0	8+Rs	2Rd	8	$Rd \oplus Rs \rightarrow DBUS; S_Z, S_p$
XORV	8	0	1	2R+1	8	$R \oplus (P+1) \rightarrow R; S_Z, S_p$
XORVC	9	0	1	2R	8	$R \oplus (P+1) \rightarrow DBUS; S_Z, S_p$
ZERO	11	0	6	2R	0	0→R
ZLBY	11	0	6	2R	2	0→R ₁₅₋₈
ZRBY	11	0	6	2R	1	0→R ₇₋₀

88A00508A-E

Notes to Table A-2

① Numbers in Table A-1 reference column identify the general format of instructions included in Table A-2.

②

CAP-16 CONVENTIONS TRANSLATED TO BINARY AND HEX EQUIVALENTS					
CAP-16 MNEMONIC	BINARY CODE	R Rd	2R 2Rd	2R+1 2Rd+1	8+R 8+Rs
A	000	0	0	1	8
X	001	1	2	3	9
Y	010	2	4	5	A
Z	011	3	6	7	B
B	100	4	8	9	C
C	101	5	A	B	D
D	110	6	C	D	E
E	111	7	E	F	F

③ CONVENTIONS USED IN OPERATION COLUMN

R register (usage determined by inst.)
 Rs source register
 Rd destination register
 EA effective address
 R_b bit in a register
 R_{b+1} next bit left
 R_{b-1} next bit right
 IV interrupt vector for I/O
 DBUS only a comparison is made, setting appropriate status register bits

/ 32K/64K mode alternate operation
 | alternate operation based on a condition for transfer
 [] condition for alternate operation
 Data transfers are full word unless bits are specifically indicated; e.g., R always means contents of register
 *R means R points to address in memory (i.e., Indirect Address)

STATUS REGISTER, S		
TERM	BIT	DESCRIPTION
S _{ISE}	S ₁₅	ISE SAVE STATUS
S _{MODE}	S ₁₄	0=32K/1=64K
S _F	S ₈	FOREGROUND
S _Z	S ₇	ZERO
S _p	S ₆	PLUS
S ₀	S ₅	OVERFLOW
S _L	S ₄	LINK
S _{Shift}	S ₃₋₀	SHIFT COUNT

LOGIC SYMBOLS

V Or
 ⊕ Exclusive-OR
 ∧ And

MATH SYMBOLS

+ Add
 - Subtract
 · Multiply
 : Divide

④ Refer to Table A-4 when the Hex Code indicates one of the following:

X	2R	X
	2R+1	
X	2b	X
	2b+1	
X	()	X

Table A-3. Memory Reference Instructions EA Calculations

CAP-16 CODING FORMAT		BINARY HEX								ADDRESSING MODE	EA CALCULATION	NEXT INSTRUCTION
OPCODE[*]address [m ₁]	15 14 13 12 CODE	11 m	10 *	9 s	8 i	7 g	6 n	5 4 3 2 1 0 Displacement				
JMP Refer to Examples Below	0 1 1 1 — 7 —								Refer to examples below for range and transitions from negative (2s complement) to positive values			
JSR	0 1 1 0 — 6 —											
LDA	0 1 0 0 — 4 —											
STA	0 1 0 1 — 5 —											
Refer to Codes	-512 ⋮ -1 0 ⋮ +511			0 0 1 0 — 2 —	0 0 0 0 — 0 —	0 0 0 0 — 0 —				Program Relative	P+1+disp	P+1
				0 0 1 1 — 3 —	1 1 1 1 — F —	1 1 1 1 — F —						
				0 0 0 0 — 0 —	0 0 0 0 — 0 —	0 0 0 0 — 0 —						
				0 0 0 1 — 1 —	1 1 1 1 — F —	1 1 1 1 — F —						
	*-512 ⋮ *-1 *0 ⋮ *511			0 1 1 0 — 6 —	0 0 0 0 — 0 —	0 0 0 0 — 0 —				Program Relative Indirect	(P+1+disp)	P+1
				0 1 1 1 — 7 —	1 1 1 1 — F —	1 1 1 1 — F —						
				0 1 0 0 — 4 —	0 0 0 0 — 0 —	0 0 0 0 — 0 —						
				0 1 0 1 — 5 —	1 1 1 1 — F —	1 1 1 1 — F —						
	0,1 ⋮ 1023,1			1 0 0 0 — 3 —	0 0 0 0 — 0 —	0 0 0 0 — 0 —				Base Relative	D+disp	P+1
				1 0 1 1 — B —	1 1 1 1 — F —	1 1 1 1 — F —						
	*0,1 ⋮ *1023,1			1 1 0 0 — C —	0 0 0 0 — 0 —	0 0 0 0 — 0 —				Base Relative Indirect	(D+disp)	P+1
				1 1 1 1 — F —	1 1 1 1 — F —	1 1 1 1 — F —						

Range of address (disp) may be numbers shown above or any CAP-16 address expression which equals these numbers.

Table A-4. Memory Reference with Indexing, EA Calculations, Ref. 2, 3, 4 in A-1 and A-2

OPCODE	OPERAND (b) [R] [m]addr[.i][.m ₂]	①					ADDRESSING MODE ①	EA CALCULATION
		15,14,13,12 OP CODE	m	*	i	7,6,5,4 { b R ext}		
RBIT	b	0 0 1 1 —3—						
LDBY	R	1 0 0 0 —8—						
STBY	R	1 0 0 1 —9—						
TBIT	b	1 0 1 0 —A—						
SBIT	b	1 0 1 1 —B—						
LDR	R	1 1 0 0 —C—						
STR	R	1 1 0 1 —D—						
CMR	R	1 1 1 0 —E—						
② { DECM INCM LARS SARS		1 1 1 1 —F—						

NOTES:

① Addressing mode and indexing are determined by M, *, i bits as follows:

- | | | | |
|-------|----------|--------|----------|
| m = 0 | Absolute | i = 00 | No index |
| = 1 | Base | = 01 | X index |
| * = 0 | Direct | = 10 | Y index |
| = 1 | Indirect | = 11 | Z index |

For instructions which reference bits (RBIT, SBIT, TBIT) and instructions referencing bytes (LDBY and STBY) an even index value refers to left byte, and an odd index value refers to right byte. Therefore, EA is divided by 2 for those instructions.

② Refer to sheet 3 and 4 for extension (ext.).

Table A-4. Memory Reference with Indexing, EA Calculations (Cont'd)
 Ref. 2, 3, 4 in A-1 and A-2

OPCODE	OPERANDS [b] [*]addr[,i][,m ₂] (R)	15,14,13,12	11,10,9,8	7,6,5,4	3,2,1,0	ADDRESSING MODE	EA CALCULATION ^③
		OP CODE	m * i	(b R ext)	DISP		
	{addr,, addr,0,}	{0 0 2}	0 0 0 0 —0—			Absolute Direct	addr→EA
	addr,X,	{0 2}	0 0 0 1 —1—			Absolute Direct Indexed	addr+X→EA
	addr,Y,	{0 2}	0 0 1 0 —2—				addr+Y→EA
	addr,Z,	{0 2}	0 0 1 1 —3—				addr+Z→EA
	{*addr,, *addr,0,}	{0 0 2}	0 1 0 0 —4—			Absolute Indirect	(addr)→EA
	*addr,X,	{0 2}	0 1 0 1 —5—			Absolute Indirect Indexed	(addr)+X→EA
	*addr,Y,	{0 2}	0 1 1 0 —6—				(addr)+Y→EA
	*addr,Z,	{0 2}	0 1 1 1 —7—				(addr)+Z→EA
	{addr,, addr,0,}	{1 1 3}	1 0 0 0 —8—			Base-relative Direct	addr+D→EA
	addr,X,	{1 3}	1 0 0 1 —9—			Base-relative Direct, Indexed	addr+D+X→EA
	addr,Y,	{1 3}	1 0 1 0 —A—				addr+D+Y→EA
	addr,Z,	{1 3}	1 0 1 1 —B—				addr+D+Z→EA
	{*addr,, *addr,0,}	{1 1 3}	1 1 0 0 —C—			Base Indirect	(addr+D)→EA
	*addr,X,	{1 3}	1 1 0 1 —D—			Base Indirect Indexed	(addr+D)+X→EA
	*addr,Y,	{1 3}	1 1 1 0 —E—				(addr+D)+Y→EA
	*addr,Z,	{1 3}	1 1 1 1 —F—				(addr+D)+Z→EA

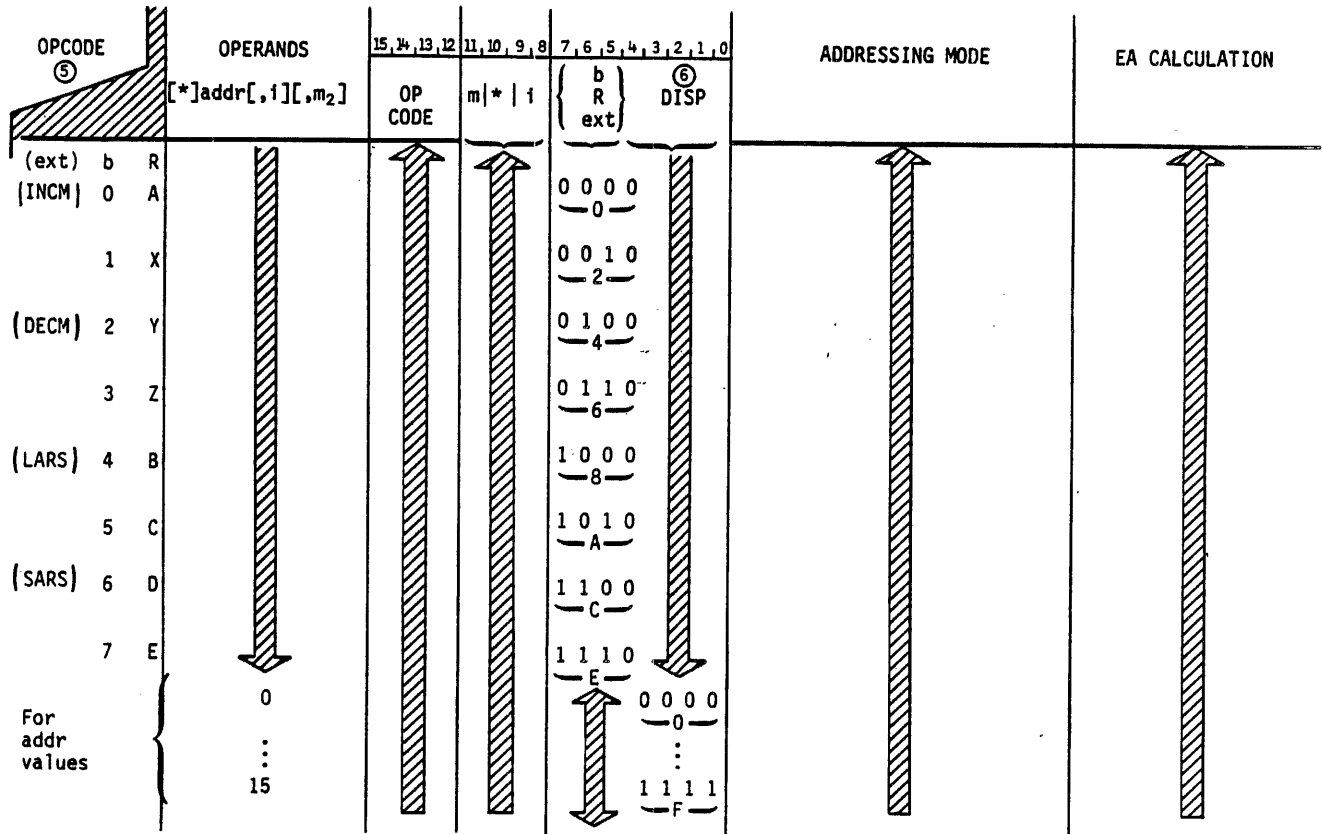
NOTES:

③ EA Calculation is determined by addressing mode. The EA is calculated from the disp field for addr values ≤ 1E. For values > 1E disp field = 1F and disp value in P+1 (2-word instruction).

When m₂ is coded 2 or 3, disp is set to 1F and all displacement value from 0 through 65,536 is in P+1.

