

SPC-16/40, SPC-16/60, SPC-16/80
SPC-16/45, SPC-16/65, SPC-16/85
system reference

88A00243A-E

SPC-16/40, SPC-16/45, SPC-16/60, SPC-16/65, SPC-16/80 SPC-16/85 system reference

GENERAL AUTOMATION, INC.
1055 South East Street
Anaheim, California 92805
(714) 778-4800

© 1972, 73, 74, 75 General Automation, Inc.

RECORD OF REVISIONS

Title: SPC-16 SYSTEM REFERENCE MANUAL

Document Number: 88A00243A

DATE	ISSUE
Jan 72	A - Original Issue
March 72	B - Changed pages 6-11, 6-18, 7-3, 7-44. Corrects ROM Disk Bootstrap, adds front panel drawing and corrects pin numbers.
June 73	C - Complete Rewrite
Feb 74	D - Changes to RCSR and RCSM Instructions.
Aug 75	E - Complete Revision

CONTENTS

Section	Title	Page
1	INTRODUCTION	1-1
2	SYSTEM ORGANIZATION	2-1
	2.1 Word Format	2-2
	2.1.1 Data Format	2-2
	2.1.2 Instruction Format	2-7
	2.2 Elements of the Central Processing Unit	2-9
	2.3 Addressing Modes	2-17
	2.3.1 Effective Address Generation - Stage 1	2-18
	2.3.2 Effective Address Generation - Stage 2	2-23
	2.3.3 Effective Address Generation - Stage 3	2-25
	2.4 Input/Output System	2-28
3	INSTRUCTION REPERTOIRE	3-1
	3.1 Memory Reference Instructions	3-6
	3.1.1 Load Register A (LDA)	3-9
	3.1.2 Store Register A (STA)	3-10
	3.1.3 Jump Unconditionally (JMP)	3-10
	3.1.4 Jump to Subroutine (JSR)	3-11
	3.2 Memory Reference with Indexing Instructions	3-12
	3.2.1 Load Register (LDR)	3-15
	3.2.2 Store Register (STR)	3-18
	3.2.3 Compare Memory with Register (CMR)	3-18
	3.2.4 Load Byte (LDBY)	3-19
	3.2.5 Store Byte (STBY)	3-20
	3.2.6 Load All Registers and Status (LARS)	3-20
	3.2.7 Store All Registers and Status (SARS)	3-21
	3.2.8 Increment Memory (INCM)	3-22
	3.2.9 Decrement Memory (DECM)	3-23
	3.2.10 Set Bit (SBIT)	3-23
	3.2.11 Reset Bit (RBIT)	3-24
	3.2.12 Test Bit (TBIT)	3-25
	3.2.13 Examples	3-26
	3.3 Conditional Jump Instructions (SKIP)	3-29
	3.3.1 Skip if Zero (SKZ)	3-32
	3.3.2 Skip If Not Zero (SKN)	3-32
	3.3.3 Skip If Plus (SKP)	3-32
	3.3.4 Skip If Minus (SKM)	3-33
	3.3.5 Skip if Overflow True (SKOT)	3-33

CONTENTS (continued)

Section	Title	Page
	3.3.6 Skip If Overflow False (SKOF)	3-33
	3.3.7 Skip If Link Set (SKS)	3-33
	3.3.8 Skip if Link Reset (SKR)	3-34
	3.3.9 Examples	3-34
3.4	Register Operate and Register Operate Compare Instructions	3-35
	3.4.1 Transfer Register (RTR)	3-36
	3.4.2 Add Registers (ADD)	3-36
	3.4.3 Subtract Registers (SUB)	3-37
	3.4.4 And Registers (AND)	3-37
	3.4.5 Exclusive Or Registers (XOR)	3-38
	3.4.6 Or Registers (OR)	3-38
	3.4.7 Add Compare Registers (ADDC)	3-39
	3.4.8 Subtract Compare Registers (SUBC)	3-39
	3.4.9 And Compare Registers (ANDC)	3-40
	3.4.10 Exclusive or Compare Registers (XORC)	3-40
	3.4.11 Or Compare Registers (ORC)	3-40
	3.4.12 Examples	3-41
3.5	Register Operate Literal and Register Operate Literal Compare Instructions	3-41
	3.5.1 Load Value to Register (LDV)	3-43
	3.5.2 Add Value to Register (ADDV)	3-43
	3.5.3 Subtract Value from Register (SUBV)	3-43
	3.5.4 And Value with Register (ANDV)	3-44
	3.5.5 Exclusive or Value with Register (XORV)	3-44
	3.5.6 Or Value with Register (ORV)	3-44
	3.5.7 Add Compare Value with Register (ADDVC)	3-45
	3.5.8 Subtract Compare Value with Register (SUBVC)	3-45
	3.5.9 And Compare Value with Register (ANDVC)	3-45
	3.5.10 Exclusive or Compare Value with Register (XORVC)	3-46
	3.5.11 Or Compare Value with Register (ORVC)	3-46
	3.5.12 Examples	3-46
3.6	Register Change Instructions	3-49
	3.6.1 Increment Register (INCR)	3-50
	3.6.2 Decrement Register (DECR)	3-50
	3.6.3 Complement Register (CMPL)	3-51
	3.6.4 Zero Register (ZERO)	3-51
	3.6.5 Zero Left Byte (ZLBY)	3-52
	3.6.6 Zero Right Byte (ZRBY)	3-52
	3.6.7 Exchange Bytes (EXBY)	3-52
	3.6.8 Register Add Link (RLK)	3-53

CONTENTS (continued)

Section	Title	Page
	3.6.9 Add Shift Counter to Register (ADDS)	3-53
	3.6.10 Transfer Status to Register (TSR)	3-54
	3.6.11 Transfer Register to Status (TRS)	3-55
	3.6.12 Read Console Switches (RCSW)	3-55
	3.6.13 Display Register Contents (DSPL)	3-56
	3.6.14 Execute Register Contents (XEC)	3-57
	3.6.15 Subroutine Return (RTRN)	3-57
	3.6.16 Exit Subroutine (EXIT)	3-58
	3.6.17 Restore Interrupt System Enable (RISE)	3-58
	3.6.18 Examples	3-59
	3.7 Shift Instructions	3-61
	3.7.1 Shift Right Arithmetic (SRA)	3-62
	3.7.2 Shift Right Circular (SRC)	3-63
	3.7.3 Shift Right Logical and Count (SRLC)	3-63
	3.7.4 Shift Right Circular Through Link (SRCL)	3-64
	3.7.5 Examples	3-64
	3.8 Control Instructions	3-68
	3.8.1 Interrupt Enable (INE)	3-69
	3.8.2 Interrupt Inhibit (INH)	3-69
	3.8.3 Foreground Mode Set (FMS)	3-69
	3.8.4 Background Mode Set (BMS)	3-70
	3.8.5 Pulse Operations Monitor Alarm (PMA)	3-70
	3.8.6 Link Set (LKS)	3-71
	3.8.7 Link Reset (LKR)	3-71
	3.8.8 Generate Sync Pulse (SYNC)	3-71
	3.8.9 Wait (WAIT)	3-72
	3.8.10 Examples	3-72
	3.9 Programmed Input/Output Instructions	3-78
	3.9.1 Data Out From Register (DTOR)	3-79
	3.9.2 Data In To Register (DTIR)	3-80
	3.9.3 Data Out from Memory (DTOM)	3-80
	3.9.4 Data In To Memory (DTIM)	3-81
	3.9.5 Output Control Function (CTRL)	3-81
	3.9.6 Test Device (TEST)	3-81
	3.9.7 Read Console Switches Into Register (RCSR)	3-82
	3.9.8 Read Console Switches Into Memory (RCSM)	3-82
	3.10 Hardware Multiply and Divide (Optional)	3-83
	3.10.1 Hardware Multiply (MPY)	3-84
	3.10.2 Hardware Divide (DIV)	3-89
	3.11 Instruction Repertoire Summary	3-96

CONTENTS (continued)

Section	Title	Page
4	PERIPHERAL PRODUCT PROGRAMMING AND DESCRIPTION	4-1
	4.1 Peripheral Product Programming	4-1
	4.1.1 Programmed I/O	4-1
	4.1.2 Direct Memory Access (DMA)	4-5
	4.2 Console Teletype	4-7
	4.2.1 Operational Description	4-9
	4.2.2 Instruction Summary	4-11
	4.2.3 Programming Examples	4-12
	4.3 High Speed Paper Tape Reader	4-13
	4.3.1 Operational Description	4-13
	4.3.2 Instruction Summary	4-14
	4.3.3 Programming Examples	4-15
	4.4 High Speed Paper Tape Punch	4-18
	4.4.1 Operational Description	4-18
	4.4.2 Instruction Summary	4-19
	4.4.3 Programming Examples	4-20
	4.5 Card Reader	4-22
	4.5.1 Operational Description	4-22
	4.5.2 Instruction Summary	4-24
	4.5.3 Programming Example	4-25
	4.6 Line Printer	4-26
	4.6.1 Operational Description	4-26
	4.6.2 Instruction Summary	4-29
	4.6.3 Programming Example	4-30
	4.7 3341/3343 Disk Storage System	4-33
	4.7.1 Operational Description	4-34
	4.7.2 Instruction Summary	4-39
	4.7.3 Programming Example	4-40
	4.8 3346/3347 Disk Storage System	4-44
	4.8.1 Operational Description	4-45
	4.8.2 Instructions Summary	4-49
	4.8.3 Programming Example	4-50
	4.9 Magnetic Tape Unit (MTU)	4-54
	4.9.1 Operational Description	4-55
	4.9.2 Instruction Summary	4-60
	4.9.3 Programming Example	4-61
5	INTERFACING	5-1
	5.1 Interfacing to the I/O Bus	5-6
	5.1.1 I/O Bus Description	5-7
	5.1.2 I/O Bus Timing and Interfacing	5-14

CONTENTS (continued)

Section	Title	Page
	5.2 Interfacing to the Data Channel Bus	5-32
	5.2.1 Data Channel Bus Description	5-35
	5.2.2 Data Channel Timing and Interfacing	5-37
	5.3 Physical Description of I/O System	5-51
6	SPC-16 CONSOLE	6-1
	6.1 Switches and Indicators	6-1
	6.2 Basic Operations	6-6
	6.2.1 Displaying/Modifying Registers	6-6
	6.2.2 Displaying/Modifying Memory Locatios	6-6
	6.2.3 Manual Execution of Stored Program	6-7
	6.2.4 Initialization of Automatic Execution	6-7
	6.3 Bootstrapping	6-7
	6.3.1 Loading the Teletype Bootstrap	6-8
	6.3.2 Executing the Teletype Bootstrap	6-9
7	SPECIAL FEATURES, OPTIONS AND CAPABILITIES	7-1
	7.1 Fail-Safe Group	7-1
	7.1.1 Relative Time Clock (RTC)	7-1
	7.1.2 Operations Monitor Alarm	7-3
	7.1.3 Power-Fail Interrupt	7-3
	7.1.4 Automatic-Restart Interrupt	7-4
	7.2 Extended Memory System (64K)	7-5
	7.2.1 System Description	7-6
	7.2.2 Mechanical Description	7-9
	7.3 Extended Memory System (128K)	7-11
	7.3.1 Map Mode	7-11
	7.3.2 Direct Addressing Mode	7-15
	7.3.3 Console Indicators	7-18
	7.4 Memory Protect Option	7-19
	7.5 High Speed Signed MPY/DIV	7-21
	7.5.1 Multiplication	7-21
	7.5.2 Division	7-23
	7.6 Arithmetic Processing Unit	7-26
8	PHYSICAL PLANNING INFORMATION	8-1
	8.1 Physical Space Considerations	8-1
	8.1.1 Overall Dimensions	8-1
	8.1.2 Racking Arrangements	8-11

CONTINUED (continued)

Section	Title	Page
	8.1.3 System Cabling	8-18
	8.2 Power Requirements	8-23
	8.2.1 AC Power Sources of Domestic Use	8-23
	8.2.2 AC Power Sources for European Use	8-25
	8.3 Environmental Specifications	8-26
	8.3.1 Operating and Storage Temperatures	8-26
	8.3.2 Cooling Considerations	8-26
Appendix A	SPC-16 Instruction Summary Hexadecimal Coding	A-1
Appendix B	Standard Character Codes	B-1
Appendix C	Summary of System Software	C-1
Appendix D	Conversion Tables	D-1
Index		I-1

Figures

2-1	Operation Codes for Instruction Sheets	2-8
2-2	Internal Organization of the CPU	2-9
2-3	Address Stages for Instructions that Reference Memory	2-19
2-4	A Typical SPC-16 System	2-29
2-5	Dedicated Memory Map	2-32
3-1	EA Generation, Memory Reference Instructions	3-8
3-2	EA Generation, Memory Reference with Indexing Instructions	3-16
3-3	EA Generation, Conditonal Jump Instructions	3-31
3-4	EA Generation, Literal Addressing Mode	3-42
4-1	Formats for Masking Internal Interrupts	4-8
5-1	Typical I/O System	5-5
5-2	Flowchart of I/O Instructions	5-16
5-3	Timing and Logic for Test and Control Instructions	5-17
5-4	XIO Control or Test	5-19
5-5	Timing and Logic for Data Output Function	5-20
5-6	Timing and Logic for Data Input Function	5-21
5-7	XIO Data Output	5-22
5-8	XIO Data Input	5-23
5-9	Logic and Timing for Program Interrupt Sequence	5-25
5-10	Timing and Logic for DMA Data Output	5-28
5-11	Timing for Consecutive DMA Data Output Transfers	5-29
5-12	Timing and Logic for DMA Input or Operate-On-Memory	5-30
5-13	Timing for Consecutive DMA Data Input Transfers	5-31
5-14	Detailed Timing of Signals on I/O Bus	5-33

CONTENTS (continued)

Figures		Page
5-15	Example Block Transfer Memory Map	5-34
5-16	Timing for Data Channel Initialization Sequence	5-38
5-17	Timing and Logic for Data Channel Data Transfer (Input)	5-40
5-18	Timing for Consecutive Data Channel Data Transfers (Input)	5-41
5-19	Timing and Logic for Data Channel Data Transfer (Output)	5-43
5-20	Timing for Consecutive Data Channel Data Transfers (Output)	5-44
5-21	Data Channel/Controller Chaining Sequence	5-45
5-22	Standard Controller to Data Channel Interface Feature Logic to Accomodate all Cases.	5-47
5-23	Initiate Timing	5-48
5-24	Normal Cycle Stealing	5-49
5-25	Chain Timing	5-50
5-26	Circuit Board Layout Models 40, 60, 80; Models 45, 65, 85; External I/O Enclosure	5-52
5-27	Pin Relationship Between Controller Card and Paddle Board	5-53
5-28	Pin Relationship Between CIT-16 Board and External I/O Bus Cable	5-55
5-29	I/O Signal Connections, External I/O Enclosure	5-58
5-30	I/O Signal Connections, Integral I/O Package (Models 40, 60 and 80)	5-59
6-1	The SPC-16 Computer Console	6-2
7-1	SPC-16/45 (65, 85) Memory Expansion	7-7
7-2	SPC-16/45 (65,85) With Extended Memory System	7-10
8-1	Installation Drawing, SPC-16/40/60/80	8-2
8-2	Installation Drawing-SPC-16/45/65/85 with 0-32K Memory Capability	8-4
8-3	Installation Drawing, SPC-16/45/65/85 with Extended 32-64K Memory	8-5
8-4	Installation Drawing SPC-16/45/65 With Extended 64-128K Memroy Option	8-6
8-5	Installation Drawing 375-Watt Processor Power Supply, (Typical of CPU and I/O Power Supply)	8-7
8-6	Installation Drawing, I/O Enclosure Model 1615	8-8
8-7	Installation Drawing, Optional Mounting Slides Model 1995	8-9
8-8	Installation Drawing, System Enclosure Model 1910	8-10
8-9	1901 System Enclosure with SPC-16/40/60/80, 1615 I/O Expansion Chassis, Paper Tape Reader and Paper Tape Punch	8-12
8-10	1901 System Enclosure with SPC-16/40/60/80, 3346/3346 Disk, Paper Tape Reader and Paper Tape Punch	8-13

CONTENTS (continued)

Figures		Page
8-11	1901 System Enclosure with SPC-16/45/65/85 and 1615 I/O Expansion Chassis	8-14
8-12	1901 System Enclosure with SPC-16/45/65/85, 1615 I/O Expansion Chassis, Paper Tape Reader (Optional) and Paper Tape Punch (Optional)	8-15
8-13	1901 System Enclosure with SPC-16/45/65/85, 1615 I/O Expansion Chassis, 3346/3347 Disk, Paper Tape Reader and Paper Tape Punch	8-16
8-14	1901 System Enclosure with SPC-16/45/65/85, Two 1615 I/O Expansion Chasses and Paper Tape Reader (Optional)	8-17
8-15	Cabling of SPC-16 Processor and Power Supply	8-19
8-16	Cabling of SPC-16 System with Extended I/O Included	8-20
8-17	Cabling for SPC-16/45/65 With 128K Memory Expansion	8-22
8-18	Power Line Attachment, 110 VAC Single-Phase 60A	8-23
8-19	Power Line Attachment, 220 VAC Single-Phase 50A	8-24
8-20	Power Line Attachment, 208 VAC Three-Phase 50A	8-24
8-21	Power Line Attachment, 220 VAC Single-Phase 30A	8-25
8-22	Power Line Attachment, 440 VAC Single-Phase 30A	8-25
8-23	Power Line Attachment, 382 VAC Three-Phase 40A	8-26

Tables		
1-1	SPC-16 Specifications Summary	1-10
3-1	EA Coding, Memory Reference Instructions	3-8
3-2	EA Coding, Memory Reference with Indexing Instructions	3-15
3-3	Instruction Repertoire Summary	3-96
4-1	Standard Data Channel Assignments for Peripheral Units	4-6
4-2	Instruction Summary for Console Teletype	4-11
4-3	Instructions Summary for High Speed Paper Tape Reader	4-15
4-4	Instruction Summary for High Speed Paper Tape Punch	4-20
4-5	Instruction Summary for the Card Reader	4-25
4-6	Paper Feed Codes for Data Printer and Tally Printer	4- 28
4-7	Instruction Summary for the Line Printer	4-30
4-8	Instruction Summary for the Disk Storage System	4-39
4-9	Instruction Summary for the Disk Storage System	4-50
4-10	Instruction Summary for the Magnetic Tape Unit	4-60
5-1	Standard Device Select Codes and Interrupt Assignments	5-2
5-2	I/O Signal Pin Assignments	5-54
5-3	I/O Cable and CIT-16 Connector Pin Assignments	5-56
8-1	Maximum Combined I/O Cable Length Between I/O Enclosure and SPC-16	8-21

SECTION 1
INTRODUCTION

The SPC-16 Automation Computers are a family of fast, unusually powerful minicomputers. They offer their users application programming efficiency and versatility usually found only in much larger computers.

All computers in the SPC-16 series support a large, powerful instruction set and many programming features that are not available in most minicomputer systems. These features not only simplify the task of application program development, they permit the solution of many data processing or process control problems that are outside the competence of other small computers.

The sophisticated processing abilities of the SPC-16 computers are made possible by the extensive use of medium scale and large scale integrated circuits in their design. These high density circuits have expanded SPC-16 hardware functions well beyond traditional minicomputer limits.

General Automation also offers a complete line of off-the-shelf peripheral controllers to interface a wide selection of standard peripherals to SPC-16 computers. They include:

- Teletype
- High Speed Paper Tape Reader and/or Punch
- Card Reader and/or Punch
- Line Printer
- CRT Display Terminal
- Disk Storage Drive
- Drum Storage Drive
- Magnetic Tape Drive
- Cassette Drive
- Floppy Disk Drive

A full range of process control and communications peripherals are also available.

All SPC-16 computer systems are backed by a variety of system software packages, covering both data processing and process control applications. The operating systems currently available include:

- Teletype Only System (TTY ONLY)
- Free Standing Operating System (FSOS-16)
- Disk Based Operating System (DBOS-16)
- Real Time Executive (RTX-16)
- Real Time Operating System (RTOS-16)

THE SPC-16 SERIES

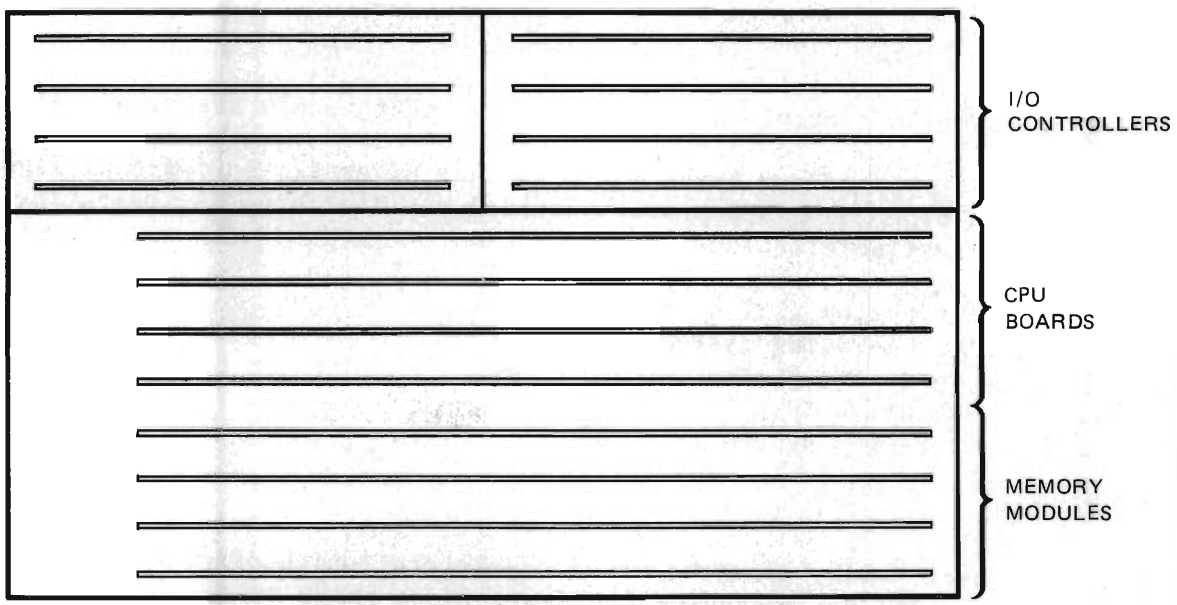
The SPC-16 processor family consists of six computers, which offer a choice of three memory cycle speeds and two packaging configurations; these are:

Computer	Cycle Speed (Nanoseconds)	Packaging Configuration
SPC-16/40	1440	Integral I/O
SPC-16/45	1440	External I/O
SPC-16/60	960	Integral I/O
SPC-16/65	960	External I/O
SPC-16/80	800	Integral I/O
SPC-16/85	800	External I/O

In those models that use the Integral I/O packaging configuration (SPC-16/40, 60 and 80), the I/O controller modules are housed in the computer mainframe. This configuration provides card slots for eight peripheral controller boards, four memory boards and four processor boards. The four processor boards are a basic requirement; the number of controller and memory boards used will depend on system requirements.

The four memory board slots allow for memory capacity of up to 32K (1K = 1024) 16-bit words. Each slot will accept any of the following types of memory modules:

- A Read/Write Memory module with 4K, 8K or 16K-word capacity. (A 16K module requires two memory board slots due to its physical size.)
- A Read/Write Memory module (4K-, 8K- or 16K-word capacity) that also includes a 32- or 64-word ROM or a Memory Protect unit (option).

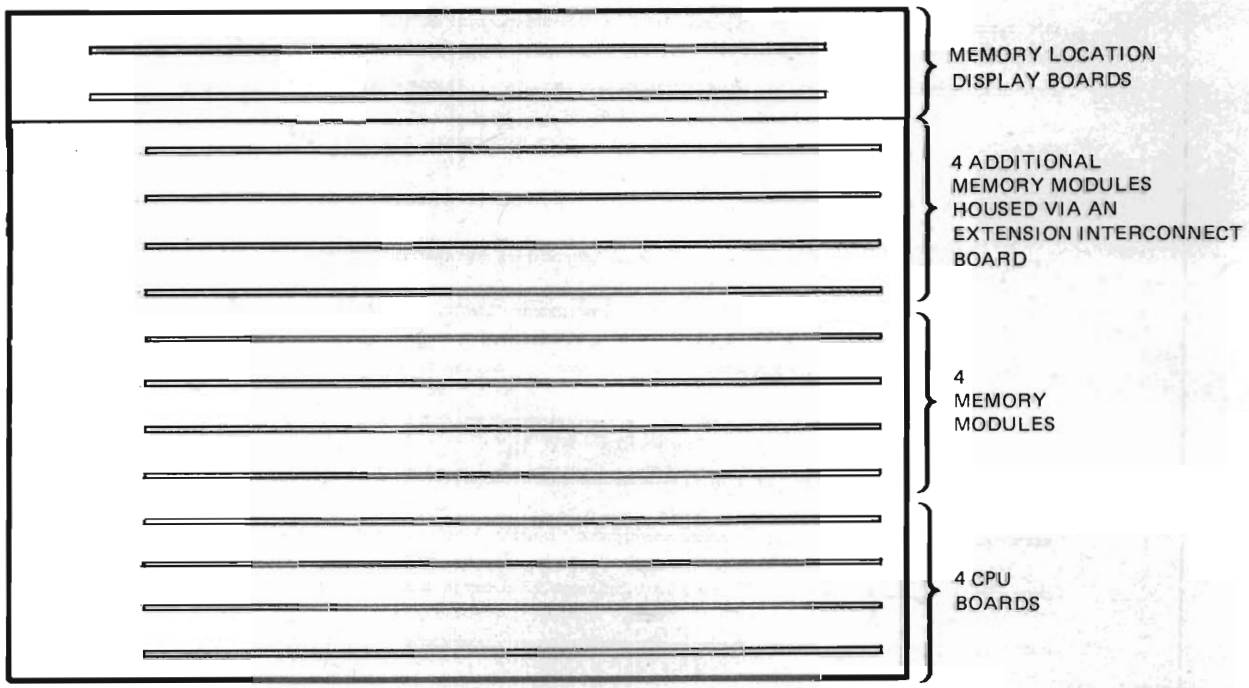


INTEGRAL I/O PACKAGING MAINFRAME CONFIGURATION
MODELS SPC-16/40, SPC-16/60, SPC-16/80 (FRONT VIEW)

If more than eight peripheral controllers are required, additional controllers can be housed in an external I/O enclosure. Each external enclosure can accommodate up to 18 periperhal controller boards; an additional card slot is reserved for a Cable Interface Translator (CIT) card. The external I/O controllers communicate with the CPU via an external I/O bus.

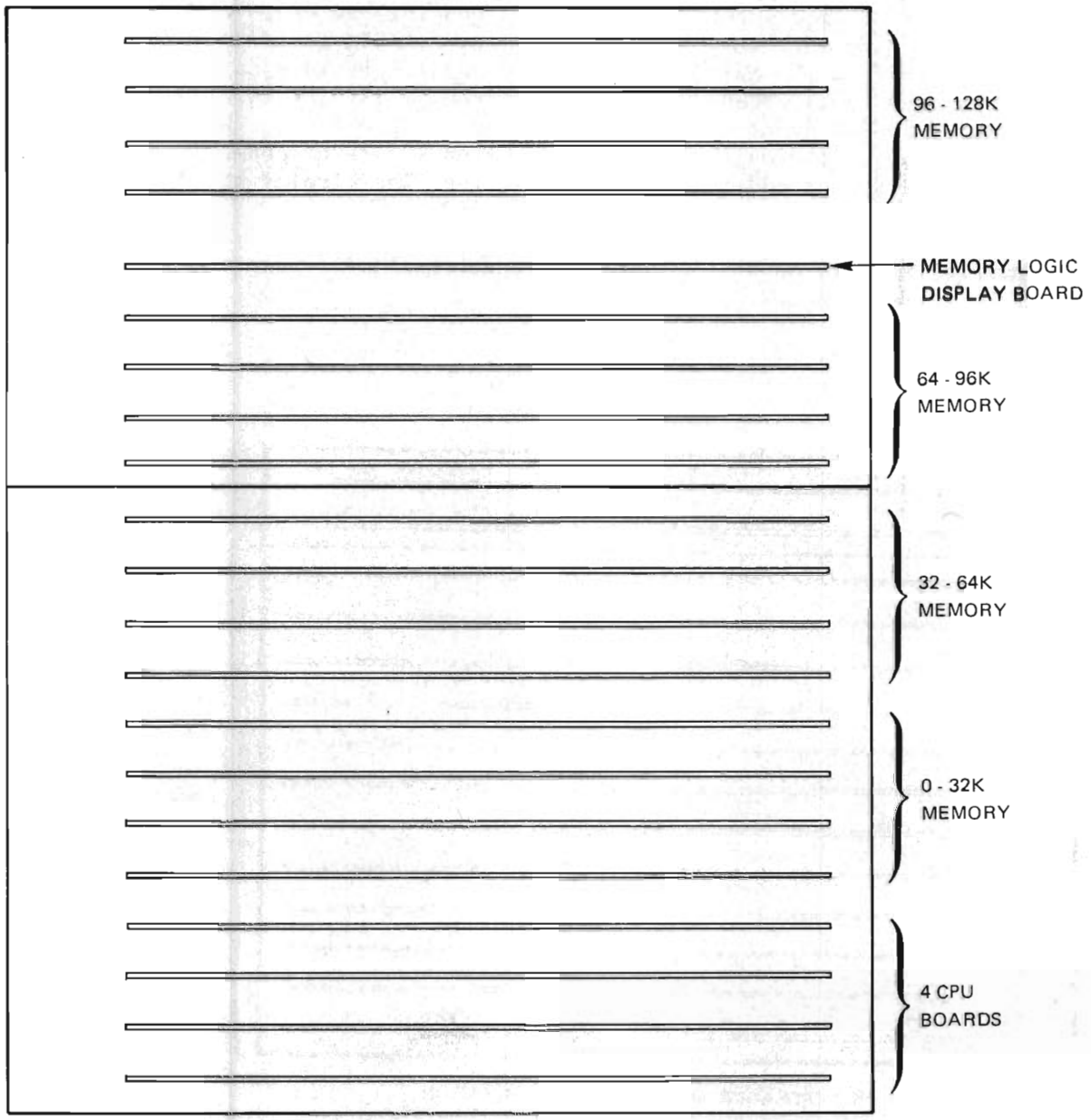
The SPC-16/45, 65, 85 model computers are designed for end users with larger applications, requiring more than 32K of memory, or for OEM users who want maximum expansion flexibility. The Mainframe for these models provides card slots for four memory modules and four processor boards. Memory capacity for this configuration can be extended to 64K by attaching a 64K extension interconnect board and a display board housing to the mainframe. The 64K extension interconnect board provides edge connectors for four additional memory modules. The display board housing includes card slots for two display and driver boards.

All I/O controllers are housed in one or more external I/O enclosures.



UNIT PACKAGING MAINFRAME CONFIGURATION
MODELS SPC-16/45, SPC-16/65, SPC-16/85
WITH 64K EXTENDED MEMORY (FRONT VIEW)

In addition, memory capacity can be increased to 128K for SPC-16/45 and SPC-16/65 by attaching a 128K extension interconnect board and a memory extension chassis to the mainframe. The 128K extension interconnect board provides the four additional edge connectors described above for extension to 64K plus edge connectors for eight additional memory modules. The memory extension chassis includes card slots for these eight additional memory modules and a card slot for one display and driver board.

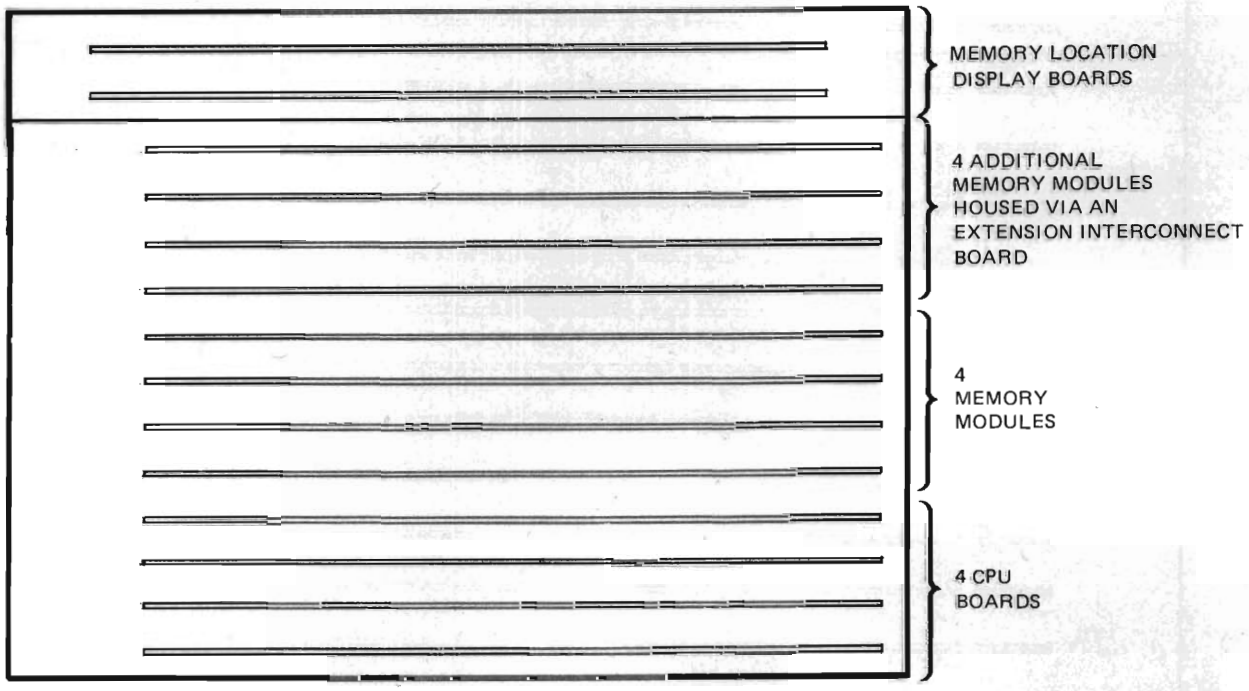


PACKAGING CONFIGURATION, MAINFRAME PLUS
128K MEMORY OPTION
MODELS SPC-16/45, SPC-16/65
(FRONT VIEW)

If more than eight peripheral controllers are required, additional controllers can be housed in an external I/O enclosure. Each external enclosure can accommodate up to 18 periperhal controller boards; an additional card slot is reserved for a Cable Interface Translator (CIT) card. The external I/O controllers communicate with the CPU via an external I/O bus.

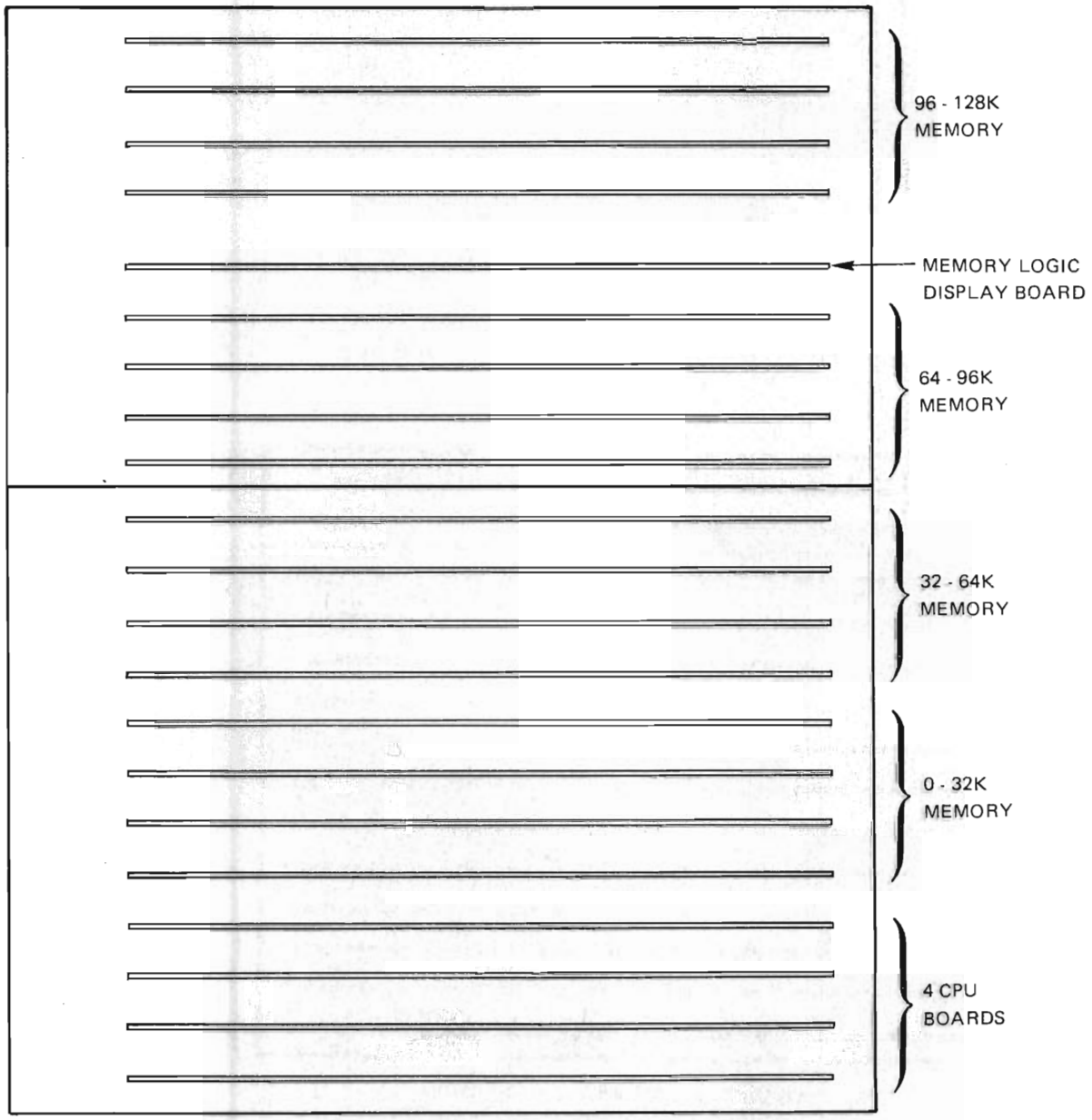
The SPC-16/45, 65, 85 model computers are designed for end users with larger applications, requiring more than 32K of memory, or for OEM users who want maximum expansion flexibility. The Mainframe for these models provides card slots for four memory modules and four processor boards. Memory capacity for this configuration can be extended to 64K by attaching a 64K extension interconnect board and a display board housing to the mainframe. The 64K extension interconnect board provides edge connectors for four additional memory modules. The display board housing includes card slots for two display and driver boards.

All I/O controllers are housed in one or more external I/O enclosures.



UNIT PACKAGING MAINFRAME CONFIGURATION
MODELS SPC-16/45, SPC-16/65, SPC-16/85
WITH 64K EXTENDED MEMORY (FRONT VIEW)

In addition, memory capacity can be increased to 128K for SPC-16/45 and SPC-16/65 by attaching a 128K extension interconnect board and a memory extension chassis to the mainframe. The 128K extension interconnect board provides the four additional edge connectors described above for extension to 64K plus edge connectors for eight additional memory modules. The memory extension chassis includes card slots for these eight additional memory modules and a card slot for one display and driver board.



PACKAGING CONFIGURATION, MAINFRAME PLUS
128K MEMORY OPTION
MODELS SPC-16/45, SPC-16/65
(FRONT VIEW)

General Automation supplies standard peripheral controllers that interface various I/O units to the SPC-16 computers. These controllers are fully compatible with all six models of the SPC-16/40+ series.

The user can, of course, use any controllers that meet the I/O bus requirements. Section 5 of this manual contains useful information regarding these requirements.

THE COMPUTER

The SPC-16 computer is a stored-program digital computer with 16-bit word length organization, a high-speed memory and a versatile arithmetic and control unit.

The standard Read/Write Memory uses three-wire core, made of lithium for greater environmental stability. Memory modules are provided on a single printed circuit board in 4096-, 8192- and 16484-word sizes. The system can be configured to support either 4K, 8K or 16K modules; however, a system that supports 8K or 16K modules can also support only one 4K module in addition to these modules.

Any Read/Write Memory module can also accommodate a small capacity Read-Only Memory. These semiconductor ROM's, which may have a 32- or 64-word capacity, are typically used for bootstraps or cold-start loaders. To be used, a small-capacity ROM is plugged into a Read/Write Memory printed circuit board, replacing a block of 32 or 64 addresses.

THE INSTRUCTION SET

All members of the SPC-16 family of computers share a common, powerful instruction set; programs written for one computer will run on any other computer in the family, except where processing speed is a factor.

Both instructions and data are stored in the SPC-16 memory. Eleven addressing modes are provided for memory reference instructions which may access any location in memory. Nine instruction groups, which include a total of 80 instructions, are standard on all processor models. (Five additional instructions, including hardware multiply and divide, are usable if the appropriate options are installed.)

The instruction set provides many operations not common in small computers. These operations include:

- Bit instructions that will test, set or reset any bit in memory, allowing the efficient use of logical variables.
- Byte instructions to load or store any byte in memory, or to switch the contents of two bytes in a word; this greatly simplifies handling of 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 instruction which reflects results in status indicators only; this allows efficient searching of memory for specified values.

- Ability to load an instruction into a register and execute it; this feature facilitates the use of procedure-only programs in Read-Only Memory.

- Jump to a subroutine and return via the address stored in the E-Register; the technique of saving the return address in a hardware 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 SPC-16 instruction set is described in detail in Section 3.

ADDRESSING MODES

A number of addressing modes are available to instructions that address memory. One instruction group uses literal addressing; these are two word instructions, where the second word of the instruction contains the operand itself. Other instruction groups 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, some instructions may specify one of three index registers to be used as a byte or word index to the referenced location.

GENERAL PURPOSE REGISTERS

A set of eight 16-bit general purpose registers is standard on all processor models. The Foreground/Background option provides an additional set of eight registers, making a total of sixteen general purpose registers, with facility for switching from one set of eight registers to the other (i.e., selecting the Background or Foreground mode). All general purpose registers may be used as accumulators for arithmetic and logical operations, as program loop counters, or as data input/output buffers.

In addition to the functional capabilities common to all general purpose registers, certain general purpose registers have special properties. One register is used in base relative addressing; when a 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 are designated index registers; when one of these registers is specified in an instruction as an index register, the specified register's contents are applied to the computer address to determine the word or byte in memory to be referenced. One register serves as a subroutine return register; it holds the return address after a jump-to-subroutine (JSR) instruction is executed or a program interrupt occurs.

INPUT/OUTPUT SYSTEM

The SPC-16 input/output system 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 feature. A program may poll devices to determine if they are ready to be serviced, or the devices may be allowed to interrupt when they require service.

A program communicates with peripheral equipment via input/output (XIO) instructions. Four classes of XIO instructions provide data input, data output, control and test functions. An XIO instruction addresses any one of up to sixty-four devices. Two devices are internal to the processor: A standard teletype controller and a controller for masking internal interrupts. The remaining 62 device select addresses may be assigned to additional devices as needed.

An active priority interrupt system minimizes interrupt response time. For each of the 64 possible devices that may be interfaced to the computer there is a dedicated memory location which contains that device's interrupt service routine address. When the processor acknowledges an interrupt, control is transferred (in a single memory cycle) to the address stored in the interrupting device's dedicated memory location; by virtue of arriving at the addressed routine, the interrupting device has been identified.

The Direct Memory Access feature allows automatic, high-speed transfer of blocks of data between the SPC-16 memory and peripherals. A DMA transfer operation is initiated by the program and runs to completion under the supervision of the peripheral controller. Each time the controller is ready to send or receive a data item, it "steals" a memory cycle from the running program; these stolen cycles do not, however, affect the course of execution of the program.

SAFETY FEATURES

Safety features are provided for the SPC-16 family of computers; these features insure reliable operation even in remote or unattended environments. The Power Fail Detection feature, standard on all models, insures that the contents of memory are preserved in the event of a power failure. The other safety features described below are standard on Models 40, 60 and 80 and optional on Models 45, 65 and 85.

1. Relative Time Clock - This feature provides a periodic interrupt that occurs every 1,000 memory cycles when the interrupt level from the Relative Time Clock (RTC) is enabled. In addition to providing a multi-programming 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.

To ensure that control is periodically returned to the monitor program, the Operations Monitor Alarm counter may be invoked.

2. Operations Monitor Alarm - 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 audio-visual alarm or automatic switchover.

   The Pulse Operations Monitor Alarm (PMA) instruction is provided to both arm the alarm and to 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 200,000 memory cycles 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 is removed from the line.

   The OMA may be cleared by auto-restart or by manually pressing the Systems Reset switch at the computer console. If the OMA is cleared by the Systems Reset switch, the program must be restarted manually.

3. Power Fail Interrupt - This feature issues a warning to the running program if the unregulated DC voltage drops below a pre-determined limit (approximately equal to 105 VAC 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 interrupt is requested, the OMA timer is initialized to time out in 100 memory cycles. Any further PMA instructions or data-channel requests are ignored. After 100 memory cycles, the OMA times out and the machine goes through a normal alarm sequence (except that the safe signal is not removed from the line). The regulated direct current voltages are guaranteed to be good at the time of the Alarm Sequence.

4. Auto-Restart Interrupt - This feature causes an orderly start-up with a special restart interrupt when power is restored after a power failure. When the unregulated DC voltage exceeds a predetermined value, the Auto-Restart Circuit requests an interrupt on its preassigned level. 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 by the hardware prior to the Auto-Restart Interrupt request.

CONSTRUCTION

The new family of SPC-16 computers is almost completely wire free; construction uses multilayer printed circuit boards, Medium Scale Integration and high speed Schottky Clamped Logic.

Every computer is tested for the full range of specified temperature and humidity, for vibration, power cycling and noise. Operating conditions are 0° to 50°C with up to 90% humidity.

Power required is 700 watts (for 4K memory) at 115 volts AC or 220 volts AC with an additional 40 watts required for each additional 4K.

All computers in the family fit in a standard 19" rack, with a height of 10.5 inches and a depth of 21.6 inches.

SUMMARY

Every SPC-16 computer, with its combination of eight general purpose registers (16 with the foreground/background option), nine powerful instruction groups, eleven addressing modes and an active priority interrupt system with unique levels, offers the following useful features:

- The ability to directly address any word, byte or bit in memory eliminates the need for paging.
- Program relative addressing allows programs to be self-relocating (i.e., they will run without modification anywhere in memory).
- Base relative addressing allows the independent location of data and instructions; temporary storage required by subroutines can be allocated dynamically, thereby minimizing memory requirements by allowing subroutines to share portions of memory.
- A concept of dynamic storage allocation, when combined with a number of special instructions such as Store All Registers and Status (SARS) and Load All Registers and Status (LARS), greatly simplifies coding of interruptable, re-entrant and recursive routines.
- The active interrupt system reduces 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.
- The ability to locate data and instructions independently and to execute the contents of a register as an instruction allows any program to be written as "pure procedure" and, consequently, to be implemented in Read-Only Memory.

These features and other specifications for the SPC-16 40/45, 60/65 and 80/85 series are summarized in Table 1-1.

Table 1-1. SPC-16 Specifications Summary

Characteristic	Specification
Memory	• Random access • 16-bit word organization • Expandable to 32K 16-bit words, using 8K or 16K memory modules. • Interchangable Read/Write and Read Only Memory boards. • Read-Only Memory modules available in 512-, 1024-, 2048- and 4096-word increments. • Read/Write Memory modules available in 4096-, 8192- and 16384-word sizes. • Read/Write Memory module may include 32- or 64-word ROM. • Lithium ferrite core stacks (with wide temperature tolerance) used on Read/Write Memory; each stack has an independent current regulator. • Metal-oxide semiconductor IC's used on Read Only Memory boards.
Addressing	• Single and double-word addressing. • Eleven addressing schemes, including: Direct and direct indexed; indirect and indirect indexed; program relative and program relative indirect; base relative and base relative indexed; base relative indirect and base relative indirect and base relative indirect indexed; literal.

Table 1-1. (continued)

Characteristic	Specification
Arithmetic	• Bit, byte and word operations (logical and arithmetic). • Parallel, binary, fixed point two's complement.
Instructions	• There are 9 standard instruction groups and one optional group (number of instructions in a group is shown in parentheses following group name): - Memory reference (4) - Memory reference, indexed (12) - Skip (±256 locations on 8 conditions) (8) - Register operate and compare (11) - Register operate literal and compare (11) - Register change (16) - Shift (0 to 16 bits) (4) - Control (9) - Input/Output for up to 64 devices (single word instruction addressing) (6) - Hardware Multiply and Divide (optional) (2) • All instructions may be stored in either Read/Write or Read Only Memory modules.
Instruction Execution Time	• Execution times vary with the complexity of the instruction and are a multiple of memory cycle time. Values below are execution times for a one-cycle instruction. • SPC-16/40 & 45 1440 ns (Read/Write Memory) 720 ns (Read Only Memory) • SPC-16/60 & 65 960 ns (Read/Write Memory) 480 ns (Read Only Memory) • SPC-16/80 & 85 800 ns (Read/Write Memory) 400 ns (Read Only Memory)

Table 1-1. (continued)

Characteristic	Specification
Input/Output	• 16-bit parallel bus. • Allows communication between SPC-16 registers/memory and up to 64 external peripheral units (via peripheral controllers). • Programmed I/O transfer rates, full 16 bits, for the three model pairs are, using back-to-back instructions: SPC-16 40/45 347.2 KHz SPC-16 60/65 520.8 KHz SPC-16 80/85 625.0 KHz • DMA port allows direct access of memory by peripheral unit(s) using a cycle-stealing technique. Transfer rates for the three model pairs are: SPC-16 40/45 694.4 KHz SPC-16 60/65 1041.6 KHz SPC-16 80/85 1250.0 KHz • Thirty-two Data channels may be optionally added to provide easy interface for cycle-stealing devices. The controller physically nearest to the CPU has priority. • Sixteen pairs of memory locations are dedicated to the Data channels.
System Safety Interrupts	• Power Fail memory protection (standard for all models). • Relative Time Clock (standard for Models 40/60/80; optional for Models 45/65/85. • Operations Monitor Alarm (standard for Models 40/60/80; optional for Models 45/65/85. • Power Fail Interrupt (standard for Models 40/60/80; optional for Models 45/65/85).

Table 1-1. (continued)

Characteristic	Specification
Other Interrupts	• Auto Restart Interrupt (standard for Models 40/60/80; optional for Models 45/65/85). • Block Protect (optional on all models). • Teletype • Console • External I/O
Physical	Console Panel Programmer's console. Sixteen data switches and indicators. Register select switches for 19 registers and status indicators. Console lock-out switch, key operated. Operations monitor alarm. Console interrupt. General Dimensions SPC-16 40/60/80 without power supply: Height = 10.5 inches Width = 19.0 inches Depth = 21.75 inches Weight = max. approx. 56 lbs SPC-16 45/65/85 without Memory Extension Option or power supply: Height = 10.5 inches Width = 19.0 inches Depth = 21.75 inches Weight = max. approx. 52 lbs (with 4 8K-memory boards) SPC-16 45/65/85 with 64K Memory Extension option (without power supply): Height = 12.2 inches Width = 19.0 inches Depth = 21.8 inches Weight = max. approx. 75 lbs. (8 8K-memory boards)

Table 1-1. (continued)

Characteristic	Specification
	SPC-16 45/65 128K Memory Extension option (without power supply): Height = 19.25 inches Width = 19.0 inches Depth = 21.6 inches Weight = 112 lbs. (16 8K-memory boards) Environment Temperature = 0°C to +50°C (+32°F to +122°F) Humidity = up to 90% relative (no condensation) AC Power 115 vac, ±10%, 47 to 63 Hz, single phase 220 vac, ±10%, 47 to 63 Hz, single phase 630 watts (4K or 8K core), add 20 watts per additional 4K, add 40 watts per additional 8K. Site Preparation No special wiring, subflooring, air conditioning, or other site preparation is required. Component Weights CPU Power Supply (51D00031A) = 47.5 lbs. Extended Memory Power Supply (51D0032A) = 25.5 lbs. 4K Memory Module (31D01470A) = 2.75 lbs. 8K Memory Module (31D01594A) = 3.0 lbs. 16K Memory Module (31D01783A) = 3.75 lbs.

SECTION 2
SYSTEM ORGANIZATION

This section describes the SPC-16 system organization at the following levels:

1. Instruction and data word formats.
2. Elements of the Central Processing Unit.
3. Addressing modes.
4. Input/Output System

The SPC-16 system organization is described in detail to provide the reader with background to evaluate information that follows; a clear understanding of the information provided in Section 2 is an essential prerequisite to subsequent sections.

The SPC-16 instruction repertoire includes eighty-five instructions, each of which identifies one of the elementary operations that can be performed by the SPC-16 computer. Any meaningful sequence of instructions is called a program. The coded representations of individual instructions (hereafter termed simply "instructions"), which comprise a program, are stored in sequence in the computer's memory.

The computer's memory is divided into a large number of cells, called memory locations. The contents of a memory location is called a "word". 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 2.1.

The Central Processing Unit (CPU) consists of the hardware elements that performs the execution of a program stored in the computer's memory. The CPU fetches an instruction from memory, performs the operations dictated by that instruction, and then fetches the next instruction in the program sequence. The hardware elements of the CPU are described in Section 2.2.

A program normally involves the creation or manipulation of data and the movement of data between the CPU and memory. Each location in memory is identified by a unique number called an address. The address of a memory location is its sequential location in memory; thus, if memory contains 4096_{10} words, addresses will range sequentially from 0 to 4095_{10}. The CPU supplies an address to the memory control circuits to access a particular location (either to "read" a word from that location or "write" a word into that location). During program execution, addressing information contained in the instructions determine which memory locations are accessed. There are eleven ways, or "addressing modes", by which an instruction can refer to a memory location. Addressing modes are discussed in Section 2.3.

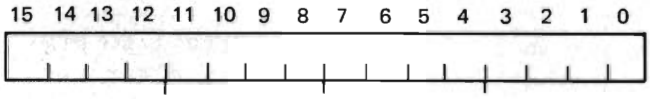
The Input/Output System allows the CPU and peripheral equipment to communicate. The general organization of the Input/Output System is described in Section 2.4.

Section 5 is devoted to the Interfacing requirements for the SPC-16/40, 45, 60, 65, 80 and 85 computers.

2.1 WORD FORMAT

The architecture of SPC-16 processors is based on the 16-bit binary word. Instructions and data are transferred between the CPU and the 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 order bit is numbered zero and the highest order bit is numbered 15.

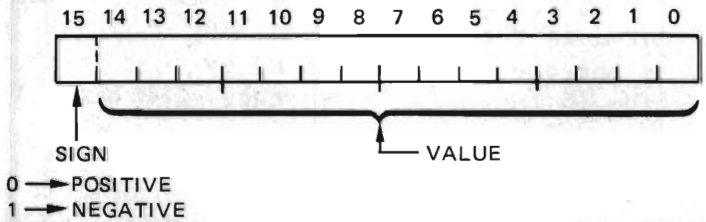


2.1.1 Data Format

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 having a significance independent of other bits.

ARITHMETIC QUANTITY

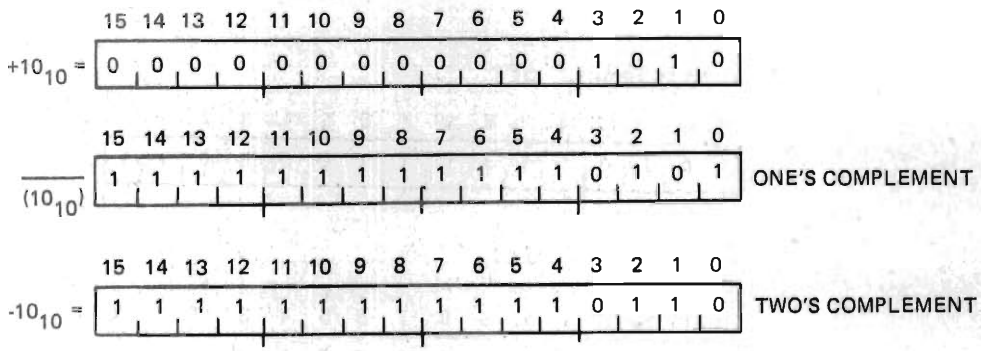
The SPC-16 hardware performs arithmetic operations in parallel on 16-bit quantities using twos complement, binary arithmetic. In a word containing a signed integer, the high order bit represents the sign.



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

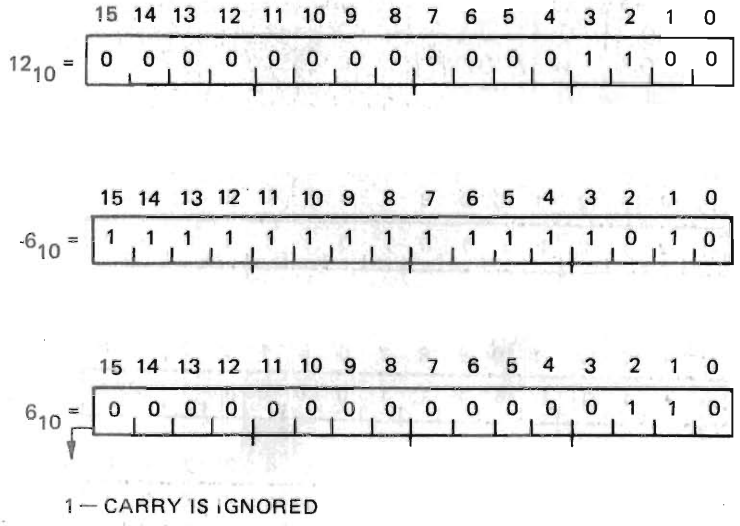
The representation for the negative of a number is obtained by taking its twos complement, with the sign bit included in the operation. A twos complement operation may be performed by taking the ones complement (or logical complement) of the number and adding 1 to the result, ignoring any carry out of bit 15.

For example, the binary representation of +10 and the conversion of the twos complement representation of -10 is shown as follows:



A 16-bit word can represent any number in the range $-32{,}768_{10}$ (8000_{16}) to $+32{,}767_{10}$ ($7FFF_{16}$). Zero has a unique representation, since the twos complement of 0000_{16} is 0000_{16}.

Arithmetic operations are particularly simple using twos complement conventions. For example, the sum of two signed numbers is found by simple addition. A number is subtracted from another by adding its twos complement. For example:



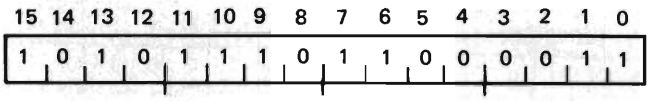
LOGICAL WORDS

The SPC-16 hardware performs logical operations in parallel on 16-bit words. Four logical operations are provided: logical complement, logical AND, logical OR and 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.

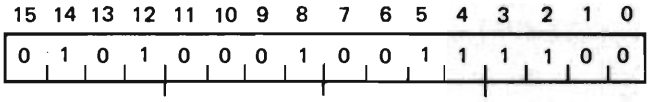
The logical complement (or ones complement) of a word is obtained by changing each 1 bit to 0 and each 0 bit to 1.

Example:

The complement of:



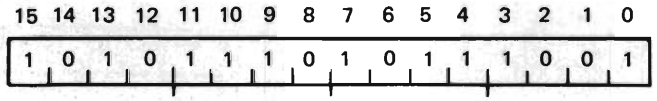
is:



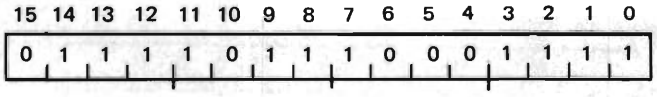
The logical AND of two words is the result obtained by the logical AND of each bit in the first word with the corresponding bit in the second word. The logical AND of two bits results in a 1 only if both bits are 1. The AND operation (indicated by $\wedge$) can be represented in a truth table as follows:

A_i	$\wedge$	B_i	=	R_i
0		0		0
0		1		0
1		0		0
1		1		1

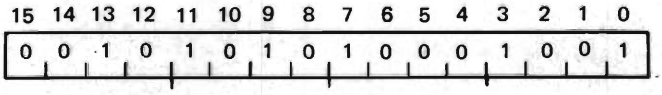
Thus



AND



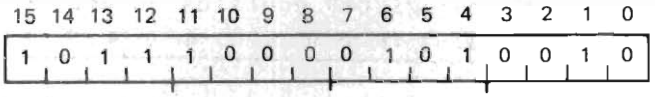
results in:



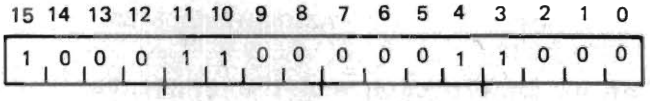
The logical OR of two words is the result obtained by the logical OR of each bit in the first word with the corresponding bit in the second word. The logical OR of two bits results in a 1 if either bit is 1. The OR operation (indicated by v) can be represented in a truth table as follows:

A_i	v	B_i	=	R_i
0		0		0
0		1		1
1		0		1
1		1		1

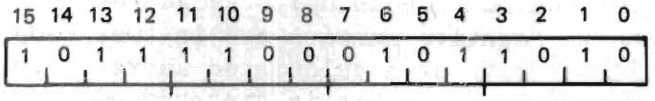
Thus



OR



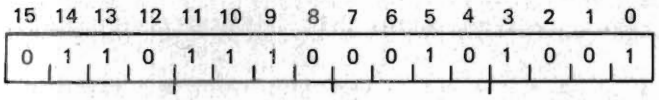
results in:



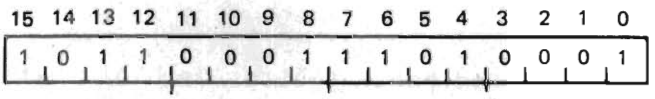
The logical XOR of two words is the result obtained by the logical XOR of each bit in the first word with the corresponding bit in the second word. The logical XOR of two bits results in a 1 only if the two bits are different. The XOR operation (indicated by ⊻ or + or ≠) can be represented in a truth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

A_i	⊻ or + or ≠	B_i	=	R_i
0		0		0
0		1		1
1		0		1
1		1		0

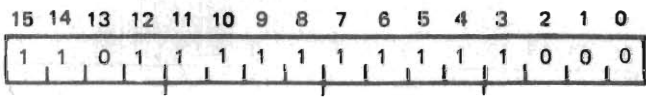
Thus



XOR



results in:

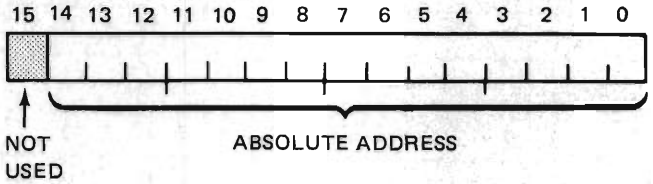


ADDRESSES

The SPC-16 memory is expandable to a capacity of 32K words (i.e., 32768_{10} words) for Models 40, 60 and 80 and to 64K words for Models 45, 65 and 85. The extra 32K on the latter models is accommodated by using an optional extended memory system, explained in Section 7.

A maximum of 32K words of memory is directly addressable in either system. A memory location is identified by a number (called its address) in the range 0000_{16} to $7FFF_{16}$. Therefore, 15 bits are required to specify any word in active memory.

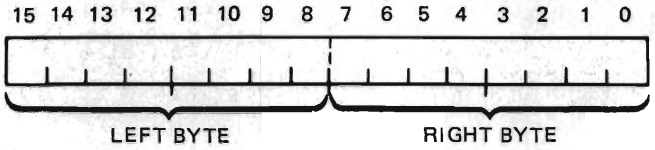
When a memory location or hardware register contains an address, only the low-order 15 bits are significant.



When the address field of an instruction and the contents of a hardware register are used in an address (e.g., indexed addressing), the address calculation is performed using 16 bit signed numbers (negative numbers are in twos complement form). However, only the 15 least-significant bits of a calculated address are supplied to the address circuits of the memory (i.e., the sign bit is disregarded).

BYTE DATA

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

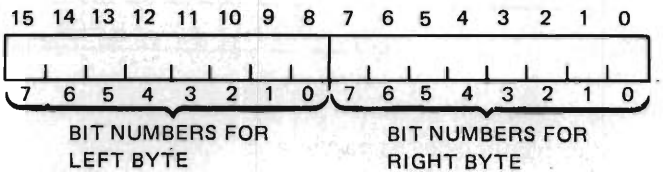


An 8-bit byte is a convenient unit for representing some forms of data. For example, any of the 128 ASCII characters (see Appendix B for ASCII character codes) can be represented in a single byte. (Only the 7 lowest order bits of a byte are significant for ASCII character data; the high order bit always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

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.

BIT DATA

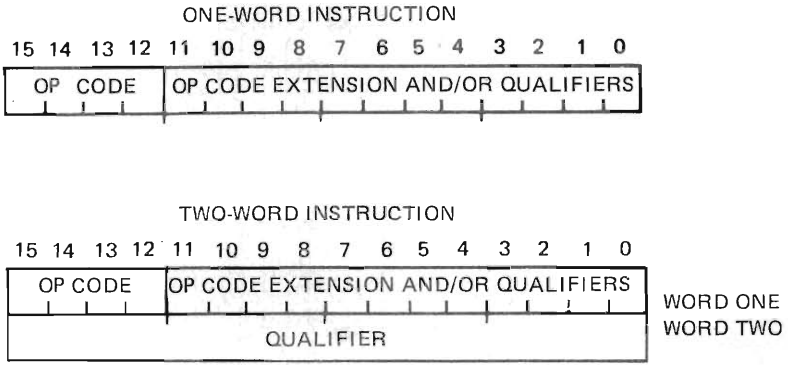
A word may be considered as 16 independent bits, each bit being identified as bit 0-7 of the left or right byte:



Instructions are provided to allow the testing, setting and resetting of individual bits in memory. Thus, logical variables (variables that have a value either "True" or "False", i.e., 1 or 0) may be packed in memory, 16 per word, without sacrificing access time to these variables.

2.1.2 Instruction Format

Instructions are represented as one or two 16-bit binary words, in the following general formats:



The operation code (bits 12 to 15 of the first word), together with any operation code extension bits, uniquely identify each of the SPC-16 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 12 to 15. Instructions requiring the most space for qualifiers are given the shortest operation codes (i.e., use only bits 12 to 15 to specify the instruction). Some instructions have a combination of operation code extension and operands in bits 0-11, and instructions which do not require any operands use all 16 bits to specify an operation.

A two-word instruction is needed 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 will always be a two-word instruction with the literal value contained in the second word. All other instructions are one-word instructions.

In order to execute an instruction, the processor logic must identify the instruction and then perform the indicated operations. The first bits which the CPU decodes are bits 12-15 of the first word. This places the instruction into one of five groups, as shown in Figure 2-1. Then, depending upon the group, additional bits are examined to identify the individual instruction.

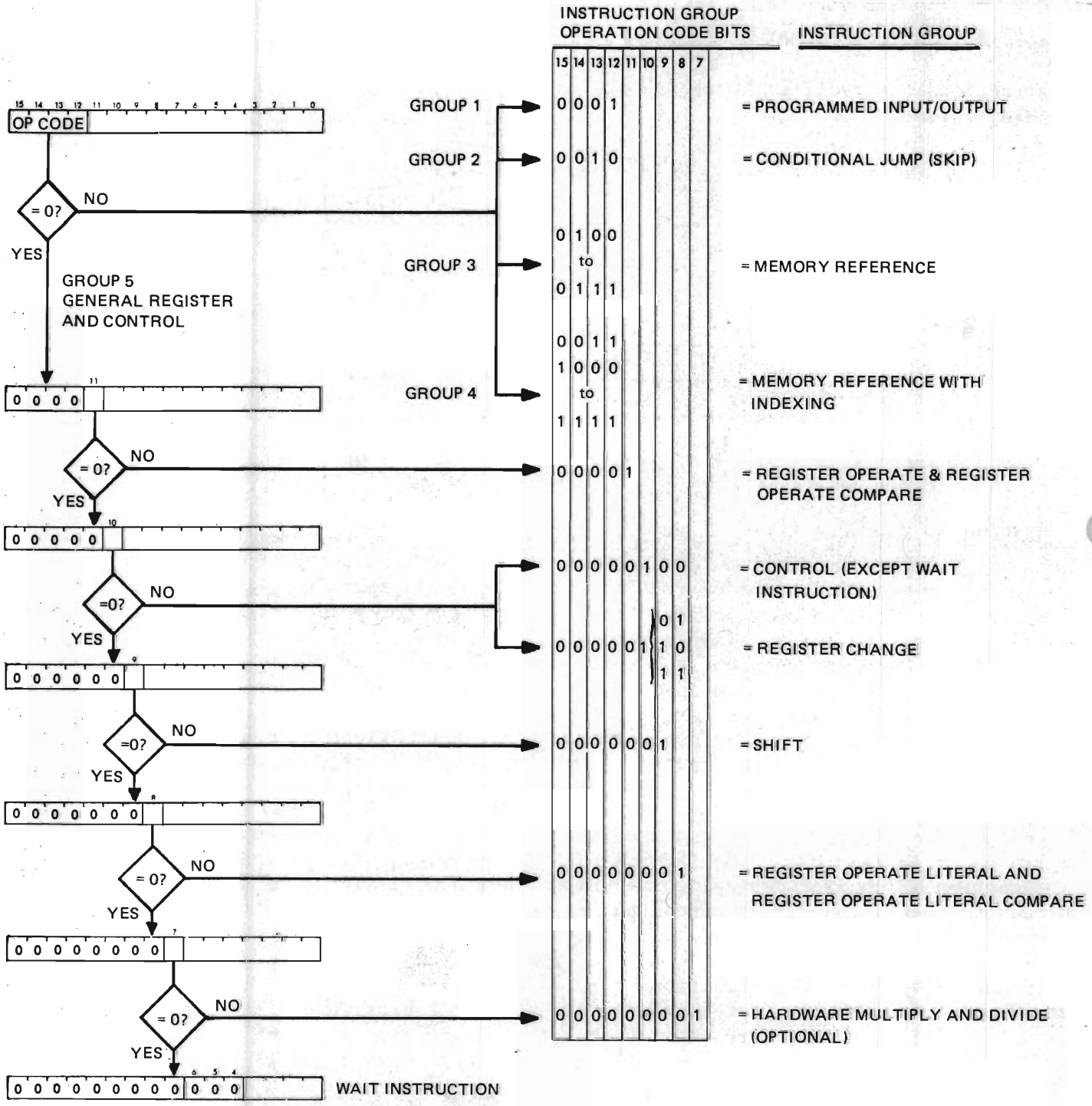


Figure 2-1. Operation Codes for Instruction Groups

Individual instruction formats, in binary representation, are presented in Section 3. The 16 digits of the binary code can be considered as four 4-bit groups which make up a four-digit hexadecimal number. The hexadecimal codes for instructions are given in Appendix A.

2.2 ELEMENTS OF THE CENTRAL PROCESSING UNIT

Figure 2-2 illustrates the internal organization of the SPC-16 computer and identifies the principle CPU elements.

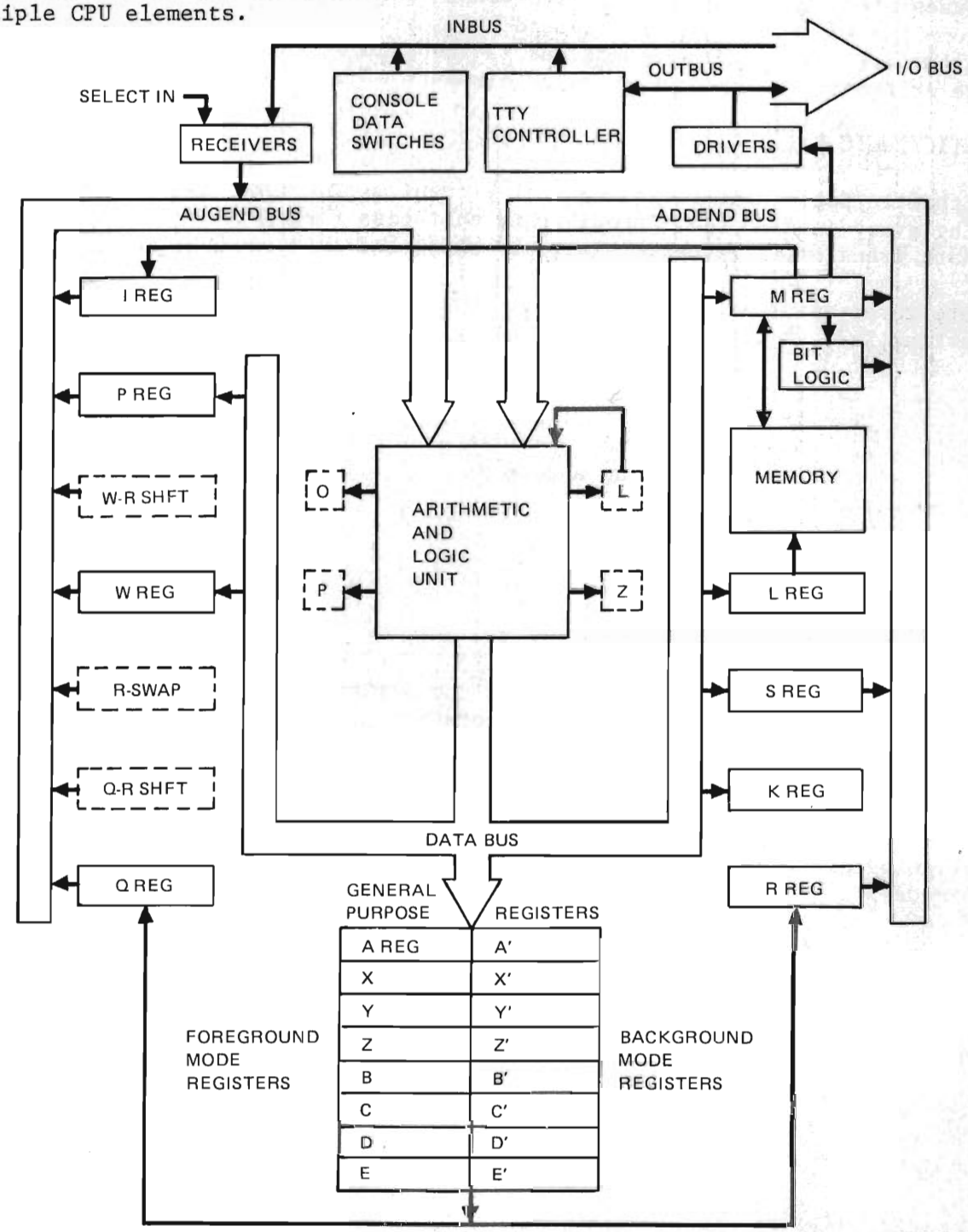


Figure 2-2. Internal Organization of the CPU

TRANSFER BUSES

Three buses provide the main communication paths within the computer:

- Augend bus (16-bits)
- Addend bus (16-bits)
- Data bus (16-bits)

These buses allow a flow of information within the computer that is both efficient and uncomplicated. The Augend and Addend buses are the paths by which all data enters the Arithmetic/Logic Unit. The Data bus is the path by which data is transferred from the Arithmetic/Logic Unit to other elements of the system.

ARITHMETIC/LOGIC UNIT

The Arithmetic/Logic Unit is the central element of the computer; it is used in executing every instruction. Through this unit pass the operands of all load, store, jump, data transfer, arithmetic, logical, shift and input/output operations.

There are three inputs to the Arithmetic/Logical Unit: The Augend bus, the Addend bus and the Link indicator. The Arithmetic/Logic Unit can perform the following operations:

- Transfer Augend bus
- Transfer Addend bus
- Add Augend to Addend
- Subtract Addend from Augend
- Add Link to Augend
- Add Augend, Addend and 1
- Complement Augend
- OR Augend and Addend
- XOR Augend and Addend
- AND Augend and Addend

The 16-bit result of any operation leaves the Arithmetic/Logic Unit via the Data bus. The Link, Zero, Plus and Overflow indicators reflect certain properties of the result of some operations (see S-Register).

GENERAL PURPOSE REGISTERS

The following eight general purpose, 16-bit hardware registers are standard on all SPC-16 models:

Register	Description	Function
A	General Purpose	Accumulator
X	Index/General Purpose	Used for indexed addressing or as A-Register.
Y	Index/General Purpose	As X-Register.
Z	Index/General Purpose	As X-Register
B	General Purpose	As A-Register
C	General Purpose	As A-Register
D	Base-Relative Addressing/General Purpose	Contains the base address for base relative addressing; otherwise used as A-Register.
E	Subroutine Linking/ General Purpose	During subroutine or interrupt execution, contains return address and Interrupt Status Enable Control. Otherwise used as A-Register.

With the foreground/background option, eight additional general purpose registers may be used. Only one set of registers is active at a given time. Instructions are provided to activate either set (i.e., change to foreground or background mode). Any reference to a register, say the A-Register, refers to the active one of the pair, either A' if in background mode of 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.

The General Purpose Registers are implemented in 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
0→ Background Registers

CPU Logic	
000	A-Register
001	X-Register
010	Y-Register
011	Z-Register
100	B-Register
101	C-Register
110	D-Register
111	E-Register

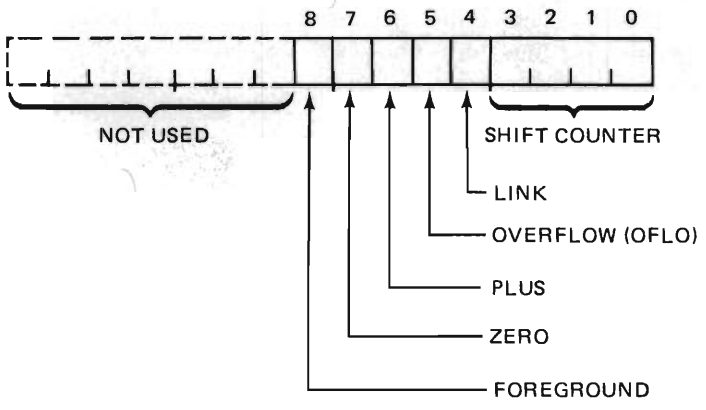
Information written into a Selected Register always comes off the Data bus. Information read from the Selected Register goes to the Q-Register, the R-Register or both.

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 and complement or clear any register
- Any register may be shifted in a variety of ways
- 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.

S-REGISTER

The S-Register or STATUS Register, contains the Shift Counter (4-bits) and the Link, Overflow, Plus, Zero and Foreground/Background indicators.



An indicator is "set" if it contains a one, and is "reset" if it contains a zero.

The Shift Counter is used in the execution of the Shift instructions. A Shift 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 the I-Register (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.

The Link indicator is affected by two types of operations: arithmetic and shift.

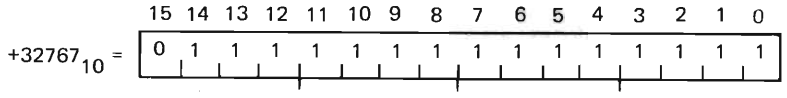
1. Arithmetic. The Link is set (=1) if there is a carry, or reset (=0) if there is no carry, from bit 15 of a previous arithmetic operation. The Link is particularly useful in multiple precision operations where more than one 16-bit word is used to represent a quantity; an instruction is provided to 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.
2. Shift. During the execution of a shift instruction, the Link indicator contains the last value shifted out of bit 0 of the selected register.

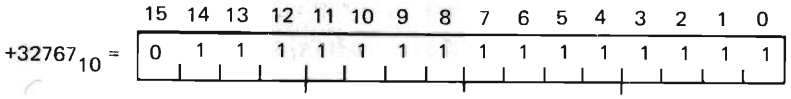
Instructions are provided for setting, resetting and testing the Link indicator.

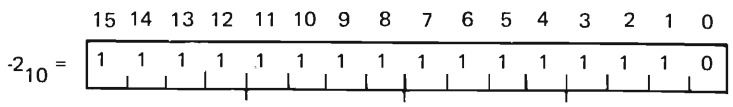
The Overflow indicator indicates an arithmetic overflow for an add or subtract instruction. The Overflow indicator is set (=1):

- 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 (subtrahend).

Overflow conditions occur when the result of an arithmetic operation is not in the allowed range (i.e., not between -32,768 and +32767). For example, if 32,767 is added to 32,767:







The Overflow indicator is not necessarily an error indicator. Indeed, if the program that generates the above operation 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 Plus indicator 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 Zero indicator 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.

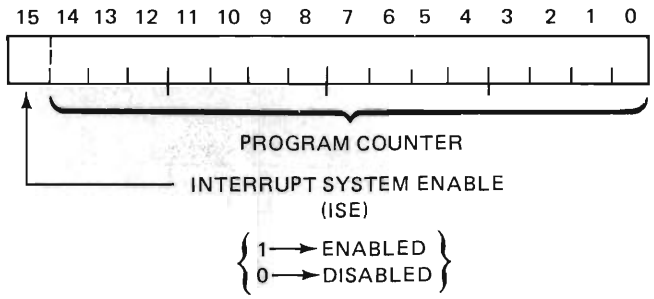
The Foreground indicator contains 1 if the Foreground Registers are active and 0 if the Background Registers are active. Instructions are provided to set or reset this bit (if the Foreground/Background option is not installed, the indicator always contains zero).

The S-Register can be loaded from the Data bus and gated onto the Addend bus. Instructions are provided to transfer the contents of the S-Register to any general purpose register and vice-versa. Its contents are 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.

P-REGISTER

Bits 0-14 of the P-Register are used as the program sequence register (called the "Program Counter").

The high order bit of the P-Register contains the Interrupt System Enable (ISE) indicator. (When loading the P-Register via the computer console, the ISE bit is not affected.)



When discussing program sequencing, the symbol "P" refers to bits 0-14 only.

The Program Counter normally contains the address of the next instruction to be executed. Its contents (bits 0-14) are transferred to the L-Register (Memory Location Register) 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, JUMP) do so by altering the contents of the P-Register. The P-Register can be changed by "jump", "conditional jump" (skip), "jump-to-subroutine" and "return" instructions.

When the Interrupt System Enable contains 1 following the execution of any instruction (except "JUMP"), the processor will acknowledge an interrupt if one is being requested. When this bit contains 0, no interrupt except Power Fail can be acknowledged (i.e., the interrupt system is disabled).

When an interrupt is acknowledged, a "jump-to-subroutine" (JSR) indirect instruction is executed, the contents of the P-Register (Program Counter = P_{0-14} and ISE = P_{15}) are saved in the E register; and the ISE bit in the P-register is reset, inhibiting other interrupts while the current interrupt is being serviced.

The ISE is controlled by "interrupt enable", "interrupt inhibit", "restore interrupt system enable", "jump-to-subroutine" and "return" instructions.

L-REGISTER

The L-Register (15-bits) is the memory location register. It contains the absolute address of the memory location from which or to which an instruction, an address or data is to be read or stored.

The L-Register is always loaded from the Data bus.

M-REGISTER

The M-Register (16-bits) is the memory buffer register. It receives all data read from and contains all data written into the computer's memory.

The M-Register can be loaded from the Data bus. It can be selected onto the Addend bus, into the I-Register or through the I/O Drivers for the Outbus. (The Outbus always reflects data in the M-Register.)

The "Bit Logic" provides the bit handling capability for "set bit", "reset bit" and "test bit" instructions. The "Bit Logic" samples and operates on the selected bit directly from the M-Register.

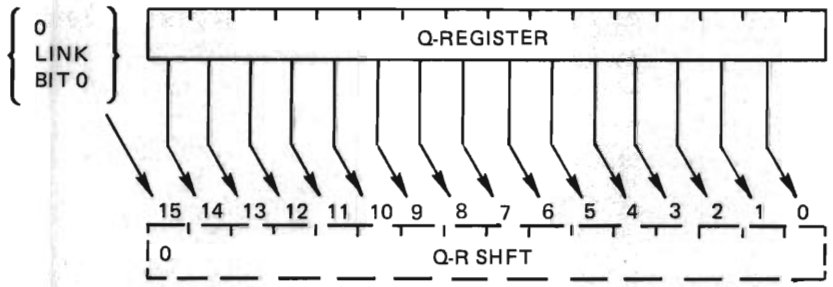
I-REGISTER

The I-Register (16-bits) always holds the instruction currently being executed (only the first word of a double-word instruction). After memory is accessed to fetch an instruction, the M-Register is transferred directly to the I-Register. The instruction decoding logic operates with the bit pattern in the I-Register to determine the sequence of events necessary to execute the instruction.

Q-REGISTER

The Q-Register serves as a buffer register between any general purpose register and the Augend bus.

A non-addressable register, labeled Q-R.SHFT in Figure 2-2 is the Q-Register with bits shifted one position to the right.



Q-R.SHFT may also be gated onto the Augend Bus.

R-REGISTER

The R-Register (16-bits) serves as a buffer register between a general purpose register and the Addend bus.

A non-addressable register (16-bits), labeled R-SWAP in Figure 2-2, is the R-Register with the bytes reversed.

$$(R)_{0-7} \rightarrow (R\text{-}SWAP)_{8-15}$$

$$(R)_{8-15} \rightarrow (R\text{-}SWAP)_{0-7}$$

This non-addressable register may be gated onto the Augend bus.

W-REGISTER

The W-Register holds the effective operand address for most address calculations. It is loaded from the Data bus and may be selected onto the Augend bus.

A non-addressable register, labeled W-R.SHFT in Figure 2-2, is the W-Register with bits shifted one position to the right. W-R.SHFT may also be gated onto the Augend bus, (used for hardware multiply only).

K-REGISTER

The contents of the K-Register (16-bits) are continuously displayed on the computer console's Register Display Indicators. An instruction is provided to transfer the contents of any general purpose register to the K-Register and, thereby, display it on the console while the computer is executing a program. (Section 6 discusses console operation).

The K-Register is loaded from the Data bus.

2.3 ADDRESSING MODES

Much of the processing power of the SPC-16 family of computers is a result of the flexible addressing scheme. A total of eleven addressing modes are provided on the SPC-16 computer.

- Absolute, Direct
- Absolute, Direct, Indexed
- Absolute, Indirect
- Absolute, Indirect, Indexed
- Base Relative, Direct
- Base Relative, Direct, Indexed
- Base Relative, Indirect
- Base Relative, Indirect, Indexed
- Program Relative, Direct
- Program Relative, Indirect
- Literal

Instructions reference memory locations either to 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 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 a relative address, which may be signed or unsigned.

Figure 2-3 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. The effective address determined in stage 1 (EA_1) is carried into stage 2, if another operation is required, and so on until the final effective address (EA FINAL) is generated. The individual operations which may occur in stage 1, stage 2 and stage 3 are discussed separately in Sections 2.3.1, 2.3.2 and 2.3.3, respectively. (See Section 3 for individual instruction formats.)