If m₂ is not coded, the assembler will provide addressing mode and will determine 1 or 2-word instruction on basis of value of addr.

Table A-4. Memory Reference with Indexing, EA Calculations (Cont'd)
 Ref. 2, 3, 4 in A-1 and A-2



NOTES:

⑤ Range of values defining b, R, or INCM, DECM, LARS or SARS instructions and MSB of displacement are in byte 3

⑥ Next instruction is determined by contents of disp field. If disp < 1E, next instruction is at P+1.
 If disp = 1F, next instruction is at P+2.
 Next instruction is also at P+2 when m₂ is coded 2 or 3.

Table A-4. Memory Reference with Indexing, EA Calculations (Cont'd)
 Ref. 2, 3, 4 in A-1 and A-2

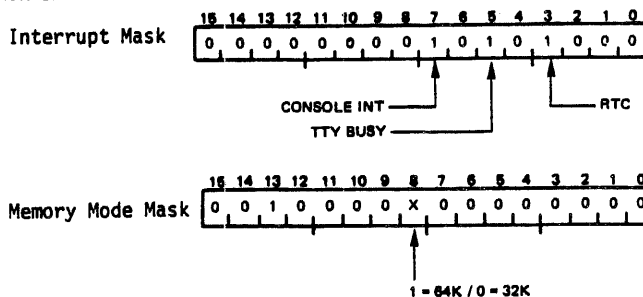
OPCODE ⑤	OPERANDS [*]addr[,i][,m ₂]	15,14,13,12 11,10,9,8 7,6,5,4 3,2,1,0				ADDRESSING MODE	EA CALCULATION
		OP CODE	m * i	$\begin{Bmatrix} b \\ R \\ \text{ext} \end{Bmatrix}$	DISP		
(ext) b R	↓	↑	↑	0 0 0 1	↓		
(INCM) 0 A				— 1 —			
1 X				0 0 1 1			
(DECM) 2 Y				— 3 —			
3 Z				0 1 0 1			
(LARS) 4 B				— 5 —			
5 C				0 1 1 1			
(SARS) 6 D				— 7 —			
7 E	1 0 0 1	— 9 —					
				1 0 1 1	— B —		
				1 1 0 1	— C —		
				1 1 1 1	— F —		
For addr values ③	16	↑	↑	0 0 0 0	— 0 —		
	⋮			1 1 1 0	— E —		
For addr values ③	32	↑	↑	1 1 1 1	— F —		
	⋮			⋮			
	65,536			1 1 1 1	— F —		

Table A-5. Standard I/O Data & Instructions (Sheet 1 of 2)

TTY DATA OF DEV ADDR	3F	INT	VECTOR	= 45
Reg to TTY	DTOR	1	8+R	7 F
Mem to TTY	DTOM	1	R	7 F
TTY to Reg	DTIR	1	8+R	B F
TTY to Mem	DTIM	1	R	B F
TEST for Not-Busy	TEST	1	0	F F
Set to Transmit	CTRL	1	0	3 F
Set to Rcv	CTRL	1	2	3 F
Set to Rcv/Echo	CTRL	1	4	3 F
Set to Break	CTRL	1	6	3 F

Refer to Appendix B for ASCII data.

MASK DATA FOR DEV ADDR 3E



OUTPUT INSTRUCTIONS TO DEV ADDR 3E

From Register	DTOR	1	8+R	7	E
From Memory	DTOM	1	R	7	E

Power up disables interrupts and sets 32K mode; 32K mode can also be permanently enabled with switch on CPU.

To set or clear mask bits, output a register with desired bits = 1 or 0 (as required) as shown in following example:

MACHINE	CAP-16	FUNCTION
0600	ZERO A	Clear Register A
011D	ORV A,X'nnnn'	Load hex equivalent of bit pattern nnnn
187E	DTOR A,X'3E'	Set the mask

INPUT INSTRUCTIONS TO DEV ADDR 3E TO READ CONSOLE SWITCHES

To Register	{ RCSR or DTIR }	1	8+R	B	E
To Memory	{ RCSM or DTIM }	1	R	B	E

DISPLAY DATA ON CONSOLE -

From Register	DSPL	0	5	2R	4
---------------	------	---	---	----	---

466-A-12

Table A-5. Standard I/O Data & Instructions (Sheet 2 of 2)

CONTROL INSTRUCTIONS TO DEV ADDR 3E

Single Step (SSTEP) CTRL	1	1	3	E
I/O Reset (IORST) CTRL	1	2	3	E

Single step control instruction executable in upper 1K of memory mode in use. NI interrupt via location X'46', with P+1 and ISE saved in locations X'7E' and X'7F', will occur after first instruction outside upper 1K of memory is executed.

ASCII character code

B

Table B-1 provides a summary of ASCII character codes. The GA-16 series software required the parity bit (7) of ASCII data to be set to one. This is referred to as system ASCII and is the way all ASCII characters for TTY are stored in memory. Odd parity and even parity representations are also shown, as they may be output from a teletype or punched on paper tape when the TTY is in LOCAL mode. GA-16 series software sets the parity bit on received ASCII characters to one.

There is a wide variety of TTY and CRT units. All provide numbers and upper-case letters. Additional symbols represent upper-case (SHIFT key pressed), printed characters, control (CTRL key pressed) and, in some cases, a combination of CTRL and SHIFT. Some machines also provide lower-case letters.

The column "TTY or CRT Implementation" shows the means to output most SHIFT or CTRL characters. A CTRL character generally does not result in a printout; however, if the paper tape punch is running, a character is punched. Even though a symbol does not appear on a key, the characters shown in Table B-1 will be output from most TTYs and some CRTs. The manufacturer's representative can usually clarify the character set capabilities of a given machine.

Table B-1. ASCII Character Code Summary

GRAPHIC OR CONTROL	SYSTEM ASCII	ODD PARITY	EVEN PARITY	REMARKS	TTY or CRT IMPLEMENTATION
NULL	80	80	00	Null, tape feed	BREAK key
SOM	81	01	81	Start of message; also SOH (start of heading)	Control A
EOA	82	02	82	End of address; also STX (start of text)	Control B
EOM	83	83	03	End of message; also ETX (end of text)	Control C
EOT	84	04	84	End of transmission; shuts off TWX machine	Control D
WRU	85	85	05	Who are you? Triggers identification (Here is...) at remote station if equipped	Control E
RU	86	86	06	Are you...	Control F
BELL	87	07	87	Rings the bell	Control G
FE	88	08	88	Format effector, also BS, backspace. Backspaces some machines	Control H
H.Tab	89	89	08	Horizontal tab. On Models 33 and 35	Control I
Line Feed	8A	8A	0A	Line feed	Control J, or Line Feed
V.Tab	8B	0B	8B	Vertical tab. On Models 33 and 35	Control K
Form	8C	8C	0C	Form feed to top of next page	Control L
Return	8D	0D	8D	Carriage return. On Models 33 and 35	Control M or CR key
SO	8E	0E	8E	Shift Out - changes ribbon color to red	Control N
SI	8F	8F	0F	Shift In - changes ribbon color to black	Control O
DCO	90	10	90	Data link escape	Control P
X-On	91	91	11	Turns on transmitter or reader	Control Q
Tape Aux. On	92	92	12	Turns on punch or auxiliary	Control R
X-Off	93	13	93	Turns off transmitter or reader	Control S
Tape Aux. Off	94	94	14	Turns off punch or auxiliary	Control T
Error	95	15	95	Error negative acknowledge	Control U
Sync	96	16	96	Synchronous idle	Control V
LEM	97	97	17	Logical end of medium; also ETB (end of transmission block)	Control W
SO	98	98	18	CAN (cancel)	Control X
S1	99	19	99	EOM (end of medium)	Control Y
S2	9A	1A	9A	SUB (substitute)	Control Z
S3	9B	9B	1B	ESC (escape on Models 33 and 35)	Control, shift K
S4	9C	1C	9C	FS (file separator on Models 33 and 35)	Control, shift L
S5	9D	9D	1D	GS (group separator on Models 33 and 35)	Control, shift M
S6	9E	9E	1E	RS (record separator on Models 33 and 35)	Control, shift N
S7	9F	1F	9F	US (unit separator on Models 33 and 35)	Control, shift O
blank	A0	20	A0	Space	Space bar
!	A1	A1	21		Shift 1
"	A2	A2	22		Shift 2
#	A3	23	A3		Shift 3
\$	A4	A4	24		Shift 4
%	A5	25	A5		Shift 5
&	A6	26	A6		Shift 6
'	A7	A7	27	Acute accent or apostrophe	Shift 7
(A8	A8	28		Shift 8
)	A9	29	A9		Shift 9
*	AA	2A	AA	Repeats on Model 37	Shift ;
+	AB	AB	2B		Shift ;
comma	AC	2C	AC	Comma	,
-	AD	AD	2D	Repeats on Model 37	-
.	AE	AE	2E	Repeats on Model 37	.
/	AF	2F	AF		/

Table B-1. ASCII Character Code Summary (Cont'd)

GRAPHIC OR CONTROL	SYSTEM ASCII	ODD PARITY	EVEN PARITY	REMARKS	TTY or CRT IMPLEMENTATION
0	B0	B0	30		0
1	B1	31	B1		1
2	B2	32	B2		2
3	B3	B3	33		3
4	B4	34	B4		4
5	B5	B5	35		5
6	B6	B6	36		6
7	B7	37	B7		7
8	B8	38	B8		8
9	B9	B9	39		9
:	BA	BA	3A		:
;	BB	3B	BB		;
<	BC	BC	3C		<
=	BD	3D	BD	Repeats on Model 37	Shift ,
>	BE	3E	BE		Shift -
?	BF	BF	3F		Shift .
@	C0	40	C0		Shift /
A	C1	C1	41		Shift P
B	C2	C2	42		A
C	C3	43	C3		B
D	C4	C4	44		C
E	C5	45	C5		D
F	C6	46	C6		E
G	C7	C7	47		F
H	C8	C8	48		G
I	C9	49	C9		H
J	CA	4A	CA		I
K	CB	CB	4B		J
L	CC	4C	CC		K
M	CD	CD	4D		L
N	CE	CE	4E		M
O	CF	4F	CF		N
P	D0	D0	50		O
Q	D1	51	D1		P
R	D2	52	D2		Q
S	D3	D3	53		R
T	D4	54	D4		S
U	D5	D5	55		T
V	D6	D6	56		U
W	D7	57	D7		V
X	D8	58	D8		W
Y	D9	D9	59		X
Z	DA	DA	5A		Y
[DB	5B	DB	On Models 33 and 35	Z
\	DC	DC	5C	On Models 33 and 35	Shift K
]	DD	5D	DD	On Models 33 and 35	Shift L
+ or ^	DE	5E	DE		Shift M
. or _	DF	DF	5F	Repeats on Model 37	Shift N
					Shift O

Table B-1. ASCII Character Code Summary (Cont'd)

GRAPHIC OR CONTROL	SYSTEM ASCII	ODD PARITY	EVEN PARITY	REMARKS	TTY or CRT IMPLEMENTATION	
-	E0	E0	60	Accent grave		
a	E1	61	E1	These characters applicable only for machines with lower case capabilities.		
b	E2	62	E2			
c	E3	E3	63			
d	E4	64	E4			
e	E5	E5	65			
f	E6	E6	66			
g	E7	67	E7			
h	E8	68	E8			
i	E9	E9	69			
j	EA	EA	6A			
k	EB	6B	EB			
l	EC	EC	6C			
m	ED	6D	ED			
n	EE	6E	EE			
o	EF	EF	6F			
p	F0	70	F0			
q	F1	F1	71			
r	F2	F2	72			
s	F3	73	F3			
t	F4	F4	74			
u	F5	75	F5			
v	F6	76	F6			
w	F7	F7	77		Repeats on Model 37	
x	F8	F8	78			
y	F9	79	F9			
z	FA	7A	FA			
{	FB	F8	7B			
	FC	7C	FC			
}	FD	FD	7D			
~	FE	FE	7E	AH mode	ALT MODE key	
DEL	FF	7F	FF	Delete, rubout repeats on Model 37	RUBOUT key	

NOTES:

REPT key causes repetition of other key pressed at same time.
 HERE IS key is option in machines which sends a pre-coded message or may send a series of NULL characters.

conversion tables

C

This appendix contains the following reference tables:

<u>Title</u>	<u>Page</u>
Hexadecimal Arithmetic	C-2
Addition Table	C-2
Multiplication Table	C-2
Powers of 16_{10}	C-3
Powers of 10_{16}	C-3
Hexadecimal-Decimal Integer Conversion	C-4
Hexadecimal-Decimal Fraction Conversion	C-10
Powers of Two	C-14
Mathematical Constants	C-14
Decimal/Binary Position Table	C-15

88A00508A-E

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	2B	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

88A00508A-E

TABLE OF POWERS OF SIXTEEN₁₀

16^n					n	16^{-n}					
				1	0	0.10000	00000	00000	00000	x	10^0
				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	474 976 710 656	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 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}
			916 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 8652 6FC1 0000	16	0.734A	CA5F	6226	F0AE x 16^{-13}
			163 4578 5D8A 0000	17	0.8877	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}

HEXADECIMAL-DECIMAL INTEGER CONVERSION

The table below provides for direct conversions between hexadecimal integers in the range 0-FFF and decimal integers in the range 0-4095. For conversion of larger integers, the table values may be added to the following figures:

Hexadecimal fractions may be converted to decimal fractions as follows:

- Express the hexadecimal fraction as an integer times 16^{-n} , where n is the number of significant hexadecimal places to the right of the hexadecimal point.

$$0. CA9BF3_{16} = CA9BF3_{16} \times 16^{-6}$$

- Find the decimal equivalent of the hexadecimal integer

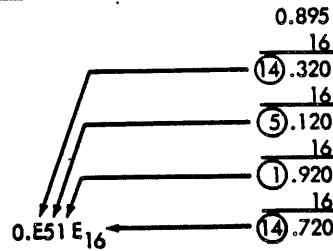
$$CA9BF3_{16} = 13\,278\,195_{10}$$

- Multiply the decimal equivalent by 16^{-n}

$$\begin{array}{r} 13\,278\,195 \\ \times 596\,046\,448 \times 10^{-16} \\ \hline 0.791\,442\,096_{10} \end{array}$$

Decimal fractions may be converted to hexadecimal fractions by successively multiplying the decimal fraction by 16_{10} . After each multiplication, the integer portion is removed to form a hexadecimal fraction by building to the right of the hexadecimal point. However, since decimal arithmetic is used in this conversion, the integer portion of each product must be converted to hexadecimal numbers.

Example: Convert 0.895_{10} to its hexadecimal equivalent



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

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255

HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
10	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
18	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
19	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
20	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
30	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
34	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
38	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
40	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
41	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
42	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
43	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
44	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
45	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
46	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
47	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
48	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
49	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
50	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
51	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
52	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
53	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
54	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
55	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
56	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
57	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
58	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
59	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535
60	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
61	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
62	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
63	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
64	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
65	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
66	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
67	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
68	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
69	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791

HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
70	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
71	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
72	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
73	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
74	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
75	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
76	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
77	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
78	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
79	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
80	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
81	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
82	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
83	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
84	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
85	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
86	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
87	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
88	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
89	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
90	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
91	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
92	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
93	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
94	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
95	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
96	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
97	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
98	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
99	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559

88A00508A-E

HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A0	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A1	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A2	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A3	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A4	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A5	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A6	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A7	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A8	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A9	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B0	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B1	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B2	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B3	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B4	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B5	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B6	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B7	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B8	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B9	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071
C0	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C1	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C2	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C3	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C4	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C5	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C6	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C7	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C8	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C9	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327

HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
D0	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D1	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D2	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D3	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D4	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D5	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D6	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D7	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D8	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D9	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E0	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E1	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E2	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E3	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E4	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E5	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E6	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E7	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E8	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E9	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F0	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F1	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F2	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F3	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F4	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F5	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F6	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F7	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F8	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F9	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

HEXADECIMAL-DECIMAL FRACTION CONVERSION

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

HEXADECIMAL-DECIMAL FRACTION CONVERSION (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 42431	.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

88A00508A-E

HEXADECIMAL-DECIMAL FRACTION CONVERSION (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 13112	.00 00 56 00	.00000 51259	.00 00 96 00	.00000 89406	.00 00 D6 00	.00001 27553
.00 00 17 00	.00000 13708	.00 00 57 00	.00000 51855	.00 00 97 00	.00000 90003	.00 00 D7 00	.00001 28149
.00 00 18 00	.00000 14304	.00 00 58 00	.00000 52451	.00 00 98 00	.00000 90599	.00 00 D8 00	.00001 28746
.00 00 19 00	.00000 14900	.00 00 59 00	.00000 53047	.00 00 99 00	.00000 91195	.00 00 D9 00	.00001 29342
.00 00 1A 00	.00000 15496	.00 00 5A 00	.00000 53643	.00 00 9A 00	.00000 91791	.00 00 DA 00	.00001 29938
.00 00 1B 00	.00000 16092	.00 00 5B 00	.00000 54239	.00 00 9B 00	.00000 92387	.00 00 DB 00	.00001 30534
.00 00 1C 00	.00000 16688	.00 00 5C 00	.00000 54835	.00 00 9C 00	.00000 92983	.00 00 DC 00	.00001 31130
.00 00 1D 00	.00000 17284	.00 00 5D 00	.00000 55431	.00 00 9D 00	.00000 93579	.00 00 DD 00	.00001 31726
.00 00 1E 00	.00000 17880	.00 00 5E 00	.00000 56027	.00 00 9E 00	.00000 94175	.00 00 DE 00	.00001 32322
.00 00 1F 00	.00000 18477	.00 00 5F 00	.00000 56623	.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 41262
.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 66161	.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 07288	.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 72121	.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 FD 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

HEXADecimal-DECIMAL FRACTION CONVERSION (Cont.)

Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00	.00000 00000	.00 00 40	.00000 00149	.00 00 80	.00000 00298	.00 00 C0	.00000 00447
.00 00 01	.00000 00002	.00 00 41	.00000 00151	.00 00 81	.00000 00300	.00 00 C1	.00000 00449
.00 00 02	.00000 00004	.00 00 42	.00000 00153	.00 00 82	.00000 00302	.00 00 C2	.00000 00451
.00 00 03	.00000 00006	.00 00 43	.00000 00155	.00 00 83	.00000 00305	.00 00 C3	.00000 00454
.00 00 04	.00000 00009	.00 00 44	.00000 00158	.00 00 84	.00000 00307	.00 00 C4	.00000 00456
.00 00 05	.00000 00011	.00 00 45	.00000 00160	.00 00 85	.00000 00309	.00 00 C5	.00000 00458
.00 00 06	.00000 00013	.00 00 46	.00000 00162	.00 00 86	.00000 00311	.00 00 C6	.00000 00461
.00 00 07	.00000 00016	.00 00 47	.00000 00165	.00 00 87	.00000 00314	.00 00 C7	.00000 00463
.00 00 08	.00000 00018	.00 00 48	.00000 00167	.00 00 88	.00000 00316	.00 00 C8	.00000 00465
.00 00 09	.00000 00020	.00 00 49	.00000 00169	.00 00 89	.00000 00318	.00 00 C9	.00000 00467
.00 00 0A	.00000 00023	.00 00 4A	.00000 00172	.00 00 8A	.00000 00321	.00 00 CA	.00000 00470
.00 00 0B	.00000 00025	.00 00 4B	.00000 00174	.00 00 8B	.00000 00323	.00 00 CB	.00000 00472
.00 00 0C	.00000 00027	.00 00 4C	.00000 00176	.00 00 8C	.00000 00325	.00 00 CC	.00000 00474
.00 00 0D	.00000 00030	.00 00 4D	.00000 00179	.00 00 8D	.00000 00328	.00 00 CD	.00000 00477
.00 00 0E	.00000 00032	.00 00 4E	.00000 00181	.00 00 8E	.00000 00330	.00 00 CE	.00000 00479
.00 00 0F	.00000 00034	.00 00 4F	.00000 00183	.00 00 8F	.00000 00332	.00 00 CF	.00000 00481
.00 00 10	.00000 00037	.00 00 50	.00000 00186	.00 00 90	.00000 00335	.00 00 D0	.00000 00484
.00 00 11	.00000 00039	.00 00 51	.00000 00188	.00 00 91	.00000 00337	.00 00 D1	.00000 00486
.00 00 12	.00000 00041	.00 00 52	.00000 00190	.00 00 92	.00000 00339	.00 00 D2	.00000 00488
.00 00 13	.00000 00044	.00 00 53	.00000 00193	.00 00 93	.00000 00342	.00 00 D3	.00000 00491
.00 00 14	.00000 00046	.00 00 54	.00000 00195	.00 00 94	.00000 00344	.00 00 D4	.00000 00493
.00 00 15	.00000 00048	.00 00 55	.00000 00197	.00 00 95	.00000 00346	.00 00 D5	.00000 00495
.00 00 16	.00000 00051	.00 00 56	.00000 00200	.00 00 96	.00000 00349	.00 00 D6	.00000 00498
.00 00 17	.00000 00053	.00 00 57	.00000 00202	.00 00 97	.00000 00351	.00 00 D7	.00000 00500
.00 00 18	.00000 00055	.00 00 58	.00000 00204	.00 00 98	.00000 00353	.00 00 D8	.00000 00502
.00 00 19	.00000 00058	.00 00 59	.00000 00207	.00 00 99	.00000 00356	.00 00 D9	.00000 00505
.00 00 1A	.00000 00060	.00 00 5A	.00000 00209	.00 00 9A	.00000 00358	.00 00 DA	.00000 00507
.00 00 1B	.00000 00062	.00 00 5B	.00000 00211	.00 00 9B	.00000 00360	.00 00 DB	.00000 00509
.00 00 1C	.00000 00065	.00 00 5C	.00000 00214	.00 00 9C	.00000 00363	.00 00 DC	.00000 00512
.00 00 1D	.00000 00067	.00 00 5D	.00000 00216	.00 00 9D	.00000 00365	.00 00 DD	.00000 00514
.00 00 1E	.00000 00069	.00 00 5E	.00000 00218	.00 00 9E	.00000 00367	.00 00 DE	.00000 00516
.00 00 1F	.00000 00072	.00 00 5F	.00000 00221	.00 00 9F	.00000 00370	.00 00 DF	.00000 00519
.00 00 20	.00000 00074	.00 00 60	.00000 00223	.00 00 A0	.00000 00372	.00 00 E0	.00000 00521
.00 00 21	.00000 00076	.00 00 61	.00000 00225	.00 00 A1	.00000 00374	.00 00 E1	.00000 00523
.00 00 22	.00000 00079	.00 00 62	.00000 00228	.00 00 A2	.00000 00377	.00 00 E2	.00000 00526
.00 00 23	.00000 00081	.00 00 63	.00000 00230	.00 00 A3	.00000 00379	.00 00 E3	.00000 00528
.00 00 24	.00000 00083	.00 00 64	.00000 00232	.00 00 A4	.00000 00381	.00 00 E4	.00000 00530
.00 00 25	.00000 00086	.00 00 65	.00000 00235	.00 00 A5	.00000 00384	.00 00 E5	.00000 00533
.00 00 26	.00000 00088	.00 00 66	.00000 00237	.00 00 A6	.00000 00386	.00 00 E6	.00000 00535
.00 00 27	.00000 00090	.00 00 67	.00000 00239	.00 00 A7	.00000 00388	.00 00 E7	.00000 00537
.00 00 28	.00000 00093	.00 00 68	.00000 00242	.00 00 A8	.00000 00391	.00 00 E8	.00000 00540
.00 00 29	.00000 00095	.00 00 69	.00000 00244	.00 00 A9	.00000 00393	.00 00 E9	.00000 00542
.00 00 2A	.00000 00097	.00 00 6A	.00000 00246	.00 00 AA	.00000 00395	.00 00 EA	.00000 00544
.00 00 2B	.00000 00100	.00 00 6B	.00000 00249	.00 00 AB	.00000 00398	.00 00 EB	.00000 00547
.00 00 2C	.00000 00102	.00 00 6C	.00000 00251	.00 00 AC	.00000 00400	.00 00 EC	.00000 00549
.00 00 2D	.00000 00104	.00 00 6D	.00000 00253	.00 00 AD	.00000 00402	.00 00 ED	.00000 00551
.00 00 2E	.00000 00107	.00 00 6E	.00000 00256	.00 00 AE	.00000 00405	.00 00 EE	.00000 00554
.00 00 2F	.00000 00109	.00 00 6F	.00000 00258	.00 00 AF	.00000 00407	.00 00 EF	.00000 00556
.00 00 30	.00000 00111	.00 00 70	.00000 00260	.00 00 80	.00000 00409	.00 00 F0	.00000 00558
.00 00 31	.00000 00114	.00 00 71	.00000 00263	.00 00 81	.00000 00412	.00 00 F1	.00000 00561
.00 00 32	.00000 00116	.00 00 72	.00000 00265	.00 00 82	.00000 00414	.00 00 F2	.00000 00563
.00 00 33	.00000 00118	.00 00 73	.00000 00267	.00 00 83	.00000 00416	.00 00 F3	.00000 00565
.00 00 34	.00000 00121	.00 00 74	.00000 00270	.00 00 84	.00000 00419	.00 00 F4	.00000 00568
.00 00 35	.00000 00123	.00 00 75	.00000 00272	.00 00 85	.00000 00421	.00 00 F5	.00000 00570
.00 00 36	.00000 00125	.00 00 76	.00000 00274	.00 00 86	.00000 00423	.00 00 F6	.00000 00572
.00 00 37	.00000 00128	.00 00 77	.00000 00277	.00 00 87	.00000 00426	.00 00 F7	.00000 00575
.00 00 38	.00000 00130	.00 00 78	.00000 00279	.00 00 88	.00000 00428	.00 00 F8	.00000 00577
.00 00 39	.00000 00132	.00 00 79	.00000 00281	.00 00 89	.00000 00430	.00 00 F9	.00000 00579
.00 00 3A	.00000 00135	.00 00 7A	.00000 00284	.00 00 8A	.00000 00433	.00 00 FA	.00000 00582
.00 00 3B	.00000 00137	.00 00 7B	.00000 00286	.00 00 8B	.00000 00435	.00 00 FB	.00000 00584
.00 00 3C	.00000 00139	.00 00 7C	.00000 00288	.00 00 8C	.00000 00437	.00 00 FC	.00000 00586
.00 00 3D	.00000 00142	.00 00 7D	.00000 00291	.00 00 8D	.00000 00440	.00 00 FD	.00000 00589
.00 00 3E	.00000 00144	.00 00 7E	.00000 00293	.00 00 8E	.00000 00442	.00 00 FE	.00000 00591
.00 00 3F	.00000 00146	.00 00 7F	.00000 00295	.00 00 8F	.00000 00444	.00 00 FF	.00000 00593

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	0.000 000 119 209 289 530 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 611 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5
2 199 023 255 552	41	0.000 000 000 000 454 747 350 886 464 118 957 519 531 25
4 398 046 511 104	42	0.000 000 000 000 227 373 675 443 232 059 478 759 765 625
8 796 093 022 208	43	0.000 000 000 000 113 686 837 721 616 029 739 379 882 812 5
17 592 186 044 416	44	0.000 000 000 000 056 843 418 860 808 014 869 689 941 406 25
35 184 372 088 832	45	0.000 000 000 000 028 421 709 430 404 007 434 844 970 703 125
70 368 744 177 664	46	0.000 000 000 000 014 210 854 715 202 003 717 422 485 351 562 5
140 737 488 355 328	47	0.000 000 000 000 007 105 427 357 601 001 858 711 242 675 781 25
281 474 976 710 656	48	0.000 000 000 000 003 552 713 678 800 500 929 355 621 337 890 625
562 949 953 421 312	49	0.000 000 000 000 001 776 356 839 400 250 464 677 810 668 945 312 5
1 125 899 906 842 624	50	0.000 000 000 000 000 888 178 419 700 125 232 338 905 334 472 656 25
2 251 799 813 685 248	51	0.000 000 000 000 000 444 089 209 850 062 616 169 452 667 236 328 125
4 503 599 627 370 496	52	0.000 000 000 000 000 222 044 604 925 031 308 084 726 333 618 164 062 5
9 007 199 254 740 992	53	0.000 000 000 000 000 111 022 302 462 515 654 042 363 166 809 082 031 25
18 014 398 509 481 984	54	0.000 000 000 000 000 055 511 151 231 257 827 021 181 583 404 541 015 625
36 028 797 018 963 968	55	0.000 000 000 000 000 027 755 575 611 628 713 510 590 791 702 270 507 812 5
72 057 594 037 927 936	56	0.000 000 000 000 000 013 877 787 807 814 456 755 295 395 851 135 253 906 25
144 115 188 075 855 872	57	0.000 000 000 000 000 006 938 893 903 907 228 377 647 697 925 567 626 953 125
288 230 376 151 711 744	58	0.000 000 000 000 000 003 469 446 951 953 614 188 823 848 962 783 813 476 562 5
576 460 752 303 423 488	59	0.000 000 000 000 000 001 734 723 475 976 807 094 411 924 481 391 906 738 281 25
1 152 921 504 606 846 976	60	0.000 000 000 000 000 000 867 361 737 988 403 547 205 962 240 695 953 369 140 625
2 305 843 009 213 693 952	61	0.000 000 000 000 000 000 433 680 868 994 201 773 602 981 120 347 976 684 570 312 5
4 611 686 018 427 387 904	62	0.000 000 000 000 000 000 216 840 434 497 100 886 801 490 560 173 988 342 285 156 25
9 223 372 036 854 775 808	63	0.000 000 000 000 000 000 108 420 217 248 550 443 400 745 280 086 994 171 142 578 125

MATHEMATICAL CONSTANTS

Constant	Decimal Value	Hexadecimal Value
π	3.14159 26535 89793	3.243F 6A89
π^{-1}	0.31830 98861 83790	0.517C C187
$\sqrt{\pi}$	1.77245 38509 05516	1.C58F 891C
$\ln \pi$	1.14472 98858 49400	1.250D 048F
e	2.71828 18284 59045	2.87E1 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	0.93CA 67E4
$\ln \gamma$	-0.54953 93129 81645	-0.8CAE 98C1
$\sqrt{2}$	1.41421 35623 73095	1.6A09 E668
$\ln 2$	0.69314 71805 59945	0.8172 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	2.4D75 3777

DECIMAL/BINARY POSITION TABLE

Largest Decimal Integer	Decimal Digits Req'd*	Number of Binary Digits	Largest Decimal Fraction
1		1	5
3		2	.75
7		3	.875
15	1	4	.937 5
31		5	.968 75
63		6	.984 375
127	2	7	.992 187 5
255		8	.996 093 75
511		9	.998 046 875
1 023	3	10	.999 023 437 5
2 047		11	.999 511 718 75
4 095		12	.999 755 859 375
8 191		13	.999 877 929 687 5
16 383	4	14	.999 938 964 843 75
32 767		15	.999 969 482 421 875
65 535		16	.999 984 741 210 937 5
131 071	5	17	.999 992 370 605 468 75
262 143		18	.999 996 185 302 734 375
524 287		19	.999 998 092 651 367 187 5
1 048 575	6	20	.999 999 046 325 683 593 75
2 097 151		21	.999 999 523 162 841 796 875
4 194 303		22	.999 999 761 581 420 898 437 5
8 388 607		23	.999 999 880 790 710 449 218 75
16 777 215	7	24	.999 999 940 395 355 244 609 375
33 554 431		25	.999 999 970 197 677 612 304 687 5
67 108 863		26	.999 999 985 098 838 806 152 343 75
134 217 727	8	27	.999 999 992 549 419 403 076 171 875
268 435 455		28	.999 999 996 274 709 701 538 085 937 5
536 870 911		29	.999 999 998 137 354 850 769 042 968 75
1 073 741 823	9	30	.999 999 999 088 677 425 384 521 484 375
2 147 483 647		31	.999 999 999 534 338 712 692 260 742 187 5
4 294 967 295		32	.999 999 999 767 169 356 346 130 371 093 75
8 589 934 591		33	.999 999 999 883 584 678 173 065 185 546 875
17 179 869 183	10	34	.999 999 999 941 792 339 086 532 592 773 437 5
34 359 738 387		35	.999 999 999 970 898 169 543 266 296 386 718 75
68 719 476 735		36	.999 999 999 985 448 034 771 833 148 193 359 375
137 438 953 471	11	37	.999 999 999 992 724 042 385 816 574 096 679 687 5
274 877 906 943		38	.999 999 999 996 362 021 192 908 287 048 339 843 75
549 755 813 887		39	.999 999 999 998 181 010 596 454 143 524 189 921 875
1 099 511 627 775	12	40	.999 999 999 999 090 505 298 227 071 762 084 960 937 5
2 199 023 255 551		41	.999 999 999 999 545 252 649 113 535 881 042 480 468 75
4 398 046 511 103		42	.999 999 999 999 772 626 324 556 767 940 521 240 234 375
8 796 093 022 207		43	.999 999 999 999 886 313 162 278 383 970 260 620 117 187 5
17 592 186 044 415	13	44	.999 999 999 999 943 156 581 139 191 985 130 310 058 593 75
35 184 372 088 831		45	.999 999 999 999 971 578 290 569 595 992 565 155 029 296 875
70 368 744 177 663		46	.999 999 999 999 985 789 145 284 797 996 282 577 514 648 437 5
140 737 488 355 327	14	47	.999 999 999 999 992 894 572 642 398 998 141 288 757 324 218 75
281 474 976 710 655		48	.999 999 999 999 996 447 286 321 199 499 070 644 378 662 109 375
562 949 953 421 311		49	.999 999 999 999 998 223 643 160 599 749 535 322 189 331 054 687 5
1 125 899 906 842 623	15	50	.999 999 999 999 999 111 821 580 299 874 767 661 094 665 527 343 75
2 251 799 813 685 247		51	.999 999 999 999 999 655 910 790 149 937 383 830 547 332 763 671 875
4 503 599 627 370 495		52	.999 999 999 999 999 777 955 395 074 968 691 915 273 666 381 835 937 5
9 007 199 254 740 991		53	.999 999 999 999 999 888 977 697 537 484 345 957 636 833 190 917 968 75
18 014 398 509 481 983	16	54	.999 999 999 999 999 944 488 848 768 742 172 978 818 416 595 458 984 375
36 028 797 018 963 967		55	.999 999 999 999 999 972 244 424 384 371 086 489 409 208 297 729 492 187 5
72 057 594 037 927 935		56	.999 999 999 999 999 986 122 212 192 185 543 244 704 604 148 864 746 093 75
144 115 188 075 855 871	17	57	.999 999 999 999 999 993 061 106 096 092 771 622 352 302 074 432 373 046 875
288 230 376 151 711 743		58	.999 999 999 999 999 996 530 553 048 046 385 811 176 151 037 216 186 523 437 5
576 460 752 303 423 487		59	.999 999 999 999 999 998 265 276 524 023 192 905 588 075 518 608 093 261 718 75
1 152 921 504 606 846 975	18	60	.999 999 999 999 999 999 132 638 262 011 596 452 794 037 759 304 046 630 859 375

*Larger numbers within a digit group should be checked for exact number of decimal digits required.

Examples of use:

1. Q. What is the largest decimal value that can be expressed by 36 binary digits?
A. 68,719,476,735.
2. Q. How many decimal digits will be required to express a 22-bit number?
A. 7 decimal digits.

binary, PGS, and data format D

Object programs for the Series-16 can exist in either of two formats, Binary or Program Generation System (PGS). Either format is valid for cards or paper tape. (Data and source language programs are in ASCII format.)

BINARY FORMAT

Binary format is simply a stream of bytes as they appear in memory. These bytes carry no control or checksum information. See Figures D-1 and D-2 for the paper tape and card representation of the Binary format.