2.3.1 Effective Address Generation - Stage 1 (ABSOLUTE, BASE RELATIVE, PROGRAM RELATIVE OR LITERAL)

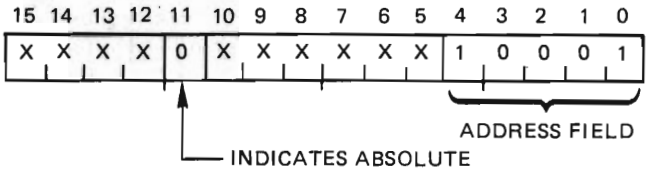
As shown in Figure 2-3, instructions that reference memory may use absolute, base relative, program relative or literal addressing. These terms describe the various methods by which the CPU can determine an effective address in stage 1 of an address generation. For some instructions, the effective address determined in stage 1 (EA_1) is the final effective address (EA FINAL) (i.e., 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 b. Change program sequencing	Program Relative, Base Relative → EA_1	Direct, Indirect → EA_2		= EA FINAL
Memory Reference with Indexing Reference word, byte or bit data.	Absolute, Base Relative → EA_1	Direct, Indirect → EA_2	Non-Indexed, Indexed → EA_3	= EA FINAL
Conditional Jump (Skip) Conditionally change program sequencing	Program Relative → EA_1			= EA FINAL
Register Operate Literal and Register Operate Compare Literal Reference a literal value	Literal → EA_1			= EA FINAL

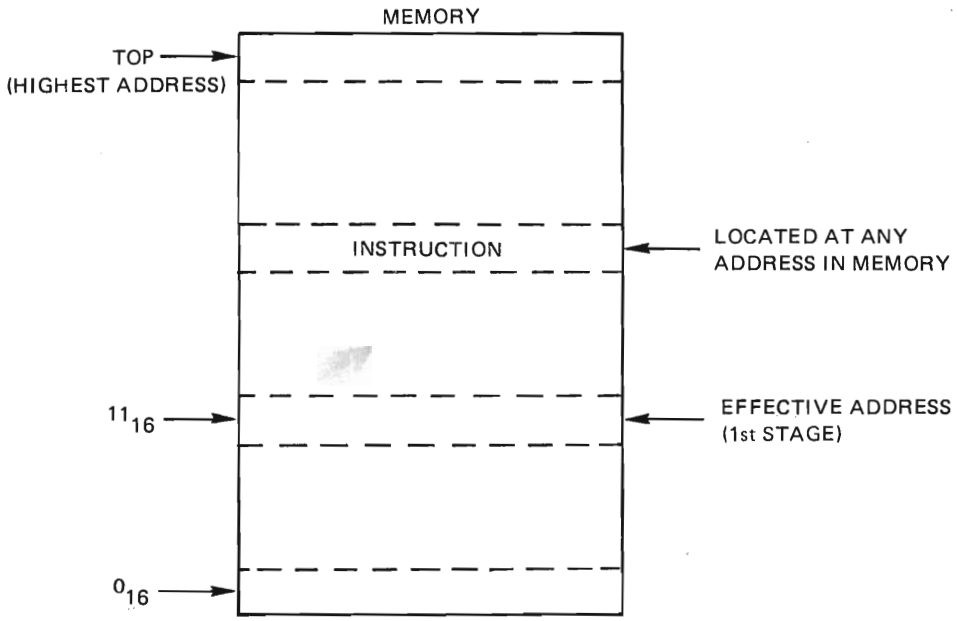
Figure 2-3. Address Stages for Instructions that Reference Memory

ABSOLUTE

The effective address is simply the address contained in the address field of the instruction. For example, the instruction (a Memory Reference with Indexing instruction is illustrated):

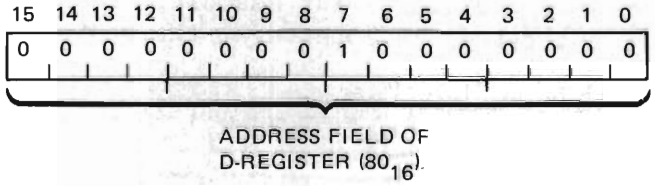


specifies the effective address, 11_{16} (or 17_{10})

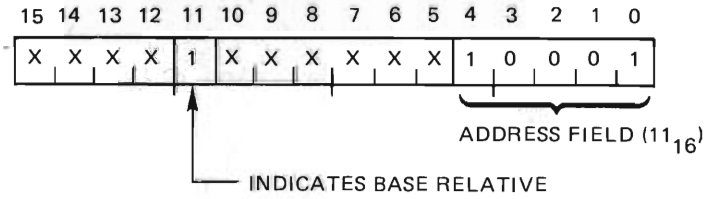


BASE RELATIVE

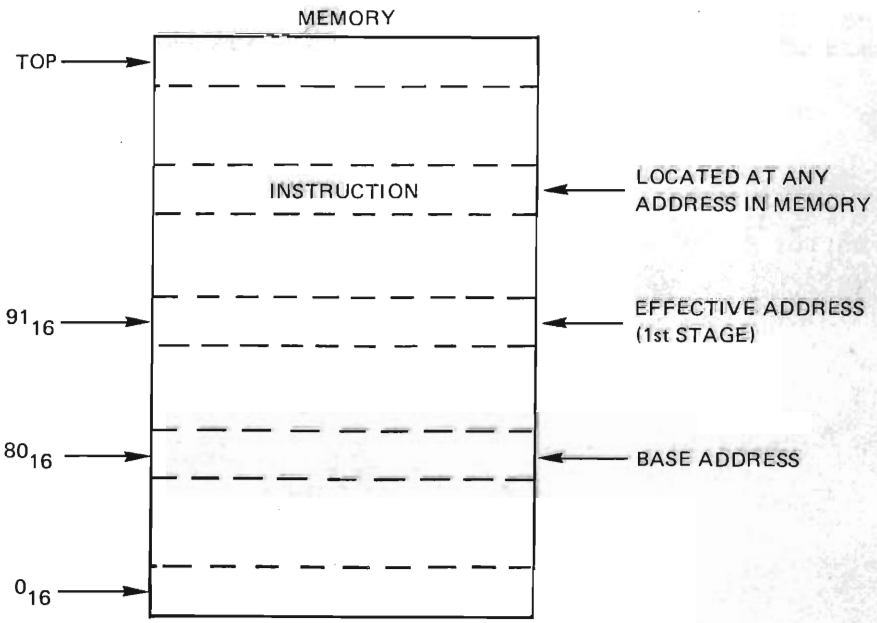
The effective address is calculated as the sum of the contents of the D-register (called the "base address") and the contents of the address field of the instruction. For example, if the D-register contains:



the instruction (a Memory Reference with Indexing instruction is illustrated)

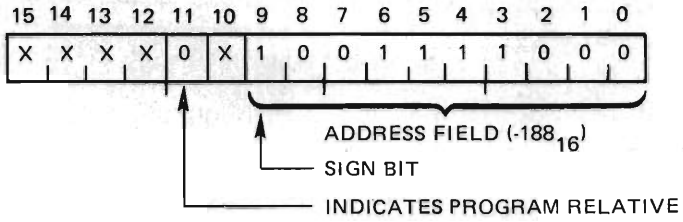


specifies the effective address, $80_{16} + 11_{16} = 91_{16}$



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 twos complement form with the highest order bit of the address field specifying the sign.) For example, if the instruction (a Memory Reference instruction is illustrated) is located at the address, $1FF_{16}$, and it appears as:



the address is determined as follows. The addressing origin is $1FF_{16} + 1_{16} = 200_{16}$, or expressed in binary notation,

$$00001000000000_2$$

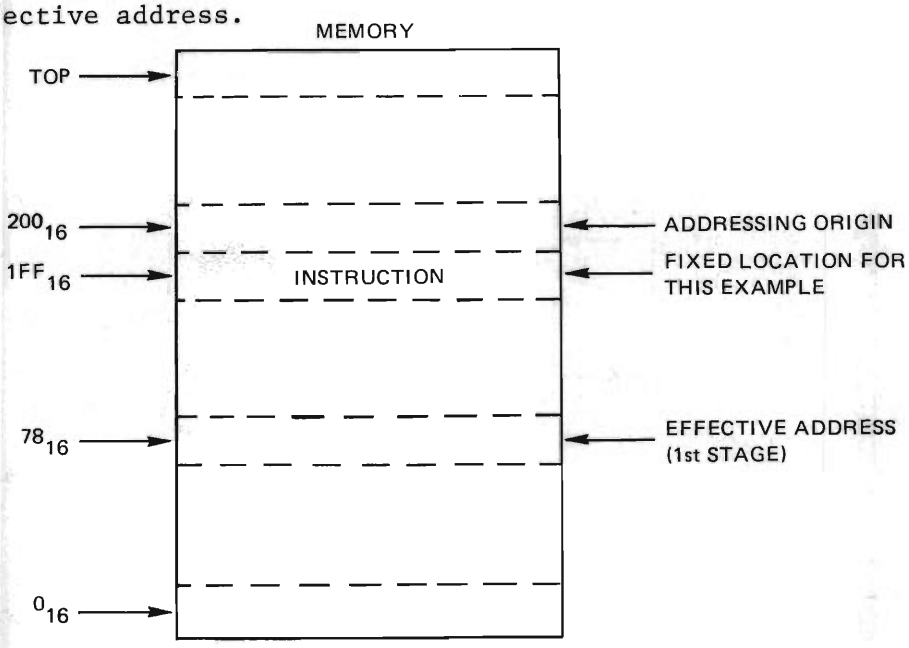
Since the address field, for program relative addressing, always contains a signed number, the binary digit indicated in the sign bit is extended to the left to form a 16-digit binary number. For the above example, the address field indicates the following twos complement number:

$$1111111001111000_2$$

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

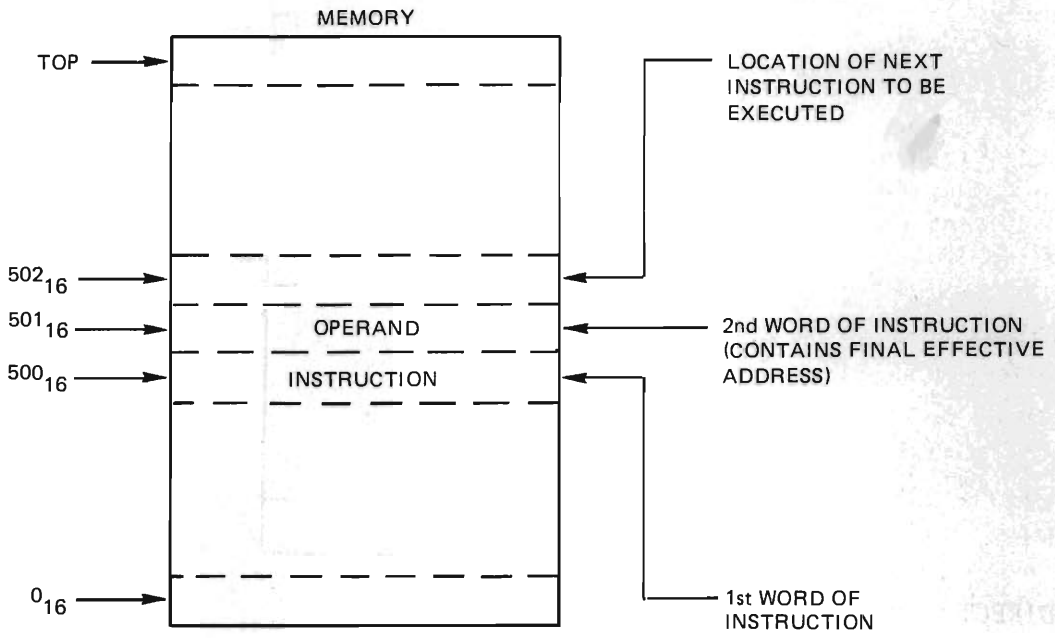
$$\begin{aligned} & 0000001000000000_2 \\ + \; & 1111111001111000_2 \\ = \; & 0000000001111000_2 \quad = \quad 78_{16} \end{aligned}$$

Thus, 78_{16} is the effective address.



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). For literal instructions, the effective address from stage 1 is always the final effective address (i.e., stage 2 is never indicated).

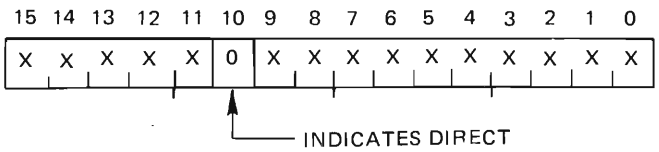


2.3.2 Effective Address Generation - Stage 2 (Direct or Indirect)

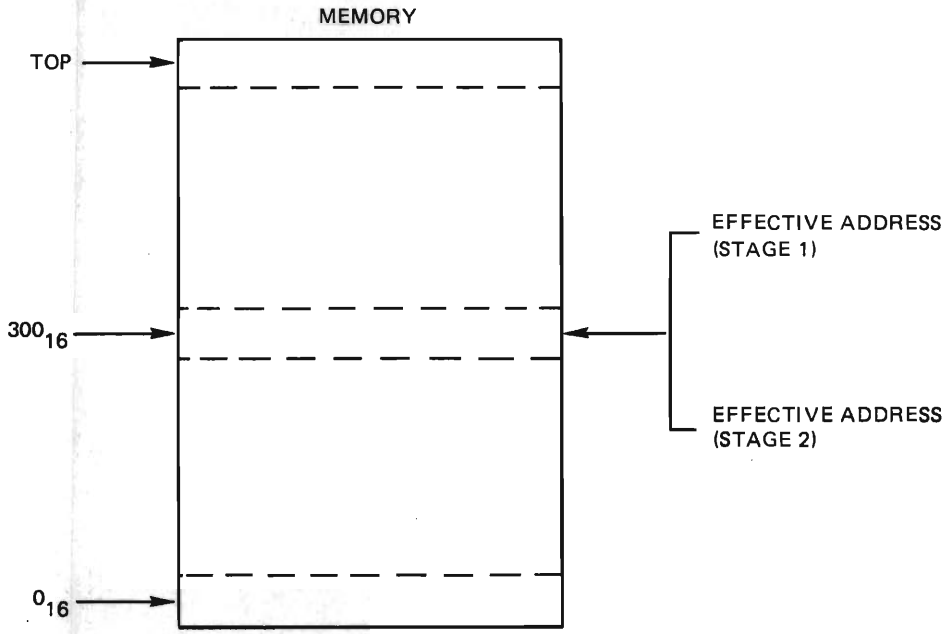
In stage 2, if a second stage is required, the CPU must determine if the effective address from stage 1 is direct or indirect. Note, in the examples below, that the CPU interrogates only one bit in the instruction word during stage 2; the address field of the instruction is no longer relevant (it is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

DIRECT

The effective address generated in stage one is simply carried into stage three, (i.e., no operation is indicated in stage two.) The result of stage one is, thus, termed a "direct address". For example, the instruction (instruction illustrated could be either a Memory Reference or a Memory Reference with Indexing instruction)

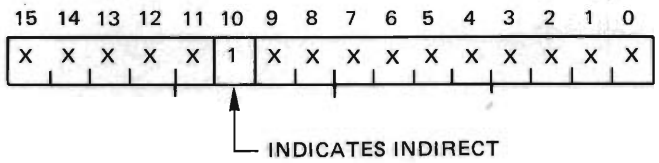


specifies the omission of stage two of operand address generation. If the effective address resulting from stage one, for the above instruction was 300_{16}, then the effective address carried into stage three is 300_{16} (i.e., $EA_1 = EA_2 = 300_{16}$).



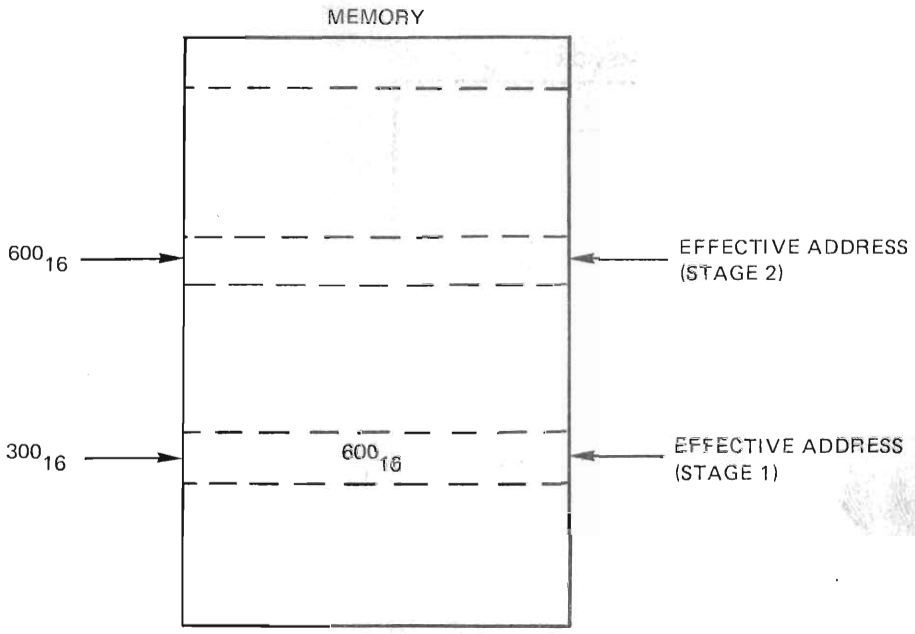
INDIRECT

The effective address generated in stage one is the address of a memory location that contains the effective address for stage two. When indirect is indicated, an extra memory cycle is required to fetch the effective address for stage two (i.e., the contents of the memory location identified by the effective address from stage one.) The effective address resulting from stage one is called an "indirect address". For example, the instruction (instruction illustrated could be either a Memory Reference or a Memory Reference with Indexing instruction)



specifies that the memory location identified by the effective address from stage one is to be accessed to retrieve the effective address that is to be carried into stage 3.

If the effective address from stage one is 300_{16} and memory location 300_{16} contains 600_{16} then 600_{16} will be the effective address resulting from stage two.

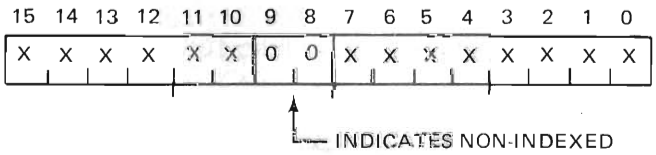


2.3.3 Effective Address Generation - Stage 3 (Indexed or Non-Indexed)

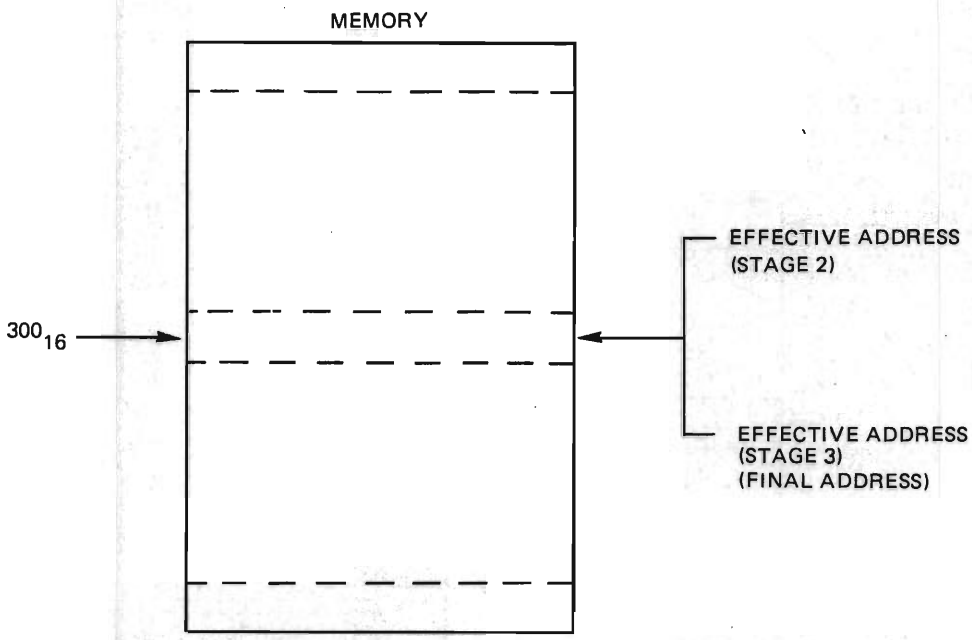
If stage three is indicated (instructions in the Memory Reference with Indexing group only), the CPU must determine whether or not to add the contents of an index register to the effective address from stage two. The address determined in stage three is the final effective address.

NON-INDEXED

The effective address resulting from stage two becomes the final effective address (i.e., no operation is indicated in stage three). For example, the instruction

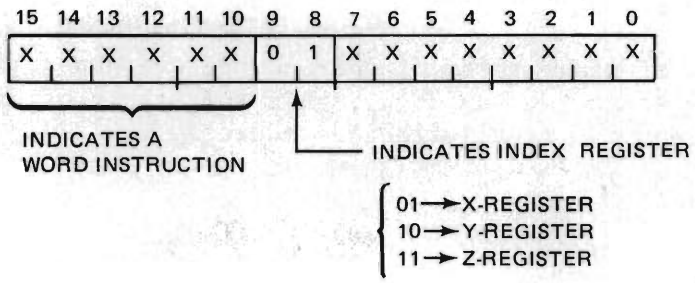


specifies the omission of stage three of operand address generation. If the effective address resulting from stage two was 300_{16}, then the result of stage three is also 300_{16}. Thus 300_{16} becomes the final effective address.

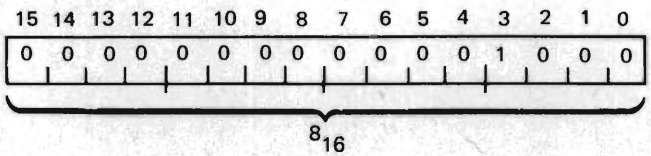


INDEXED

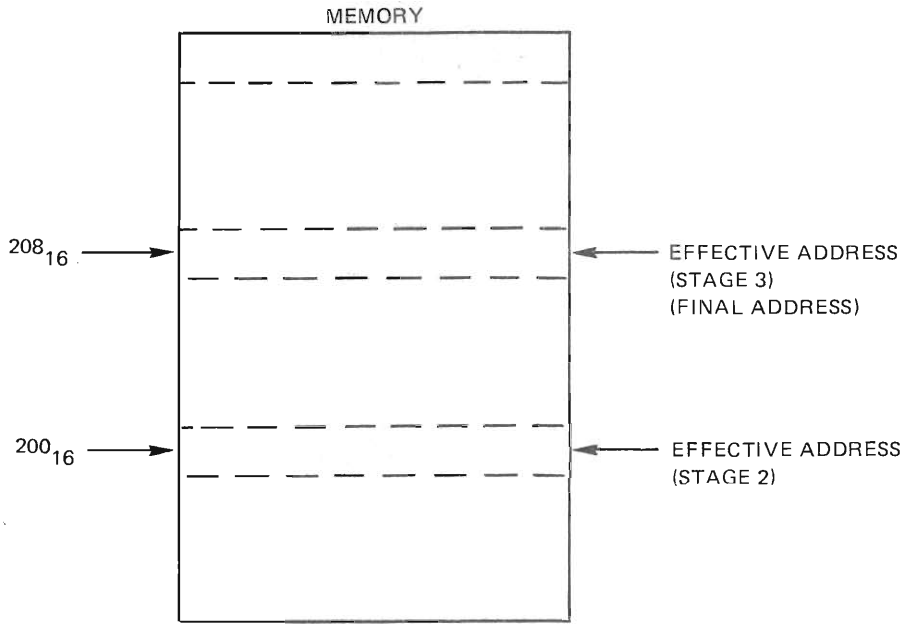
The effective address from stage two is modified by the contents of the index register (X,Y or Z register) indicated by the instruction, to determine the final effective address. If the instruction is one that operates on "word" data, the contents of the selected index register are simply added to the effective address from stage two to determine the final address. If the instruction is one that operates on "byte" or "bit" data, the contents of the selected index register are divided by two and then added to the effective address from stage two to determine the operand address. If the contents of the index register are even, the left byte is selected; if the contents are odd, the right byte is selected. (For "bit" instructions, a 3-bit field in the instruction itself indicates which of the 8-bits in the selected byte is to be referenced.) For example, the following "word" instruction;



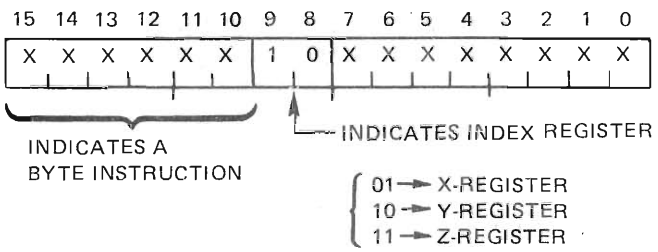
specifies that the contents of the X-register are to be used in determining the operand address. If the X-register contains:



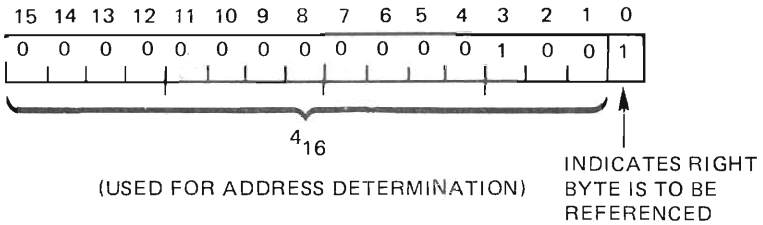
and the effective address from stage two is 200_{16}, the operand address is $200_{16} + 8_{16} = 208_{16}$.



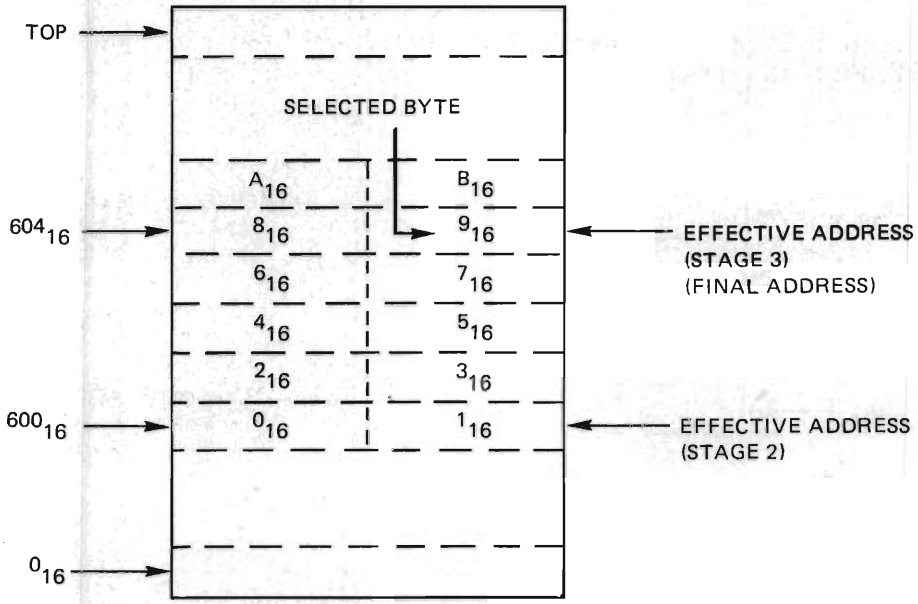
As an example of "byte" indexing, the following "byte" instruction;



specifies that the contents of the Y-register are to be used in determining the operand address. If the Y-register contains:



(Note that dividing the contents of the register by 2 is equivalent to using bits 1 - 15 instead of bits 0 - 14) and the effective address from stage two is 600_{16}, the operand address is $600_{16} + 4_{16} = 604_{16}$.



(Note the actual contents of the Y-register is 9_{16}. 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.)

2.4 INPUT/OUTPUT SYSTEM

The SPC-16 Input/Output System is the means by which peripheral equipment and the SPC-16 computer communicate.

The Input/Output Bus (I/O Bus) is the central element of the input/output system. All the signals necessary for the control of peripheral equipment and the transfer of data between the computer and peripheral units are provided on the I/O Bus.

The relationship between an SPC-16 computer and its peripheral units is shown in Figure 2-4. For Models 40, 60 and 80, the peripheral controller cards for the system illustrated would be housed in the computer's mainframe. For Models 45, 65 and 85, the controllers would be housed in an External I/O Enclosure.

A peripheral controller incorporates the hardware logic necessary to interface its peripheral unit to the I/O Bus. A controller must respond to and generate appropriate signals on the I/O Bus and thereby perform the operations necessary to transfer data between its peripheral unit and the CPU. A detailed description of the I/O Bus and interfacing requirements is provided in Section 5.

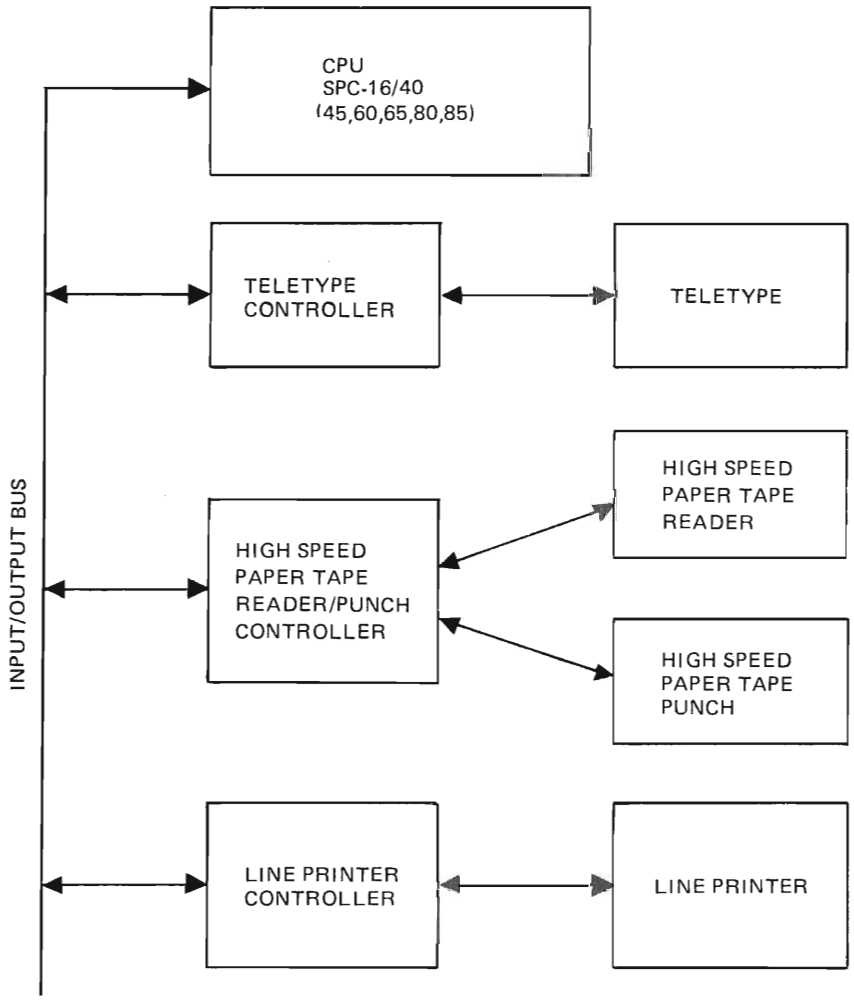


Figure 2-4. A Typical SPC-16 System

There are three kinds of operations that may occur on the I/O Bus:

- Programmed I/O operations without interrupts
- Programmed I/O operations with interrupts
- Direct Memory Access (DMA)

PROGRAMMED I/O OPERATIONS WITHOUT INTERRUPTS

A programmed I/O operation occurs whenever the CPU executes an instruction in the input/output group (XIO instructions - see Section 3.9). XIO instruction format allows for 64 unique device select codes (all XIO instructions include a 6-bit field that may contain a device select code in the range X'00' to X'3F'). For the purpose of XIO instructions each controller is represented by one or more logically independent devices; each device responds when and only when its unique device select code appears on the I/O Bus during the execution of an XIO instruction. Device select codes have been assigned to standard GA supplied controllers; these assignments are summarized in Table 5-1.

There are basically four XIO instructions:

- Data Input Instruction
- Data Output Instruction
- Control Instruction
- Test Instruction

When a Data Input instruction is executed, the selected controller (the controller whose device select code has appeared on the I/O Bus) must place a data word on the I/O Bus. Certain peripheral units (e.g., teletype, high speed paper tape reader) handle data one character at a time; a Data Input instruction is issued by the running program each time the peripheral unit has a character ready to input. For other peripheral units (e.g., disk); the Data Input instruction is used to input a status word from the controller; individual bits in the input data word reflect various status conditions in the controller.

When a Data Output instruction is executed, the selected controller must accept a data word from the I/O Bus. In the case of a peripheral unit such as a paper tape punch, the data would be a character that is actually output to the peripheral unit via the controller. For other devices the data could be used to set or reset certain conditions in the device.

The format of a Control instruction allows one of eight control functions to be specified for each controller (most controllers do not utilize all eight). A variety of operations are possible with Control instructions. Control instructions can be used to set or reset flags in a controller; for example, one Control instruction could be assigned to enabling program interrupts from a controller and another to disabling interrupts. For a card reader, a Control instruction is used to initiate the reading of a card. For a paper tape reader, a Control instruction is used to advance the tape to the next data frame.

The format of Test instructions allows one of eight tests to be performed on each controller. One Test instruction might determine whether or not a controller is ready to input a character. Another might be used to determine if an error flag is set in the controller (i.e., an error occurred during the last operation).

In all cases, the effect of an XIO instruction issued to a particular controller depends on the design of the controller. The instructions that have been implemented on standard GA controllers and their effect on the operation of their respective peripherals are described in Section 4.

PROGRAMMED I/O OPERATIONS WITH INTERRUPTS

A facility is provided to allow a peripheral controller to interrupt the running program when it requires service (e.g., it has a character ready to input). This facility is called the Priority Interrupt System.

When a controller requests an interrupt, the CPU generates an indirect jump through a memory location dedicated to the interrupting device. This dedicated memory location normally contains the address (interrupt vector) of the interrupt service routine responsible for handling interrupts from that controller. The system provides 64 priority levels with each level having a dedicated memory location for the indirect jump. Figure 2-5 lists the dedicated memory locations for the six internal interrupts. Certain locations have been assigned to GA's standard controllers and these assignments are summarized in Table 5-1.

Overall control of the Priority Interrupt System is governed by programmed and automatic control of the Interrupt Status Enable flip-flop (ISE). The ISE is copied into the high order bit (bit 15) of the P register. If the ISE contains a 1, the CPU will accept interrupt requests; if the ISE contains a 0, the CPU will not acknowledge interrupt requests (except the Power-fail and Auto-restart interrupts, which cannot be disabled). When the processor acknowledges an interrupt request, the ISE is automatically cleared, thereby disabling further interrupts. In addition to the ISE, which controls all interrupts, each controller has its own interrupt mask flip-flop which can be enabled or disabled under program control for optimum control of the system.

The following list summarizes the conditions necessary for the CPU to acknowledge an interrupt from a controller:

a. The CPU has just completed an instruction that is not a JMP instruction (or an instruction that clears the ISE) and there is no DMA transfer request pending. DMA transfers may occur during any memory cycle, even in the middle of a multi-cycle instruction, and have priority over interrupt requests (except Power-fail interrupt).

b. At least one controller is requesting an interrupt (which implies that the controller's interrupt mask is enabled).

c. The master interrupt system is enabled, i.e., ISE is set.

Hexadecimal	Contents
0000 ↓ 000F	Fast Access Locations for Operating System
0010 ↓ 001E	Fast Access Locations for User
001F	Reversed for E-Save
0020	Scan Control Register (SCR), Data Channel 0 (DC0)
0021	Channel Address Register (CAR), Data Channel 9 (DC0)
003E	SCR, DC15
003F	CAR, DC15
0040	Power Fail - Indirect Vector
0041	Auto Restart - Indirect Vector
0042	Memory Protect Violation - Indirect Vector
0043	Relative Time Clock - Indirect Vector
0044	*
0045	Console Teletype Not Busy - Indirect Vector
0046	*
0047	Console Interrupt - Indirect Vector
0048	*
004F	*
0050	*
0057	*
0078	*
007F	*
0080	Fixed Location Table
00BF	

*Available for external devices.

Figure 2-5. Dedicated Memory Map

d. No DMA requests are waiting that the CPU has not honored.

The following list summarizes the operation of the Priority Interrupt System:

a. The ISE flip-flop is under program control.

b. The interrupt masks on individual controllers are controlled by XIO instructions (see Section 4).

c. The highest priority interrupt request, at the completion of the current instruction, forces a JSR indirect through the controller's dedicated memory location.

d. All JSR instructions automatically disable the interrupt system (ISE is cleared) and save the return address and the old status of the ISE flip-flop in the E-register.

e. Subroutine return is accomplished by executing the 'RTRN E' command (return to the address contained in the E-register) that transfers the contents of the E-register back to the P-register (Program Counter and ISE flip-flop). Both the forced JSR and the RTRN commands require only 1 memory cycle each.

f. If nesting of subroutines is required, the program must save the E-register.

g. Load-All-Registers and Status (LARS) and Store-All-Registers and Status (SARS) instructions have been provided to minimize the overhead in interrupt service routines (the E-register is included in "all" registers).

h. When any of the instructions that reset the ISE flip-flop (INH, RISE, RTRN, JSR) are executed, the results of the instruction are instantaneously in effect, i.e., an interrupt sequence cannot occur following an INH, etc.

i. When any of the instructions that set the ISE flip-flop (INE, RISE, RTRN) are executed, the affect on the interrupt system depends on whether the interrupt is one of the internal or external interrupts, If it is an internal (console TTY, RTC, etc.) interrupt then the results are instantaneously in effect, i.e., a RTC interrupt can occur immediately following an INE, etc., If it is an external interrupt then at least two memory cycles must elapse (after the INE) before the interrupt can occur.

j. The CPU will not allow an interrupt immediately following a jump (JMP) instruction. Following execution of the JMP instruction, the instruction at the EA will be executed before the CPU will allow itself to be interrupted.

Since the memory location dedicated to a controller's interrupt normally contains the address of the routine responsible for servicing that controller, program control is transferred directly to the appropriate routine when a controller's interrupt is acknowledged. By virtue of arriving at the service routine, the controller that caused the interrupt has been identified. However, when more than one condition or "status-bit" associated with a single controller can request an interrupt, the program must "sense" the status of the controller (e.g., by using Test Instructions) to determine the interrupting condition.

When the processor acknowledges an interrupt, the system is disabled and no further interrupt requests can be acknowledged until the system is enabled again. When any given interrupt occurs and is acknowledged, the interrupting condition remains active (even though the system is disabled) until such time as the program removes the interrupting condition. Therefore, to prevent the same interrupt from interrupting itself, the program must mask-off the interrupt or remove the interrupting condition prior to enabling the system.

The interrupt priorities are established by the controller's physical location on the I/O Bus, the closest controller having the highest priority. The internal interrupts are of higher priority than the external controllers. To establish different priorities other than the physical one, the program must mask the interrupts individually.

To remove the interrupting condition, the program generally has to perform one of the following tasks:

a. Execute a new command to the controller that proceeds independent of past history. For example, if a paper tape reader requested an interrupt because it was waiting with a character, the interrupting condition could be removed by inputting the character or by instructing the reader to step to the next character.

b. For controllers that have a small number of "status-bits" one Control instruction (code) is assigned to that controller to reset all "status-bits". In these controllers each "status-bit" must be tested individually with the Test instruction. Resetting the "status-bit" would remove the interrupting condition.

c. For controllers that require a large number of "status-bits" one data-input address code can be assigned to that controller for retrieving the status information of which several of the "bits" could have caused an interrupt. When this data-input instruction is executed, the status-bits are reset and any interrupting condition existing at that time is removed.

DIRECT MEMORY ACCESS (DMA)

The Input/Output System is designed to allow automatic transfer of data between a peripheral controller and the computer's memory. The running program initiates the data transfer operation by specifying such information as the number of data items to be transferred, the beginning memory address for the transfer and the direction of the transfer (i.e., in or out of memory). Once initiated the data transfer operation runs

to completion, without program intervention, under the supervision of the peripheral controller. At the completion of the operation, the controller normally requests a program interrupt to signal the running program that the data has been transferred.

A Direct Memory Access operation requires one memory cycle to store or retrieve each data word being transferred. When a controller's DMA request is acknowledged by the CPU, a memory address is supplied to the CPU; the next memory cycle is stolen from the running program in order to transfer the data word to/from the specified memory location. Other than the fact that the running program's execution is retarded by one memory cycle during each data item transfer, the DMA operation does not affect normal program execution or interrupt servicing for other controllers.

Controllers supplied by GA that use DMA utilize a Data Channel controller and associated dedicated memory locations for program initialization of the block count and beginning memory address for a DMA transfer operation. A consecutive pair of memory locations is assigned to each Data Channel controller on a system (see Dedicated Memory Map - Figure 2-5). The first word in the pair (Scan Control Register (SCR)) is initialized by the program to contain the number of data items to be transferred. The second word (Channel Address Register (CAR)) is initialized to contain the beginning memory address (minus one) for the data transfer. After initialization of the SCR and the CAR, the program issues a control command to the controller to start the transfer operation. Details on the operation of individual peripheral units that use the DMA feature are given in Section 4.

SECTION 3
INSTRUCTION REPERTOIRE

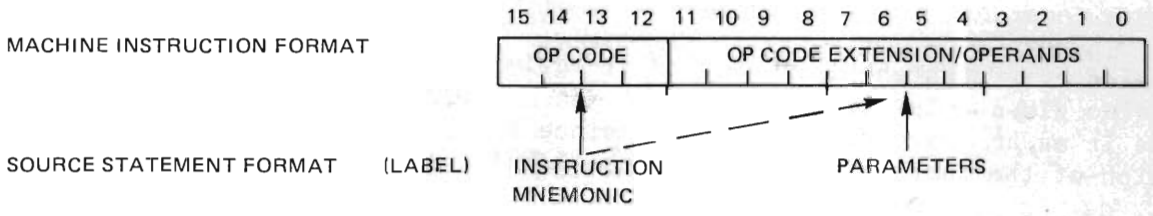
This section describes the 85 instructions which comprise the SPC-16 instruction set. The instructions are grouped as follows:

Memory Reference Instructions (4)
Memory Reference with Indexing Instructions (12)
Conditional Jump Instructions (Skip) (8)
Register Operate and Register Operate Compare Instructions (11)
Register Operate Literal and Register Operate Compare
Literal Instructions (11)
Register Change Instructions (16)
Shift Instructions (4)
Control Instructions (9)
Programmed Input/Output Instructions (8)
Hardware Multiply and Divide Instructions (Optional) (2)

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

Examples in this section use CAP-16 assembly language syntax . A "source statement" consists of an instruction mnemonic (eg., JMP for the jump instruction) 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. The 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 their machine language equivalents. The general relationship between the machine language format and the source statement format for SPC-16 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 the 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.

ADDRESSING PARAMETERS

Two instruction groups (Memory Reference and Memory Reference with Indexing) have some addressing parameters that may or may not be present. These addressing parameters correspond to the three addressing stages (see Section 2.3) 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. The first stage addressing mode can be specified by the programmer by appending an additional parameter to the source statement.

[*] Stage 2 Addressing (Direct, Indirect)
For both groups, an asterisk indicates indirect addressing; if no asterisk is present, direct addressing is implied.

[,i] Stage 3 Addressing (Indexed)
The Memory Reference with Indexing instructions may have an indexed address by naming an index register as a parameter; if no index register is given, the address is unindexed.

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 the source statement. The value given a label, however, is independent of the source statement which includes it as an addressing parameter, since the label value is determined by the location of the instruction on which the label appears.

A CAP-16 assembly language program generally consists of two sections: a program section and a data section. The program section normally contains all the executable instructions in the program. The data section normally contains all the storage for variable data items on which the instructions in the program sector operate.

Labels in the program section are assigned a value relative to the beginning of the program section. If PB ("program base") symbolizes the absolute address of the first instruction in a program section, then PB+0, PB+1 and PB+2 identify the locations of the first three instructions. If these instructions are given the labels ONE, TWO and THREE, respectively, then the label ONE has the value PB+0, the label TWO has the value PB+1, and the 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:

Value of Label	Label	Mnemonic	Parameters
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 addressing 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 addressing 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 twos complement form).

Since the absolute address PB is eliminated in the 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 program section may be located anywhere in memory without affecting the coding of the individual instructions (i.e., PB can have any value).

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

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

DATA0 has the value $$+0 and DATA1 has the value $$+1. The value of $$, the base address, is taken as the contents of the D register and is the addressing origin. The instructions in the program section that reference these locations will be assembled into machine code using base relative addressing. The address field of the LDA instruction is calculated as DATA0 - base address, or $$+0 - $$ = 0. The address field of the STA instruction is calculated as DATA1 - base address, or $$+1 - $$ = +1.

Since the coding of the instructions is independent of the location of the data section (i.e., $$ may have any address value), the actual memory locations referenced by these instructions depends on the value contained in the D register when these instructions are executed. The program itself may therefore dynamically allocate memory locations for its data section by setting the D register to the beginning of any available block of memory.

PROCESSOR CYCLE TIMES

The times required to complete one processor cycle for the SPC-16 Series is as follows:

Model	Cycle Speed (nanoseconds)
SPC-16/40 & 45	1440
SPC-16/60 & 65	960
SPC-16/80 & 85	800

The number of cycles required to execute an instruction is given in each instruction subsection. For a two-word Memory Reference with Indexing instruction (DISP=X'1F'), or whenever indirect addressing is used, one memory cycle time should be added to the instruction execution time. If both DISP=X'1F' and indirect addressing is selected, two memory cycle times should be added to the instruction time. (One memory cycle time is identical to one processor cycle time.)

Certain instructions execute twice as fast if fetched from a Read-Only memory as from a Read-Write memory. All Register Operate instructions have this characteristic, as do all Register Operate Compare, Register Change and Control instructions. Therefore, when estimating execution times for ROM-originated instructions in these groups, the above cycle times should be halved.

CONVENTIONS

The sequence of operations involved in executing the instruction is given in each instruction subsection, and the indicators and general purpose registers (A,X,Y,Z, B,C,D,E) that are affected by the execution of the instruction are listed.

In discussing the sequence of operations that occurs when an instruction is executed, the following conventions are observed:

a. The sequence is shown by one or more arrows, e.g., 1→ Link means that the content of the Link indicator is replaced by a 1.

b. A single letter code is used to represent the contents of a register. For example, D refers to the contents of the D register.

c. Bits of a word may be represented as subscripts, e.g., B_{0-3} refers to bits 0 through 3 of the B register.

d. The term Effective Address, EA, refers to the address value being calculated during the execution of an instruction which addresses a memory location. The EA is the value at any step in the evaluation process.

e. A memory address is represented by an address symbol without parentheses. 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 the EA.

f. The symbol P refers to the absolute address from which the instruction currently being executed was fetched. P+1 refers to the next sequential address. Thus, if the instruction at P is a one-word instruction which does not affect program sequencing, P+1 is the address of the next instruction to be executed. If P is a two-word instruction, P is the address of the first word and P+1 is the address of the second word. For a two-word instruction, (P+1) is the second word of the instruction.

   The CAP-16 assembly language equivalent of P is the symbol $, and the equivalent of P+1 is $+1.

g. In examples using base relative addressing, the symbol $$ refers to the base address (i.e., the contents of the D register).

REGISTER IDENTIFIER CODES

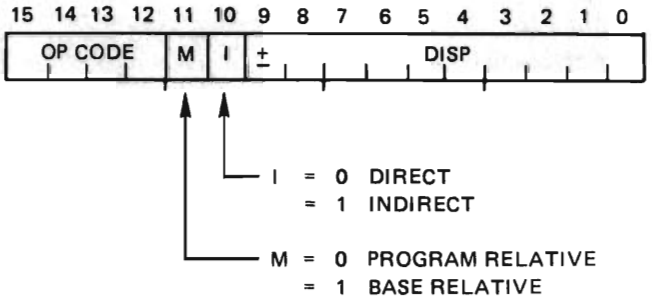
Several instruction groups use register identifier codes. These codes are listed below; they are the same for all instructions.

Bit Code	Register	Bit Code	Register
000	A	100	B
001	X	101	C
010	Y	110	D
011	Z	111	E

3.1 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 the A register or effect a jump to another instruction or subroutine.

Memory Reference instructions have the following machine instruction format:



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

Mnemonic	Parameters
LDA STA JMP JSR	[*]a[,m]

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

a is an address expression

m indicates addressing mode (bit 11)
= 0 Program Relative
= 1 Base Relative

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

1. If m is absent from the CAP-16 source statement, addressing mode is determined by the assembler:

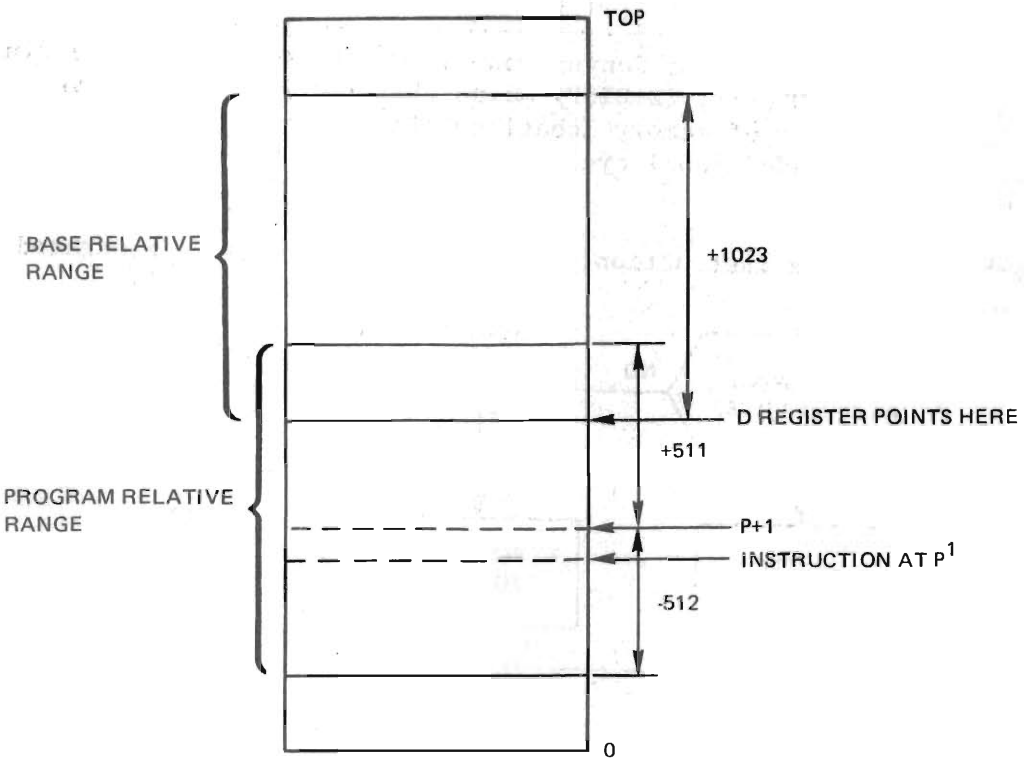
   Program Relative: a - $+1 → bits 0 - 9 (twos complement form)
   Base Relative: a - $$ ⟶ bits 0 - 9

2. If m is specified: a⟶ bits 0 - 9

NOTE

If "m" is specified, the address expression is a displacement and must evaluate to a constant within the range 0 to 1023 for m = 1 and -512 to +511 for m = 0 (i.e., may not include a relocatable label value).

The addresses which may be accessed directly by memory reference instructions are illustrated below. Locations outside the given range are addressable indirectly via an in-range location. (The CAP-16 Assembler automatically performs this function for directly addressed locations that are out of range.)



[1] MEMORY REFERENCE INSTRUCTION LOCATED ANYWHERE IN MEMORY.

Effective address coding is summarized in Table 3-1.

Table 3-1. EA Coding, Memory Reference Instructions

Type of Address	M	I	Range	EA[1)]	Next Instruction	Additional Cycles [2)]
Program Relative	0	0	-512≤DISP≤+511	P+1 ± DISP	P+1	0
Program Relative, Indirect	0	1		(P+1 ± DISP)	P+1	1
Base Relative	1	0	0≤DISP≤1023	D + DISP	P+1	0
Base Relative, Indirect	1	1		(D + DISP)	P+1	1

1. Recall that as described under Convention e), D+DISP means that the final Effective Address is D+DISP; (D+DISP) means that the final Effective Address is the contents of memory location D+DISP.
2. Instruction cycles + additional cycles = total processor cycles for instruction.

Upon executing a Memory Reference instruction, the effective address is generated as shown in Figure 3-1.

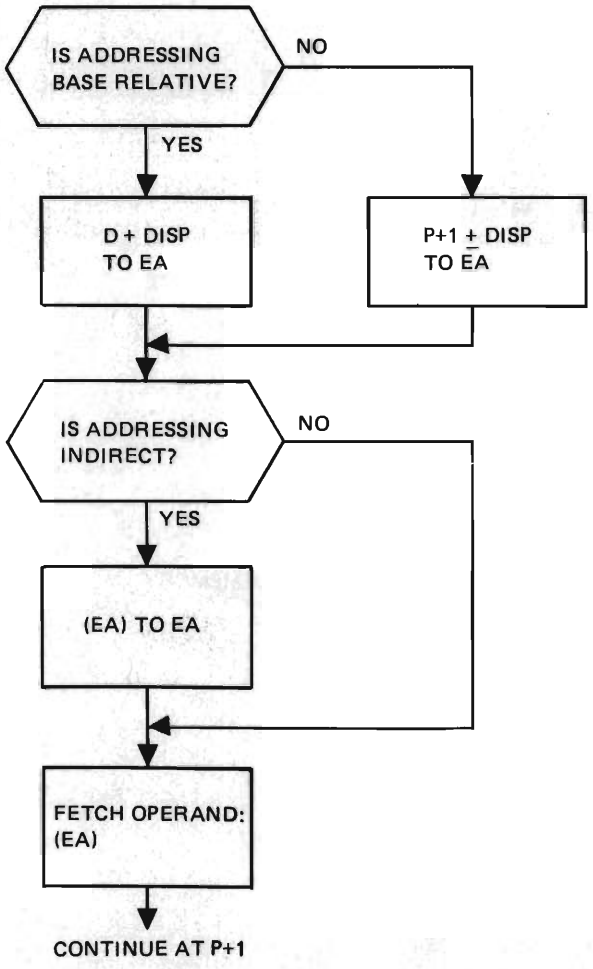
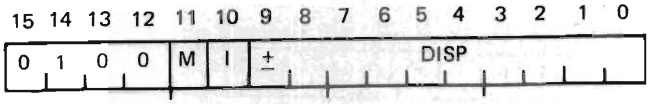


Figure 3-1. EA Generation, Memory Reference Instructions

3.1.1 Load Register A (LDA)

2 cycles



The contents of the location as specified by the EA replaces the contents of the A register.

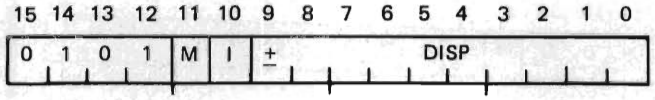
$$(EA)_{0-15} \rightarrow A_{0-15}$$

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) are loaded into the A register.
LDA SMRT	The contents of location SMRT are loaded into the A register.
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 of the D register are loaded into the A register.
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 are used as an address, which may specify any location in memory. The contents of the latter location are loaded into the A register.

3.1.2 Store Register A (STA) — 2 cycles



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

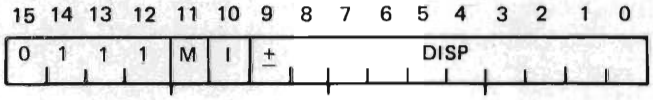
$$A_{0-15} \rightarrow (EA)_{0-15}$$

Indicators affected: None
Registers affected: None

Example:

STA *VAL	The contents of memory location VAL provide the address where the contents of the A register 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.

3.1.3 Jump Unconditionally (JMP) — 1 cycle



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

$$EA_{0-14} \rightarrow P_{0-14}$$

Indicators affected: None
Registers affected: None

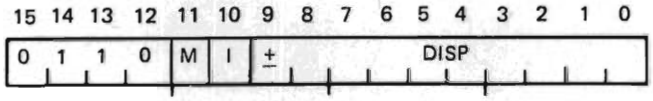
This instruction is not interruptable. Following execution of the JMP instruction, the instruction at the EA will be executed before the CPU will allow itself to be interrupted.

Examples:

JMP START	Where START is in the program section. Program control is transferred to the instruction having the label STAPT.

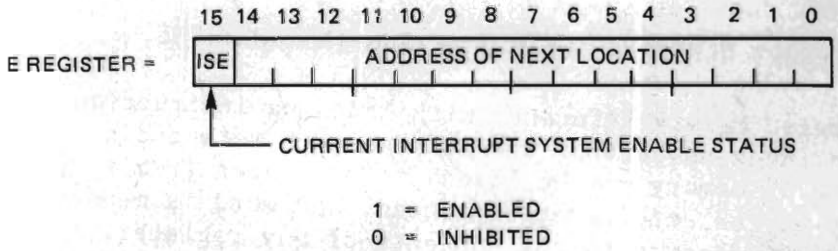
```
   JMP  *START        Program control is transferred to the instruction
START {define address}whose address is contained in the next memory
                      location (at START). The instruction to be executed
                      next may be located anywhere in memory.
```

3.1.4 Jump to Subroutine (JSR) — 1 cycle



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

$P \rightarrow E_{0-14}$
$ISE \rightarrow E_{15}$
$EA_{0-14} \rightarrow P_{0-14}$
$0 \rightarrow ISE$



Indicators affected: ISE
Registers affected: E

The contents of the A,B,C,D,X,Y and Z registers 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 3.2.13).

The return from a subroutine is normally made using a RTRN E instruction (Section 3.6.15). The ISE of the calling program may be duplicated in the subroutine with the RISE E instruction (Section 3.6.16), or interrupts may be specifically enabled with the INE instruction (Section 3.8.1). Otherwise, interrupts remain disabled during the execution of the subroutine.

Since the address transferred to the E register is $+1, a JSR instruction will not be properly executed if it is fetched from the last location of memory.

When the CPU acknowledges an 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 the E register.

Examples:

```
        JSR   SUBRUT
GO      LDA   MEAS
```

where SUBRUT is a label in the program section. The address of the instruction labeled GO and the current interrupt status are stored in the E register; the Interrupt System Enable is inhibited, and program control is transferred to location SUBRUT.

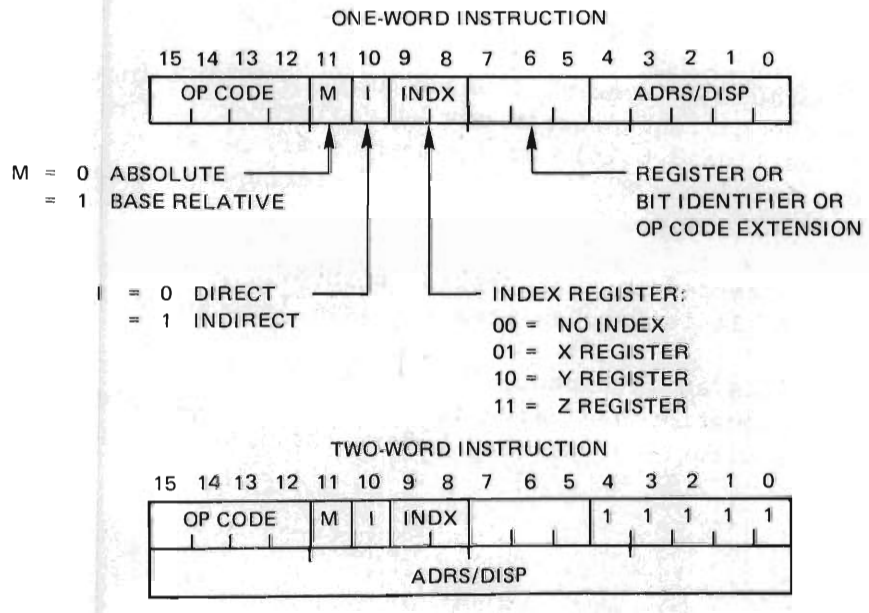
```
        JSR   *$$+1
```

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 the E register and the Interrupt Status Enable is inhibited. Control is then transferred to the subroutine.

3.2 MEMORY REFERENCE WITH INDEXING INSTRUCTIONS

The twelve Memory Reference with Indexing instructions 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/store all registers and status, allowing efficient interruptable, reentrant subroutine coding.

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



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

Command	Parameters
LDR STR CMR LDBY STBY LARS SARS INCM DECM SBIT RBIT TBIT	(∂)[*]a [,[i] [,m]]

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

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

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

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

a is an address expression

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

m indicates the addressing mode (absolute or base relative) and instruction length as follows:
= 0 one word instruction (bits 0-4 = ADRS) with absolute addressing (bit 11=0)
= 1 one word instruction (bits 0-4 = DISP) with base relative addressing (bit 11=1)
= 2 two word instruction (first word bits 0-4 = X'1F' and second word bits 0-14 = ADRS) with absolute addressing (bit 11=0)
= 3 two word instruction (first word bits 0-4 = X'1F' and second word bits 0-15 = DISP) with base relative addressing (bit 11=1)

The ADRS/DISP is determined as follows:

Absolute ADRS: a→ bits 0-4 (or 0-15 of second word)

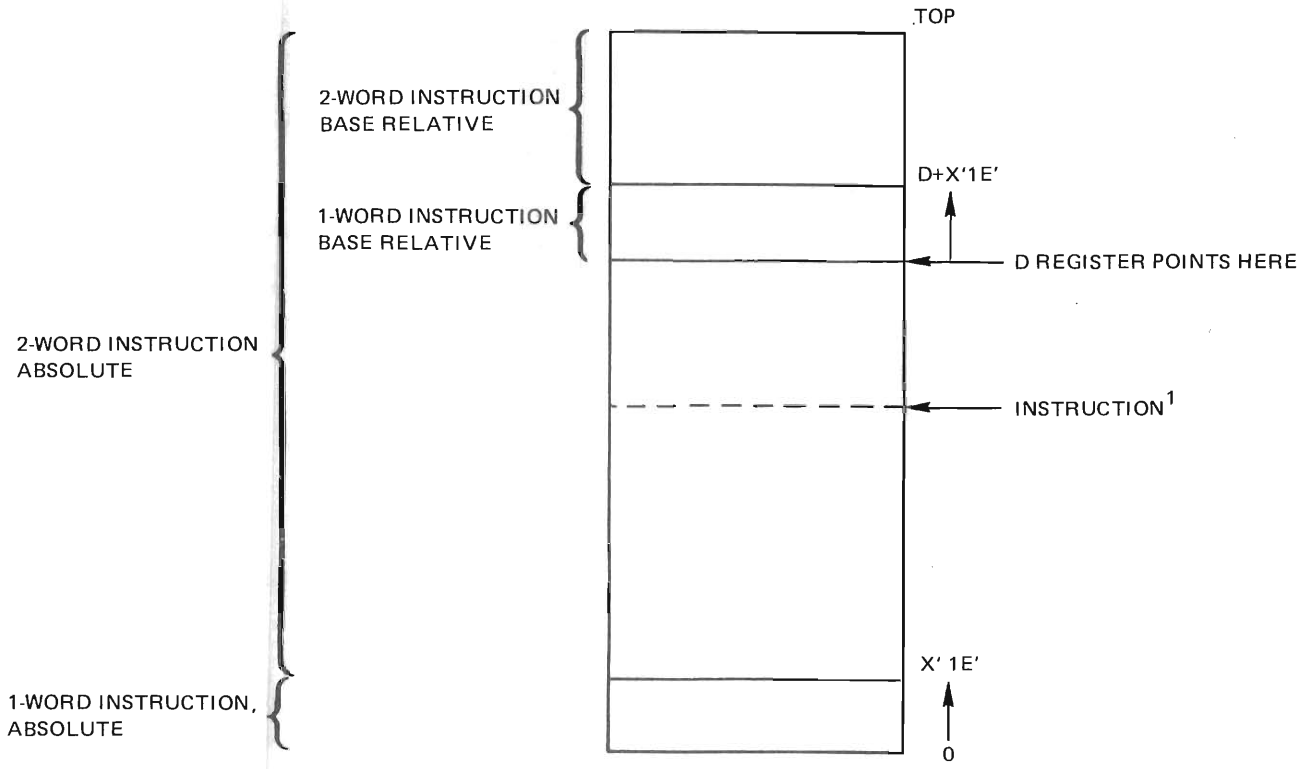
Base Relative DISP: if m is absent:
a - $$→ bits 0-4 (or 0-15 of second word)

if m is present:
a→ bits 0-4 (or 0-15 of second word)

NOTE

If the programmer wishes to specify m where 'a' contains a relocatable label value or a label value outside the range 0-30_{10}, a two-word instruction (m=2 or m=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.

Effective address coding is summarized in Table 3-2.

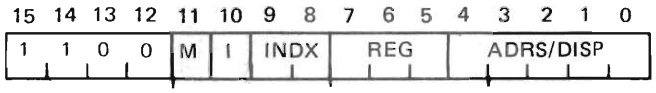
Table 3-2. EA Coding, Memory Reference with Indexing Instructions

Absolute	M	I	Indx	Adrs	EA[1]	Next Instruction	Additional Cycles[2]
Direct	0	0	0	<1F	ADRS	P+1	0
Direct, Indexed	0	0	1-3	<1F	ADRS + INDX	P+1	0
Direct	0	0	0	=1F	(P+1)	P+2	1
Direct, Indexed	0	0	1-3	=1F	(P+1) + INDX	P+2	1
Indirect	0	1	0	<1F	(ADRS)	P+1	1
Indirect, Indexed	0	1	1-3	<1F	(ADRS) + INDX	P+1	1
Indirect	0	1	0	=1F	((P+1))	P+2	2
Indirect, Indexed	0	1	1-3	=1F	((P+1)) + INDX	P+2	2
Base Relative				DISP			
Direct	1	0	0	<1F	D + DISP	P+1	0
Direct, Indexed	1	0	1-3	<1F	D + DISP + INDX	P+1	0
Direct	1	0	0	=1F	D + (P+1)	P+2	1
Direct, Indexed	1	0	1-3	=1F	D + (P+1) + INDX	P+2	1
Indirect	1	1	0	<1F	(D + DISP)	P+1	1
Indirect, Indexed	1	1	1-3	<1F	(D + DISP) + INDX	P+1	1
Indirect	1	1	0	=1F	(D + (P+1))	P+2	2
Indirect, Indexed	1	1	1-3	=1F	(D + (P+1)) + INDX	P+2	2

1) INDX/2 for LDRY, STBY, SBIT, RBIT and TBIT instructions.
2) Instruction cycles + additional cycles = total processor cycles for instruction.

Upon execution of a Memory Reference with Indexing instruction, the effective address is generated as shown in Figure 3-2.

3.2.1 Load Register (LDR)



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

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

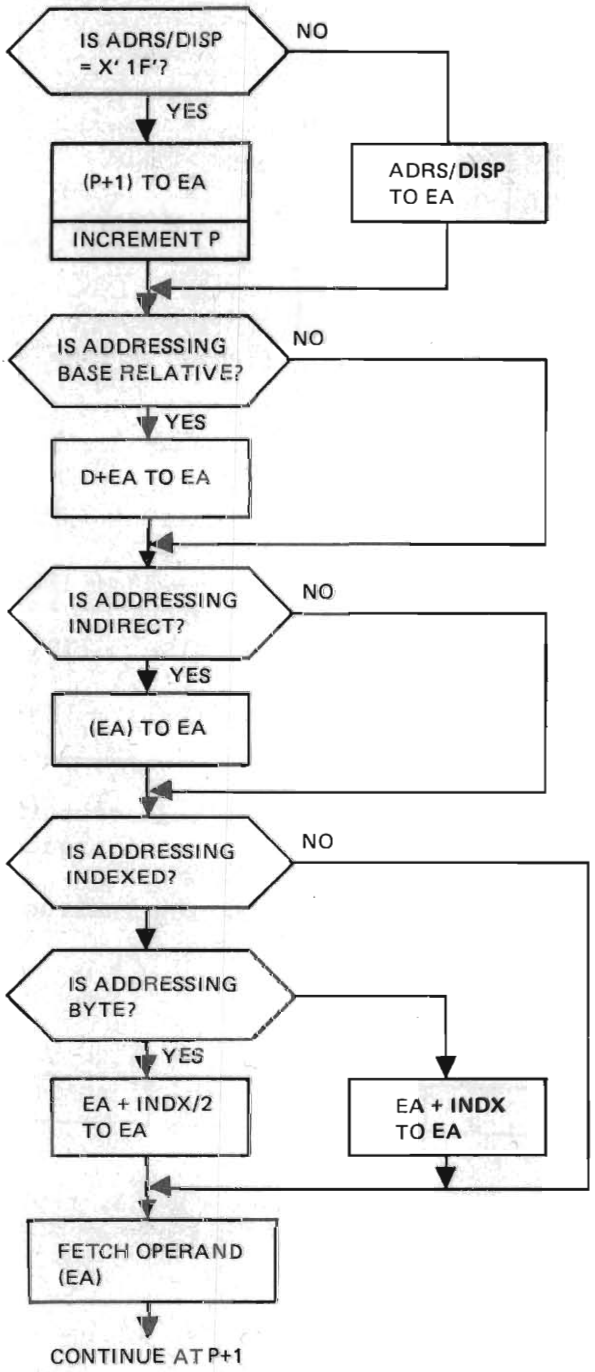


Figure 3-2. EA Generation, Memory Reference with Indexing Instructions

Indicators affected: None
Registers affected: Selected register

Examples:

```
LDR  A,0
```
Addressing is direct and absolute and the instruction is contained in a single word. The contents of memory location 0 are loaded into the A register.

```
LDR  B,*0
```
Addressing is indirect and absolute, and 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 the B register.

```
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 the X register is added to this address to specify the location whose contents are loaded into the A register.

```
LDR  Z,*20,,1
  or
LDR  Z,*$$+20
```
Addressing is indirect and base relative. Since the first stage address is within 31 locations from the base address, a one-word instruction is used where m is specified. For either instruction, location $$+20 contains the address of the location whose contents are loaded into the Z register.

```
LDR  B,ADRS,Y,2
```
Addressing is direct and absolute, with indexing. The contents of the absolute memory location, whose address is the sum of ADRS and the contents of the Y register, are loaded into the B register. ADRS may identify any memory location, since a two-word instruction is specified, and may be relocatable.

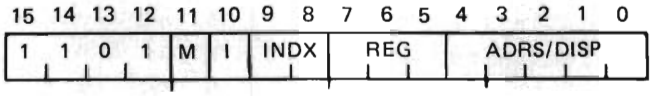
```
LDR  C,*X'1000',X,3
  or
LDR  C,*$$+X'1000',X
```
Addressing is indirect and base relative, with indexing. Either of these source statements will cause a two-word instruction to be generated; the second word will contain 1000_{16}. When this instruction is executed, the effective address is generated 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 = Final EA

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

3.2.2 Store Register (STR)

2 cycles



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

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

Indicators affected: None
Registers affected: None

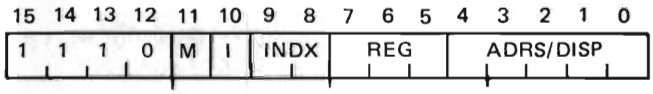
Example:

```
STR  A,LOC
```

Where LOC ≥ X'1F' and is in the data section. Addressing is direct and base relative, and a 2-word instruction would be generated. The store location, LOC, may be any forward reference from the base address. (Although the programmer will rarely use the capability, the store location could also be any reverse reference from the base address since the second word of the instruction may specify a negative displacement in twos complement form.) When this instruction is executed, the contents of the A register are stored at location LOC.

3.2.3 Compare Memory With Register (CMR)

2 cycles



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 result is different from the sign of the register contents. The Link will be set if there was a carry out of bit 15.

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

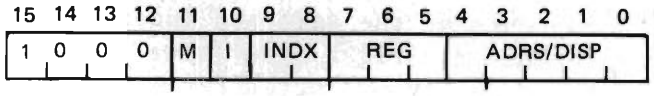
Indicators affected: Zero, Plus, Overflow, Link
Registers affected: None

Example:

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

3.2.4 Load Byte (LDBY)

2 cycles



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

If contents of index register are odd:

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

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

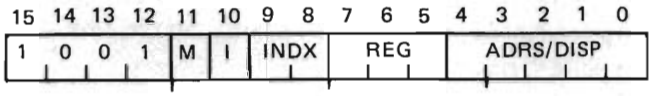
$$(EA)_{8-15} \rightarrow R_{0-7}$$

Indicators affected: None
Registers affected: Bits 0-7 of selected register

Example:

```
LDBY  A,ADDR4,X   Where ADDR4 is a label in the data section. Addressing
                  is direct, base relative, with indexing. The amount of
                  displacement is ADDR4-$$, and the EA is calculated as
                  the contents of the D register plus the amount of dis-
                  placement, plus the contents of the X register divided by
                  2. The contents of this location, bits 0-7 if X is
                  odd or bits 8-15 if X is even, are loaded into the A
                  register, bits 0-7.
```

3.2.5 Store Byte (STBY) — 2 cycles



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

If contents of selected index register are odd:

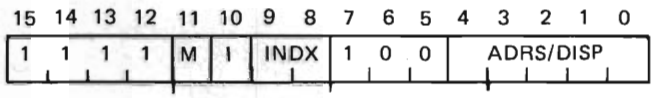
$$R_{0-7} \rightarrow (EA)_{0-7}$$

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

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

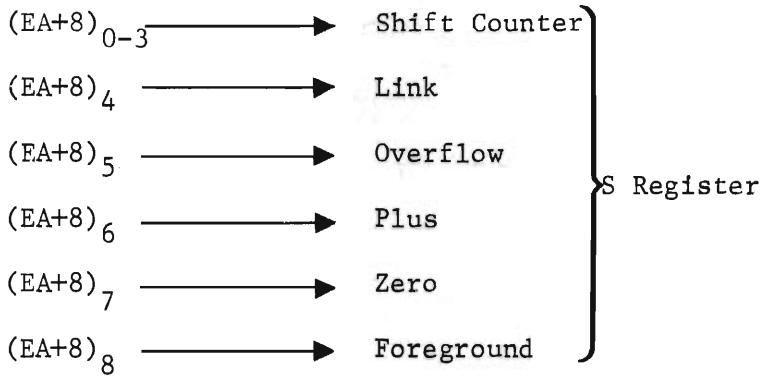
Indicators affected: None
Registers affected: None

3.2.6 Load All Registers and Status (LARS) — 10 cycles



The registers A,X,Y,Z,B,C,D, and E, and the Shift Counter and indicators (S Register) are loaded from the contents of nine consecutive locations as specified by EA through EA+8.

$$(EA)_{0-15} \rightarrow A_{0-15}$$
$$(EA+1)_{0-15} \rightarrow X_{0-15}$$
$$(EA+2)_{0-15} \rightarrow Y_{0-15}$$
$$(EA+3)_{0-15} \rightarrow Z_{0-15}$$
$$(EA+4)_{0-15} \rightarrow B_{0-15}$$
$$(EA+5)_{0-15} \rightarrow C_{0-15}$$
$$(EA+6)_{0-15} \rightarrow D_{0-15}$$
$$(EA+7)_{0-15} \rightarrow E_{0-15}$$

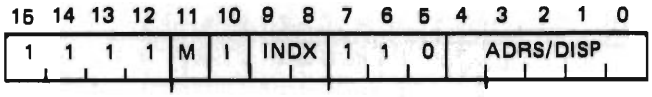


Indicators affected: Zero, Plus, Overflow, Link, Foreground
Registers affected: A,X,Y,Z,B,C,D,E

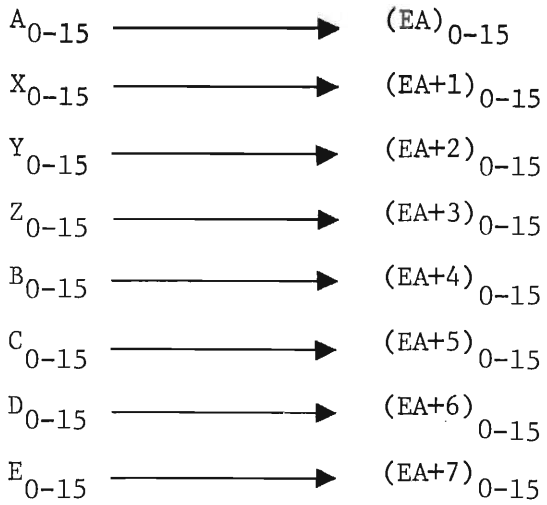
Example:

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).

3.2.7 Store All Registers and Status (SARS) 10 cycles



The registers A,X,Y,Z,B,C,D and E, and the Shift Counter and status indicators (S register), replace the contents of nine consecutive memory locations as specified by EA through EA+8.



S Register

- Shift Counter → $(EA+8)_{0-3}$
- Link → $(EA+8)_4$
- Overflow → $(EA+8)_5$
- Plus → $(EA+8)_6$
- Zero → $(EA+8)_7$
- Foreground → $(EA+8)_8$

Indicators affected: None
Registers affected: None

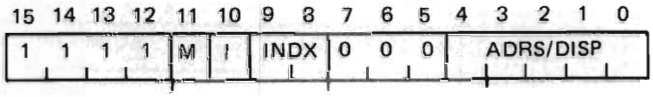
Example:

```
SARS  *0
```

The contents of the registers and indicators are stored into 9 consecutive memory locations. The address of the first memory location is contained in location 0 (i.e., 0 is an indirect address).

3.2.8 Increment Memory (INCM)

2 cycles



The contents of the location as specified by the EA are incremented by 1.

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

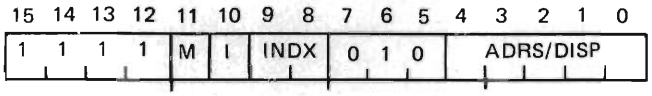
Indicators affected: Zero, Plus, Link (Note that overflow is not affected.)
Registers affected: None

Example:

```
INCM  ADR+4
```

The contents of memory location ADR+4 are incremented.

3.2.9 Decrement Memory (DECM) — 2 cycles



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

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

Indicators affected: Zero, Plus, Link (Note that overflow is not affected.)
Registers affected: None

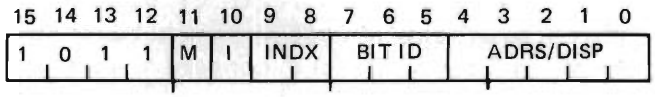
Example:

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

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

```
DECM  0,,1
```

3.2.10 Set Bit (SBIT) — 2 cycles



A 1 is placed in the specified bit of the location as 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 (0-7) of the right byte is set. If the contents of the selected index register are even or no index register is selected, the specified bit (0-7) of the left byte is set.

If index is odd:

$$(\overline{EA})_k \longrightarrow \text{Zero Indicator}$$

$$1 \longrightarrow (EA)_k$$

If index is even (or no index):

$(\overline{EA})_{k+8} \longrightarrow$ Zero Indicator

$1 \longrightarrow (EA)_{k+8}$

Indicators affected: Zero
Registers affected: None

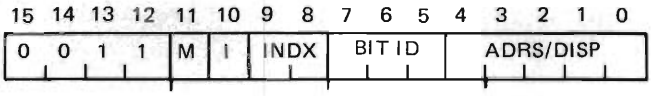
Example:

```
SBIT  4,BITS,Z
```

BITS identifies the beginning address of a bit array, and the value of Z divided by two identifies the word that contains the referenced bit. If the contents of Z are odd, bit 4 at the EA is set. If the contents of Z are even, bit 12 (i.e., bit 4 of the left byte) is set. The Zero indicator will contain zero if the specified bit was 1 before the operation or 1 if the specified bit was zero before the operation.

3.2.11 Reset Bit (RBIT)

2 cycles



A zero is placed in the specified bit of the location as determined by the EA. The previous status (zero, non-zero) of the specified bit is shown in the Zero indicator. 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 (0-7) of the right byte is reset. If the contents of the selected index register are even or no index register is selected, the specified bit (0-7) of the left byte is reset.

If contents of index register are odd:

$(\overline{EA})_{k} \longrightarrow$ Zero Indicator

$0 \longrightarrow (EA)_{k}$

If contents of index register are even (or no index):

$(\overline{EA})_{k+8} \longrightarrow$ Zero Indicator

$0 \longrightarrow (EA)_{k+8}$

Indicators affected: Zero
Registers affected: None

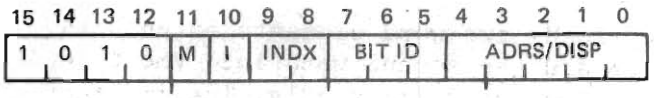
Example:

```
RBIT  2,4,Y,1
```

The beginning address of the bit array is calculated as the contents of the D register plus 4, and the value of Y divided by two identifies the word element in which the specified bit is contained. If the contents of Y are odd, bit 2 at the EA is reset; if the contents of 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 one if the specified bit was zero before the operation.

3.2.12 Test Bit (TBIT)

2 cycles



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

If contents of index register are odd:

$$(\overline{EA})_k \longrightarrow \text{Zero indicator}$$

If contents of index register are even (or no index register):

$$(\overline{EA})_{k+8} \longrightarrow \text{Zero indicator}$$

Indicators affected: Zero
Registers affected: None

Example:

```
TBIT  3,OME,Y
```

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

3.2.13 Examples

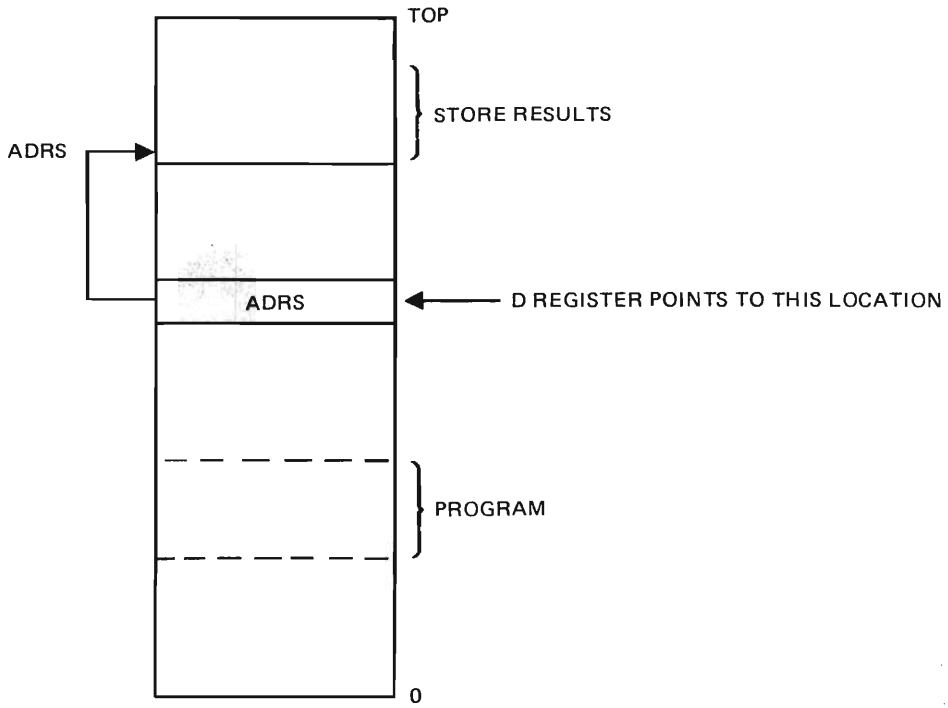
1. Beginning at the location pointed to by the base address, this instruction sequence performs an operation (not defined in the example) on consecutive elements of an array. When a test for exit is successful, the routine exits.

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

2. This sequence loads two-word elements of an array, placing the first word into the B register and the second word into the C register. 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 (Y register) is added to locate the array element.

```
            ZERO  Y              Zero index register to begin.
LOOP        LDR   B,*0,Y,1       Load B register with 1st word
            INCR  Y              Increment index.
            LDR   C,*0,Y,1       Load C register with 2nd word.
            .                    (Perform operations and test for exit.)
            .
            .
            INCR  Y              Increment to next 1st word.
            JMP   LOOP           Jump to load next 1st word.
```

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



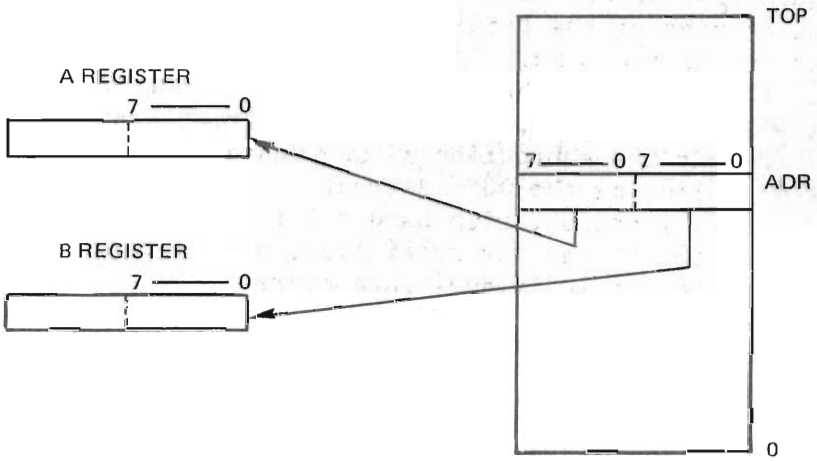
Main Program:

```
                ZERO   X              Zero index to start STORE at beginning
                                      of store array.
                .                     Do 1st computation, leaving result in A.
                JSR    STORE          Call subroutine STORE.
                .                     Do 2nd computation, leaving result in A.
                JSR    STORE          Call subroutine STORE.
                .
                .                     (Continue computations, calling STORE
                .                     for each result.)
                .
```

Subroutine:

```
STORE           STR    A,*0,X,1       Store A result.
                INCR   X              Increment index.
                RTRN   E              Return to main program.
```

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




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

* Has 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.

```
                ZERO   X
LOOP            .                     (Perform computation, leaving result
                .                     in right byte of A.)
```

```
                STBY  A,*$$,X        Store byte result.
                INCR  X             Increment index.
                .                   (Test for exit.)
                JMP   LOOP          Jump to computation for 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  temp            Store all registers/status.
                .                   (Perform computations.)
                LARS  temp          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.

   Main program:

```
                JSR   SUBR
                DC    LIST          (Define beginning location of parameters.)
HOH             .
                .
```

   Subroutine:

```
SUBR            SARS  *STACK        Store register/status.
                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.
```

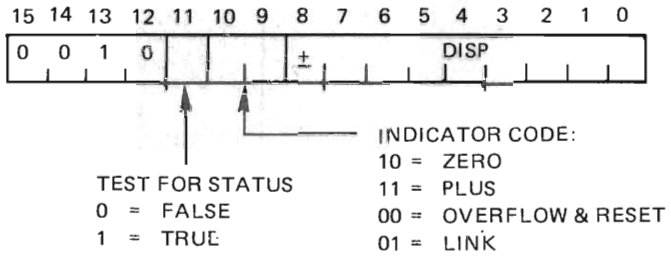
```
EXIT        LARS    *STACK      Restore original register contents.
            INCR    E           Increment E to return at location HOH.
            RTRN    E           Return to main program at HOH.
```

3.3 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:



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

Command	Parameters
SKZ	
SKN	
SKP	
SKM	
SKOT	a
SKOF	
SKS	
SKR	

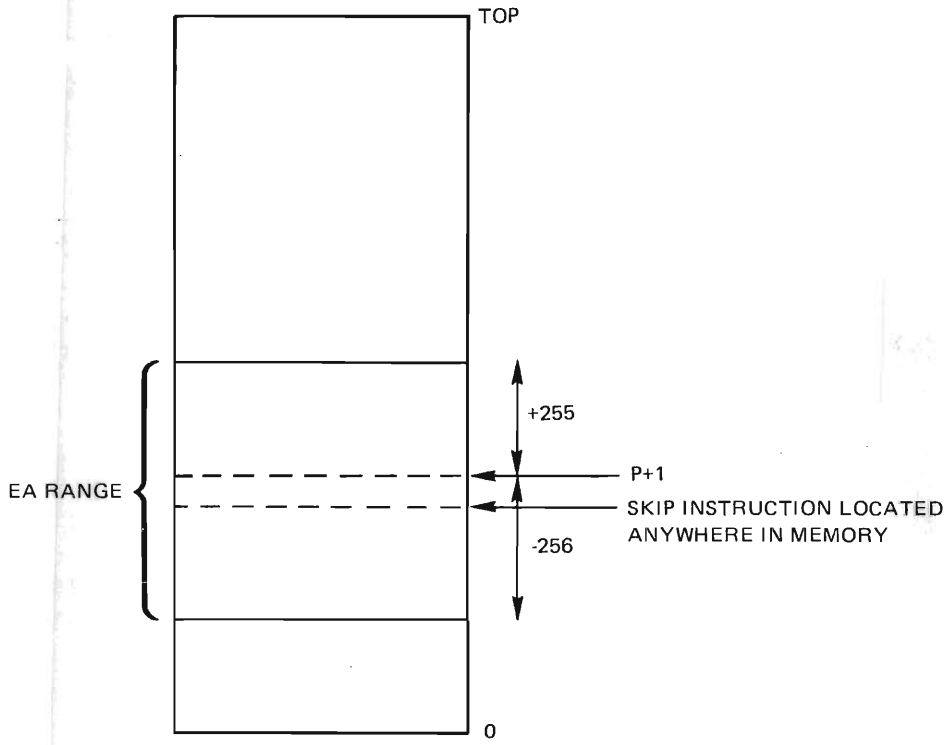
where: a is an address expression.

The addressing mode for this instruction group is always program relative, direct. The address displacement is determined as follows:

a - $+1→ bits 0-8 (twos complement form)

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

The addressing range is illustrated below:



The effective address, EA, is generated using the signed displacement, which is represented in twos complement form in bits 0-8. Bit 8 is extended to bits 9 through 15 to compute the effective address. EA generation is diagrammed in Figure 3-3.

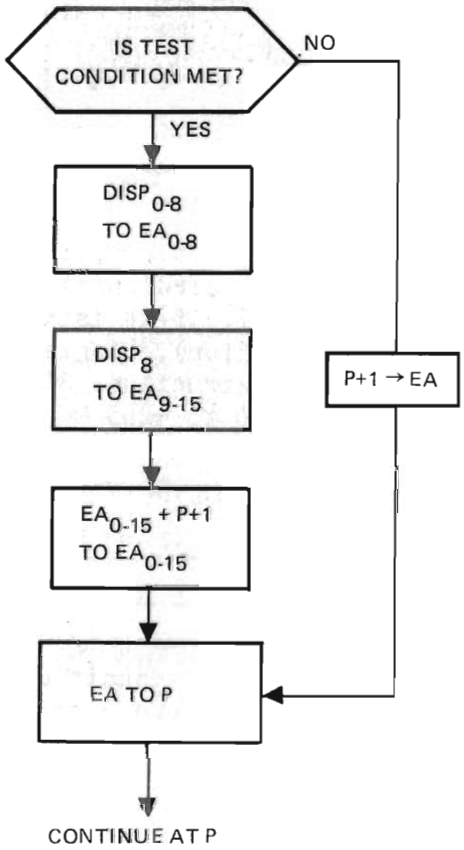
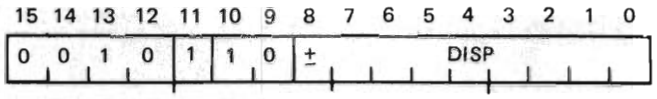


Figure 3-3. EA Generation, Conditional Jump Instructions

3.3.1 Skip if Zero (SKZ) — 1 cycle



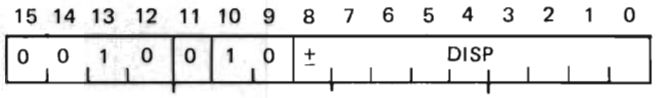
If the Zero indicator is set (zero condition exists), control is transferred to the EA. If the Zero indicator is not set (non-zero condition exists), execution continues with the next instruction.