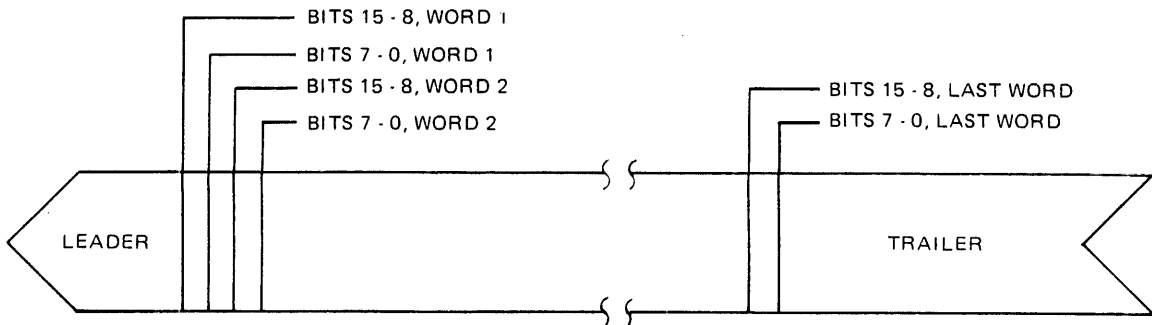
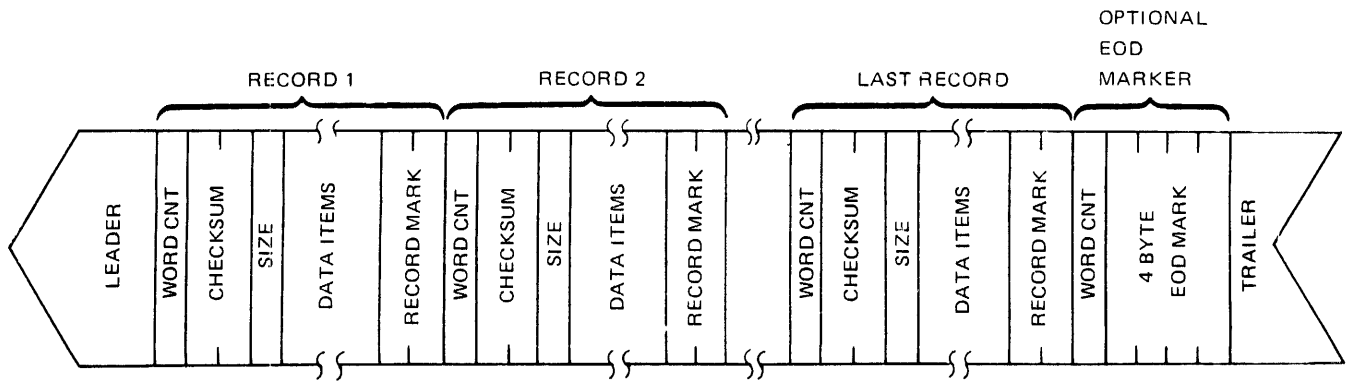


Figure D-1. Binary Paper Tape Format



NOTE: OPTIONAL EOD MARKER IS A FOUR-BYTE ENTRY CONTAINING 44281040₁₆

Figure D-2. PGS Paper Tape Format

PGS format is the universal Series-16 object program representation. Object programs may be either absolute or relocatable. All program loaders read PGS format (except HSPFL) and all object output is punched in PGS format (BUS-16 can punch Binary). The major divisions of PGS data is the 54 word (108 byte) record. Within a record are data items consisting of a control byte and its associated data bytes. See Figures D-3 and D-4.

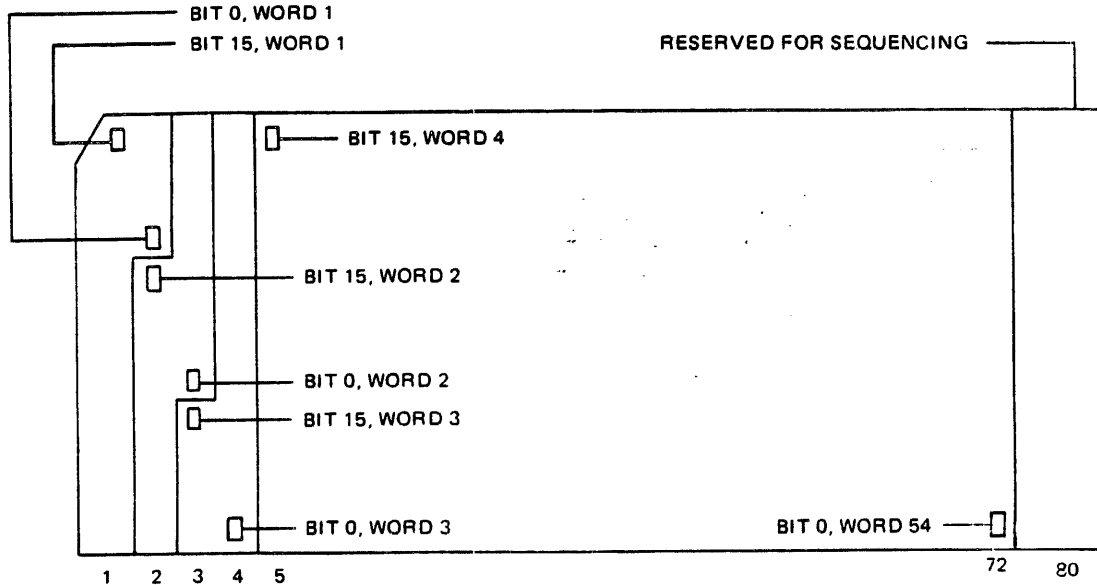
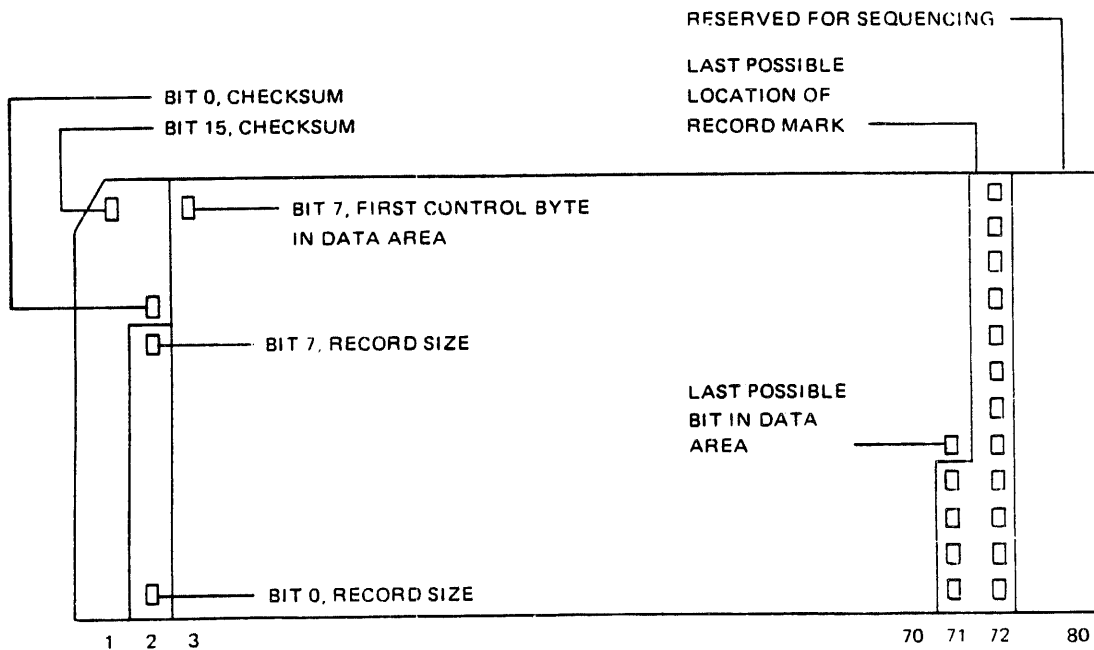


Figure D-3. Binary Card Format



NOTE: ONE RECORD PER CARD REGARDLESS OF LENGTH

466-D-2

Figure D-4. PGS Card Format

PGS RECORD

The PGS record consists of up to 54 words (108 bytes) of information and appears as follows:



466-D-3

where:

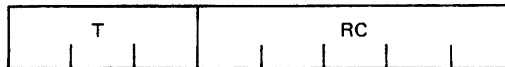
- Checksum is the one-word result of a summation of the remainder of the record up to the record mark.
- Data size is a one-byte count of the number of bytes in the data area. This may be a maximum of 103_{16} (67_{16}).
- Data area contains the data items and may contain up to 103_{16} (67_{16}) bytes.
- Record mark is a 2-byte record terminator containing $E7E7_{16}$ (on older tapes this terminator is $FFFF_{16}$).

Each record is preceded by a one-byte word count on physical paper tape only, indicating the number of words in the record. The maximum count, per record, is 54_{16} (36_{16}).

PGS DATA ITEMS

The Data Area in a PGS record is made up of individual data items. Each data item consists of a control byte followed by one or more data bytes.

A control byte defines the type of data item and appears as:



466-D-4

where:

- T is a 3-bit value indicating the type of item.
- RC is a 5-bit field used to further define the item.

88A00508A-E

When T=0, 1, 2, 5 or 7, the RC field is divided into a 3-bit R field and a 2-bit C field as follows:



where:

- R=0 Absolute
- 1 COMMON
- 2 Program (PSECT)
- 3 GLOBAL
- 4 Data (DSECT)
- 5 Unused
- 6 External

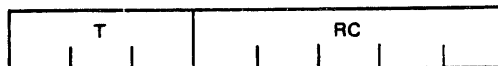
C = number of data bytes-1 in the item.

When T=3 or 4, the RC field is divided into a 2-bit R field and a 3-bit C field as follows:



where R and C have meaning as described above.

When T=6, the RC field is a 5-bit field:



where:

466-D-5

- RC = 0 A secondary control byte follows.
- RC ≠ 0 This field contains a double word count of absolute data items that follow the control byte.

CONTROL BYTE TYPES

- T=0 Data - from one to four data bytes follow depending on field C. They are loaded according to the relocation specified in field R.
- T=1 Define Origin - two data bytes follow which replace the contents of the loading location counter.
- T=2 Define PSECT - two data bytes follow which replace the contents of the PSECT relocation base.

T=3 External Reference - data bytes which follow are an external reference by name whose length in bytes is equal to C+1. R defines the type of reference as follows:

- R=0 Primary reference (REF)
- R=1 Secondary reference (SREF)
- R=2 Labeled COMMON

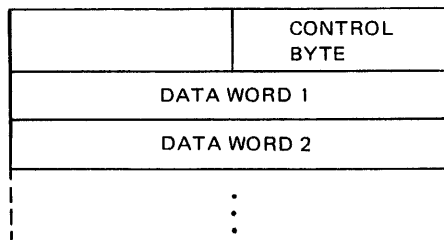
T=4 External Definition - the data bytes which follow contain the name of an entry point in the program being loaded followed by the 2-byte address of the entry point. C-1 equals number of bytes in the name. C+1 equals the total number of data bytes in the item. The R field signifies the relocation base of the address as follows:

- R=0 Absolute
- R=1 Program (PSECT) Relocatable
- R=2 Data (DSECT) Relocatable

T=5 Define DSECT - two data bytes follow which replace the contents of the DSECT relocation base. If R=6, this value must be zero.

- T=6
- a. RC = 0 Special - see following section.
 - b. RC ≠ 0

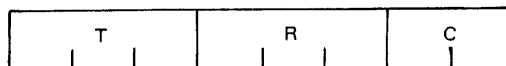
n absolute data words follow, where $n = (RC) * 2$. All data words in the count are contained in the same record. This type of control byte always appears as the right-hand byte of the preceding word:



T=7 End item - signifies the end of object text. If C=3, the 2-byte value following is the execution address of the program.

SECONDARY CONTROL BYTE TYPES

When the item type is T=6, RC=0, the primary control byte is followed by a secondary control byte which specifies one of six functions. This secondary control byte also has a 3-bit type field, and except as indicated, R and C fields are defined as for primary control bytes.



Secondary Control Byte types:

1. $T_s=0$ R is not significant
C=0 Byte addressing mode
C=1 Word addressing mode
C=2 End of core resident (main) module
C=3 The current sector number for writing overlays is stored at the current location counter value.