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.

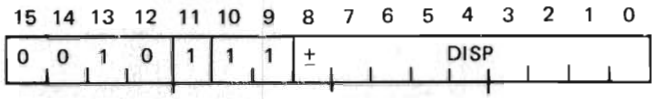
3.3.2 Skip If Not Zero (SKN) — 1 cycle



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

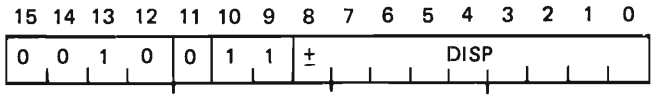
3.3.3 Skip If Plus (SKP) — 1 cycle



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

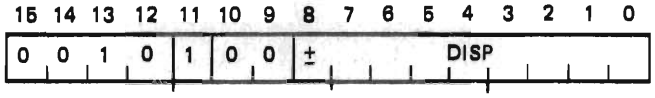
3.3.4 Skip If Minus (SKM) — 1 cycle



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

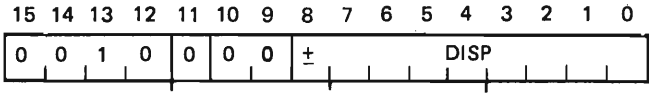
3.3.5 Skip If Overflow True (SKOT) — 1 cycle



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→ Overflow
Registers affected: None

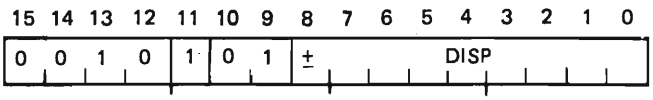
3.3.6 Skip If Overflow False (SKOF) — 1 cycle



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

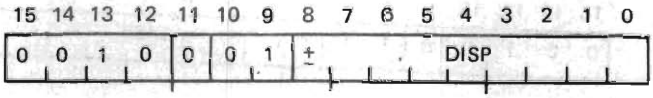
3.3.7 Skip If Link Set (SKS) — 1 cycle



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

3.3.8 Skip If Link Reset (SKR) — 1 cycle



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

3.3.9 Examples

1. 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 C register.
            .
            .                    (Perform computations.)
            .
NEXT                             (Test for done.)
            INCR  X              Increment index for next byte.
            JMP   LOOP           Jump to test next byte.
```

3.4 REGISTER OPERATE AND REGISTER OPERATE COMPARE INSTRUCTIONS

Register Operate and Register Operate Compare instructions are all one-word, single cycle, register-to-register operations. All Register Operate and Register Operate Compare instructions execute twice as fast when fetched from a Read-Only memory as from a Read/Write memory.

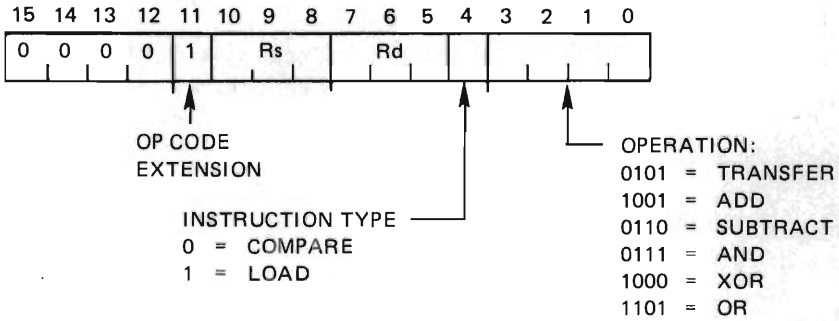
Register Operate instructions may be used to :

- Transfer the contents of one register to another.
- Add or subtract the contents of two registers.
- Perform one of the 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.

There are 11 instructions in this group; each has the following machine instruction format:



If bit 4=1, the operation determined by bits 0-3 is performed, the result replaces the contents of the destination register, and the conditions of the result are placed into the indicators. If bit 4=0, the operation determined by bits 0-3 is performed and the conditions of the result are placed into the indicators only; the destination register remains unchanged.

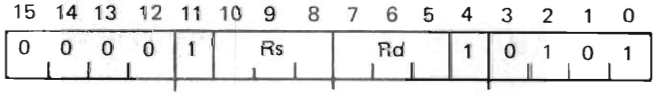
The results of register transfer (RTR) and logical operations are shown in the Zero and Plus indicators. The results of arithmetic operations (addition, subtraction) are shown in the Zero, Plus, Overflow and Link indicators.

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

Command	Parameters
RTR ADD SUB AND XOR OR ADDC SUBC ANDC XORC ORC	Rd,Rs

where: Rd is the destination register (determines the coding for bits 5-7).
Rs is the source register (determines the coding for bits 8-10).

3.4.1 Transfer Register (RTR) — 1 cycle



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

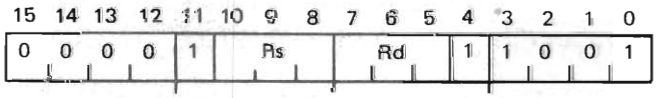
$$Rs_{0-15} \longrightarrow Rd_{0-15}$$

Indicators affected: Zero, Plus
Registers affected: Destination Register

Example:

RTR D,X	The contents of the X register replace the contents of the D register. The X register is unchanged.

3.4.2 Add Registers (ADD) — 1 cycle



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.

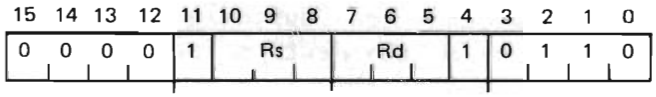
$Rd_{0-15} + Rs_{0-15} \rightarrow Rd_{0-15}$

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

Example:

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

3.4.3 Subtract Registers (SUB) — 1 cycle

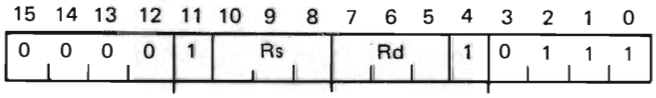


The contents of the source register are subtracted from the contents of the destination register. The results is stored in the destination register; the source register is unchanged.

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

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

3.4.4 And Registers (AND) — 1 cycle



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.

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

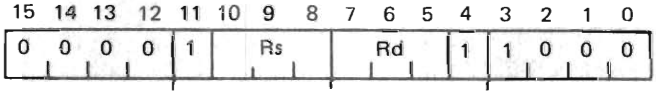
Indicators affected: Zero,Plus
Registers affected: Destination Register

Example:

```
AND  A,B        If: A =      0 ———— 01010
                    B =      0 ———— 01100
                             ————————————
                The result:  0 ———— 01000
```

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

3.4.5 Exclusive Or Registers (XOR) — 1 cycle



The contents of the source register are logically EXCLUSIVE ORed (⊻ symbolizes the XOR function) with the contents of the destination register and the results replace the contents of the destination register.

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

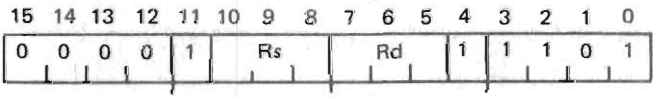
Indicators affected: Zero, Plus
Register affected: Destination Register

Example:

```
XOR  A,B        If: A =      0 ———— 01010
                    B =      0 ———— 01100
                             ————————————
                The results: 0 ———— 00110
```

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

3.4.6 Or Registers (OR) — 1 cycle



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.

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

Indicators affected: Zero, Plus
Registers affected: Destination Register

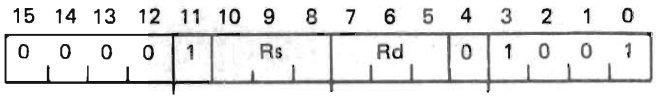
Example:

```
OR  A,B        If: A =    0--------01010
                   B =    0--------01100

               The result: 0--------01110
```

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

3.4.7 Add Compare Registers (ADDC) — 1 cycle



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

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

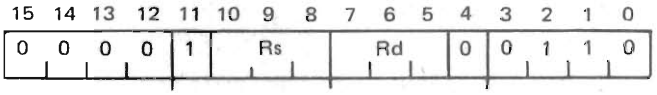
Indicators affected: Zero, Plus, Overflow, Link
Registers affected: None

Example:

```
ADDC  A,B
```

The contents of A are added to the contents of B. The results are reflected in the Zero, Plus, Overflow and Link indicators.

3.4.8 Subtract Compare Registers (SUBC) — 1 cycle

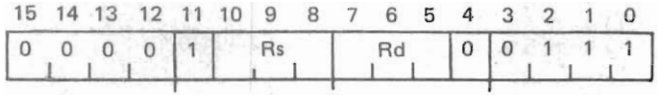


The contents of the source register are subtracted from the contents of the destination register, and the results are reflected in the indicators. Neither the source nor destination register is changed.

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

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

3.4.9 And Compare Registers (ANDC) — 1 cycle



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.

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

Indicators affected: Zero, Plus
Registers affected: None

3.4.10 Exclusive Or Compare Registers (XORC) — 1 cycle

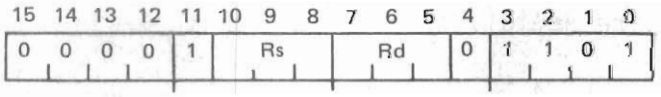


The contents of the source register are logically EXCLUSIVE ORed with the contents of the destination register, and the results are reflected in the indicators. Neither the source nor destination register is changed.

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

Indicators affected: Zero, Plus
Registers affected: None

3.4.11 Or Compare Registers (ORC) — 1 cycle



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.

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

Indicators affected: Zero, Plus
Registers affected: None

3.4.12 Examples

1. This sequence subtracts the number in the A register from the number in the B register, and places the result in the C register if result > 0, or clears the C register if result ≤ 0.

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

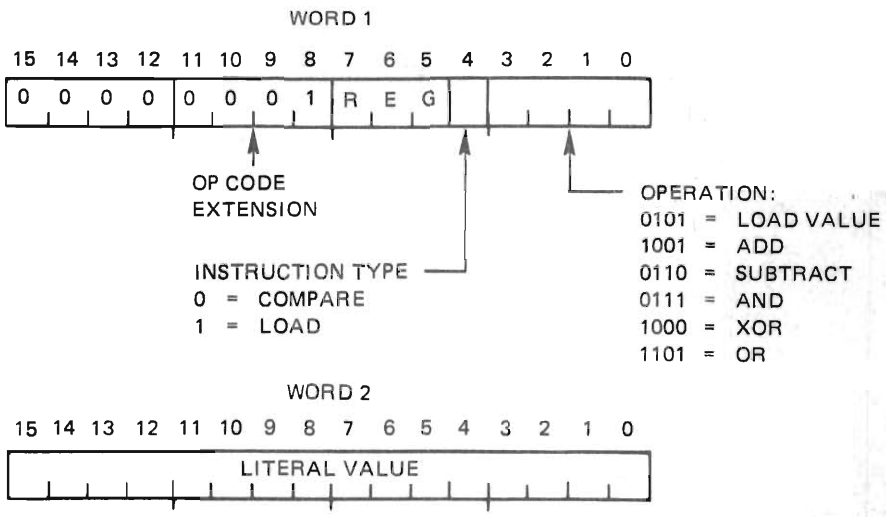
2. This sequence tests for overflow before adding the A and B registers.

```
                ADDC  A,B       Add compare A and B.
                SKOT  OFL       Skip if overflow true.
                ADD   A,B       Perform addition, leaving result in A.
                .
                .
OFL             .
                .
```

3.5 REGISTER OPERATE LITERAL AND REGISTER OPERATE LITERAL COMPARE INSTRUCTIONS

These 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 instructions in this group. Each instruction occupies two words and has the following format:



If bit 4=1, the operation determined by bits 0-3 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 0-3 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:

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

where: R specifies one of the eight general purpose registers;
R determines the coding of bits 5-7 in the first word of the instruction.

lit is the literal value. The literal value may be coded in CAP-16 assembly language in either decimal form (e.g., 15) or hexadecimal form (e.g., X'0F'). 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 EA is generated as the address of the next sequential location. EA generation is shown in Figure 3-4.

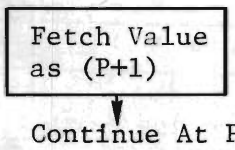
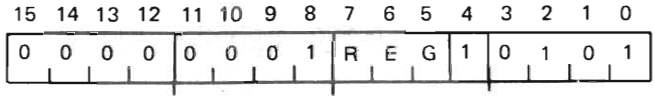


Figure 3-4. EA Generation, Literal Addressing Mode

3.5.1 Load Value to Register (LDV) — 2 cycles



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

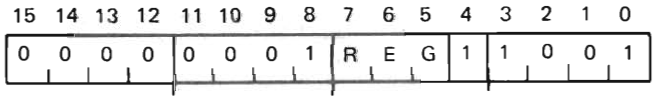
$$(P+1)_{0-15} \longrightarrow R_{0-15}$$

Indicators affected: Zero, Plus
Registers affected: Specified Register

Example:

```
LDV  X,10      The binary equivalent of 10(10) is loaded into
               the X register.
```

3.5.2 Add Value to Register (ADDV) — 2 cycles



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

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

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

Example:

```
ADDV  A,X'1F02'      The binary equivalent of 1F02(16) is
                     added to the contents of the A register,
                     and the result is placed into the
                     A register.
```

3.5.3 Subtract Value From Register (SUBV) — 2 cycles

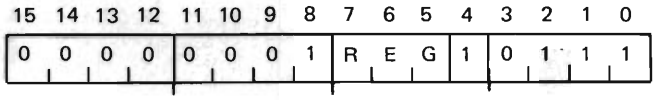


The second word of the instruction (i.e., the literal) is subtracted from the contents of the specified register; the result replaces the register contents.

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

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

3.5.4 And Value With Register (ANDV) — 2 cycles



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

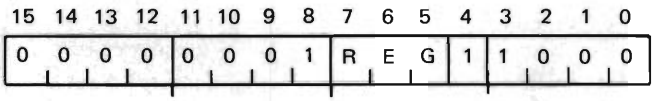
$R_{0-15} \wedge (P+1)_{0-15} \rightarrow R_{0-15}$

Indicators affected: Zero, Plus
Registers affected: Specified Register

Example:

```
ANDV  B,X'0063'    X'0063'          = 0————————01100011
                   If B register    = 0————————00011111
                                      ————————————————
                   The result:        0————————00000011
                   is placed into the B register.
```

3.5.5 Exclusive Or Value With Register (XORV) — 2 cycles

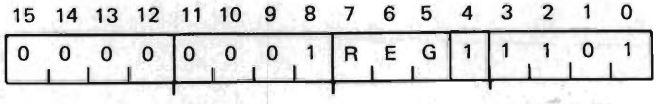


The second word of the instruction (i.e., the literal) is logically EXCLUSIVE ORed with the contents of the specified register; the results are placed in the specified register.

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

Indicators affected: Zero, Plus
Registers affected: Specified Register

3.5.6 Or Value With Register (ORV) — 2 cycles

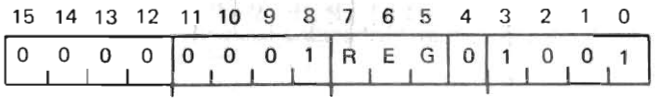


The second word of the instruction (i.e., the literal) is logically ORed with the contents of the specified register; the results are placed in the specified register.

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

Indicators affected: Zero, Plus
Registers affected: Specified Register

3.5.7 Add Compare Value With Register (ADDVC) — 2 cycles



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

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

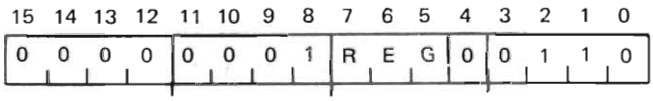
Indicators affected: Zero, Plus, Overflow, Link
Registers affected: None

Example:

```
ADDVC  X,50
```

The binary equivalent of 50_{10} is added to the contents of the X register, with the results appearing in the indicators only. The X register is not changed.

3.5.8 Subtract Compare Value With Register (SUBVC) — 2 cycles

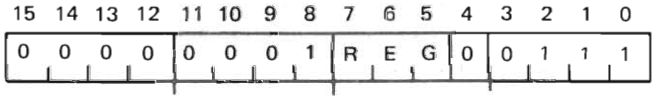


The second word of the instruction (i.e., the literal) is subtracted from the contents of the specified register; the indicators are set according to the results.

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

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

3.5.9 And Compare Value With Register (ANDVC) — 2 cycles

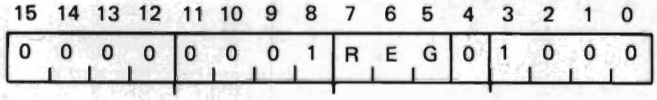


The second word of the instruction (i.e., the literal) and the specified register are logically ANDed; the indicators are set according to the results.

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

Indicators affected: Zero, Plus
Register affected: None

3.5.10 Exclusive Or Compare Value With Register (XORVC) 2 cycles

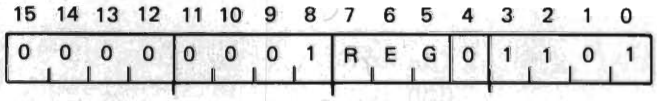


The second word of the instruction (i.e., the literal) and the specified register are logically EXCLUSIVE ORed; the indicators are set according to the results.

$R_{0-15} \veebar (P+1)_{0-15} \longrightarrow$ Data Bus

Indicators affected: Zero, Plus
Registers affected: None

3.5.11 Or Compare Value With Register (ORVC) 2 cycles



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

$R_{0-15} \vee (P+1)_{0-15} \longrightarrow$ Data Bus

Indicators affected: Zero, Plus
Registers affected: None

3.5.12 Examples

1. This routine searches for an item in an unordered list of 100 items. The item to be found in the list is contained in the A register. 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 the Y register 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 the Y register by the LDV instruction. The index is decremented to the next list item after a comparison has been made by the CMR instruction. Upon completing the search, the absolute address of the list item is in the B register. If the item is not in the list, B is 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 = 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 list beginning address in answer
                                      register.
                ADD    B,Y            Add index position of element.
EXIT            .                     Answer is in B register.
                .
```

2. ANDVC may be used to test a byte or a bit of a register for zero. R is the specified register.

   To test right byte:

```
                ANDVC  R,X'00FF'
                SKZ    ZERO           Jump if right byte is zero.
```

   To test left byte:

```
                ANDVC  R,X'FF00'
                SKZ    ZERO           Jump is left byte is zero.
```

   To test bit 'n' of register R for zero, $0 \leq n \leq 15$, the literal is a mask composed of 15 zeros and a 1 in the bit position to be tested:

```
                ANDVC  R,lit          where lit = 2**n
                SKZ    ZERO
```

   For example, to test bit 3 of the A register:

```
                ANDVC  A,X'0008'      8=2**3
                SKZ    ZERO
```

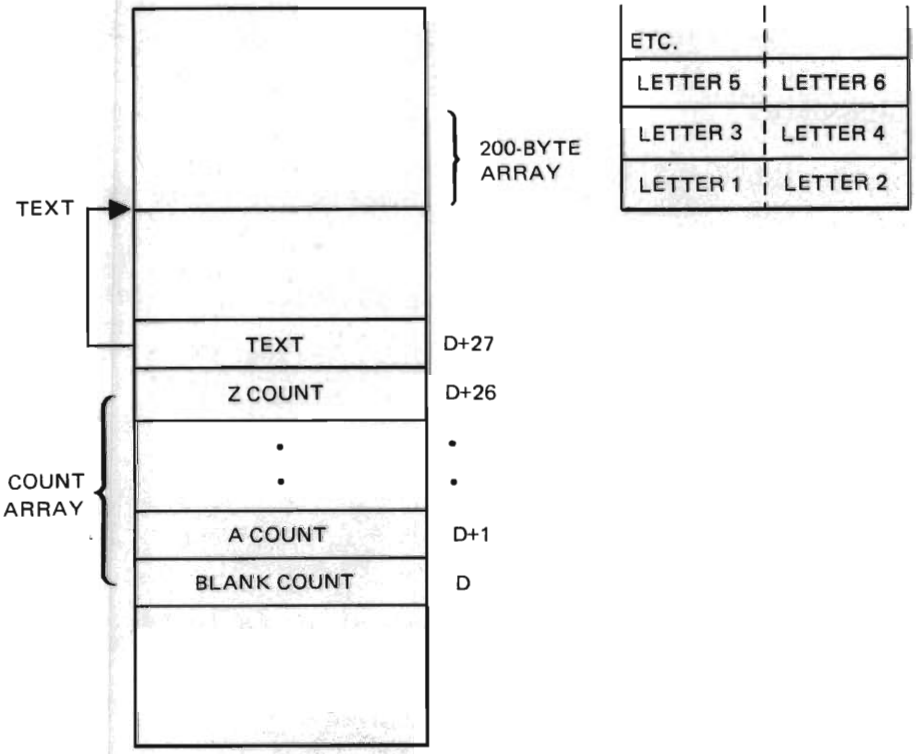
3. A function of NUM, with NUM in the A register, is to be completed according to the following formula:

```
                f(NUM)=NUM+f(NUM-2)   for NUM ≥ 10
                f(NUM)=0              for NUM < 10
```

   In the sequence, successive values of NUM are added to the subtotal in the B register 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 B register.
```

4. This sequence counts the number of times each letter of the alphabet occurs in a piece of text. The ASCII-coded text (see Appendix B) is contained in a 200-byte array labeled TEXT; the beginning address, TEXT, is contained at location D+27. A loop counter in the X register 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-Z and Blank.) This is illustrated as follows:



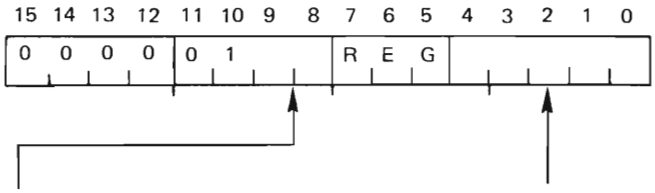
```
                ZERO  X              Zero TEXT array index to start.
LOOP            LDBY  Y,*27,X,1      Load letter into Y.
                ANDV  Y,X'1F'        Strip 3 high order bits to get Count
                                     Array Index.
                INCM  0,Y,1          Add 1 to letter count.
                INCR  X              Next letter.
                SUBVC X,200          End of table test.
                SKM   LOOP           If not done, fetch next letter.
DONE            .
                .
```

3.6 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.

Each of these instructions requires one cycle, with the exception of the Execute Register Contents instruction, which takes one cycle plus the cycles required to execute the specified instruction. All Register Change instructions execute twice as fast when fetched from a Read-Only memory as from a Read/Write memory.

There are 16 Register Change instructions, and each of these instructions has the following format:



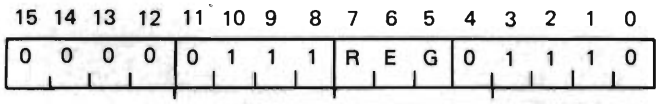
11	INCREMENT REGISTER	01110
11	DECREMENT REGISTER	00010
11	COMPLEMENT REGISTER	00000
10	ZERO REGISTER	00000
10	ZERO LEFT BYTE	00010
10	ZERO RIGHT BYTE	00001
10	EXCHANGE BYTES	00100
11	REGISTER ADD LINK	00001
11	ADD SHIFT COUNTER TO REGISTER	01011
10	TRANSFER STATUS TO REGISTER	01000
01	TRANSFER REGISTER TO STATUS	01000
10	READ CONSOLE SWITCHES	10000
01	DISPLAY REGISTER CONTENTS	00100
01	EXECUTE REGISTER CONTENTS	10000
01	SUBROUTINE RETURN	00011
01	RESTORE INTERRUPT SYSTEM ENABLE	00001

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

Command	Parameters
INCR DECR CMPL ZERO ZLBY ZRBY EXBY RLK ADDS TSR TRS RCSW DSPL XEC RTRN RISE	R

where: R specifies one of the eight general purpose registers, coded into bits 5-7.

3.6.1 Increment Register (INCR) — 1 cycle



The specified register is incremented by one.

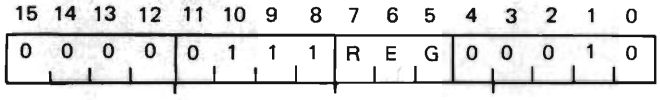
$$R_{0-15} + 1 \rightarrow R_{0-15}$$

Indicators affected: Zero, Plus, Link (Note that overflow is not affected.)
Registers affected: Specified Register

Example:

```
INCR  A       One is added to the contents of the A register.
```

3.6.2 Decrement Register (DECR) — 1 cycle



The specified register is decremented by 1.

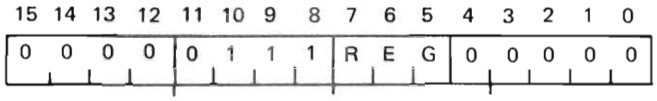
$$R_{0-15} - 1 \rightarrow R_{0-15}$$

Indicators affected: Zero, Plus, Link (Note that overflow is not affected.)
Registers affected: Specified Register

Example:

```
DECR  B      Subtract one from the contents of the B register.
```

3.6.3 Complement Register (CMPL) — 1 cycle



The contents of the specified register is replaced by its ones complement.

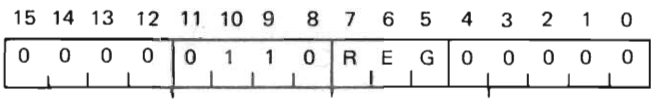
$$\overline{R}_{0-15} \rightarrow R_{0-15}$$

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 the B register =  1110001011000000
             The 1's complement:  0001110100111111
             replaces the value in the B register
             and the Link is set.
```

3.6.4 Zero Register (ZERO)



The specified register is cleared

$$0 \rightarrow R_{0-15}$$

Indicators affected: None
Registers affected: Specified Register

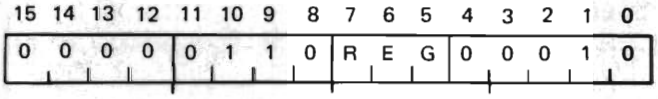
Example:

```
ZERO  X      The contents of the X-register are replaced
             with zeros.
```

The two-cycle, double word instruction:

```
LDV   X,0    Places all zeros in the X register but, in
             addition, affects the Zero and Plus indicators.
```

3.6.5 Zero Left Byte (ZLBY) — 1 cycle



The left byte of the specified register is cleared. The right byte is not affected.

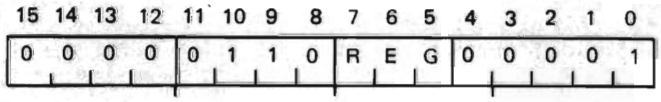
$0 \rightarrow R_{8-15}$

Indicators affected: None
Registers affected: Bits 8-15 of the specified register.

Example:

```
ZLBY  A       If A contained X'FFFF', ZLBY would replace
              this value with X'00FF'.
```

3.6.6 Zero Right Byte (ZRBY)



The right byte of the specified register is cleared. The left byte is not affected.

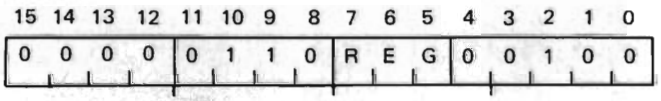
$0 \rightarrow R_{0-7}$

Indicators affected: None
Registers affected: Bits 0 - 7 of the specified register.

Example:

```
ZRBY  C       If C contained X'FFFF', ZRBY would replace
              this value with X'FF00'.
```

3.6.7 Exchange Bytes (EXBY) — 1 cycle



The left and right bytes of the specified register are interchanged.

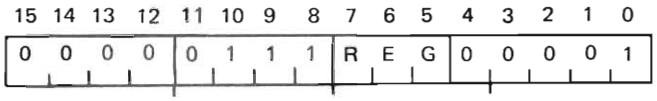
$R_{0-7} \leftrightarrow R_{8-15}$

Indicators affected: None
Registers affected: Specified Register

Example:

```
EXBY  C      If register C contains: 1111111100000000
             After exchanging left and right bytes,
             C contains:             0000000011111111.
```

3.6.8 Register Add Link (RLK) — 1 cycle



The content of the Link indicator is added to the contents of the specified register.

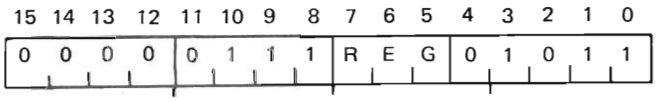
$$R_{0-15} + L \rightarrow R_{0-15}$$

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

Example:

```
RLK  A       If the Link is set (=1), the A register is
             incremented. If the Link is reset (=0), the
             A register is not changed.
```

3.6.9 Add Shift Counter to Register (ADDS) — 1 cycle



The contents of the Shift Counter are added to the selected register.

$$S_{0-3} + R_{0-15} \rightarrow R_{0-15}$$

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

ADDS is normally used after the Shift Right Logical and Count instruction (Section 3.7.3). The SRLC instruction shifts bits from bit 0 into the Link, stopping if a one is shifted into the Link or 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.

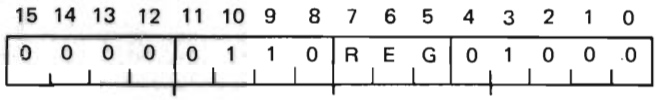
The value of the Shift Counter (0-15) specifies the bit position (0-15) of the one bit that was shifted into the Link (i.e., the first low-order one bit). The value of the Shift Counter plus one indicates the number of shifts that were performed. (The Link should be tested to determine if the operation halted as a result of the shift count being exhausted or a 1 being shifted into the Link.)

Example:

To find the bit position of the first low order 1 bit in the A register and place the bit position in the B register:

```
          ZERO   B        Zero answer register.
          SRLC   A,16     Shift until low order 1 bit is in Link.
          SKR    NONE     Exit if Link=0.
          ADDS   B        Read bit position.
          .
NONE      .
          .
```

3.6.10 Transfer Status to Register (TSR) — 1 cycle



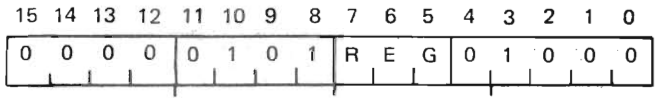
The contents of the Indicators and the Shift Counter are transferred to the specified register.

S Register:

- Shift Counter → R_{0-3}
- Link → R_4
- Overflow → R_5
- Plus → R_6
- Zero → R_7
- Foreground → R_8
- 0 → R_{9-15}

Indicators affected: None
Registers affected: Specified Register

3.6.11 Transfer Register To Status (TRS) — 1 cycle



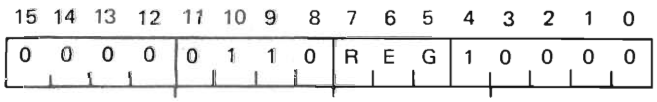
The appropriate bits of the specified register are loaded into the Indicators and Shift Counter. This instruction changes the state of the indicators and may, therefore, affect the mode (foreground, background).

$R_{0-3} \rightarrow$ Shift Counter
$R_4 \longrightarrow$ Link
$R_5 \longrightarrow$ Overflow
$R_6 \longrightarrow$ Plus
$R_7 \longrightarrow$ Zero
$R_8 \longrightarrow$ Foreground

} S Register

Indicators affected: Zero, Plus, Overflow, Link, Foreground
Registers affected: S Register

3.6.12 Read Console Switches (RCSW) — 1 cycle



The contents of the data entry switches on the computer console are loaded into the specified register. Switch down =1; switch up =0.

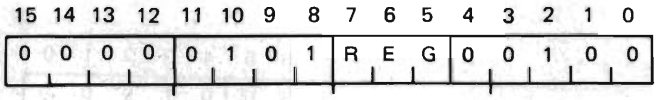
$(\text{CNSL SW})_{0-15} \rightarrow R_{0-15}$

Indicators affected: None
Registers affected: Specified Register

CAUTION

When this instruction is immediately followed by a cycle steal or an interrupt, the console switch data may be OR'ed with the cycle steal address or with the interrupt vector. To avoid this problem, use the Read Console Switches into Register command (RCSR) or the Read Console Switches into Memory command (RCSM). See Sections 3.9.7 and 3.9.8, respectively.

3.6.13 Display Register Contents (DSPL) 1 cycle



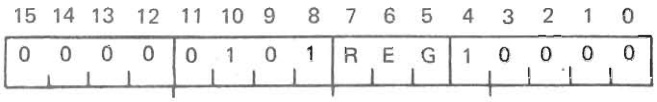
The contents of the specified register are loaded into the Console Display register (K Register), and the register contents are displayed on the console indicator lights. The display remains (during continued program execution) until the Console Display register is changed.

$$R_{0-15} \rightarrow K_{0-15}$$

Indicators affected: None
Registers affected: None

The least significant four bits of the K register are used as input to a 4-bit Digital-to-Analog Converter. The analog voltage is available on the I/O bus as signal VU+I. This voltage can be viewed on an oscilloscope for troubleshooting purposes.

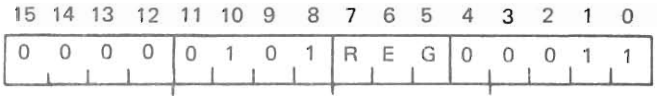
3.6.14 Execute Register Contents (XEC) — 1 cycle+



The contents of the specified register are loaded into the instruction register (I register) and executed as a single word instruction. Only single-word instructions can be executed with the XEC instruction. This excludes two-word Memory Reference with Indexing instructions (i.e., ADDR/DISP = X'1F') and Register Operate Literal/Compare instructions.

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.

3.6.15 Subroutine Return (RTRN) — 1 cycle



The contents of the specified register (normally the E register) are transferred to the P register and to the Interrupt System Enable.

$R_{0-14} \longrightarrow P_{0-14}$

$R_{15} \longrightarrow ISE$

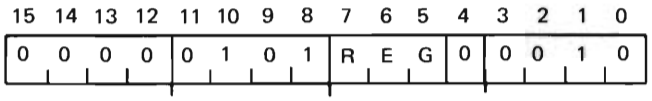
Indicators affected: ISE
Registers affected: None

If the restored status enables the interrupt system, interrupts may occur immediately following execution of the RTRN instruction in the case of internal interrupts only. External interrupts must wait at least two cycles (see Section 2.4).

Example:

	RTRN E	Control is returned to the calling program at the location specified by the contents of the E register, bits 0-14. The ISE status of the calling program prior to the jump-to-subroutine, which was placed into bit 15 of the E register, is placed back into the ISE, restoring the interrupt status of the calling program.

3.6.16 Exit Subroutine (EXIT) 1 cycle

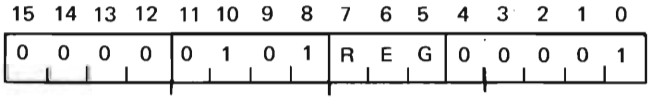


32K MODE

Bits 0-14 of the specified register are transferred to the corresponding positon of the P register ISE is unchanged.

R_{0-14}	P_{0-14},	ISE	Unchanged

3.6.17 Restore Interrupt System Enable (RISE) 1 cycle



This instruction transfers the content of bit 15 of the selected register to the Interrupt System Enable. (Logical 1 enables interrupts).

$R_{15} \longrightarrow$ ISE

Indicators affected: ISE
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. The E register is normally specified, since the Interrupt System Enable status was stored in the E register, bit 15, during the execution of the jump-to-subroutine (JSR) instruction.

If the transferred status enables the interrupt system, interrupts may occur immediately following execution of the RISE instruction in the case of internal interrupts only. External interrupts must wait at least 2 cycles (see Section 2.4).

Example:

Calling Program:

```
        .
        JSR     SUBR
        .
```

Subroutine:

```
SUBR    SARS    temp    Subroutine SUBR will now have the same
        RISE    E       interrupt status as the calling program.
        .
        .
```

3.6.18 Examples

1. To compute the absolute value of a number that is in the A register:

```
        ANDVC   A,X'8000'       Test sign bit.
        SKZ     EXIT            Exit if number is positive.
        CMPL    A               } Change negative number (2's complement)
        INCR    A               } to positive representation.
EXIT    .
        .
```

2. A number of programs are being run, and it is desired to show at the console which program is running. This can be done by numbering the programs and executing a DSPL instruction to place the program number in the console indicator lights.

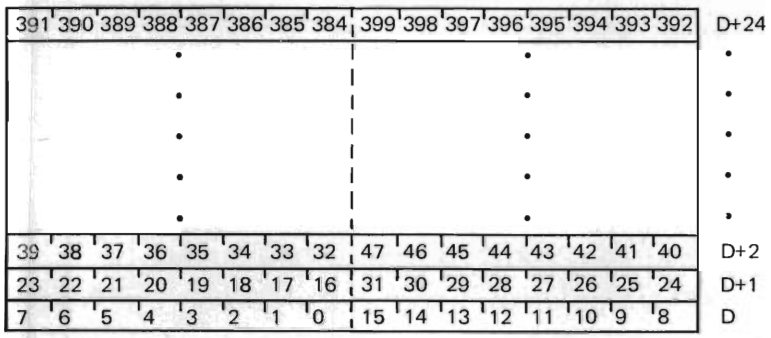
   For each program, where n is the program number:

```
        LDV     A,n
        DSPL    A
        .
        .
```

   While the program is running, the value n will be dislayed on the console.

3. This example shows how the XEC instruction may be used to specify a parameter of an instruction at execution time. The instruction which is to be executed is the Test Bit (TBIT) instruction.

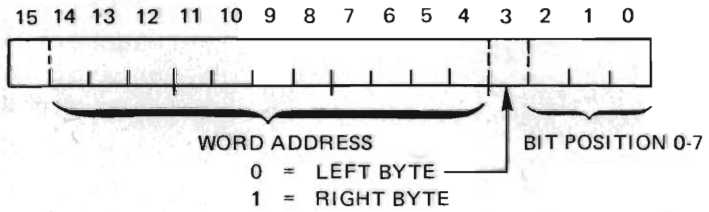
An array of 400 bits is stored in memory in the first 25 words beginning at D:



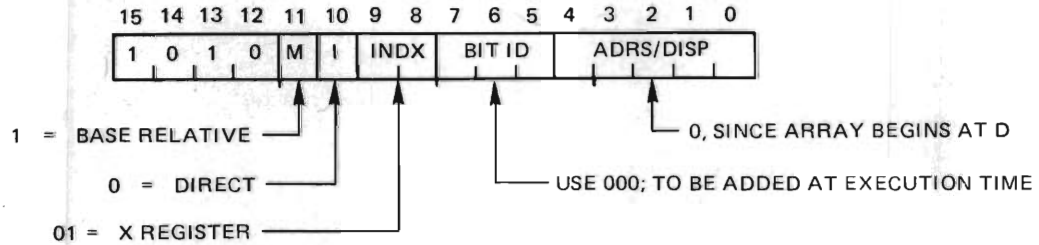
The result of a calculation in the main program gives a bit element number (from 0 to 399) in the X register, which must be converted to reference the bit element as bit 0-7 of the left or right byte of a particular word.

For each bit element number, the binary representation can be defined as follows:

X register = bit number (0 to 399)



The TBIT instruction is contained in the subroutine 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 the X register and put them in bits 5-7 of the TBIT instruction. Then it shifts off the three bits, leaving the correct byte address in the X register.

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.

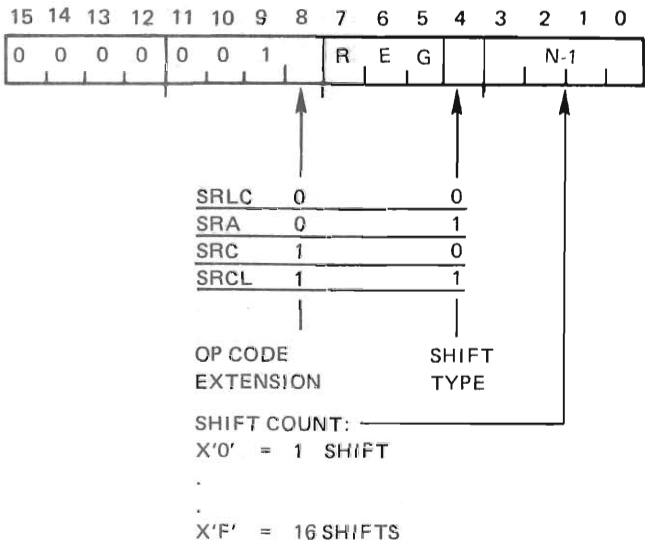
Subroutine:

```
TSTB    LDV   A,X'0007'    Set up mask for bit position.
        AND   A,X          Put bit position (only) in A register.
        SRC   A,11         Shift bit position to bits 5-7 for
                           TBIT instruction.
        ORV   A,X'A900'    Fill in rest of TBIT instruction.
        SRA   X,3          Make byte index in X register.
        XEC   A            Execute TBIT instruction.
        RTRN  E            Result is in Zero indicator.
```

3.7 SHIFT INSTRUCTIONS

Shift instructions shift the bits of a selected register a specified number of positions. 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.

There are four instructions in the shift group; each has the following format:



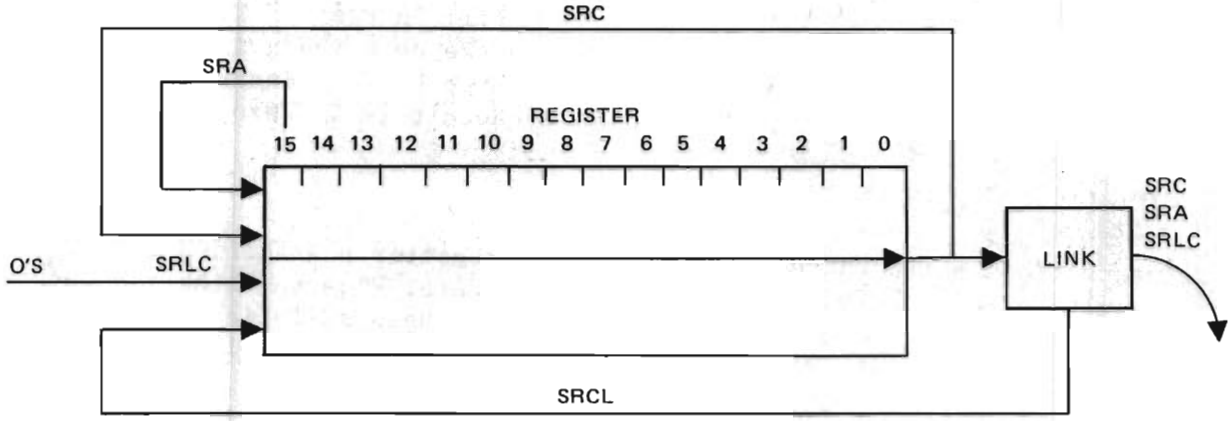
The CAP-16 source statement format for this instruction group is as follows:

Command	Parameters
SRA SRC SRLC SRCL	R,n

where: R is a general purpose register, coded into bits 5-7.

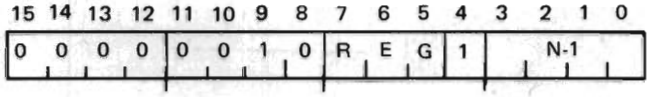
n is the actual number of shifts to be performed. n-1 is coded into bits 0-3 of the machine instruction.

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. Each shift instruction takes one cycle plus an additional half cycle for each shift after the first shift. Since up to 16 shifts may be performed, the maximum number of cycles is 8-1/2.

3.7.1 Shift Right Arithmetic (SRA) 1+0.5n cycles



The selected register 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.

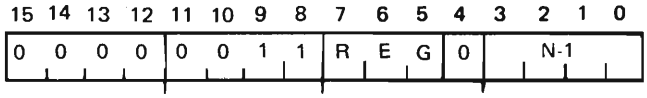
Indicators affected: Zero, Plus, Link
Registers affected: Selected Register

Examples:

```
SRA  A,10     If the A register = 0000111100001111
              Upon completing 10 shifts, the A register
              contains 0000000000000011 and the Link
              contains the original bit 9 (=1).

SRA  B,15     If the B register = 1000000000000000
              Upon completing 15 shifts, the A register
              contains 1111111111111111 and the Link
              contains the original bit 14 (=0).
```

3.7.2 Shift Right Circular (SRC) 1+0.5n cycles



The selected register 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.