2. $T_s=1$ Address Chain - The value following this control byte is the starting address in a chain. The core location at the starting address contains the next address in the chain. This chain is followed until some location in the chain has contents zero, signaling the end of the chain. Into bits 14-0 of each location in the chain is placed the current value of the loading location counter. (This value may be set by an ORG item before the address chain.)

R is the appropriate relocation specification as for primary control bytes.

C is the length of the following item (=1).

3. $T_s=2$ Deflection Chain - The value following this control byte is the starting address in a chain. Bits 7-0 of the starting location contain a displacement back to the next item in the chain. The chain continues until bits 7-0 of some location in the chain have contents zero, signaling the end of the chain. Each link of the chain is resolved by placing the difference between the current location counter value and chain address into bits 7-0 of the chain location.

R is the appropriate relocation specification as for primary control bytes.

C is the length of the following item (=1).

4. $T_s=3$ Define COMMON, dynamic storage, or GLOBAL size - The value following gives the absolute size of the COMMON, dynamic, or GLOBAL block specified by R (R=1 COMMON, R=2 dynamic storage, R=3 GLOBAL). The program with the largest COMMON requirement must be loaded first. When R=1 or 3, this item type is always preceded by an item type T=3 (External Reference).

C is length of the following item.

5. $T_s=4$ Memory Module Select - The value following is a data word for extended memory mapping. This data word is output via an XIO instruction specifying the Extended Memory device select code (X'39').

R is not significant.

C is length of the following item (=1).

6. $T_s=5$ Not used.
7. $T_s=6$ Write Overlay to Bulk - The two addresses following are the beginning and ending addresses of the overlay module. The overlay module is written to bulk beginning at the current (first available) sector in the scratch file. The current sector number is then updated to the next available sector.

R is the appropriate relocation specification as for primary control bytes.

C is length of the following item (=3).

8. $T_s=7$ Not used.

ASCII PAPER TAPE FORMAT

ASCII records contained on paper tape (or input from the teletype keyboard) consist of a stream of ASCII characters with special form and record control characters. An ASCII record has the following format:

Line Feed	Start of record
ASCII Characters	Characters within record
Carriage Return	End of record indicator

Any amount of leader may exist prior to the line feed and after the Carriage Return characters. Within a record the following characters have special meaning:

Line Feed	Ignored
Rubout	Delete the entire record and start a new record
←(Back arrow)	Delete the previous character. Multiple ← will delete as many characters as the number of ←.

summary of **E** system console ROM commands

SYMBOLS

a - 4-digit hexadecimal (hex) address (leading zero's may be omitted)

b - beginning address

e - Ending address

n - hex number or pattern

r - Register number (A X Y Z B C D E S)
0 1 2 3 4 5 6 7 8

␣ - space bar

(cr) RETURN key

If you make a mistake, RUBOUT erases the entire command

Command Mnemonic	Function	Format, Example & Comments
BREAK	Interrupt CPU and gain access to Console ROM ¹	Press BREAK key on TTY or CRT
R	Display all registers	R (cr) (You cannot change any registers)
R	1. Display specified register 2. Change register	rR (cr) 3R (cr) Display Z register. Then ␣ display next reg. Simply type new hex value after desired register display. Then ␣ enters it and displays next register or (cr) enters it and terminates.
(cr)	1. Display memory location	a (cr) 3FC0 (cr) Both the address and its contents appear on CRT.

¹Requires CPU-2 switches preset as follows:

BKDS switch not in BKDS position (Break enabled)

BKINT switch in BKINT position (Break interrupt enabled)

Command Mnemonic	Function	Format, Example & Comments
	2. Change memory location	Simply type new hex value after the desired location. Then, \backslash enters it and displays next location - or (cr) enters it and terminates.
M	Display block of memory	b/eM (cr) 10/22M (cr) Display memory from hex 0010 to 0022. (You cannot modify memory)
Z	Store a pattern in a block of memory	b/e/nZ (cr) 23AB/24AB/1111Z (cr) Store all "1's" from 23AB to 24AB.
S	Single step	1. aS (cr) 60AS (cr) Execute instruction at 060A and return to Console ROM. 2. S (cr) Execute instruction following last single step or at last Trap, whichever occurred last.
T	Set TRAP	1. aT (cr) 1000T (cr) Set Trap at hex 1000. 2. T (cr) Remove Trap
	Clear TRAP	
G	Go to specified address and start execution.	1. aG (cr) 100G (cr) Start execution at hex 0100 2. G (cr) Start execution at last TRAP or after last single step.
Y	Set bias for relative addressing	aY (cr) 3000Y (cr) Bias to be used in relative addressing is 3000.
A	Address modes used if a bias has been set for relative addressing Absolute	100S,A (cr) Single step at absolute address 0100; ignore the bias.
B	Relative	100S,B (cr) Single step at relative address 0100 plus bias.

Command Mnemonic	Function	Format, Example & Comments
There are 3 other console functions which permit resetting I/O and which permit loading binary tapes (such as a bootstrap loader or punching binary tape).		
L	Load absolute binary tape	aL (cr) Load from TTY or PTR starting at specified addr.
I	Punch absolute binary tape	b/eI (cr) Output to TTY or PTP the specified block of memory.
!	I/O reset	!(cr) Initialize all peripheral controllers.

early GA-16/110 and GA-16/220 configurations **F**

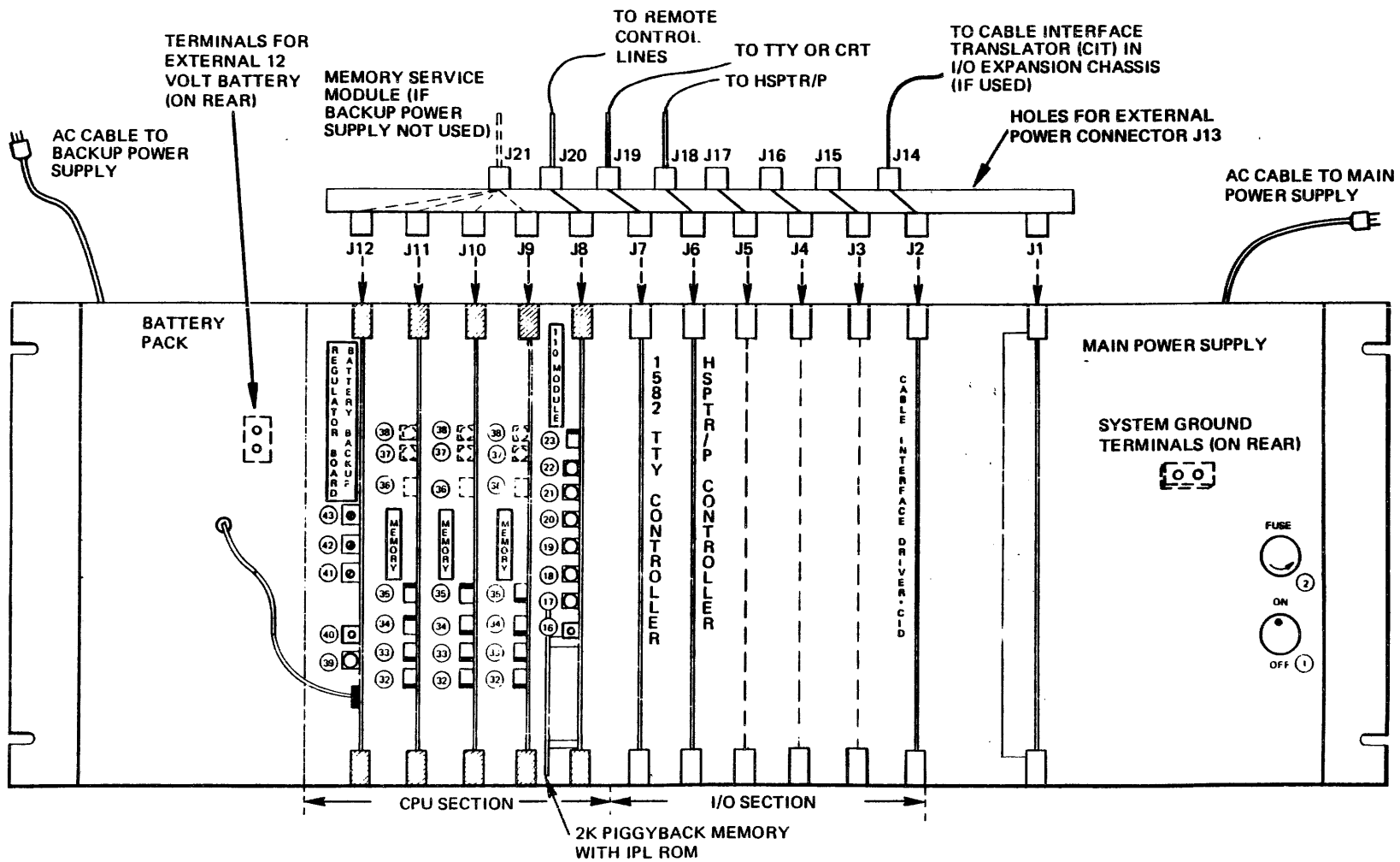
Figure F-1 and F-2 show the configurations of the compact chassis built prior to December 1976. The major changes made in currently shipped units provide additional space between CPU-1 and CPU-2 modules to accommodate a re-designed SCI module. (See Section 3, Figure 3-2.) Table F-1 shows connector assignments for early compact chassis in both GA-16/110 and GA-16/220 configurations.

Figures F-3 and F-4 show the configurations of jumbo chassis built prior to October 1976. To provide more space between CPU-1 and CPU-2 modules to accommodate redesigned SCI module (See Section 3, Figure 3-4), an I/O slot has been eliminated in currently shipped units. Table F-2 shows connector assignments for early jumbo chassis in both GA-16/110 and GA-16/220 configurations.

The circled numbers are keyed to Table 3-3 in Section 3 for descriptions of controls and indicators.

Figure F-5 shows the priority interrupt chain wiring for early compact and jumbo chassis.

F-2



88A00508A-E

508-F-1

Figure F-1. GA-16/110 System in Compact Chassis (No MPP)