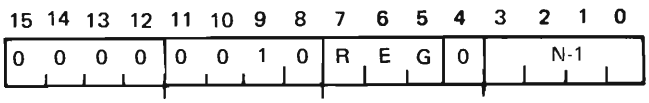
Indicators affected: Zero, Plus, Link
Registers affected: Selected Register

Examples:

SRC A,1 — If the A register = 1111000011110000 Upon completing 1 shift, the A register contains: 0111100001111000 and the Link contains the original bit 0 (=0).

SRC A,5 — If the A register = 1111000011110000 Upon completing 5 shifts, the A register contains: 1000011110000111 and the Link contains the original bit 4 (=1).

3.7.3 Shift Right Logical and Count (SRLC) 1+0.5n cycles



The selected register is shifted right N positions as specified by the shift count or until the Link contains a 1, 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 '1' bit that terminates the shifting; this number is also the number of shifts performed minus one. The value of the Shift Counter may be read by the ADDS instruction (Section 3.6.9) or by the TSR instruction (Section 3.6.10).

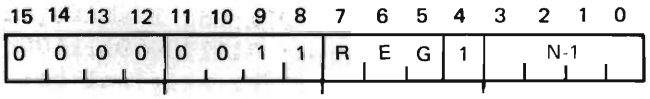
Indicators affected: Zero, Plus, Link
Registers affected: Selected Register

Examples:

```
SRLC  B,10     If register B = 1101000000000000
               Upon completing 10 shifts, the B
               register contains:0000000000110100
               and the Link contains the original
               bit 9 (=0). The Shift Counter will
               contain 9.

SRLC  C,6      If  register C = 1000000000011100
               Three shifts will be made. Upon
               completion, the C register contains:
               0001000000000011 and the Link contains
               the original bit 2 (=1). The Shift
               Counter will contain 2.
```

3.7.4 Shift Right Circular Through Link (SRCL) 1+0.5n cycles



The selected register 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 contains: 0000000011110000
               and the Link is set (=1), upon completing
               5 shifts, the A register will contain:
               0000100000000111 and the Link contains
               the original bit 4 (=1).
```

3.7.5 Examples

1. To count the number of 1's in the A register, leaving count in X.

```
        ZERO   X       Zero count to begin
LOOP    SRLC   A,16    Shift until Link contains a 1.
        SKR    DONE    Skip if no 1 was shifted into Link.
        INCR   X       Count the 1.
        JMP    LOOP    Go to shift again.
DONE    .
        .
```

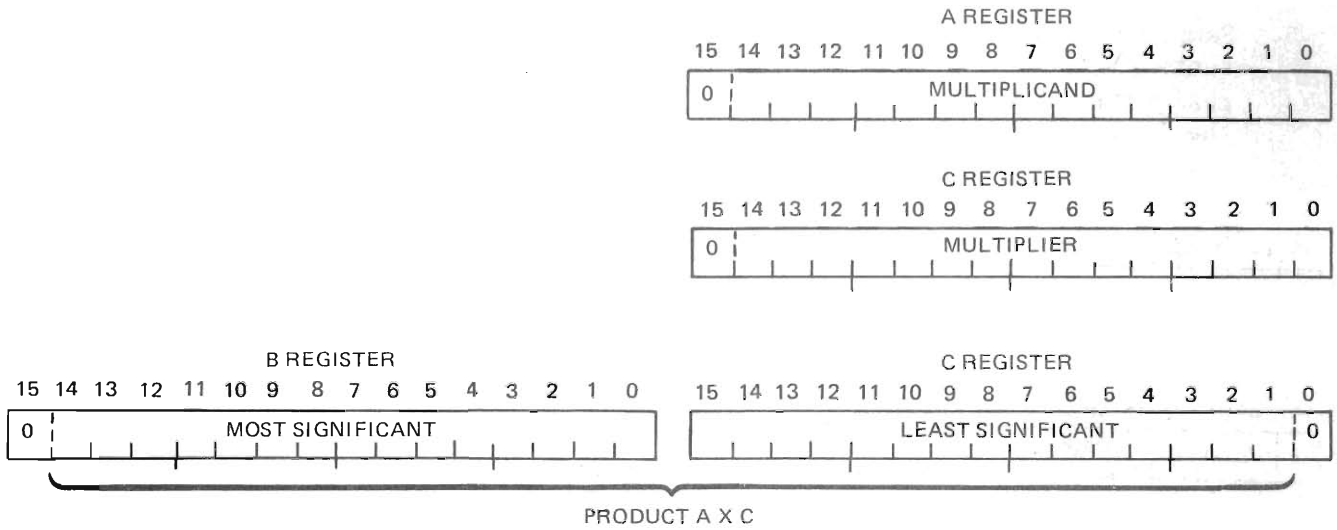
2. The Right Shift Circular instruction may be executed to perform an effective left shift by specifying a count which is the number of left shifts subtracted from 16. For example:

```
SRC    A,10    A circular right shift 10 positions is
               equivalent to a circular left shift of
               16 - 10 = 6 positions.
```

3. The ADD registers instruction may also be used to shift left by specifying the same register for both operands:

```
ADD    R,R     The contents of the specified register,
               R, are doubled, effectively performing a
               left shift one position. The carry out of
               bit 15 goes into the Link, and a zero is
               placed into bit 0.
```

4. Sample multiply routine for positive integers. Subroutine MPY multiplies the number in the A register by the number in the C register and places the result as a double length product in the B and C registers.



In binary multiplication, the partial product of the multiplicand and multiplier is:

> Zero if the multiplier bit is zero, or the shifted multiplicand if the multiplier bit is one.

For example, in multiplying 9 x 5:

```
multiplicand:  9(10) =     1001
multiplier:    5(10) =      101
                           1001    multiplier bit is one
                          0000     multiplier bit is zero
                         1001      multiplier bit is one
Total = 45:              101101
```

The same procedure is followed in the multiply routine, except that:

1. The product of each multiplier bit is added to a running subtotal rather than summing the partial products afterwards.

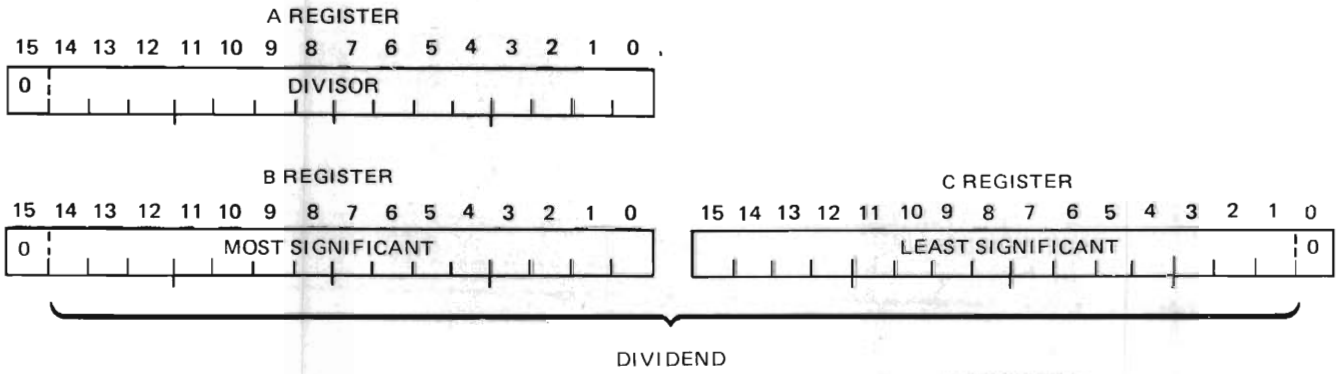
2. If the multiplier bit is zero, the add operation is simply skipped.

3. The subtotal is shifted right instead of shifting the partial products left.

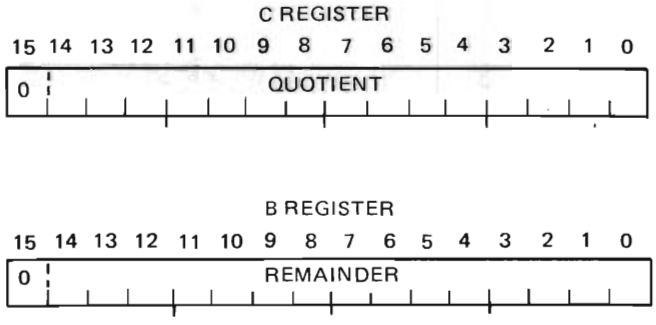
Subroutine:

```
MPY     LDV   X,15    Initialize multiplier bit count.
        ZERO  B       Zero subtotal to begin.
LOOP    SRLC  B,1     Shift B, C right 1, putting low order
        SRCL  C,1     multiplier bit into the Link.
        SKR   NEXT    If bit=0, no add.
        ADD   B,A     If bit=1, add multiplicand to subtotal.
NEXT    DECR  X       Decrement multiplier bit count.
        SKP   LOOP    Loop is more to go.
        RTRN  E       Product is in B,C registers.
```

5. Sample divide routine for positive integers. Subroutine DIV divides the contents of the A register into a double length dividend contained in the B,C registers. The quotient is placed into the C register and the remainder is in the B register.



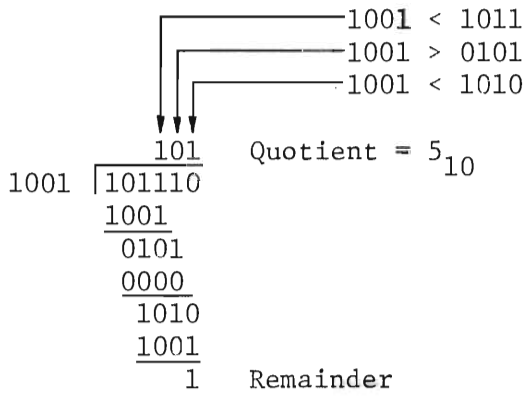
Result of B,C ÷ A:



In binary division, a quotient bit is:

1 if the divisor is less than or equal to the partial dividend, or
0 if the divisor is greater than the partial dividend.

For example, in dividing $46_{10} \div 9_{10}$



The divide routine shifts B, C left one, bringing a zero into bit 0 of the quotient. Then a subtract compare of the partial dividend (B register) from the divisor (A register) is performed: If B > A, the zero bit of the quotient is not changed.

Subroutine:

```
DIV             LDV     X,14    Initialize quotient bit count.
LOOP            ADD     B,B  ]
                ADD     C,C  } Shift B, C left one, ignoring carry out
                RLK     B    ] of B (Note: Initially a zero bit).
                SUBC    B,A     Subtract compare divisor from high-order
                                dividend bits.
                SKM     ZERO    If result negative, leave 0 bit in quotient.
                SUB     B,A     Otherwise, subtract.
                INCR    C       And put 1 bit in quotient.
ZERO            DECR    X       Decrement quotient bit count.
                SKP     LOOP    Loop if more to go.
                RTRN    E       Quotient is in C, remainder is in B.
```

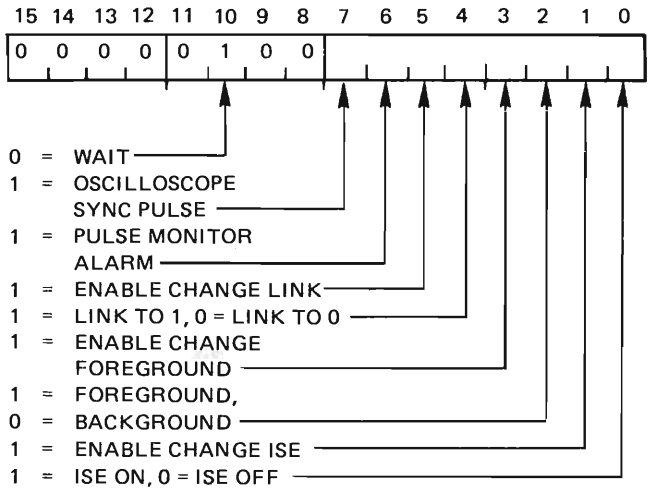
3.8 CONTROL INSTRUCTIONS

The Control group includes instructions to perform the following functions:

- Enable/inhibit system interrupt
- 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 "freeze" in a wait state

A Control instruction takes one processor cycle. All Control instructions execute twice as fast when fetched from a Read-Only memory as from a Read/Write memory.

There are nine instructions in the Control group; they have the following format:

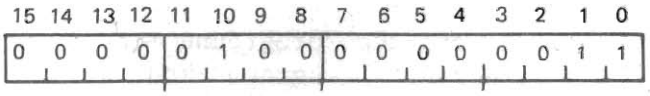


The source statement format for this group is as follows:

Command	Parameters
INE	
INH	
FMS	
BMS	
PMA	
LKS	
LKR	
SYNC	
WAIT	

No mnemonic parameters are specified for Control instructions. As shown, all 16 bits are used as an operation code.

3.8.1 Interrupt Enable (INE) — 1 cycle



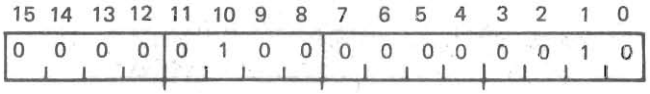
Following the execution of this instruction, interrupt requests may interrupt the normal program sequence and demand service by an interrupt subroutine.

1→ ISE

Indicators affected: ISE
Registers affected: None

The interrupt system will allow an interrupt immediately following execution of the Interrupt Enable instruction in the case of internal interrupts only. External interrupts must wait at least two cycles. (See Section 2.4).

3.8.2 Interrupt Inhibit (INH) — 1 cycle



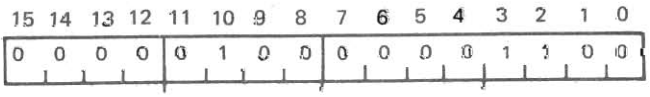
Following the execution of this command, normal program sequencing will not be interrupted by interrupt requests. Note: The power-fail and automatic restart interrupts cannot be inhibited.

0→ ISE

Indicators affected: ISE
Registers affected: None

Interrupts are inhibited immediately following execution of the INH instruction.

3.8.3 Foreground Mode Set (FMS) — 1 cycle



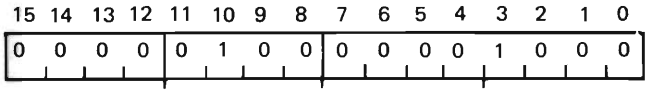
Following the execution of this instruction, the eight Foreground registers (A, X, Y, Z, B, C, D, E) are used by all instructions that specify registers. The Background registers (A', X', Y', Z', B', C', D', E') are not addressable while the computer is operating in the Foreground mode.

1→ Foreground

Indicators affected: Foreground
Registers affected: Set of 8 foreground mode registers become accessible; set of 8 background mode registers are not accessible.

The computer may also be placed into the Foreground mode by pressing the console SYSTEMS RESET switch if the foreground/background mode option is installed. If the foreground/background mode option is not installed, the execution of this instruction has no effect (i.e., the Background registers are always the active set).

3.8.4 Background Mode Set (BMS) 1 cycle

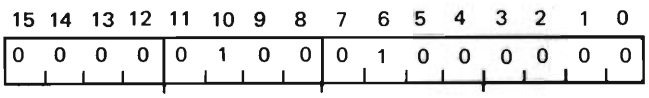


Following the execution of this instruction, the eight Background registers (A', X', Y', Z', B', C', D', E') are used by all instructions that specify registers. The Foreground registers (A, X, Y, Z, B, C, D, E) are not addressable while the computer is operating in the Background mode.

0→ Foreground

Indicators affected: Foreground
Registers affected: Set of 8 background mode registers become accessible; set of 8 foreground mode registers are not accessible.

3.8.5 Pulse Operations Monitor Alarm (PMA) 1 cycle

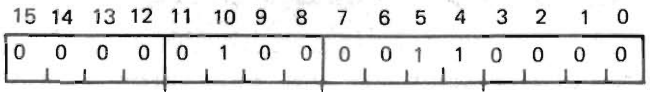


This instruction resets the Operations Monitor Alarm (OMA) timer each time the instruction is executed. Failure to execute this instruction within 200,000 cycles of the previous execution causes the computer to automatically switch from Run mode to Idle mode and the safe signal to be removed from the safe line (SFEC-I).

The OMA timer will not initially start until the PMA instruction is executed for the first time. Thereafter, it must be continually executed to prevent the OMA from becoming activated.

Indicators affected: None
Registers affected: None

3.8.6 Link Set (LKS) — 1 cycle

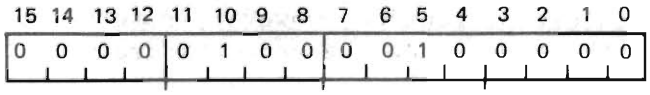


Execution of this instruction places a 1 into the Link.

1→ Link

Indicators affected: Link
Registers affected: None

3.8.7 Link Reset (LKR) — 1 cycle

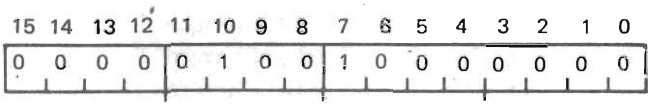


Execution of this instruction places a 0 into the Link.

0→ Link

Indicators affected: Link
Registers affected: None

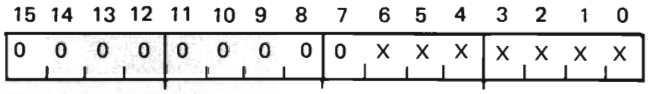
3.8.8 Generate Sync Pulse (SYNC) — 1 cycle



Execution of this instruction places a 1/4 cycle pulse on the SYNC-I 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.

Indicators affected: None
Registers affected: None

3.8.9 Wait (WAIT) — 1 cycle



Execution of this instruction causes program execution to freeze; the console IDLE and RUN indicators are illuminated. The WAIT instruction operates by preventing the program counter (P register) 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 0-6 of the WAIT instruction. A particular WAIT branch can be identified by examining the contents of the I Register (which will contain the operand value) upon execution of the WAIT instruction.

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, however, returns control to the location containing the WAIT instruction since the return address is taken as the contents of the P register.

The program can be advanced past the WAIT condition by manually pressing the STEP switch on the console. (If an interrupt routine increments the return address, i.e., the contents of the E register, before returning to the interrupted program, the program is advanced past the WAIT instruction.)

3.8.10 Examples

1. The five controls:
   bit 7 = 1 Generate SYNC pulse
   bit 6 = 1 Reset OMA
   bit 5 = 1 Enable Change Link
   bit 3 = 1 Enable Change Foreground
   bit 1 = 1 Enable Change ISE

   are operated upon independently. More than one control could be specified by selectively setting bits 0-7 in one Control instruction. For example, the following source statement will generate a machine code of X'04FF':

```
DC   X'04FF'
```

   If this word is executed as an instruction, it will:

   Generate SYNC pulse
   Reset OMA
   1→ Link
   1→ Foreground
   1→ ISE

2. The primary function of SYNC is to allow an external device to be synchronized with the execution of segments of programs or of special hardware functions.

   For example, if a number of programs are interactively running, each program could be assigned a number and when it was running this number would be displayed on the console display register (see Section 3.6.17, Example 2). The 4 low order bits of the Console Display Register are also input to a 4-bit Digital-to-Analog Converter, signal VU+I, which varies from 0 to +7.5V, and VU could be displayed on an oscilloscope. SYNC would be located at some recurring point in the program and would be used to trigger the oscilloscope display. Thus, the value of VU would represent the running program at any point in time, and the time could be determined from its relativity to SYNC.

   SYNC is also used as a synchronizing-only pulse (i.e., it does no other work) for CPU and I/O debugging/testing.

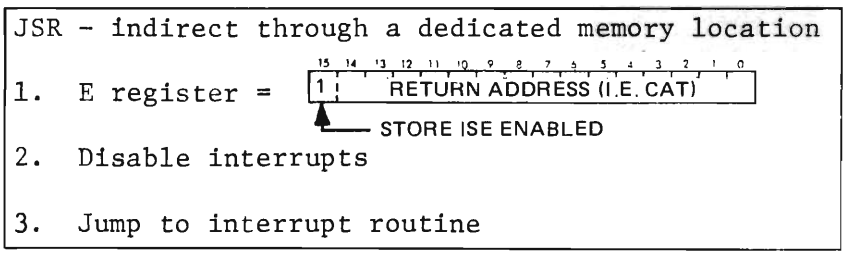
3. This example shows how a non-interruptable interrupt subroutine could be constructed, requiring a total of 6 processor cycles for interrupt transfer, store and restore operations. (Normally only small interrupt programs would be constructed this way, since the interrupt system is disabled while the interrupt routine is executing.)

   The interrupt routine is executed using the background registers; thus, only the contents of the S (status) register need to be saved by the interrupt routine. This is done by storing the interrupted program's S register status in the interrupt routine's (background) E register. (In an interruptable program, the E register cannot be used as a general purpose register since an interrupt, which changes the E register, may occur at any time. This restriction does not apply to a non-interruptable routine.)

   Note that the RTRN E refers to the foreground E register, while the TSR/TRS instructions reference the background E register.

   Interruptable program (using foreground registers):

```
                INCR  A
                DECR  B
                      ←Interrupt occurs:
CAT             CMPL  C
                .
                .
```



Interrupt routine:

```
BMS           Set background mode.
TSR    E      Save program's indicators status.
.
.            (Perform operations.)
.
TRS    E      Restore program's indicators status.
FMS           Set foreground mode.
RTRN   E      Return to interrupted program (at CAT).
```

4. A group of subroutines may share a temporary storage area in core by indirectly addressing a specified location (e.g., address X'0000') which contains the beginning address of the common storage area. Thus, if a program calls a series of subroutines in sequence and each of the subroutines requires no more than 20 "working" memory locations, only 20 memory locations would need to be set aside, as a dynamic storage area, for all of the subroutines to use.

   Combining the use of a common storage area with the SARS/LARS instructions allows any subroutine to be reentrant. Interrupts can use the same set of registers as the interrupted program by storing all registers and status, performing the interrupt operations, and then restoring the interrupted program's registers and status. (This method also allows recursive programming i.e., a subroutine may call itself or subroutines may call each other.)

   GA standard software modules use the contents of location X'0000' as a dynamic storage area pointer; it always contains the address of the next available location in the dynamic storage area. The code, shown below, is the standard subroutine entry and exit sequences for reentrant, interruptable subroutines. The subroutine stores the registers and status of the calling or interrupted program indirectly through location X'0000' (i.e., in nine sequential locations beginning at the first available location of the dynamic storage area). It then sets its base address (D register) to this same dynamic storage area location (i.e., D register now contains the address of the location where the A register for the calling or interrupted program was just saved). The routine updates the pointer in location X'0000' to point to the "new" next available location of the dynamic storage area by adding 9 (required by SARS) plus the number of temporary storage locations required by the subroutine itself. Then the interrupt enable status is restored, so that interrupts may occur while the subroutine performs its work.

   If the subroutine is interrupted, the interrupting routine (which is assumed to have the same entry sequence) will not affect the interrupted routine's registers or temporary storage locations. While the subroutine is performing its actual function or calculation, its temporary memory locations are accessed using base relative addressing.

   When a subroutine exits, it relinquishes the locations that it reserved during its entry sequence by resetting the dynamic storage area pointer to its previous value, then restores the previous registers and status.

Note that interrupts are inhibited while the subroutine is performing the entry and exit operations described above.

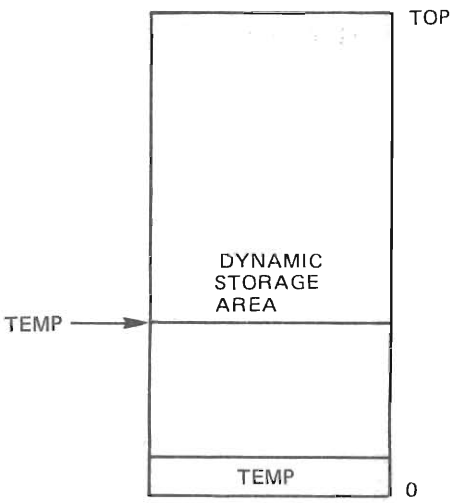
```
SUB             SARS   *0              Store all registers and status.
                LDR    D,0             Set up new base address.
                RTR    X,D             Transfer to X register.
                ADDV   X,length        Add length (=storage needed by sub-
                                       routine)
                STR    X,0             Store updated dynamic stg area pointer.
                RISE   E               Restore interrupt enable status.
                .
                .
                .                      (Perform calculations.)
                .
                .
                INH                    Inhibit interrupts.
                STR    D,0             Restore previous dynamic stg area pointer.
                LARS   *0              Restore all registers and status.
                RTRN   E               Return to calling or interrupted pro-
                                       gram.
```

For example, an interruptable program calls two subroutines, SUB1 and SUB2, and SUB2 may also call SUB1. Assume that the following sequence occurs:

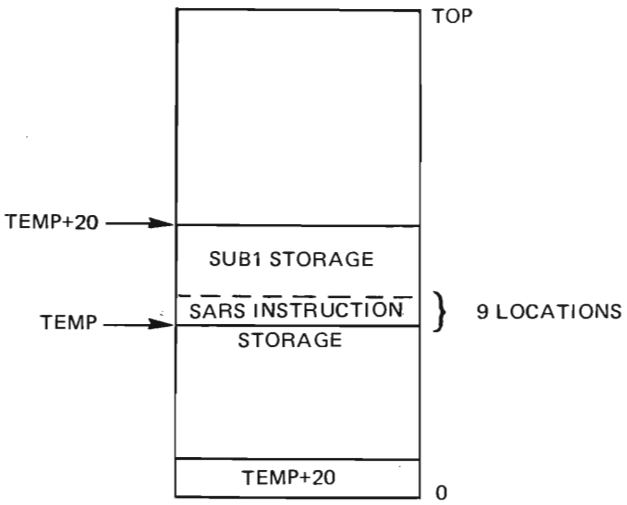
Calling Sequence:

```
1.  Program:  JSR  SUB1          (SUB1 uses 20 temporary storage locations)
2.  Program:  JSR  SUB2          (SUB2 uses 25 temporary storage locations)
3.  SUB2:     JSR  SUB1 <----    Interrupt occurs. (Interrupt uses 20 temporary
                                 storage locations)
```

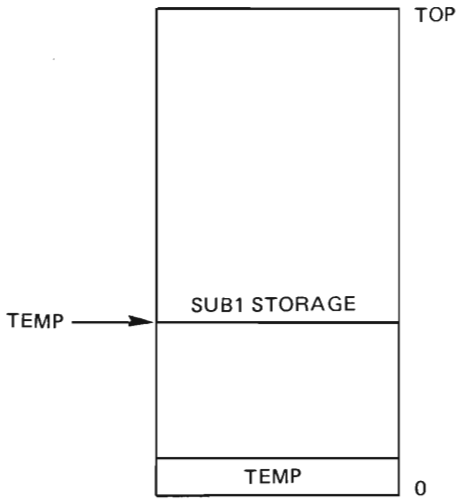
Initially location X'0000' contains the address of the first available space in the dynamic storage area:



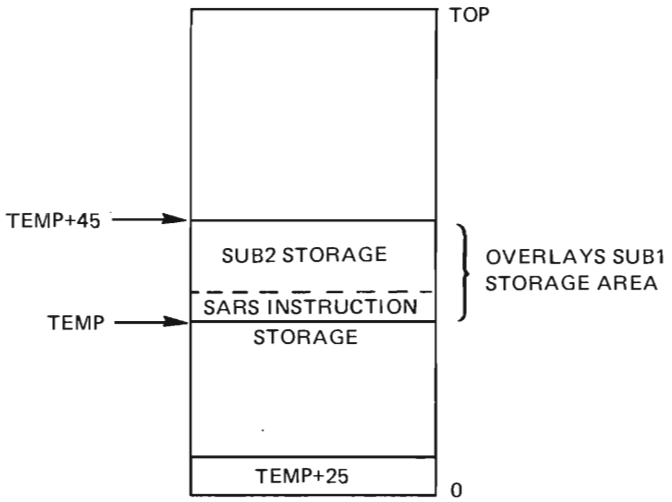
1. SUB1 would take TEMP as its base address, add 20 to TEMP, and store this address back at location 0. The first 9 of the 20 locations are used by the SARS instruction, which includes storing the contents of the E register (from the main program), which contains SUB1's return address.



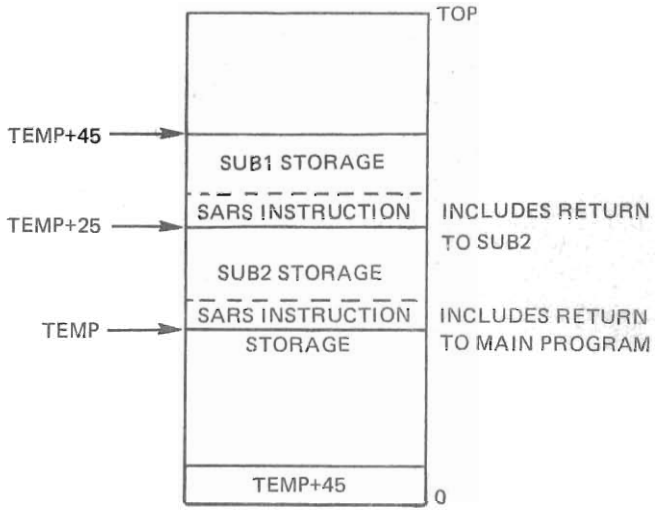
When SUB1 has finished its calculations, it restores the original address TEMP at location 0:



2. SUB2 would also use TEMP as its base address, add 25 to it, and store this address back at location 0:

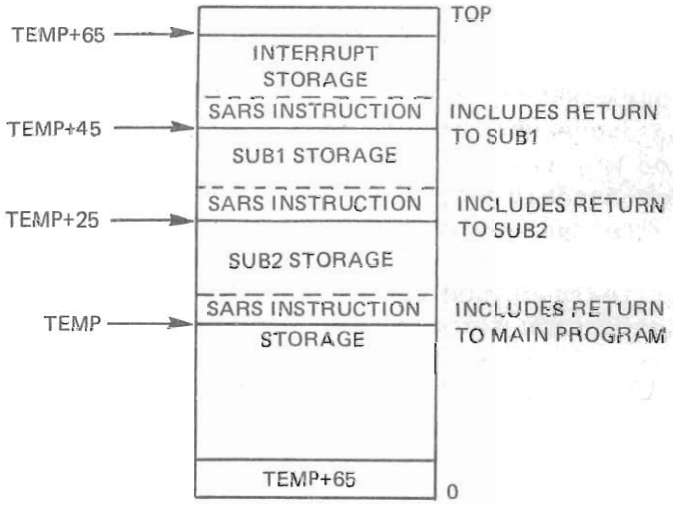


3. During its execution, SUB2 calls SUB1. SUB1 would take TEMP+25 as its base address, add 20 to it, and store this address back at location 0. The first 9 locations of SUB1 again are used by the SARS instruction, which includes storing the contents of the E register (from SUB2), which contains SUB1's return address.

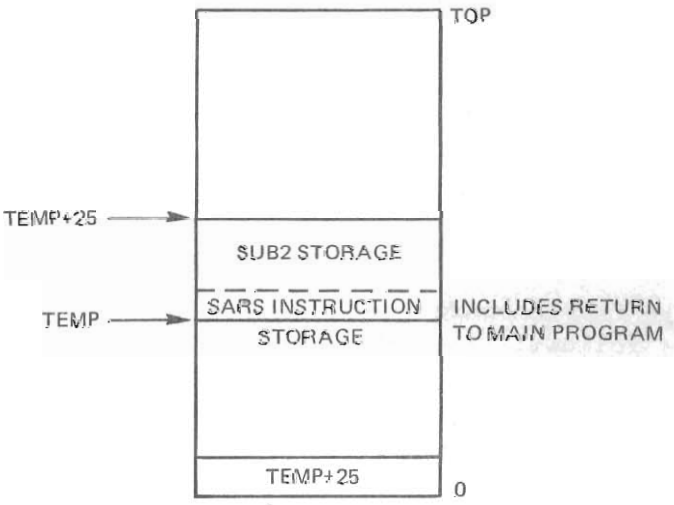


While SUB1 is executing, an interrupt occurs. The interrupt will take TEMP+45 as its base address, add 20 to it, and store this address back at location 0.

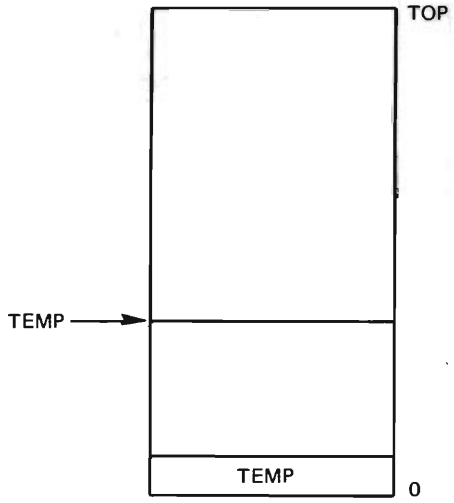
When the interrupt is finished, it relinquishes its storage locations by restoring the base address to TEMP+45, and returns to SUB1.



When SUB1 is finished, it relinquishes its storage locations and returns to SUB2:



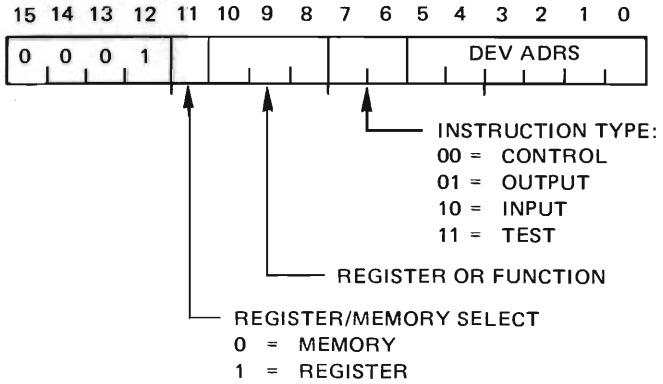
When SUB2 is finished, it restores the original address TEMP at location 0 and returns to the main program:



3.9 PROGRAMMED INPUT/OUTPUT INSTRUCTIONS

Programmed input/output instructions cause information to be transmitted between the processor and external devices, extended memory, or the console. The information may be a word of data transmitted to/from a register or memory location or from console switches, a control pulse output to an external device, or an external device test which provides a conditional skip of the next instruction.

All instructions in the programmed input/output group have the following machine instruction format:



The CAP-16 source statement format for the Programmed Input/Output instructions is as follows:

Command	Parameters
DTOR, DTIR, DTOM, DTIM, CTRL, TEST	∂,dev
RCSR, RCSM	∂ (dev = $3E_{16}$ implied)

where: ∂ specifies one of the general purpose registers, coded into bits 8-10 for data in/out instructions (DTOR, DTIR, DTOM, DTIM, RCSR, RCSM).

or ∂ specifies one of eight control functions for CTRL instructions (bits 6, 7 = 00) or one of eight tests for TEST instructions (bits 6, 7 = 11).

dev specifies one of a possible 64 devices, coded into bits 0-5.

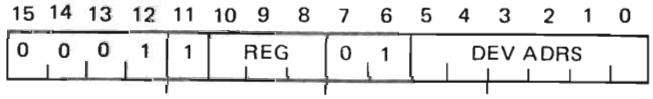
For a data word to/from a register, the register is specified by bits 8-10. For a data word to/from a memory location, the address of the memory location is taken as the contents of the specified register.

The instructions presented in this section may be used to control any of up to 64 devices interfaced to the SPC-16 (a peripheral may constitute one or more devices). Each device has a unique select code. The individual device is responsible for recognizing its own select code and interpreting and performing the required functions when it is selected. For example, Test 2 required of device X is a hardware function of X and may not be the same operation as Test 2 required of device Y.

When the processor executes an I/O instruction, it outputs the device select code specified in the instruction, together with control signals and 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 processor via the I/O bus.

The following subsections describe each of the Programming Input/Output instructions with general examples. Additional examples specific to particular devices are included in Section 5, which discusses individual peripheral controllers.

3.9.1 Data Out From Register (DTOR) 2 cycles



The contents of the specified register are output to the addressed device.

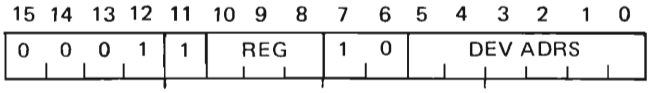
$R_{0-15} \longrightarrow$ I/O BUS DATA OUTPUT LINE

Indicators affected: None
Registers affected: None

Example:

```
DTOR  A,X'10'  The contents of the A register are
               output to the device whose device
               select code is 10(16).
```

3.9.2 Data In To Register (DTIR) — 2 cycles



A word of data from the addressed device replaces the contents of the specified register. If the selected device inputs a word of less than 16 bits, the other data bits will contain zero.

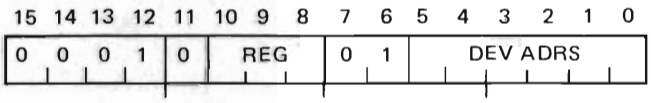
I/O BUS DATA INPUT LINE $\rightarrow R_{0-15}$

Indicators affected: None
Registers affected: Specified Register

Example:

```
DTIR  B,X'21'  Data from the device whose select
               code is 21(16) are input to the B register.
```

3.9.3 Data Out From Memory (DTOM) — 2 cycles



The specified register contains the address of the memory location from which data will be output to the addressed device.

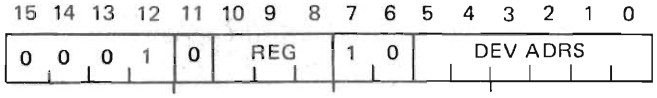
$(R)_{0-15} \longrightarrow$ I/O BUS DATA OUTPUT LINE

Indicators affected: None
Registers affected: None

Example:

```
DTOM  B,X'12'  The B register contains the address
               of the memory location whose contents
               are output to the device whose
               device select code is 12(16).
```

3.9.4 Data In To Memory (DTIM) — 2 cycles



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.

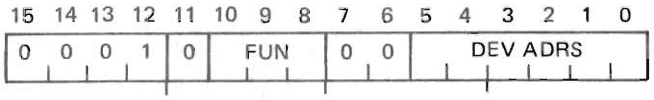
I/O BUS DATA INPUT LINE⟶ $(R)_{0-15}$

Indicators affected: None
Registers affected: None

Example:

```
DTIM  C,X'17'  The C register contains the address
               of the memory location whose contents
               will be replaced by data read from
               the device whose device select code
               is 17(16).
```

3.9.5 Output Control Function (CTRL) — 2 cycles



The specified control function is output as a control pulse to the addressed device.

$PULSE_i$⟶ I/O BUS OUTPUT CONTROL LINE, i=f(bits 8-10)

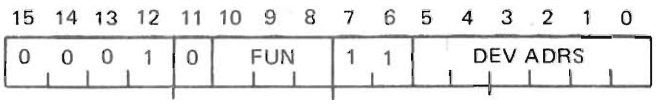
Indicators affected: None
Registers affected: None

Example:

If control function 0 enables interrupts and device X'08' is the Paper Tape Reader:

```
CTRL  0,X'08'  The execution of this instruction will
               enable interrupts by the Paper Tape Reader.
```

3.9.6 Test Device (TEST) — 2 cycles



One of eight tests (specified by the FUN bits) is performed on the selected device. The result must be returned: If the test condition is met, the next instruction will be skipped. If the test condition was not met, no skip is made.

$TEST_i$ I/O BUS OUTPUT CONTROL LINE, i=f (bits 8-10)

RESULT I/O BUS COMMON TEST LINE (INPUT)

If the test condition is met:

P+1 → P

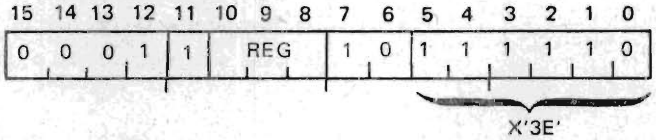
If the test condition is not met, continue with next instruction.

Indicators affected: None
Register affected: None

Example:

TEST 6,X'33' The specified device performs its Test 6; if the result returned is 'true', the next instruction is skipped.

3.9.7 Read Console Switches Into Register (RCSR) 2 cycles



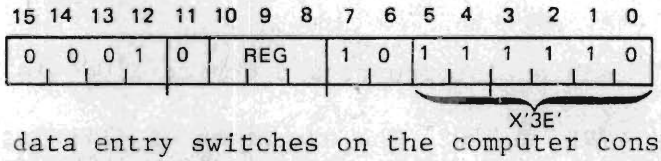
The contents of the data entry switches on the computer console are loaded into the specified register. The device address is always X'3E'.

Switch down = 1
Switch up = 0

(CNSL SW) → R
0-15 0-15

Indicators affected: None
Registers affected: Specified Register

3.9.8 Read Console Switches Into Memory (RCSM) 2 cycles



The contents of the data entry switches on the computer console are loaded into the memory address indicated in the specified register. The device address is always X'3E'.

Switch down = 1
Switch up = 0
(CNSL SW) → (R)
0-15 0-15

Indicators affected: None
Registers affected: None

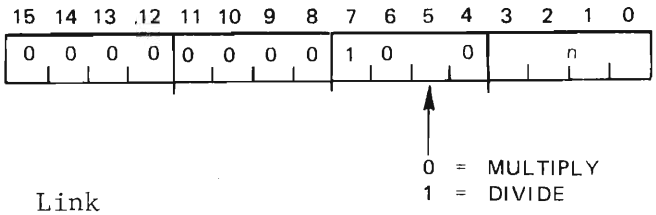
NOTE

The two instructions above are really special case DTIR/DTIM's. They are to be used in lieu of RCSW to insure that the switch data is properly read. See RCSW, Section 3.6.12.

3.10 HARDWARE MULTIPLY AND DIVIDE (OPTIONAL)

With the Hardware Multiply and Divide option (Section 7.4) installed, multiplication and division are performed directly by hardware circuitry rather than through software subroutines such as described in the examples of Section 3.7. These instructions operate on fixed point, positive numbers only.

Hardware Multiply and Divide instructions are of the following format:



Indicators affected: Link
Registers affected: B,C

The CAP-16 source statement format for the Hardware Multiply and Divide instructions is as follows:

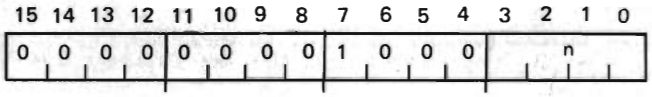
Command	Parameters
MPY DIV	n

where: n is a positive integer, contained in bits 0-3, n determines the number of shift/add operations in multiplication or the number of shift/subtract operations in division.

Note: If n is omitted, n=15 is assumed.

The multiply and divide examples given in Section 3.7 were coded such that a fixed number of operations were performed. The Hardware Multiply and Divide instructions (and the GA Multiply-Divide Simulator program SIM$) allow the number of operations to be variable. Thus, the processor cycles required for the multiply or divide can be minimized by specifying an n which processes only the significant digits.

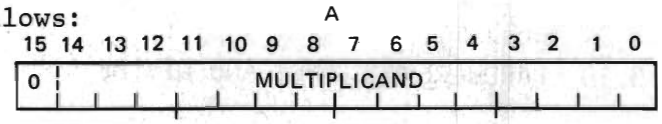
3.10.1 Hardware Multiply (MPY) 1.5 + 0.5n cycles



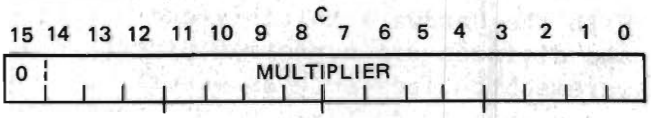
This instruction multiplies the contents of the A register by the contents of the C register and places the result as a double length product in the B and C registers.

The operand registers are summarized as follows:

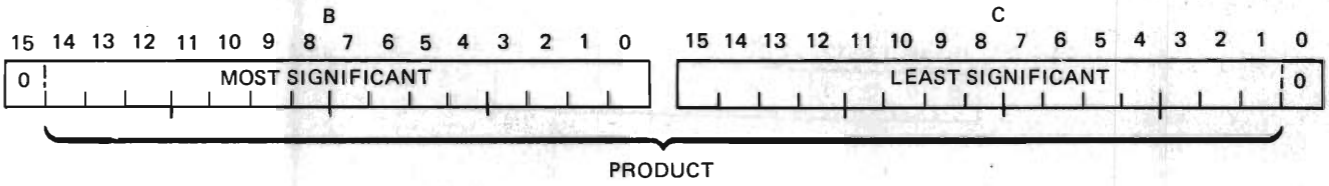
(A) = Multiplicand (15 bits)



(C) = Multiplier (15 bits)



(B,C) = Product (A) x (C), (0-30 bits)



All 15 bits of the multiplicand (A register) are used. Bit 15 of the A and C registers must be zero.

The answer will be contained in the B and C registers, with the digit in bit 15 of C considered as immediately following the digit in bit 0 of B. Bits 0 of C and 15 of B will always be zero.

During the multiply operation, the multiplier bits are shifted (right) out of bit 0 of the C register and product bits are shifted (right) into bit 15 of the C register from the B register. n must be large enough so that all '1' bits of the multiplier are shifted out of the C register.

After a multiply, the content of the Link is indeterminant.

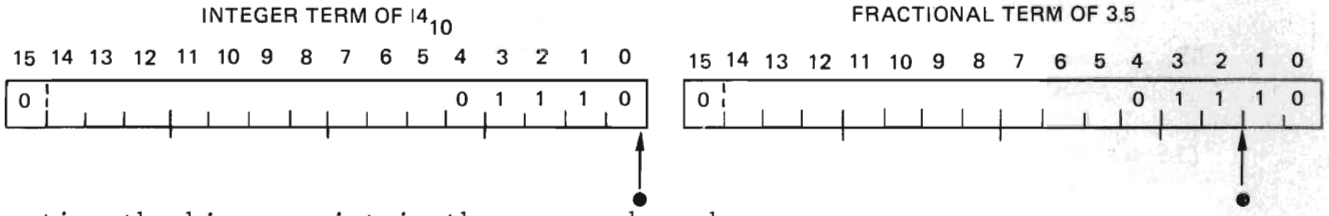
The number of processor cycles required for the multiply operations is given by:

1.5 + 0.5 n cycles

For example, with n=15, 9 cycles are required. With n=8, 5.5 cycles are required.

The 15-bit multiplicand and multiplier terms can 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.

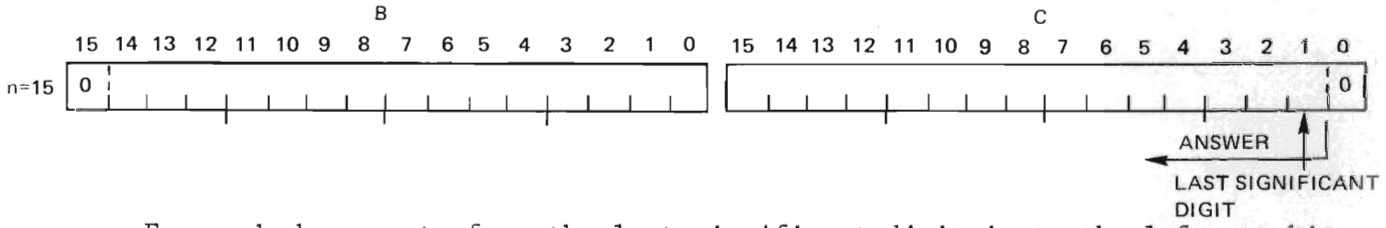
For an integer term, the binary point is to the right of bit 0. A term can also be considered as having a fractional part, with the binary point between the integer and fractional portions. For example:



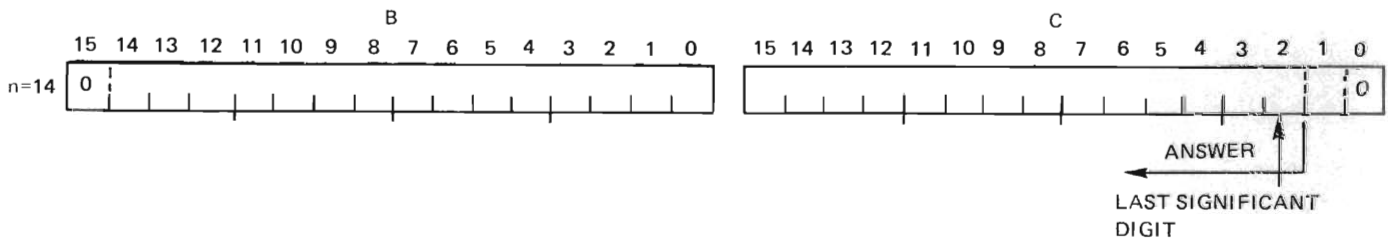
Locating the binary point in the answer depends upon:

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 x .03 = .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:



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 zeros, 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.

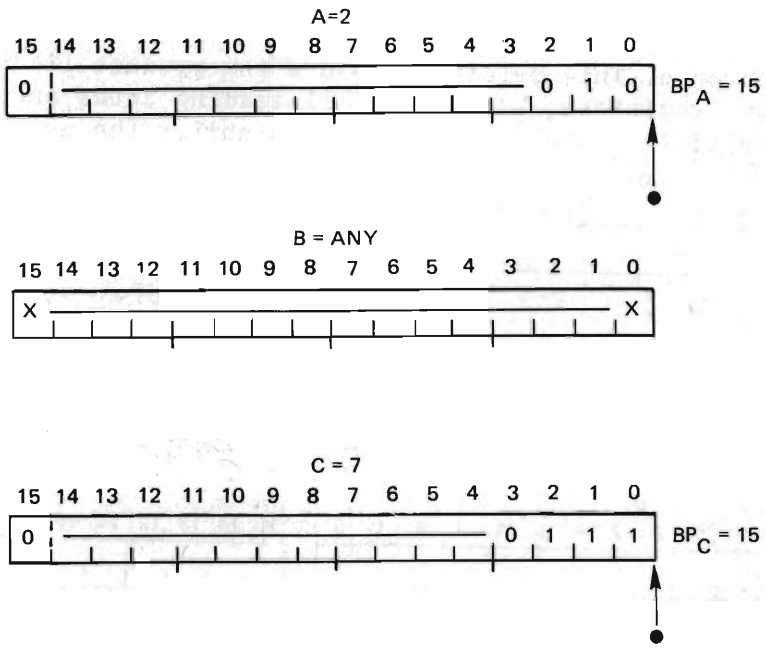
BP_C is the number of integer digits of the multiplier.

$BP_{B,C}$ 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 1 of the C register.)

Examples:

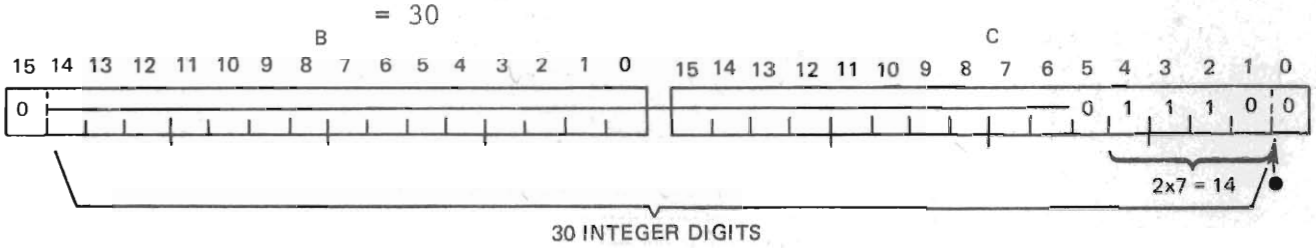
1. Given the following bit configuration for two integer terms prior to multiplication:



a. MPY 15 Multiply the contents of the A register by the low order 15 bits of the C register, with the answer in the B and C registers.

Result: $BP_{BC} = BP_A + BP_C - (15-n)$

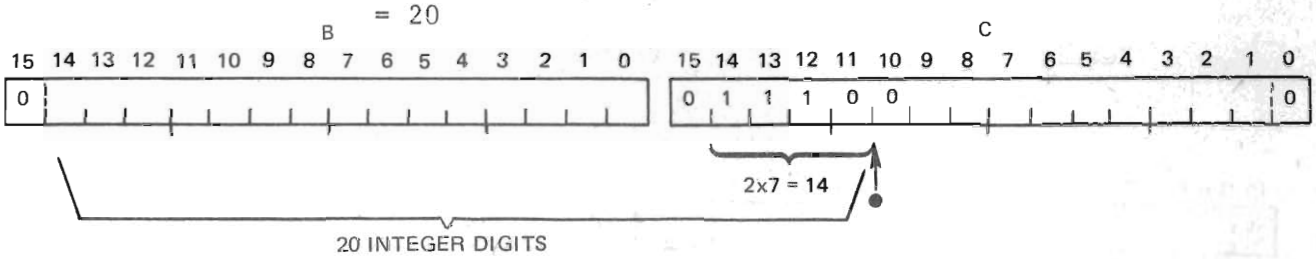
$= 15 + 15-0$
$= 30$



b. MPY 5 Multiply the contents of the A register by the low order 5 bits of the C register, with the answer in the B and C registers.

Result: $BP_{BC} = BP_A + BP_C - (15-n)$

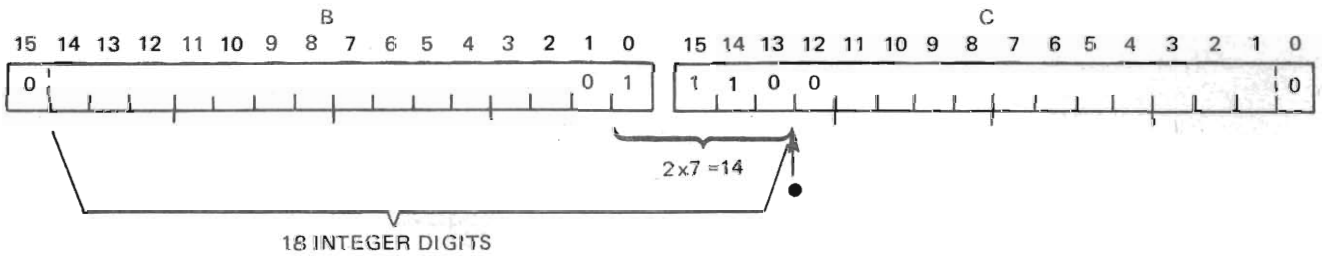
$= 15 + 15 - (15-5)$
$= 20$



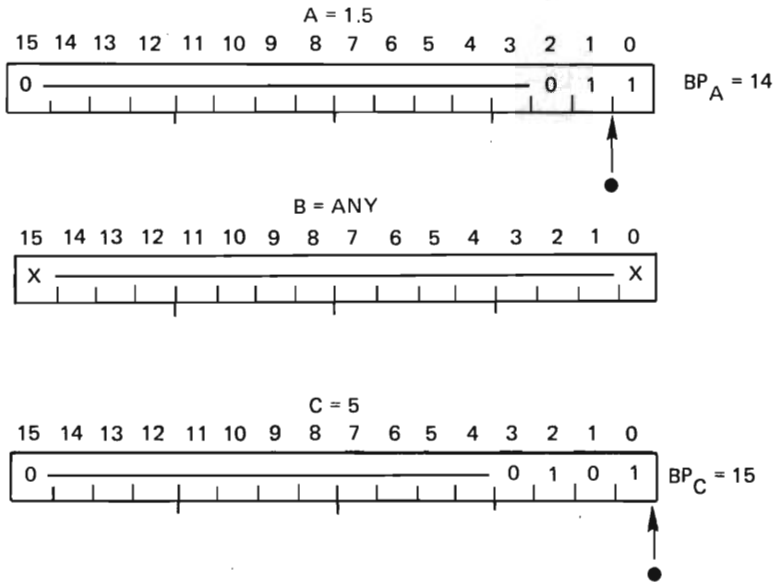
c. MPY 3 Multiply the contents of the A register by the low order 3 bits of the C register, with the answer in the B and C registers.

Result: $BP_{B,C} = BP_A + BP_C - (15-n)$

$= 15 + 15 - (15-3)$
$= 18$



2. Given the following bit configuration for one integer term and one fractional term:

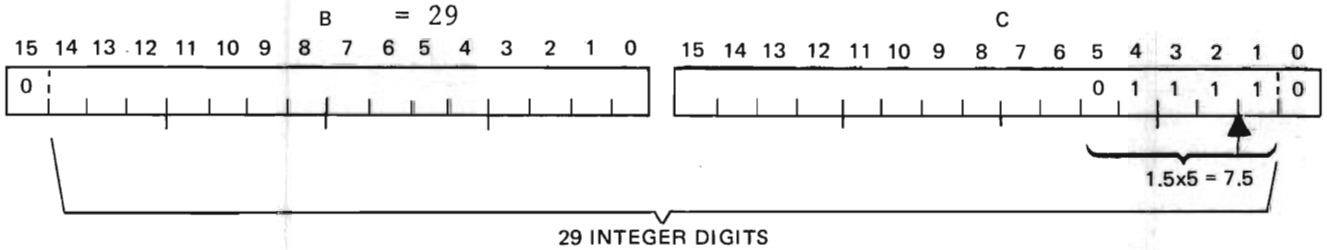


a. MPY 15 Multiply the contents of the A register by the low order 15 bits of the C register, with the answer in the B and C registers.

Result: $BP_{B,C} = BP_A + BP_C - (15-n)$

$= 14 + 15 - (15-15)$

$= 29$

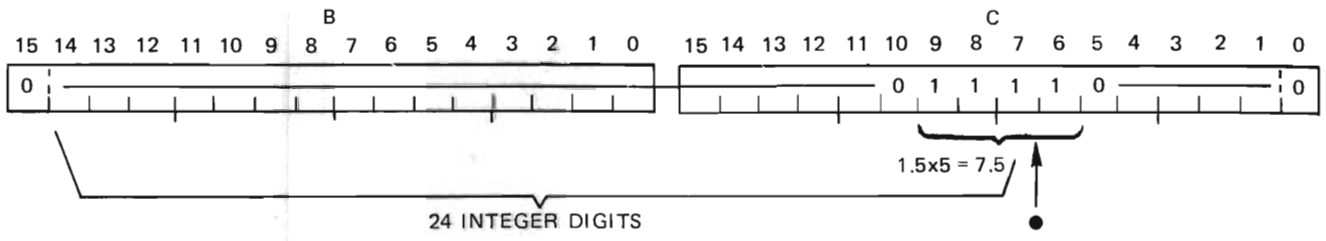


b. MPY 10 Multiply the contents of the A register by the low order 10 bits of the C register, with the answer in the B and C registers.

Result: $BP_{B,C} = BP_A + BP_C - (15-n)$

$= 14 + 15 - (15-10)$

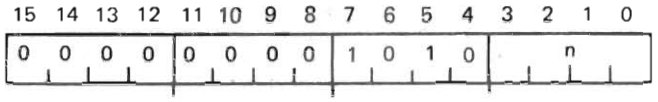
$= 24$



3. Negative (twos complement) numbers can be multiplied using the Hardware Multiply instruction by multiplying their absolute values and 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'   )
            SKZ     CREG        |
            CMPL    A           } Get absolute value of multiplicand.
            INCR    A           |
            INCR    X           )  If negative, change sign of answer.
CREG        ANDVC   C,X'8000'   )
            SKZ     NEXT        |
            CMPL    C           } Get absolute value of multiplier.
            INCR    C           |
            INCR    X           )  If negative, change sign of answer.
NEXT        MPY                    Perform multiplication.
            SRA     X,1            Put sign bit into Link.
            SKR     DONE           If positive, done.
            CMPL    B           )
            CMPL    C           |
            INCR    C           } Otherwise, change B, C result to negative.
            RLK     B           )
DONE        .
            .
```

3.10.2 Hardware Divide (DIV) 1.5 + n cycles

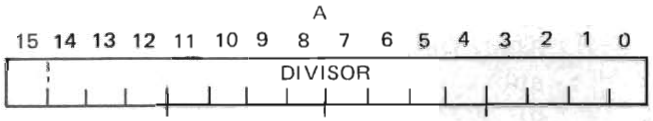


This instruction divides a double length dividend taken as the contents of the B and C registers by the contents of the A register. The resulting quotient is in the C register, and the B register holds the remainder.

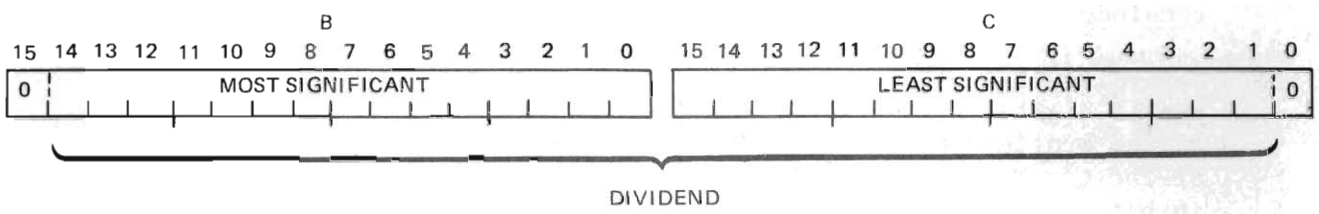
The number in B before division must be less than the number in A.

The contents of the source/destination registers are summarized as follows:

(A) = Divisor (15 bits)

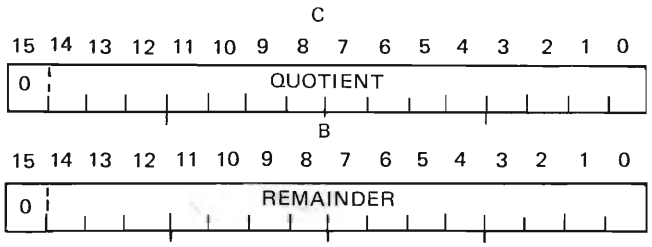


(B,C) = Dividend (30 bits)



(C) = Quotient (B,C)/(A), (0-15 bits)

(B) = Remainder (15 bits)



All 15 bits of the divisor (A register) are used. Bit 15 of the A register must be zero.

During the divide operation, the bits of the dividend are shifted (left) as a unit, e.g., after one shift, bit 15 of C is moved left one place to bit 0 of B. With each shift/subtract, one digit of the quotient is formed (at C_0). The number of quotient digits, n, must be large enough so that n shifts will remove all '1' bits of the dividend from the C register. Bits 15 of B and 0 of C must be zero prior to the divide operation.

Upon completion of the division, the quotient is contained in the C register and the remainder is in the B register. 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 will be set to the status of bit (15-n) of the C register at the end of the division.

The number of processor cycles required for the divide operation is given by:

1.5 + n cycles

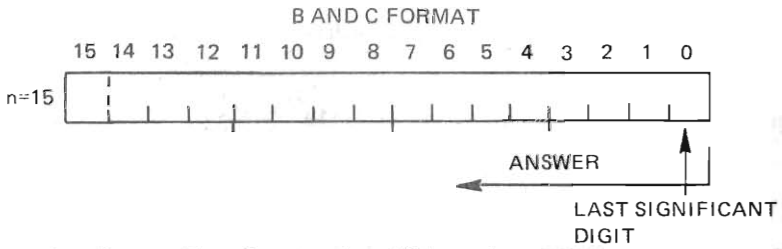
For example, with n=15, 16.5 cycles are required. With n=8, 9.5 cycles are required.

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.

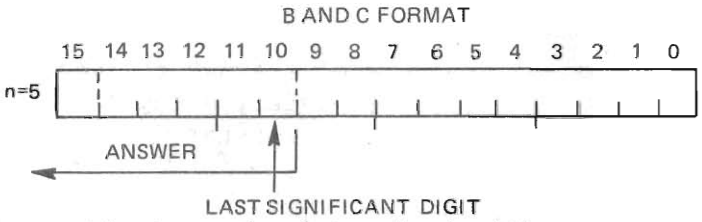
Locating the binary point in the answer depends upon:

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 .0649 ÷ .03 = 2.16, remainder of .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 C and the last significant digit of the remainder in bit 0 of B:



For each decrement of n, the last significant digit is to the left one bit. For example:



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 the C register):

$$BP_C = BP_{B,C} - BP_A - (n-15)$$

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 the C register.)

The binary point in the remainder is calculated as follows:

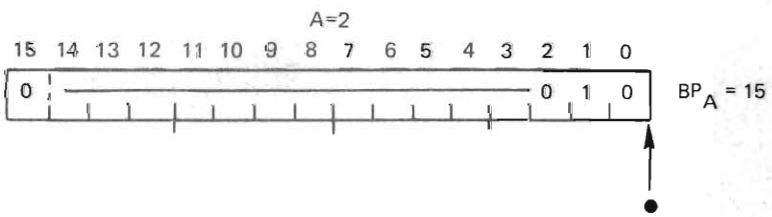
$$BP_B = BP_{B,C} - n$$

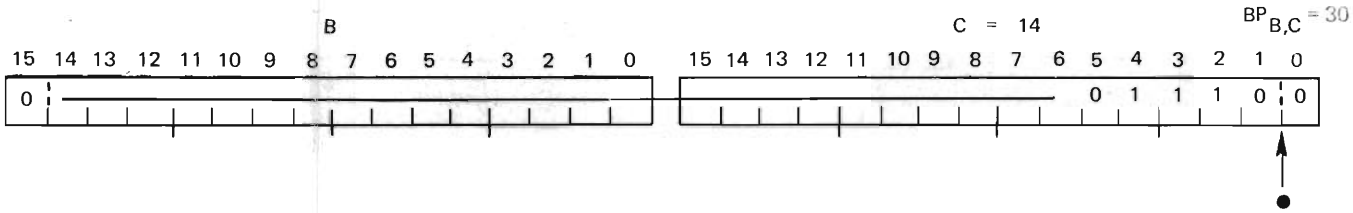
where: $BP_{B,C}$ is the number of integer digits in the dividend.

BP_B is the number of integer digits in the remainder.

Examples:

1. Given the following bit configurations prior to division:



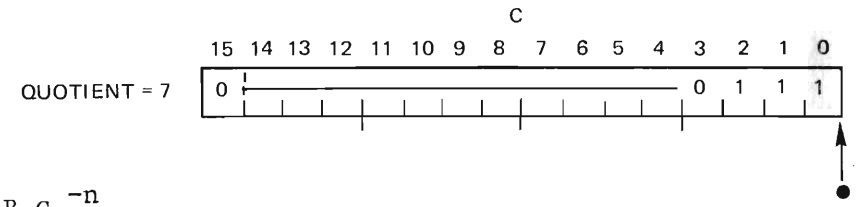


DIV 15 — Divide the contents of the B,C registers by the contents of the A register, using all 15 bits of the dividend in the C register.

Result: $BP_C = BP_{B,C} - BP_A$ (n-15)

$= 30 - 15 - (15-15)$

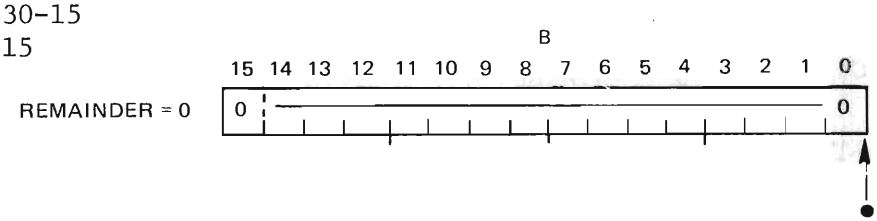
$= 15$



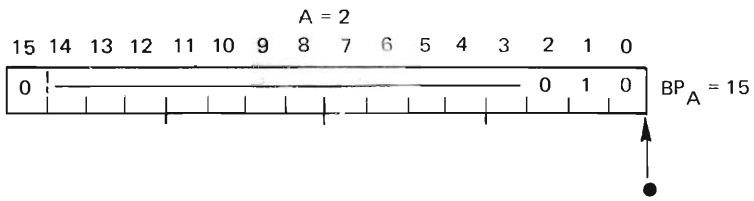
$BP_B = BP_{B,C} - n$

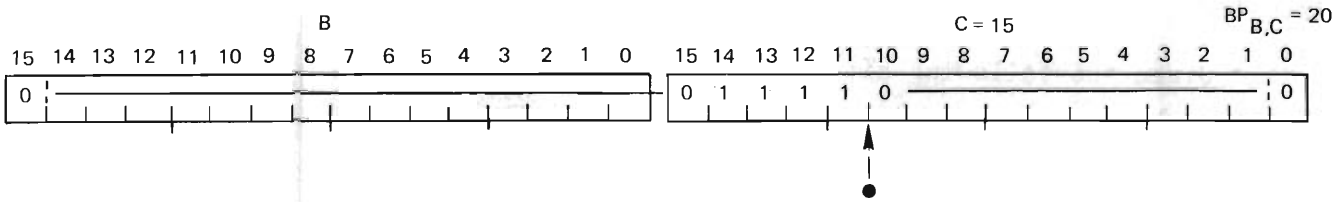
$= 30-15$

$= 15$



2. Given the following bit configurations prior to division:

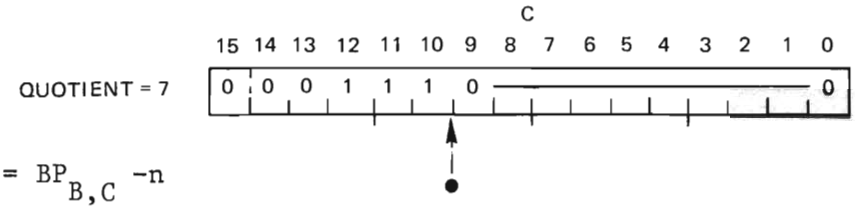




a. DIV 15 Divide the contents of the B, C registers by the contents of the A register, using all 15 bits of the dividend in the C register.

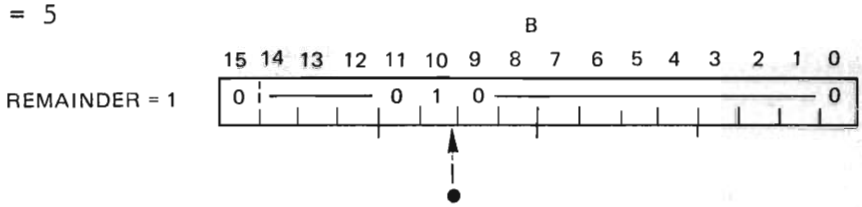
Result: $BP_C = BP_{B,C} - BP_A - (n-15)$

$= 20 - 15 - (15-15)$
$= 5$



$BP_B = BP_{B,C} - n$

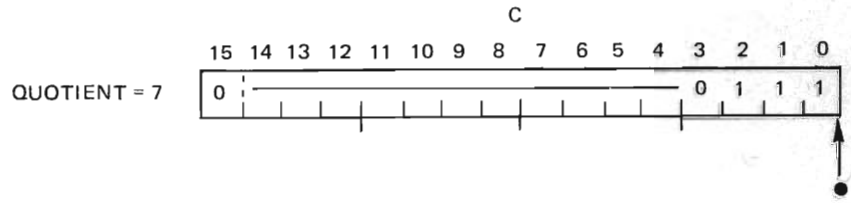
$= 20 - 15$
$= 5$



b. DIV 5 Divide the contents of the B, C registers by the contents of the A register, using 5 high order bits of the C register.

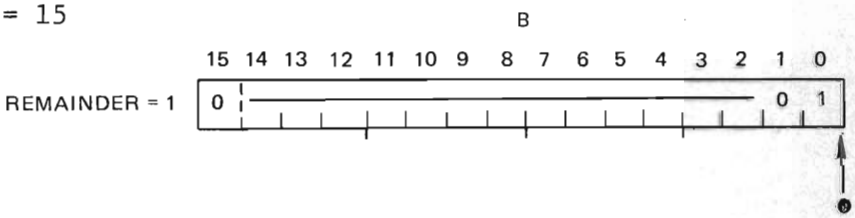
Result: $BP_C = BP_{B,C} - BP_A - (n-15)$

$= 20 - 15 - (5-15)$
$= 15$

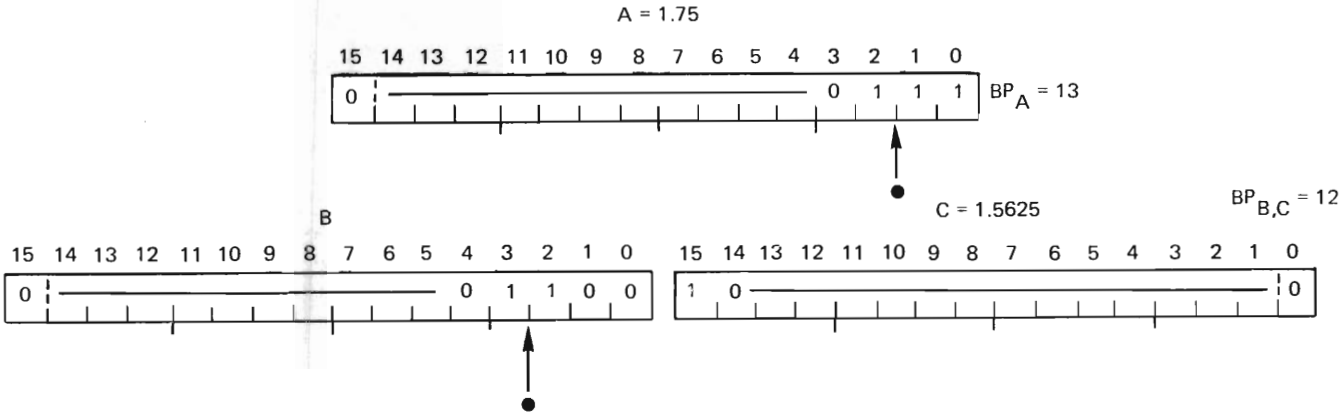


$BP_B = BP_{B,C} - n$

$= 20 - 5$
$= 15$



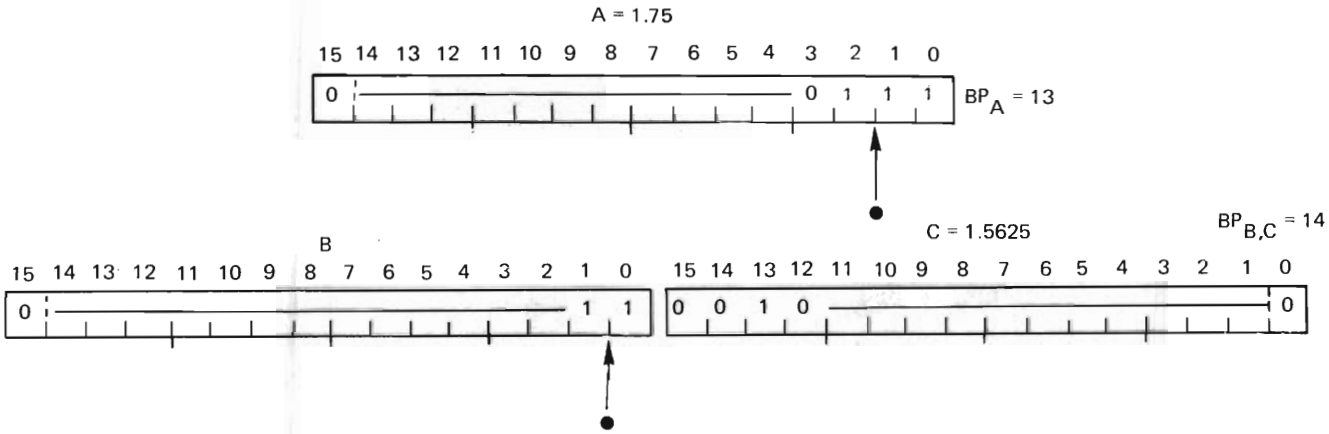
3. Given the following bit configuration prior to division:



Result: The result will be meaningless. The number in B must be less than the number in A prior to division (regardless of their values as determined by the binary points).

Attempting to perform the division with B initially greater than A is equivalent to "skipping over" computing the first significant digits of the answer.

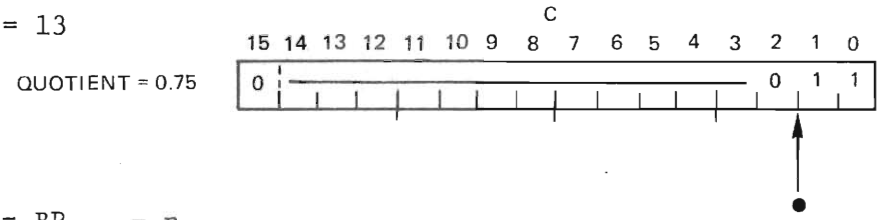
4. Given the following bit configuration prior to division:



DIV 3 Divide the contents of the B, C registers by the contents of the A register, using 2 high order bits of the C register.

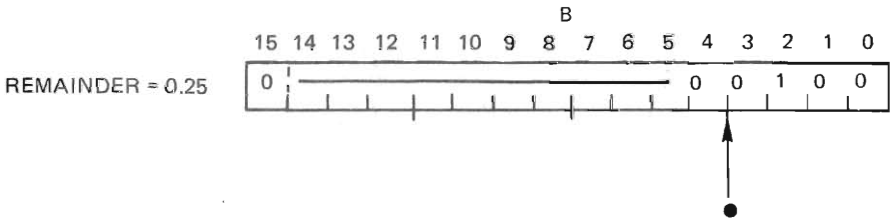
Result: $BP_C = BP_{B,C} - BP_A - (n-15)$

$= 13$



$BP_B = BP_{B,C} - n$

$= 11$



3.11 INSTRUCTION REPERTOIRE SUMMARY

Table 3-3. Instruction Repertoire Summary

Description	Command	Parameters	Indicators Affected	Cycles
Memory Reference				†
Load register A	LDA	[*]a [,m]	None	2
Store register A	STA	[*]a [,m]	None	2
Jump unconditionally	JMP	[*]a [,m]	None	1
Jump to subroutine	JSR	[*]a [,m]	ISE	1
Memory Reference with Indexing				† ‡
Load register	LDR	r,[*]a [,[i] [,m]]	None	2
Store register	STR	r,[*]a [,[i] [,m]]	None	2
Compare memory with register	CMR	r,[*]a [,[i] [,m]]	Ze,Pl,Ov,Lk	2
Load byte	LDBY	r,[*]a [,[i] [,m]]	None	2
Store byte	STBY	r,[*]a [,[i] [,m]]	None	2
Load all registers and status	LARS	[*]a[,[i] [,m]]	Ze,Pl,Ov,Lk,Fg	10
Store all registers and status	SARS	[*]a[,[i] [,m]]	None	10
Increment memory	INCM	[*]a[,[i] [,m]]	Ze,Pl,Lk	2
Decrement memory	DECM	[*]a[,[i] [,m]]	Ze,Pl,Lk	2
Set bit	SBIT	b,[*]a [,[i] [,m]]	Ze	2
Reset bit	RBIT	b,[*]a [,[i] [,m]]	Ze	2
Test bit	TBIT	b,[*]a [,[i] [,m]]	Ze	2
Conditional Jump (Skip)				
Skip if zero	SKZ	adrs	None	1
Skip if non-zero	SKN	adrs	None	1
Skip if plus	SKP	adrs	None	1
Skip if minus	SKM	adrs	None	1
Skip if overflow true	SKOT	adrs	Ov	1
Skip if overflow false	SKOF	adrs	Ov	1
Skip if link set	SKS	adrs	None	1
Skip if link reset	SKR	adrs	None	1
Register Operate				
Transfer register	RTR	Rd,Rs	Ze,Pl	1
Add registers	ADD	Rd,Rs	Ze,Pl,Ov,Lk	1
Subtract registers	SUB	Rd,Rs	Ze,Pl,Ov,Lk	1
And registers	AND	Rd,Rs	Ze,Pl	1
Exclusive Or registers	XOR	Rd,Rs	Ze,Pl	1
Or registers	OR	Rd,Rs	Ze,Pl	1

† Add 1 cycle for indirect addressing
‡ Add 1 cycle for two-word instruction

Table 3-3. (continued)

Description	Command	Parameters	Indicators Affected	Cycles
Register Operate Compare				
Add compare registers	ADDC	Rd,Rs	Ze,Pl,Ov,Lk	1
Subtract compare registers	SUBC	Rd,Rs	Ze,Pl,Ov,Lk	1
And compare registers	ANDC	Rd,Rs	Ze,Pl	1
Exclusive Or compare registers	XORC	Rd,Rs	Ze,Pl	1
Or compare registers	ORC	Rd,Rs	Ze,Pl	1
Register Operate Literal				
Load value to register	LDV	R,lit	Ze,Pl	2
Add value to register	ADDV	R,lit	Ze,Pl,Ov,Lk	2
Subtract value from register	SUBV	R,lit	Ze,Pl,Ov,Lk	2
And value with register	ANDV	R,lit	Ze,Pl	2
Exclusive Or value with register	XORV	R,lit	Ze,Pl	2
Or value with register	ORV	R,lit	Ze,Pl	2
Register Operate Compare Literal				
Add compare value to register	ADDVC	R,lit	Ze,Pl,Ov,Lk	2
Subtract compare value with register	SUBVC	R,lit	Ze,Pl,Ov,Lk	2
And compare value with register	ANDVC	R,lit	Ze,Pl	2
Exclusive Or compare value with register	XORVC	R,lit	Ze,Pl	2
Or compare value with register	ORVC	R,lit	Ze,Pl	2
Register Change				
Increment register	INCR	R	Ze,Pl,Lk	1
Decrement register	DECR	R	Ze,Pl,Lk	1
Complement register	CMPL	R	Ze,Pl,Lk	1
Zero register	ZERO	R	None	1
Zero left byte	ZLBY	R	None	1
Zero right byte	ZRBY	R	None	1
Exchange bytes	EXBY	R	None	1
Register add link	RLK	R	Ze,Pl,Ov,Lk	1
Add shift counter to register	ADDS	R	Ze,Pl,Ov,Lk	1
Transfer status to register	TSR	R	None	1
Transfer register to status	TRS	R	Ze,Pl,Ov,Lk,Fg	1
Read console switches	RCSW	R	None	1
Display register contents	DSPL	R	None	1
Execute register contents	XEC	R	†	‡
Subroutine return	RTRN	R	ISE	1
Restore interrupt enable	RISE	R	ISE	1

† Same indicators as normal execution of specified instruction

‡ One cycle plus cycles required by specified instruction

Table 3-3. (continued)

Description	Command	Parameters	Indicators Affected	Cycles
Shift				
Shift right arithmetic	SRA	R,n	Ze,Pl,Lk	1+0.5n
Shift right circular	SRC	R,n	Ze,Pl,Lk	1+0.5n
Shift right logical and count	SRLC	R,n	Ze,Pl,Lk	1+0.5n
Shift right circular through link	SRCL	R,n	Ze,Pl,Lk	1+0.5n
Control				
Interrupt enable	INE		ISE	1
Interrupt inhibit	INH		ISE	1
Foreground mode set	*FMS		None	1
Background mode set	*BMS		None	1
Pulse operations monitor alarm	*PMA		None	1
Link set	LKS		Lk	1
Link reset	LKR		Lk	1
Generate sync pulse	SYNC		None	1
Wait	WAIT		None	1
Programmed Input/Output				
Data out from register	DTOR	R,dev adrs	None	2
Data in to register	DTIR	R,dev adrs	None	2
Data out from memory	DTOM	R,dev adrs	None	2
Data in to memory	DTIM	R,dev adrs	None	2
Read console switches into register	RCSR	R,X'3E'	None	2
Read console switches into memory	RCSM	R,X'3E'	None	2
Test device	TEST	fun,dev adrs	None	2
Output control function	CTRL	fun,dev adrs	None	2
Hardware Multiply and Divide (Optional)				
Hardware multiply	*MPY	n	Lk	1.5+0.5n
Hardware divide	*DIV	n	Lk	1.5+n

* Effective only if appropriate option is installed.

SECTION 4
PERIPHERAL PRODUCT PROGRAMMING AND DESCRIPTION

This section describes the standard peripheral I/O units that are available with the SPC-16 and discusses some basic considerations of I/O programming.

Section 4.1 discusses Programmed I/O and Direct Memory Access transfers. The remaining sections describe the following peripheral units:

- Console Teletype (Section 4.2)
- High Speed Paper Tape Reader (Section 4.3)
- High Speed Paper Tape Punch (Section 4.4)
- Card Reader (Section 4.5)
- Line Printer (Section 4.6)
- 3341/3343 Disk Storage System (Section 4.7)
- 3346/3347 Disk Storage System (Section 4.8)
- Magnetic Tape Unit (Section 4.9)

Sections 4.2 through 4.9 include examples of the types of programs that may be written to implement I/O transfers for each unit. These examples are included only as illustrations of one approach to I/O programming; they are not intended to serve as a directly usable I/O system.

4.1 PERIPHERAL PRODUCT PROGRAMMING

In programming I/O transfers, the programmer must understand the fundamental distinction between Programmed I/O and Direct Memory Access. Programmed I/O involves data transfers that are conducted under program control; the transfer of each data item requires the execution of programmed instructions. Direct Memory Access (DMA), however, frees the program from the item-by-item transfer. The program is only required to initiate the operation; following initiation, the peripheral controller conducts the specified transfer without further intervention from the program.

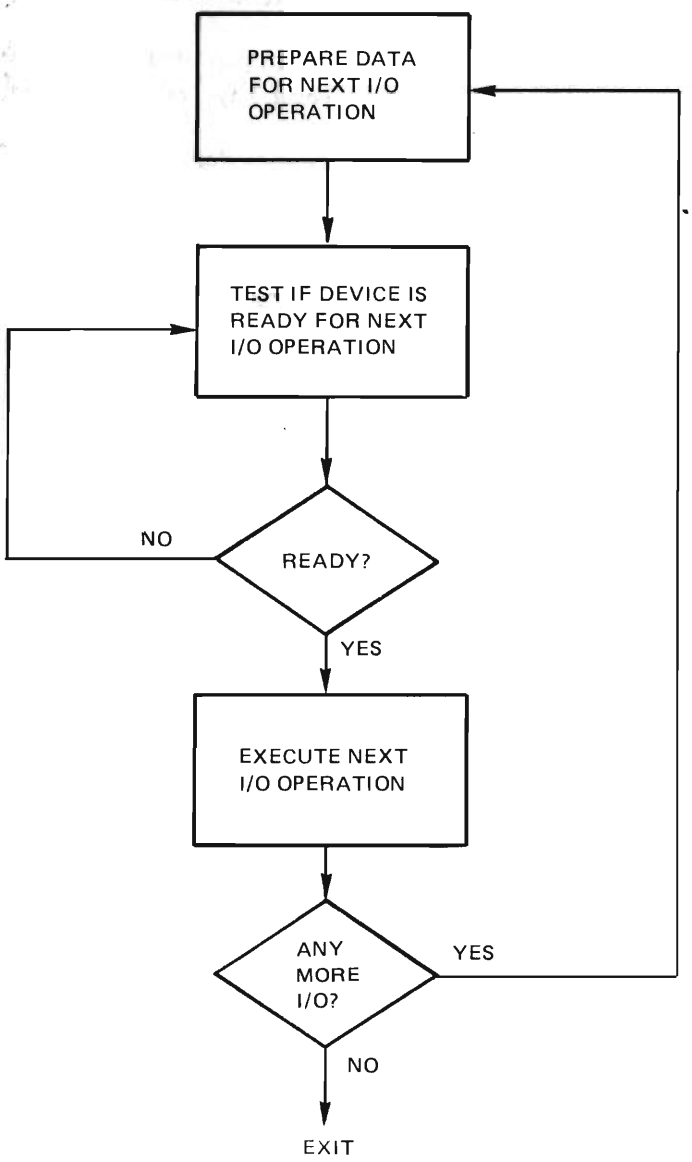
4.1.1 Programmed I/O

Programmed I/O may be either test loop-driven or interrupt-driven. Test loop-driven programmed I/O requires the CPU's total dedication to the I/O transfer from time of initiation until it has been completed. Interrupt-driven programmed I/O requires the program to initiate the transfer and to relinquish control to a routine that transfers one data item each time the unit is ready to send/receive a data item (i.e., each time the controller issues an interrupt).

TEST LOOP-DRIVEN PROGRAMMED I/O

Test loop-driven programmed I/O is perhaps the simplest I/O method to program since it involves the least number of instructions and the data transfer sequence is coded within one routine. Test loop-driven programmed I/O consists of a series of loops. The initial loops test the availability of the unit (i.e., the "busy" and "ready" status). When the unit is ready to accept the transfer instruction, the transfer is conducted, item-by-item, until completed.

In general, the I/O program steps look like this:



A test loop-driven programmed I/O routine that outputs one data item from Register R to Controller IJ could be programmed as follows:

```
HERE    TEST    M,IJ        Test if controller IJ is ready to
                            receive data (M stands for a code
                            that specifies the particular test
                            line being sampled. If the controller
                            is not busy, skip the next instruction.
```

```
JMP     HERE    The controller was busy, jump
                back to test again.

DTOR    R,IJ    Execute the data out instruction.
```

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.

Examples of test loop-driven programmed I/O are given in Sections 4.2.3, 4.3.3 and 4.4.3.

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. When 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 μseconds; 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, Direct Memory Access (DMA), which allows a block of data to be transferred under control of the peripheral controller, is required.

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 program 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 SPC-16 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 trans-
                  ferred.

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.

Examples of interrupt-driven programmed I/O are given in Sections 4.3.3 and 4.4.3.