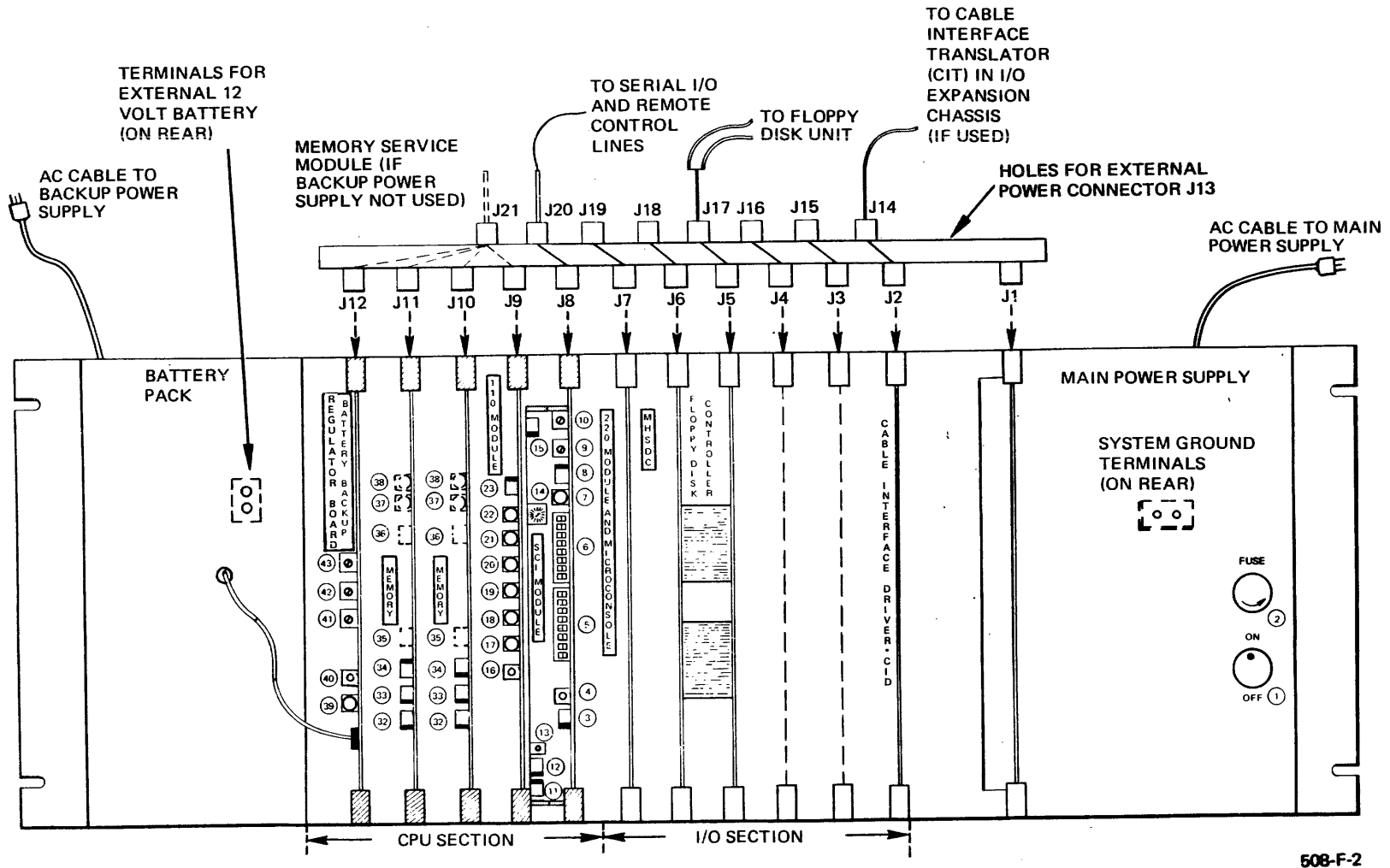
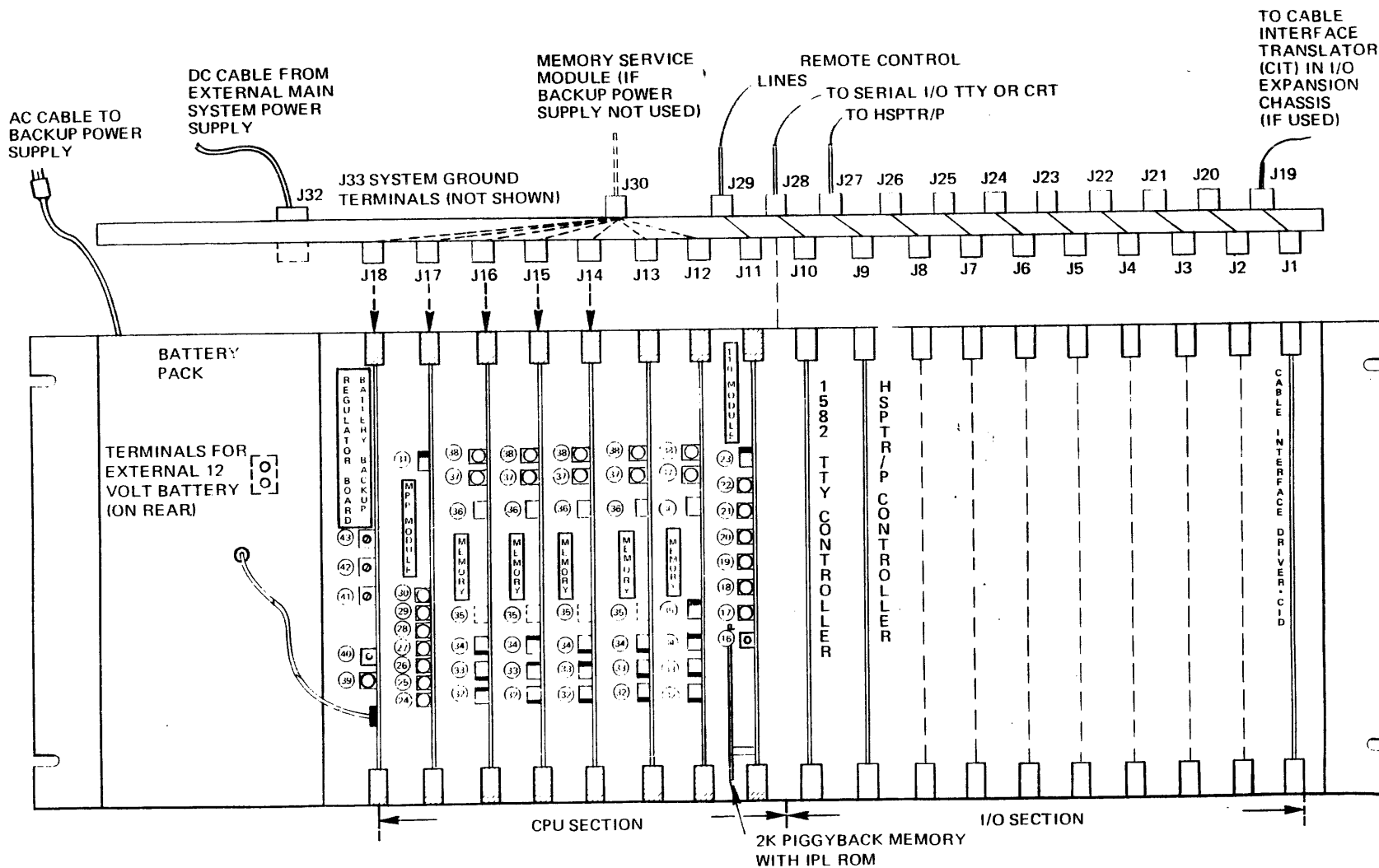


Figure F-2. GA-16/220 System in Compact Chassis (No MPP)

Table F-1. Connector Assignments for Compact Chassis Computer System

Front/Module (110) (See Figure F-1)		Front/Module (220) (See Figure F-2)	
J1	Main power supply	J1	Main power supply
J2	Cable interface driver (CID)	J2	Cable interface driver (CID)
J3	I/O controllers or shorting boards. Shorting boards are used in empty slots between controllers or CID to maintain prior chain continuity.	J3	I/O controllers or shorting boards. Shorting boards are used in empty slots between controllers or CID to maintain interrupt chain continuity.
J4		J4	
J5		J5	
J6		J6	
J7	CPU-1 module with optional piggyback memory and IPL ROM.	J7	MHSDC controller, if used, should go into J7.
J8	4K/8K memory module, or Memory Parity Protect module.	J8	CPU-2 module with optional System Console Interface.
J9		J9	CPU-1 module with optional piggyback memory.
J10	J12 also used for battery backup regulator module. Backup power supply battery pack (use regulator board in J12).	J10	4K/8K memory module, or Memory Parity Protect module. J12 also used for battery backup regulator module. Backup power supply battery pack (use regulator board in J12).
J11		J11	
J12		J12	
-		-	
Rear/Interface (110)		Rear/Interface (220)	
-	AC input connector and system grounding terminals on power supply.	-	AC input connector and system grounding terminals on power supply.
J13	Holes for optional external power supply connector.	J13	Holes for optional external power supply connector.
J14	Cables to I/O expansion chassis (6-feet maximum).	J14	Cable to I/O expansion chassis (6-feet maximum).
J15	Paddleboards and cables interfacing controllers (in J3 through J7) to peripheral devices. Access to cold-start (CLDS-) memory mode (64KM+) and remote control lines (IPLSW-, PFD-, RSET-, RUN-).	J15	Paddleboards and cables interfacing controllers (in J3 through J7) to peripheral devices. Serial I/O paddleboard with connector to TTY or CRT also provides access to cold-start (CLDS-) jumpers and remote control lines (IPLSW-, PFD-, RSET-, RUN-).
J16		J16	
J17		J17	
J18		J18	
J19		J19	
J20	Memory Service Module if backup power is not used.	J20	Memory Service Module if auxiliary power supply is not used.
-	AC cord for backup power supply.	-	AC cord for backup power supply.
-	Terminals for external 12-volt battery on rear of battery pack.	-	Terminals for external 12-volt battery on rear of battery pack.

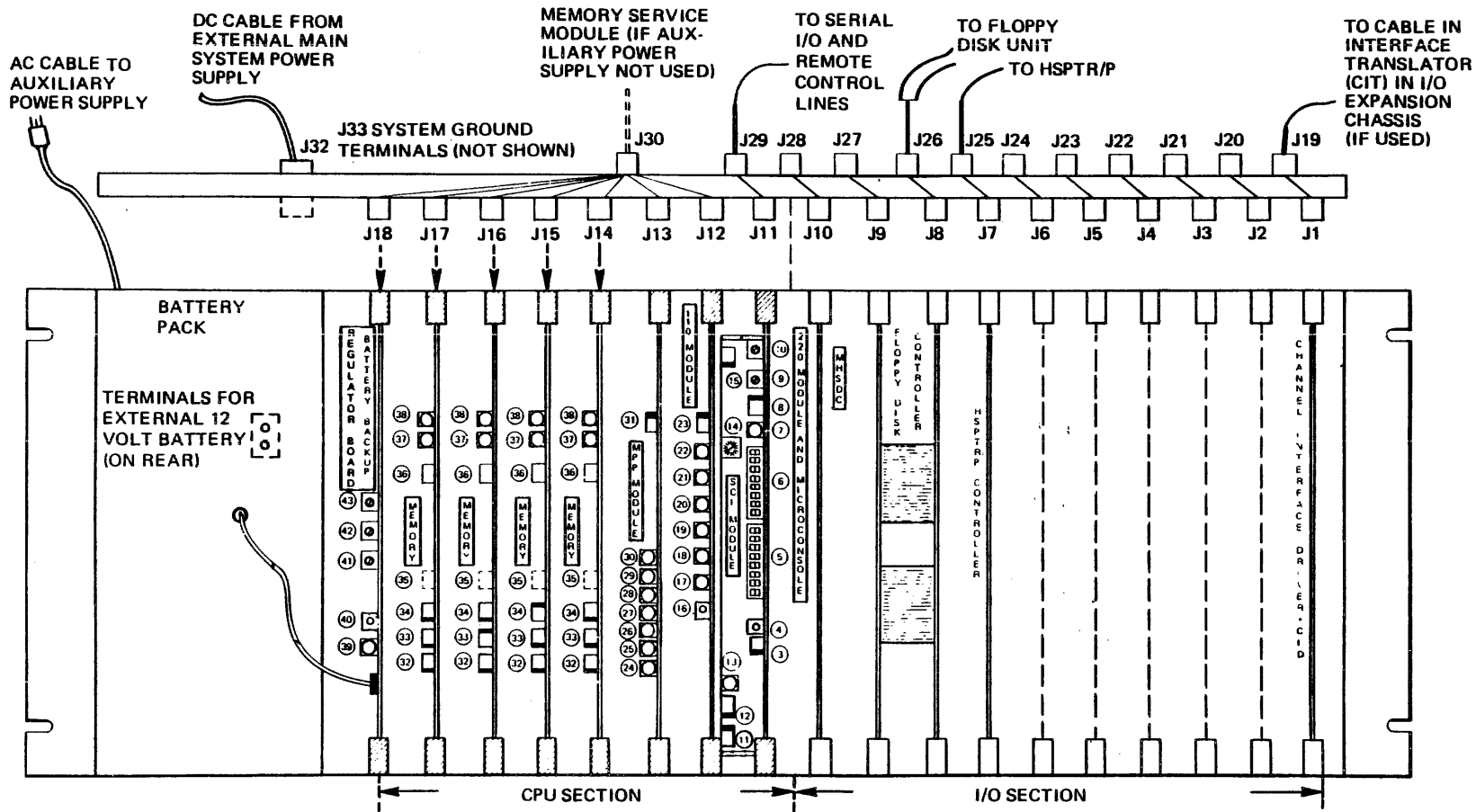
F-5



88A00508A-E

508-F-3

Figure F-3. GA-16/110 System in Jumbo Chassis (with MPP)



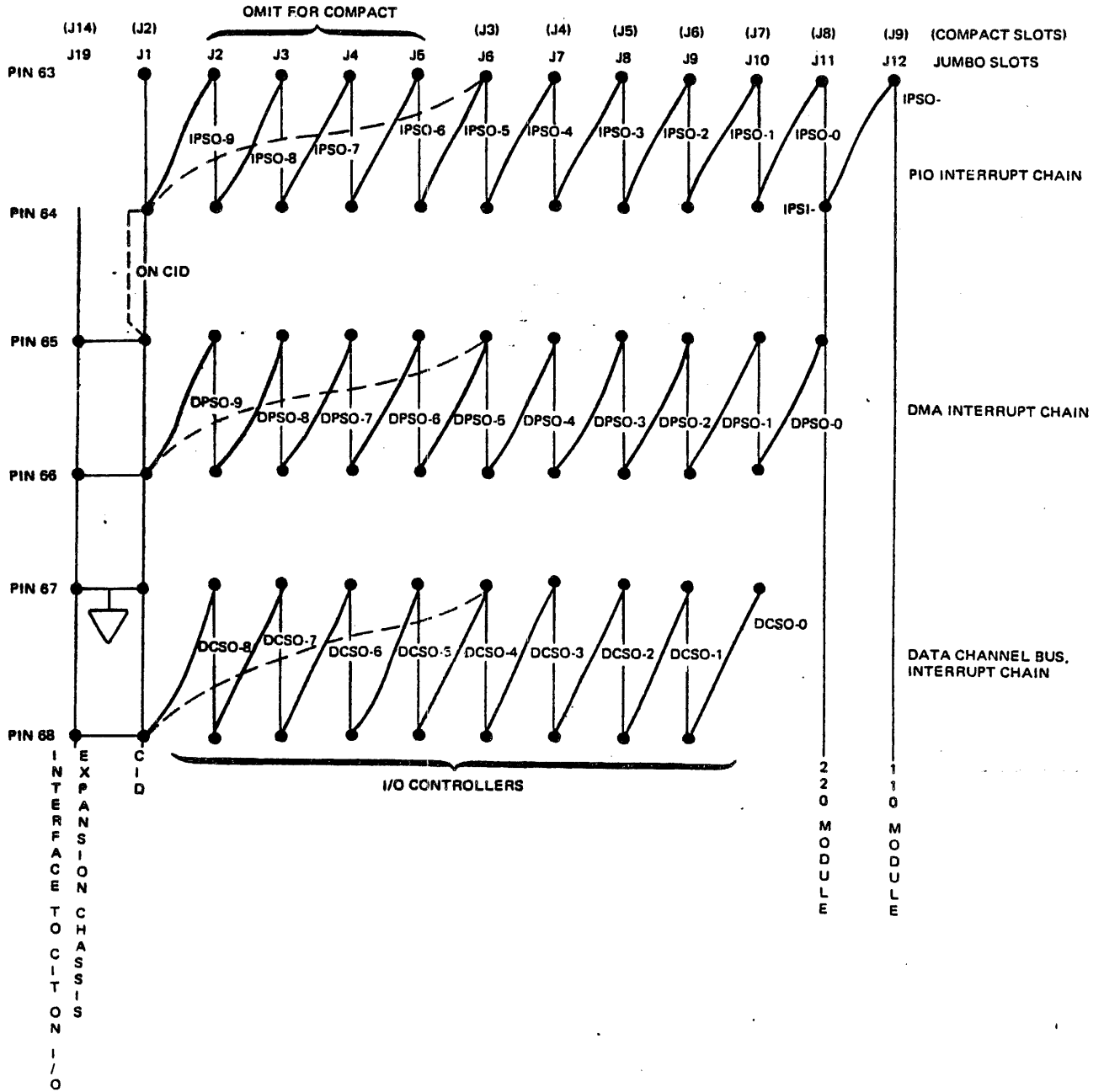
508-F-4

88A00508A-E

Figure F-4. GA-16/220 System in Jumbo Chassis (with MPP)

Table F-2. Connector Assignments for Jumbo Chassis Computer System

Front/Module (110) (See Figure F-3)		Front/Module (220) (See Figure F-4)	
J1	Cable interface driver (CID)	J1	Cable interface driver (CID).
J2	I/O controllers or shorting boards. Shorting boards are used in empty slots between controllers or CID to maintain interrupt chain continuity.	J2	I/O controllers or shorting boards. Shorting boards are used in empty slots between controllers or CID to maintain interrupt chain continuity. MHSDC controller, if used, should go into J10.
J3		J3	
J4		J4	
J5		J5	
J6		J6	
J7		J7	
J8		J8	
J9		J9	
J10		J10	
J11	CPU-1 module with optional piggy-back memory and IPL ROM.	J11	CPU-2 module with optional System Console Interface.
J12	4K/8K memory module (can't be serviced properly by MPP).	J12	CPU-1 module with optional piggy-back memory.
J13	4K/8K memory module, or memory parity protect module. J18 also used for battery backup regulator module.	J13	4K/8K memory module, or memory parity protect module. J18 also used for battery backup regulator module.
J14		J14	
J15		J15	
J16		J16	
J17		J17	
J18		J18	
-	Backup power supply battery pack (use regulator board in J18).	-	Backup power supply battery pack (use regulator board in J18).
Rear/Interface (110)		Rear/Interface (220)	
J19	Cable to I/O expansion chassis, (6-feet maximum). Paddleboards and cables interfacing I/O controllers (in J2 thru J10) to peripheral devices.	J19	Cable to I/O expansion chassis (6-feet maximum). Paddleboards and cables interfacing I/O controllers (in J1 thru J10) to peripheral devices
J20	Access to cold-start (CLDS-) memory mode (64KM+), and remote control lines (IPLSW-, PFD-, RSET-, RUN-).	J20	Serial I/O paddleboard with connector to TTY or CRT. Also provides access to cold-start (CLDS-) jumper, and remote control lines (IPLSW-, PFD-, RSET-, RUN-).
J21		J21	
J22		J22	
J23		J23	
J24		J24	
J25		J25	
J26		J26	
J27		J27	
J28		J28	
J29		J29	
J30	Memory Service Module if backup power supply is not used.	J30	Memory Service Module if auxiliary power supply is not used.
J32	Power cable from external power supply.	J32	Power cable from external power supply.
J33	System grounding terminals.	J33	System grounding terminals.
-	AC cord for backup power supply.	-	AC cord for backup power supply.
-	Terminals for external 12-volt battery on rear of battery pack.	-	Terminals for external 12-volt battery on rear of battery pack.



redesigned **G** GA-16/110/220 processor boards

This appendix provides a description of the newly designed GA-16/110 (CPU-1) and GA-16/220 (CPU-2) processor boards. These boards can be identified by the following assembly numbers:

- CPU-1 Board - 31D02573A
- CPU-2 Board - 31D02574A

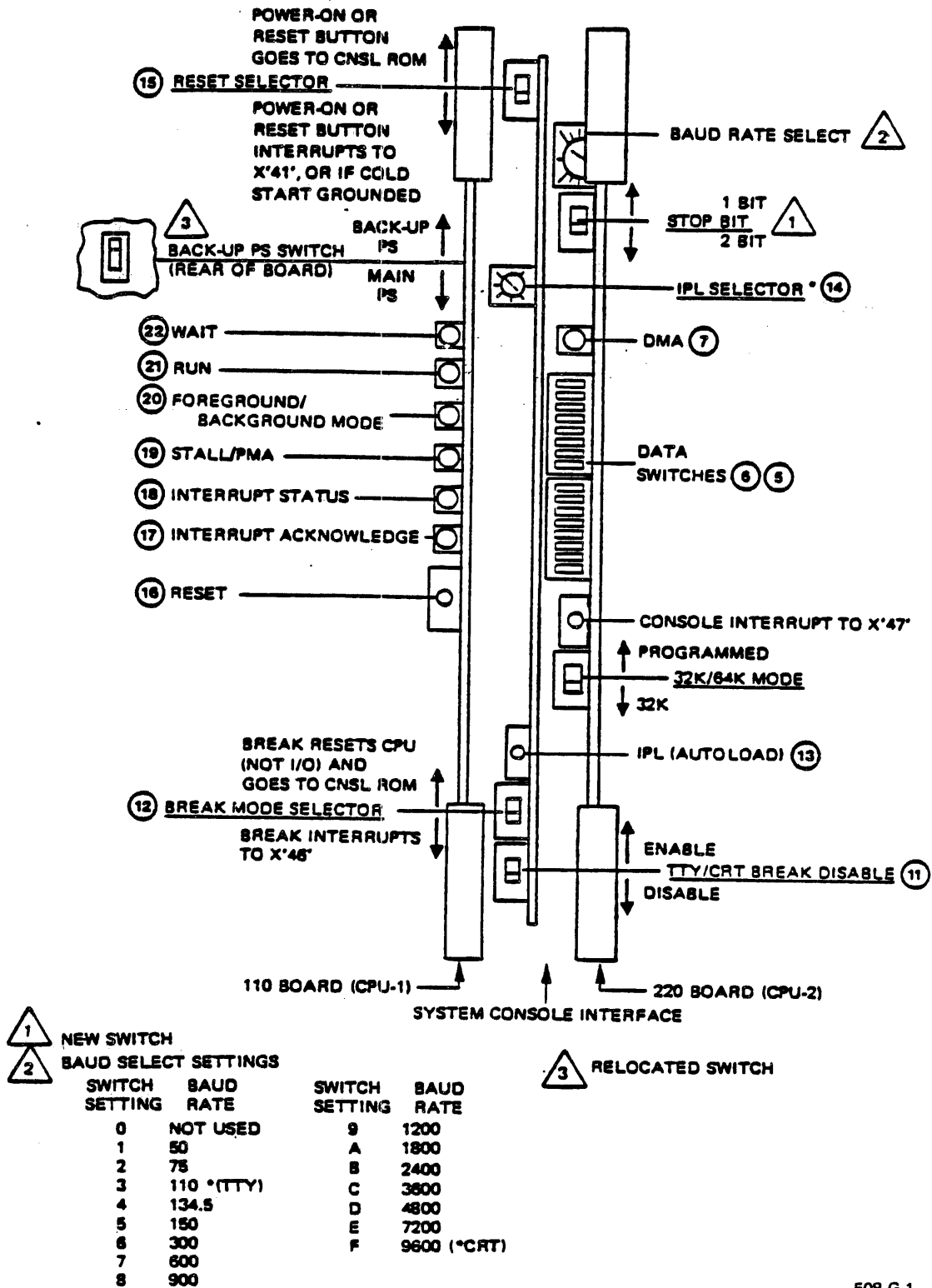
The main change in the CPU-1 board is the relocation of the Battery Back-Up Power Supply Switch which is now located between board coordinates D1 and E1. When this switch is in the up position, +5VB power for the memories originates from the back-up power supply or the batteries (when AC power is disconnected). When the switch is in the down position, memory power is supplied by the main power supply via the memory service module.

NOTE

The memory service module must be used when the switch is in the down position.

The main change in the CPU-2 board is the removal of the two-position 110/9600 baud switch. This switch was replaced by a rotary switch which provides fifteen selectable baud rates. A STOP BIT select switch has also been added which allows a selection of a one- or two-bit "STOP BIT."

Figure G-1 provides a composite view of the CPU-1 and CPU-2 boards containing these switches. The selectable baud rates are also shown in Figure G-1. The remaining switches and indicators have not been changed and can be found in Table 3-3 next to the key numbers (circled).



508-G-1

Figure G-1. Redesigned GA-16/110/220 Processor Boards

COMMENT SHEET

Document No. _____

Title: _____

Page: _____

FROM:

NAME: _____

BUSINESS ADDRESS: _____

Does this publication meet your requirements?

Yes

No

If no, please explain. _____

Do you wish a reply?

Yes

No

COMMENTS: (Describe any errors, suggested ideas, additions, or deletions etc. Please include page number.)

All comments and suggestions become the property of General Automation, Inc.

No Postage Stamp Necessary If Mailed In the U.S.A.

(See Other Side)

Fold On Dotted Lines And Staple

YOUR COMMENTS, PLEASE . . .

This publication serves as a reference for systems analysts, programmers and operators of General Automation systems. Your answers to the questions on the back of this form help us produce better publications for your use.

FOLD

First Class
Permit No. 423
Anaheim, California

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

**General Automation, Inc.
1055 South East Street
Anaheim, California 92803**

Attention: Publications Services

FOLD

General Automation, Inc.
1055 South East Street
Anaheim, California 92803

CUT ALONG · NE

GENERAL AUTOMATION

1055 SOUTH EAST STREET, ANAHEIM, CA 92805

TELEPHONE: (714) 778-4800