Like test loop-driven programmed I/O, interrupt-driven programmed I/O may prove insufficiently 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.

4.1.2 Direct Memory Access (DMA)

If the controller for a particular peripheral unit is designed to use the Direct Memory Access facility, the programmer's calling program needs only to initiate the transfer with the controller. The calling program must tell the controller the number of data words to be transferred, the buffer address in memory, and the type of data transfer. Once initiated, the DMA data transfer proceeds without program intervention. The controller performs the transfer by "stealing" memory cycles from the CPU each time the peripheral unit is ready to send/receive a word.

The SPC-16 system makes provision for data channels in groups of 8 each through the addition of high speed data channel (MHSDC) modules for DMA transfers. Up to 4 MHSDC's may be installed per system (limit of one MHSDC per I/O adapter). Each data has been allocated two dedicated memory words, one each for its word count (scan control register (SCR)) and data table address (channel address register (CAR)), as discussed in Section 2.4. A number of data channels have been assigned to standard GA peripheral units that use DMA; these assignments are summarized in Table 4-1.

Table 4-1. Standard Data Channel Assignments for Peripheral Units

Channel	(Hex)	(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/Punch
4	28	29	"Floppy" disk or cassettes
5	2A	2B	Card Punch or Punch portion of Card Reader/Punch
6	2C	2D	Not assigned
7	2E	2F	Line Printer

Direct Memory Access tranfers may be programmed in either a non-interrupting or an interrupting mode.

In the non-interrupting mode, the program consists of a series of test loops that determine when the unit is ready to accept a transfer command. The program is responsible for loading the word count and data buffer address into the dedicated SCR and CAR locations and, when the unit is ready, must output the transfer command to the peripheral controller. Since interrupts are not permitted, the program must also either loop until the operation is completed or periodically check for completion. An example of DMA programming in the non-interrupting mode is given in Section 4.5.3.

The I/O routine for DMA transfers in the interrupting mode are constructed similar to an interrupt-driven programmed I/O routine (see Section 4.1.1) except that the interrupt service routine is needed to check only status indicators and is not involved in the actual data transfer. Examples of DMA programming in the interrupting mode are given in Sections 4.6.3, 4.7.3, 4.8.3 and 4.9.3.

4.2 CONSOLE TELETYPE

The ASR Model 33 Teletype (TTY) has a keyboard, printer, paper tape reader and paper tape punch. The teletype can be operated independently, or it can be used on-line to input or output data to the CPU. The maximum data transmission rate is ten characters per second. The teletype can request an interrupt when it requires service, or an executing program can test the teletype to see if it is ready to be serviced.

The teletype punches standard ASCII character codes compatible with other perpipheral units which handle eight channel paper tape.

The controller for the console teletype is housed in the computer chassis as an internal serial/parallel and parallel/serial converter operating on the programmed I/O bus. This internal controller operates identically to an external controller, except for the technique used in setting and resetting the interrupt mask.

For peripheral units with external controllers, the interrupt masks are normally set and reset using an I/O Control command addressed to the device, where one combination of the function bits (8-10) is assigned to set the mask and another combination is assigned to reset the mask. Console teletype interrupts are initiated in a similar manner, except that the mask is set up separately (either in a register or a memory location). The mask also selectively enables three other internal interrupts:

- The Memory Protect Violation,
- Relative Time Clock and
- Console Interrupt

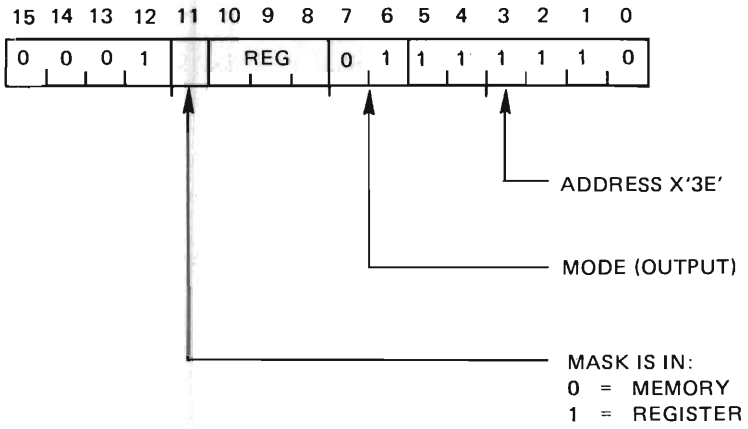
One of the I/O device select codes, X'3E', is assigned to address these internal interrupts for interrupt masking. An I/O Data Output command referencing the mask location and issued with a device select code of X'3E' will selectively enable/disable all four interrupts. The instruction and mask formats are illustrated in Figure 4-1.

The internal mask register is not affected by SYSTEM RESET. When power is turned on in the processor, the mask register will assume an unknown state. The Auto-restart interrupt has no associated mask register bit; it is always enabled during a power-up sequence. It is the responsibility of the program to initialize the state of the mask register after power has been turned on (and thereafter).

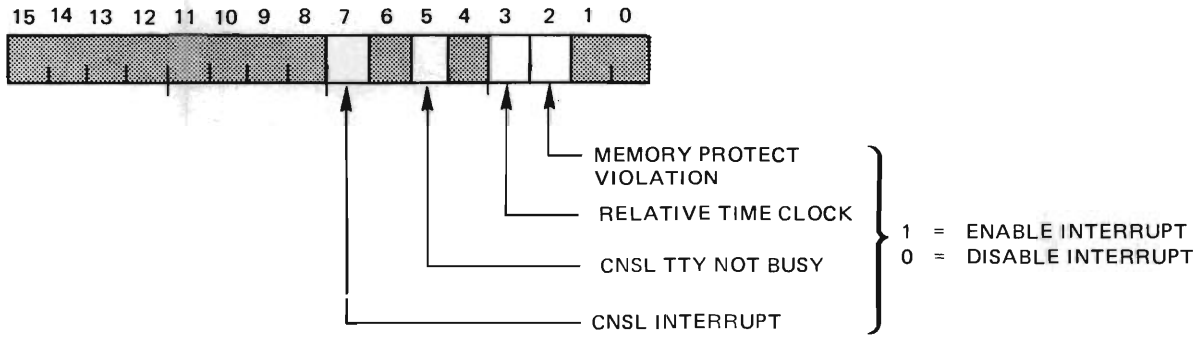
SYSTEM RESET does not clear the system enable flip-flop which is a master control over all interrupts with the exception of the POWER FAIL interrupts. When power first comes up, "SYRT" is automatically generated, but "INE" is temporarily forced to the enable state by the Auto-restart condition. The "INE" control allows the Auto-restart interrupt, "INE" is disabled. Its further use and control is the responsibility of the program.

ASSEMBLER MNEMONICS: DTOR R,DEV
DTOM R,DEV

INSTRUCTION FORMAT



MASK FORMAT



EXAMPLE: TO ENABLE ALL INTERNAL INTERRUPTS USING A MASK IN THE A REGISTER:

```
LDV     A,X'AC'     (OR ANY NUMBER THAT PUTS ONES IN
                    BITS 2, 3, 5 AND 7)
DTOR    A,X'3E'     ENABLE INTERNAL INTERRUPTS
```

Figure 4-1. Formats for Masking Internal Interrupts

4.2.1 Operational Description

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.

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 SPC-16 (depending on the operating mode). TTY NOT BUSY also goes false when the teletype's operating mode is changed and when the teletype is turned off.

The teletype controller operates in four modes:

- TRANSMIT
- RECEIVE ONLY
- RECEIVE AND ECHO
- BREAK

as governed by which one of four Control instruction codes the controller receives from the SPC-16. Additionally, when the SPC-16 system is initialized, the console teletype is placed into the RECEIVE ONLY mode in a "busy" state.

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 Data Out instruction). TTY NOT BUSY goes true and initiates another service request. Changing to the BREAK mode (at end of transmission) or to either RECEIVE or RECEIVE & ECHO (if data is to be subsequently input) will remove the service request.

During RECEIVE ONLY mode operation, the controller sends teletype data to the CPU. During RECEIVE & ECHO mode 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 when the teletype is waiting with a character to send. The service request is removed when the character has been input.

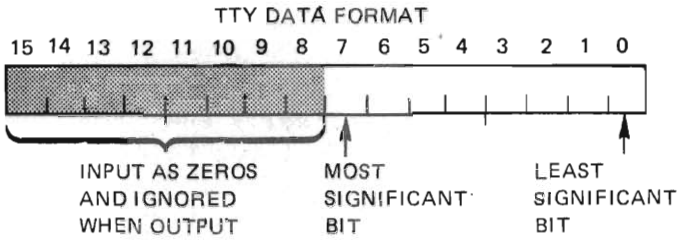
The BREAK mode is intended to be a "transmission ended" mode, during which a timer may cause the teletype to shut off if the mode is not changed within a pre-determined time period.

Control functions are summarized below:

Mode	Description	To Remove Service Request	Specific Rules
TRANSMIT Mode	Transmit to printer and/ or punch.	1. Output character. 2. Change mode.	1. New character starts when DTOR or DTOM occurs. 2. Service request occurs when not busy.
RCV ONLY Mode	Teletype input to processor buffer only.	1. Input a character. 2. Receive subsequent sync character. 3. Change from one receive mode to another receive mode.	1. New character starts when not busy and mark-to-space transition occurs. Mark=current flow =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.
RCV & ECHO Mode	Teletype input to processor buffer and to teletype printer and/ or punch.		
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.		

4.2.2 Instruction Summary

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:



The mode (initialized to RECEIVE ONLY) can be changed by executing a CTRL instruction. A mode change should occur only when the teletype is not busy.

The console teletype I/O instructions are summarized in Table 4-2.

Table 4-2. Instruction Summary for Console Teletype

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
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

4.2.3 Programming Examples

These examples show how the console teletype can be programmed without the use of interrupts.

1. Programmed input/output instructions are used in the teletype bootstrap which, when executed, will load data or programs from the teletype. The D register is manually set to the beginning load address; the X register is used to reference sequential byte elements from the base address location.

```
        ZERO    X           Initialize byte index.
LOOP    DSPL    X           Display the number of the byte to be read.
        TEST    0,X'3F'     Test for teletype not busy.
        JMP     $-1         Loop until teletype is ready.
        DTIR    A,X'3F'     Input character.
        STBY    A,0,X,1     Store character
        INCR    X           Increment byte index.
        JMP     LOOP        Continue looping.
```

When end of tape (or keyed input) is reached, the program loops at TEST/JMP $-1 instructions until the bootstrap execution is manually halted.

2. Subroutine RLINE reads one line of input (terminated by a carriage return) from the console teletype, and stores the input beginning at the address indirectly accessed by the D register.

Subroutine:

```
RLINE   ZERO    X           Initialize byte index.
        CTRL    4,X'3F'     Set Receive & Echo mode.
LOOP    TEST    0,X'3F'
        JMP     LOOP        Loop until TTY has a character to input.
        DTIR    A,X'3F'     Input character to A register.
        SUBVC   A,X'8D'     Test for carriage return.
        SKZ     LF
        STBY    A,*0,X,1    Store character.
        INCR    X           Increment byte index.
        JMP     LOOP        Jump to input next character.
LF      CTRL    0,X'3F'     Set Transmit mode.
        LDV     A,X'8A'     Load Line Feed code.
        DTOR    A,X'3F'     Output Line Feed.
        RTRN    E           Array (D) to (D)+X contains input line.
```

4.3 HIGH SPEED PAPER TAPE READER

The High Speed Paper Tape Reader (HSPTR) allows the high speed input of data from an eight channel, punched paper tape medium. The HSPTR is a character oriented peripheral unit; it operates completely under program control. The program may interrogate the HSPTR to determine if it requires service, or the HSPTR may be allowed to request a program interrupt when it requires service.

4.3.1 Operational Description

The operational status of the HSPTR is reflected by the state of three indicators on the controller: READY, OPERATION COMPLETE and COMMAND REJECT. Three Test instruction codes have been assigned to interrogate the state of these indicators.

READY is true whenever the controller is ready to accept a STEP command to advance the paper in the reader. READY is false when a condition on the HSPTR requires manual intervention (e.g., the HSPTR is turned off). READY is also false while the HSPTR is in the process of advancing the paper tape to the next character position.

OPERATION COMPLETE becomes true after a STEP command has been issued and the paper tape has been advanced to the next character position (the next data frame is under the read head). OPERATION COMPLETE also becomes true whenever COMMAND REJECT becomes true. OPERATION COMPLETE is cleared by an acceptable STEP command, a RESET STATUS command or a DATA INPUT command. If the interrupt mask for the HSPTR is enabled when OPERATION COMPLETE is true, the device requests a program interrupt. Clearing OPERATION COMPLETE will remove the interrupt request.

COMMAND REJECT becomes true if a STEP command is given to the controller while it is performing another operation or if the HSPTR is turned off (i.e., READY is false). COMMAND REJECT will cause OPERATION COMPLETE to become true, causing an interrupt if the interrupt mask is enabled. COMMAND REJECT is cleared when OPERATION COMPLETE is cleared.

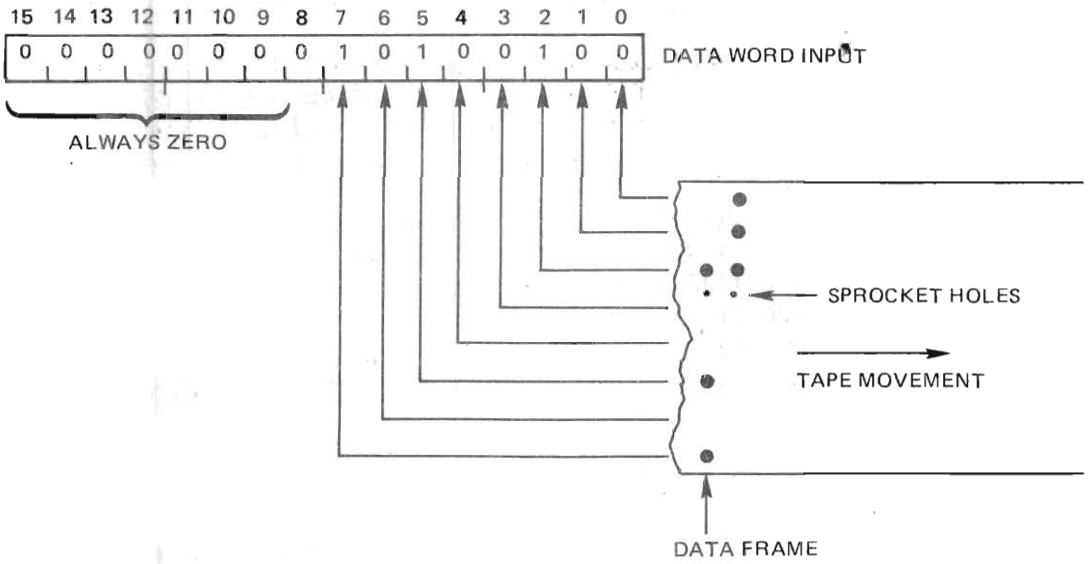
Four Control instruction codes have been assigned to the HSPTR. These commands perform the following functions: ENABLE INTERRUPT, DISABLE INTERRUPT, STEP and RESET STATUS.

The ENABLE INTERRUPT command sets the interrupt mask on the controller to allow the controller to request an interrupt with OPERATION COMPLETE is true. The DISABLE INTERRUPT command resets the interrupt mask, preventing the HSPTR controller from requesting an interrupt. The state of the mask remains in effect until an enable/disable command is given or the SYSTEMS RESET switch on the computer's console is depressed (which disables interrupts on all peripheral controllers).

When the program issues a STEP command, the HSPTR drive mechanism begins advancing the paper tape to the next data frame (each data frame is marked by a sprocket hole in the paper tape). The time required to advance the tape depends upon which HSPTR model is being used; typically, the time required is on the order of 3 to 5 milliseconds. When the drive mechanism has positioned the next frame over the read head, OPERATION COMPLETE becomes true.

The RESET STATUS command can be used to clear the OPERATION COMPLETE and COMMAND REJECT indicators; it has no effect on the READY indicator. Since the STEP and DATA INPUT commands also perform this function, RESET STATUS is normally used only when no operation other than clearing the status is required.

One Data Input instruction is used with the HSPTR. This command causes the reading of the buffer, which holds the data frame which was over the read head at the last OPERATION COMPLETE. The Data Input instruction reads a full word (16-bits) of data; however, in the case of the HSPTR, only the eight least significant bits reflect the contents of the data frame.



4.3.2 Instruction Summary

The controller for the HSPTR has been assigned a device select code of X'8'. The memory location dedicated to this controller's interrupt vector is X'48'.

Table 4-3 summarizes the XIO instructions that apply to the HSPTR. The instructions are given in CAP-16 Assembly Language format (see Section 3.9 for their machine language equivalents).

Table 4-3. Instructions Summary for High Speed Paper Tape Reader

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' or DTIM R,X'8'	INPUT DATA from current data frame into register (R) or into memory

R = A,X,Y,Z,B,C,D or E

4.3.3 Programming Examples

The HSPTR may be operated in a non-interrupting or interrupting mode. Examples are given for each mode.

NON-INTERRUPT SERVICING

When the HSPTR is not allowed to interrupt (i.e., its interrupt mask is disabled), the program must interrogate its OPERATION COMPLETE flag to determine when the Reader has completed the STEP operation and a character is ready to input. The following instruction sequence may be used to advance the tape one frame and input the character.

(NOTE: Interrupt mask for HSPTR is disabled.)

```
TEST    1,X'8'    Is HSPTR READY?
JMP     $ - 1     No - test again
CTRL    3,X'8'    Yes - STEP reader
TEST    0,X'8'    Is OPERATION COMPLETE?
JMP     $ - 1     No - test again
DTIR    A,X'8'    Yes - INPUT DATA to A register
```

INTERRUPT SERVICING

The programming example below illustrates a subroutine that may be called by the main program to input a specified number of characters from the HSPTR and place the characters in sequential locations in a specified data buffer.

The subroutine consists of two parts:

- An Initialization Routine (PTRIN)
- An Interrupt Service Routine (INTRPT)

The initialization routine sets up the interrupt vector for the HSPTR's interrupt to point to the interrupt service routine. In addition, it sets up the transfer parameters, the character count and buffer address, in common locations (TABLE) and initiates the first STEP operation. Each time an interrupt occurs (when a STEP operation has been completed) program control is transferred to the interrupt service routine. The interrupt service routine reads in the character, issues a new STEP command and returns to the interrupted main program.

In addition, the subroutine defines and maintains a memory word (HSPTR) that the programmer must interrogate in order to check whether a previously-initiated operation has been completed. Bit 15 of this word is set when the operation begins and is reset when all characters have been input.

```
        JSR       PTRIN      Calling sequence
        DC        20         No. of characters to input
        DC        BUFFER     Address of data buffer
        (Return when input has been initiated, execute code as required
        and monitor HSPTR for Bit 15(10)=0)
TABLE   DS        1          Will contain character count
        DS        1          Will contain buffer address
        DS        1          Will contain byte index
HSPTR   DS        1          Bit 15 will contain operation
                              complete indicator
PTRIN   SARS      *0         Save calling program's registers
        LDR       B,0   }
        RTR       C,B   }
        ADDV      C,9   }    Entry sequence
        STR       C,0   }
        SBIT      7,HSPTR    Set operation not complete
        LDV       D,TABLE    Setup D register
        RTR       X,E        Setup to fetch parameters
        INE                  Enable interrupts
        LDR       A,0,X      Fetch character count
        STR       A,0,,1     Store in TABLE
        LDR       A,1,X      Fetch buffer address
        STR       A,1,,1     Store in TABLE+1
        ZERO      A
        STR       A,2,,1     Zero byte index (TABLE+2)
        LDV       A,INTRPT
        STR       A,X'48'    Setup interrupt vector
        TEST      1,X'8'     Is HSPTR READY?
        JMP       $ - 1      No - test again
```

```
            CTRL    3,X'8'      Yes - STEP to next data frame
            CTRL    0,X'8'      Enable interrupt from HSPTR
            INH                 Exit sequence
            STR     B,0         Restore registers
            LARS    *0          and adjust
            ADDV    E,2         return address
            RTRN    E           Return to calling program

INTRPT      SARS    *0          Entry sequence
            LDR     B,0         Save registers, etc.
            RTR     C,B
            ADDV    C,9
            STR     C,0

            CTRL    1,X'8'      Disable interrupt from HSPTR*
            INE                 Enable interrupts (ISE)*

            TEST    3,X'8'      Is interrupt due to COMMAND
            JMP     $ + 2       REJECT? Yes - Test READY
            JMP     $ + 4       No - Input character
            TEST    1,X'8'      Test READY?
            JMP     $ - 1       Not READY - try again
            JMP     STEP        READY - advance tape
            DTIR    A,X'8'      Read in character
            LDV     D,TABLE     Set up D register
            LDR     X,2,,1      Fetch byte index
            STBY    A,*1,X,1    Store char. in data buffer
            INCM    2,,1        Increment byte index
            DECM    0,,1        Decrement character count
            SKZ     EXIT        If zero - operation is complete
STEP        CTRL    3,X'8'      STEP tape to next character

            INH                 Inhibit master interrupt (ISE)**

            CTRL    0,X'8'      Enable interrupt from HSPTR

            STR     B,0         Restore registers of
            LARS    *0          interrupted program
            RTRN    E           Return
EXIT        INH                 Inhibit master interrupt (ISE)
            RBIT    7,HSPTR     Indicate operation complete
            STR     B,0         Restore registers of
            LARS    *0          interrupted program
            RTRN    E           Return - operation completed
```

* These commands are optional, their inclusion improves system interrupt response.
** Not required if the two optional commands above are deleted.

4.4 HIGH SPEED PAPER TAPE PUNCH

The High Speed Paper Tape Punch (HSPTP) punches data, output by the computer, on eight-channel paper tape. The HSPTP is a character oriented peripheral unit; it operates completely under program control. The program may interrogate the HSPTP to determine if it requires service, or the HSPTP may be allowed to request a program interrupt when it requires service.

4.4.1 Operational Description

The operational status of the HSPTP is reflected by the state of three indicators on the controller: READY, OPERATION COMPLETE and COMMAND REJECT. Three Test instruction codes have been assigned to interrogate the state of these indicators.

READY is true whenever the controller is ready to accept an OUTPUT command (explained below). READY is false when a condition of the HSPTP requires manual intervention (e.g., the HSPTP is turned off). READY is also false while the HSPTP is in the process of advancing the paper tape to the next character position. The RESET STATUS command does not affect this indicator.

OPERATION COMPLETE becomes true after an OUTPUT command has been issued and the paper tape has been advanced to the next character position and the character has been punched. OPERATION COMPLETE also becomes true whenever COMMAND REJECT becomes true. OPERATION COMPLETE is cleared by an acceptable OUTPUT command or a RESET STATUS command. If the interrupt mask for the HSPTP is enabled when OPERATION COMPLETE is true, the controller requests a program interrupt. Clearing OPERATION COMPLETE will remove the interrupt request.

COMMAND REJECT becomes true if an OUTPUT command is given to the controller while it is performing another operation or if the punch is turned off (i.e., READY is false). COMMAND REJECT will cause OPERATION COMPLETE to become true, causing an interrupt if the interrupt mask is enabled. COMMAND REJECT is cleared when OPERATION COMPLETE is cleared.

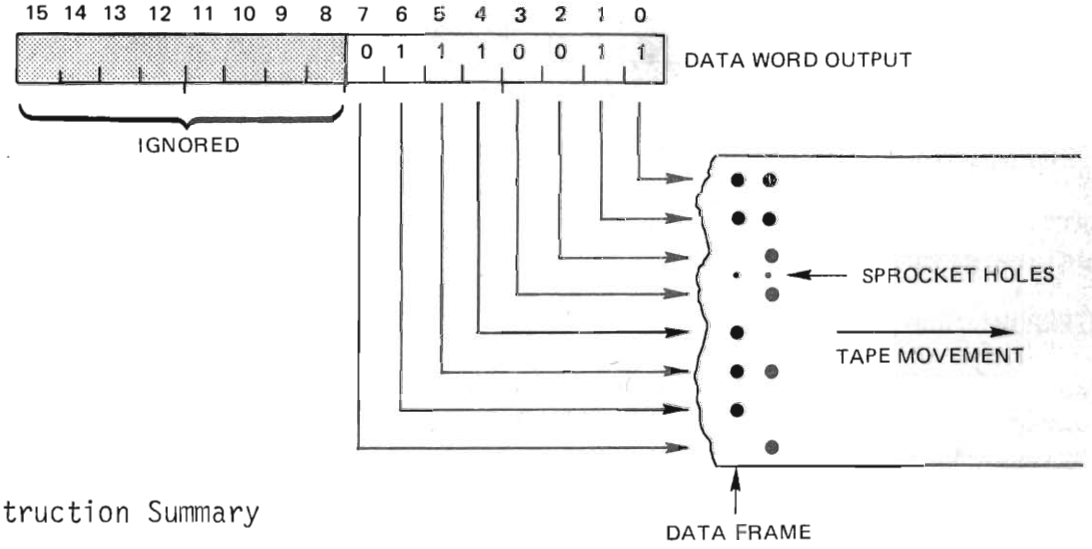
Three Control instruction codes have been assigned to the HSPTP. These commands perform the following functions:

- ENABLE INTERRUPT
- DISABLE INTERRUPT
- RESET STATUS

The ENABLE INTERRUPT command sets the interrupt mask on the controller to allow the controller to request an interrupt when OPERATION COMPLETE is true. The DISABLE INTERRUPT command resets the interrupt mask, preventing the HSPTP controller from requesting an interrupt. The state of the mask remains in effect until one of these commands is given or the SYSTEMS RESET switch on the computer's console is depressed (which disables interrupts on all peripheral controllers).

The RESET STATUS command can be used to clear OPERATION COMPLETE and COMMAND REJECT. Since the OUTPUT command also performs this function, RESET STATUS is normally used only when no operation other than clearing the status is required.

One Data Output instruction is assigned to the HSPTP to perform both a tape advance and a character punch operation. When the program issues an OUTPUT command, the character in the specified register (or in the memory location whose address is contained in the specified register) is output to the HSPTP controller. The controller advances the paper tape to the next character position; when the tape is in position, the data frame and sprocket hole is punched. After the character is punched, OPERATION COMPLETE becomes true. The OUTPUT command should be issued only when the controller is READY; otherwise, a COMMAND REJECT will result. The OUTPUT instruction outputs a full word (16-bits) of data; however, only the low order byte (bits 0-7) is accepted by the controller and punched.



4.4.2 Instruction Summary

The controller for the HSPTP has been assigned a device select code of X'09'. The memory location dedicated to this controller's interrupt vector is X'49'.

Table 4-4 summarizes the XIO instructions that apply to the HSPTP. The instructions are given in CAP-16 Assembly Language format (see Section 3.9 for their machine language equivalents).

Table 4-4. Instruction Summary for High Speed Paper Tape Punch

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' or DTOM R,X'9'	OUTPUT data from register (R) or from memory to HSPTP controller; advance tape; punch character

R = A,X,Y,Z,B,C,D or E

4.4.3 Programming Examples

The HSPTP will operate in either a non-interrupting or an interrupting mode. Examples are given for each mode.

NON-INTERRUPT SERVICING

When the HSPTP is not allowed to interrupt (i.e., its interrupt mask is disabled), the program must interrogate its OPERATION COMPLETE flag to determine when the unit has completed the last OUTPUT operation. The following instruction sequence may be used to punch a character contained in the A register.

```
PUNCH   TEST    1,X'9'      Is HSPTP READY?
        JMP     $ - 1       No - test again.
        DTOR    A,X'9'      Yes - OUTPUT character, advance
                             tape, and punch
        TEST    0,X'9'      OPERATION COMPLETE?
        JMP     $ - 1       No - test again
        TEST    3,X'9'      Yes - test COMMAND REJECT

        JMP     PUNCH       Yes - retry operation
        .                   Loop control logic
        .
        .
        CTRL    2,X'9'      No - RESET STATUS, clear last operation
                             complete indication.
```

INTERRUPT SERVICING

When the HSPTP is allowed to interrupt, the output routine normally consists of two parts: an initialization routine (INITP), which sets up the interrupt vector and transfer parameters (character count and data buffer address); and an interrupt service routine (IPUNCH), which responds to the interrupt and outputs the next character.

In addition, the subroutine defines and maintains a memory word that the programmer must interrogate in order to check whether a previously-initiated operation has been completed. Bit 15 of this word is set when the operation begins and is reset when all characters have been output.

```
        JSR     INITP     Calling sequence
        DC      30        No. of characters to output
        DC      PBUFFR    Address of data buffer
                (Return when output is started to continue
                execution. Program must monitor HSPTP, Bit 15(10)=0
                when all data bytes output)
TABLE   DS      1         Will contain character count
        DS      1         Will contain buffer address
        DS      1         Will contain byte index
HSPTP   DS      1         Will contain operation complete indicator

INITP   SARS    *0        }
        LDR     D,0       }
        RTR     C,D       }  Entry sequence
        ADDV    C,9       }
        STR     C,0       }
        SBIT    7,HSPTP   Set operation not complete
        RTR     X,E       Set up to fetch parameters
        ADDV    E,2
        STR     E,7,,1
        LDV     D,TABLE   Set up D register
        INE               Enable interrupts
        LDR     A,0,X     Fetch character count
        STR     A,0,,1    Store in TABLE
        LDR     A,1,X     Fetch buffer address
        STR     A,1,,1    Store in TABLE+1
        ZERO    A
        STR     A,2,,1    Zero byte index
        LDV     A,IPUNCH
        STR     A,X'49'   Set up interrupt vector
        TEST    1,X'9'    Is HSPTP READY?
        JMP     $ - 1     No - test again
        JMP     0FIRST    Jump to output 1st character

IPUNCH  SARS    0*        Entry sequence
        LDR     B,0       Save register, etc.
        RTR     C,B
        ADDV    C,9
        STR     C,0
```

```
OFIRST CTRL      1,X'9'      Disable interrupt from HSPTP
       INE                   Enable master interrupt (ISE)
       LDV       D,TABLE     Setup D register
       LDR       X,2,,1      Fetch byte index
       LDBY      A,*1,X,1    Fetch character to output
       DTOR      A,X'9'      OUTPUT character
       INCM      2,,1        Increment byte index
       DECM      0,,1        Decrement character count
       SKM       EXIT        Test for exit
       INH                   Not all characters have been output
       CTRL      0,X'9'      Enable interrupt from HSPTP
       STR       B,0         Restore registers, etc.
       LARS     *0
       RTRN      E           Return to main program

EXIT   INH                   All characters have been output
       RBIT      7,HSPTP     Indicate operation complete
       CTRL      2,X'9'      Reset Interrupt
       STR       B,0         Restore registers, etc.
       LARS      *0          Note: Interrupt from HSPTP
                                   is not enabled
       RTRN      E           Return - operation completed
```

4.5 CARD READER

The Card Reader (CR) is a record oriented device that inputs data from an 80 column, punched card medium. The initiation of a data transfer operation (i.e., the reading of each card) for the CR is under program control. However, for the actual transfer of data words the CR controller employs a Data Channel (Direct Memory Access); the controller supervises the transfer of a data word directly to the computer's memory as each card column is read. The CR can be interrogated by the program to determine if it requires service, or it can be allowed to request a program interrupt when it requires service.

4.5.1 Operational Description

Five indicators on the CR controller reflect the status of the CR:

- READY
- OPERATION COMPLETE
- ERROR
- COMMAND REJECT
- NOT BUSY

Five Test instruction codes have been assigned to interrogate these status indicators.

READY is true whenever the CR is ready to accept a READ command. READY is false when the CR is in the process of reading a card (i.e., a READ command has been issued but the operation has not been completed). READY is also false when a condition that requires manual intervention exists on the CR (i.e., power off, hopper empty, stacker full or ERROR). READY is not affected by the RESET STATUS command.

OPERATION COMPLETE becomes true after a READ command has been given, a card is fed, read and properly stacked. OPERATION COMPLETE also becomes true when ERROR and/or COMMAND REJECT are true. When OPERATION COMPLETE is true, the CR controller requests a program interrupt if its interrupt mask is enabled. OPERATION COMPLETE is cleared by the RESET STATUS command or by an acceptable READ command.

ERROR becomes true when a read operation cannot be successfully completed due to an equipment malfunction, i.e., a "pick" failure, jam or read check (light/dark check) occurs. ERROR causes READY to become false since manual intervention is required to correct the failure. ERROR is not affected by the RESET STATUS command.

COMMAND REJECT becomes true if a READ command is given to the CR controller while it is performing a previous read operation or while READY is false. COMMAND REJECT causes an OPERATION COMPLETE interrupt. COMMAND REJECT is reset when OPERATION COMPLETE is reset.

NOT BUSY is similar to READY. NOT BUSY is true whenever the reader is ready to accept a READ command and is false when the CR is in the process of reading a card. However, NOT BUSY (unlike READY) does not go false when the CR executes a condition that requires manual intervention. NOT BUSY is not affected by the RESET STATUS command.

Four Control instruction codes have been assigned to the CR. These instructions perform the following operations:

- ENABLE INTERRUPT
- DISABLE INTERRUPT
- RESET STATUS
- READ

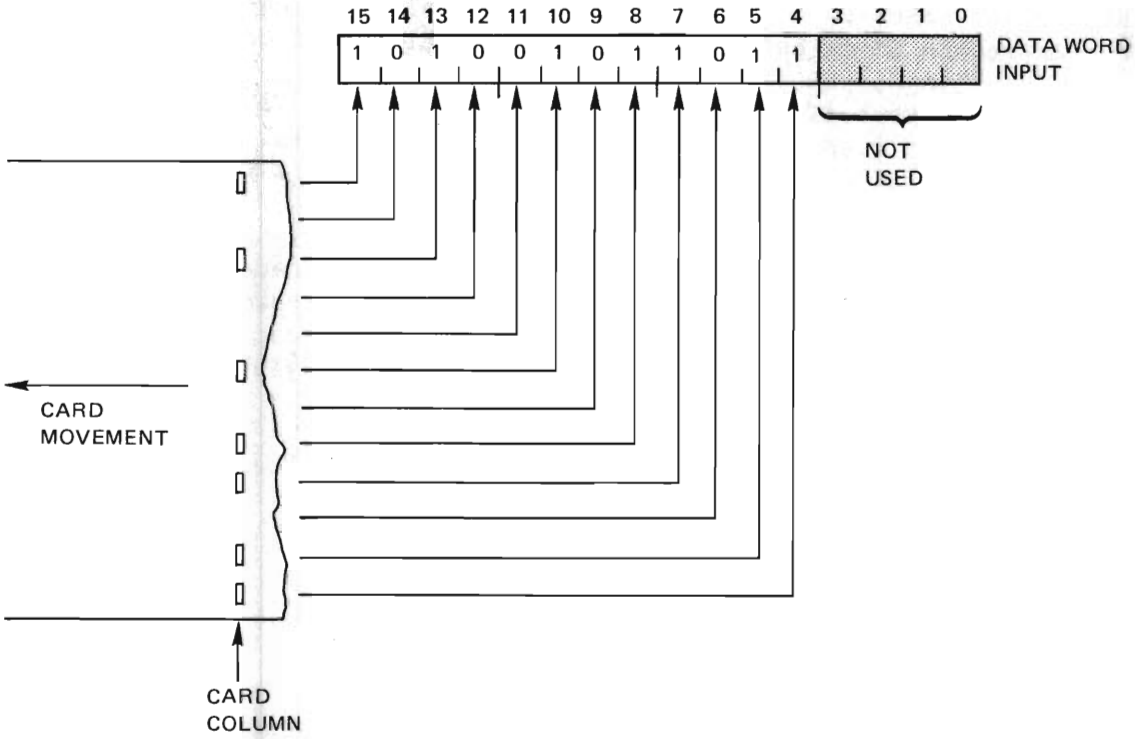
The ENABLE INTERRUPT command sets the interrupt mask on the controller to allow the CR controller to request an interrupt when OPERATION COMPLETE is true. The DISABLE INTERRUPT command resets the interrupt mask, preventing the CR controller from requesting an interrupt . The state of the mask remains in effect until one of these commands is given or the SYSTEMS RESET switch on the computer's console is depressed (which disables interrupts on all peripheral controllers).

The RESET STATUS command is used to clear OPERATION COMPLETE and/or COMMAND REJECT. The READ command has the same effect on these indicators.

The READ command causes the CR controller to initiate the reading of a card from the CR. Before issuing this command, the program must initialize the pair of memory locations dedicated to the CR's data channel. The first location, the Scan Control Register (SCR), specifies the number of card columns to be read; this is ≤ 80 since the CR controller always transfers 80 columns of data. The second location, the Channel Address Register (CAR), specifies the beginning memory address (minus one) for the data transfer. When the READ command is given, the CR drive mechanism starts a card moving through the unit. As each column moves under the read station it is read by the controller and transferred as a data word directly to the computer's memory. The data word read from the first card column is transferred to the memory

address specified by (CAR+1); data from sequential columns are transferred to sequential memory locations. Data transfer ceases when the specified number of data words have been transferred. OPERATION COMPLETE becomes true when the card has been passed completely through the reader.

The data word input for each card column has the following format:



4.5.2 Instruction Summary

The controller for the CR has been assigned a device select code of X'B'. The memory location dedicated to this controller's interrupt vector is X'4B'. The memory locations dedicated to the CR's SCR and CAR are X'26' and X'27', respectively.

Table 4-5 summarizes the XIO instructions that apply to the Card Reader. The instructions are given in CAP-16 Assembly Language format (see Section 3.9 for their machine language equivalents).

Table 4-5. Instruction Summary for the Card Reader

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)

4.5.3 Programming Example

The Card Reader operates in either a non-interrupting or an interrupting mode.

The example below illustrates a subroutine that reads 80 card columns into a specified data buffer and returns to the calling program when the operation is completed. The example uses non-interrupting mode.

```
        JSR     CARD    Calling sequence
        DC      80      Read 80 columns
        DC      BUFFER  Data buffer address
        (Return when operation completed)
        .
        .
CARD    SARS    *0      Save registers, etc.
        LDR     B,0
        RTR     C,B
        ADDV    C,9
        STR     C,0
        RTR     D,E     Set up D to fetch parameters
        RISE    E       Restore previous interrupt status
        LDR     A,0,,1  Fetch column count
        STR     A,X'26' Put in SCR location
        LDR     A,1,,1  Fetch buffer address
        DECR    A
        STR     A,X'27' Put in CAR location
TST     TEST    1,X'B'  Is CR READY?
        JMP     $ - 1   No - test again
        CTRL    3,X'B'  Yes - READ card
        TEST    3,X'B'  COMMAND REJECT?
```

```
        JMP     TST     Yes - retry operation
        TEST    0,X'B'  No - OPERATION COMPLETE?
        JMP     $ - 1   No - test again
        TEST    2,X'B'  Yes - ERROR?

        JMP     $+2     Yes - Call ERROR
        JMP     EXIT    No - finished
        JSR     ERROR   Call error routine (prints message
                         to operator)
        JMP     TST     Return from ERROR - retry operation
EXIT    INH             Exit sequence
        STR     B,0     Restore registers, etc.
        LARS    *0
        INCR    E       Increment return address
        INCR    E       Increment again
        RTRN    E       Return (to loc. following DC BUFFER)
```

4.6 LINE PRINTER

The Data Printer and Tally Printer are record oriented devices that print data output from the computer. The initiation of a line printer operation (i.e., a PRINT or PAPER FEED operation) is under program control. For the actual transfer of data words, however, the line printer (LP) controller employs a Data Channel (Direct Memory Access). The computer executes a Data Output instruction (DTOR or DTOM) outputting to the controller an object word from a register (DTOR) or from memory (DTOM); the object word specifies the PRINT or PAPER FEED operation. The controller supervises the transfer of data words from memory to the line printer. The LP controller can be interrogated by the program to determine if it requires service, or it can be allowed to request a program interrupt.

4.6.1 Operational Description

Six indicators on the LP controller reflect the status of the line printer:

- OPERATION COMPLETE
- READY
- BOTTOM OF FORM
- COMMAND REJECT
- TRANSFER COMPLETE
- BUSY

Six TEST instruction codes have been assigned to interrogate these status indicators.

OPERATION COMPLETE normally becomes true after a PRINT or PAPER FEED command has been given and the printing or paper feed operation is completed. If the AUTO LINE FEED bit of the object word (described below) is true, the OPERATION COMPLETE condition occurs after the linefeed is completed. This condition will also occur if a PRINT or PAPER FEED command is given to the LP controller while READY is false. OPERATION COMPLETE also becomes true when COMMAND REJECT is true. When OPERATION COMPLETE is

true, the controller requests a program interrupt if its interrupt mask is enabled. OPERATION COMPLETE is cleared by a RESET OPERATION COMPLETE command or by an acceptable PRINT or PAPER FEED command.

READY is true whenever the controller is ready to accept a PRINT or PAPER FEED command. READY is false when the controller has accepted a PRINT or PAPER FEED command but the operation is not yet complete (see OPERATION COMPLETE). READY is also false when a condition that requires manual intervention exists at the line printer (i.e., any condition that places the line printer in the stop mode - power off, out of paper, etc.) . READY is not affected by the RESET STATUS command.

A BOTTOM OF FORM condition occurs when a hole is detected in channel 2 of the line printer's VFU tape. It is reset when a hole is detected in channel 8 of the VFU tape (Top of Form).

COMMAND REJECT becomes true if a PRINT or PAPER FEED command is given to the controller while BUSY is true or READY is false. COMMAND REJECT causes an OPERATION COMPLETE interrupt. COMMAND REJECT is cleared by a RESET STATUS command or by an acceptable PRINT or PAPER FEED command.

A TRANSFER COMPLETE condition occurs only after a PRINT command when all the data to be printed has been transferred to the line printer; an OPERATION COMPLETE condition will not occur until the printing is actually completed. When TRANSFER COMPLETE is true, the controller requests a program interrupt if its interrupt mask is enabled; a second interrupt occurs when OPERATION COMPLETE goes true. TRANSFER COMPLETE is cleared only by a RESET STATUS command.

BUSY is true when the controller has accepted a PRINT or PAPER FEED command but the operation is not yet complete (see OPERATION COMPLETE).

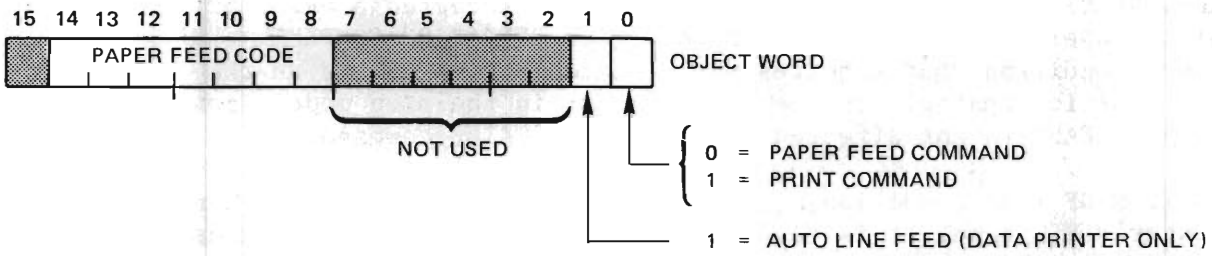
Four Control instruction codes have been assigned to the line printer:

- ENABLE INTERRUPT
- DISABLE INTERRUPT
- RESET STATUS
- RESET OPERATION COMPLETE

The ENABLE INTERRUPT command allows the controller to request an interrupt when OPERATION COMPLETE or TRANSFER COMPLETE are true. The DISABLE INTERRUPT command prevents the controller from requesting an interrupt. The state of the interrupt mask remains in effect until one of these commands is given or the SYSTEMS RESET switch on the computer console is depressed (which disables interrupts on all peripheral controllers).

The RESET STATUS command is used to clear COMMAND REJECT and/or TRANSFER COMPLETE. The RESET OPERATION COMPLETE command is used to clear OPERATION COMPLETE.

The object word, transferred from the computer to the controller as the result of a Data Output instruction, can specify a PRINT or PAPER FEED command. A typical operation would use a series of alternate PRINT and PAPER FEED commands. The format of the object word is as follows:



The AUTO LINE FEED bit is useful when using the Data Printer if each PAPER FEED command is to be a simple single line feed. For this list type of printing, the PAPER FEED commands can be eliminated by setting the AUTO LINE FEED bit, which will cause an automatic line feed after each PRINT command. The Tally Printer will automatically line feed after each PRINT command whether the AUTO LINE FEED bit is used or not. The AUTO LINE FEED, however, does not affect the performance of a PAPER FEED command which is always performed according to the paper feed code. Table 4-6 lists the various paper feed codes that can be programmed into bits 8-14 of the object word.

Table 4-6. Paper Feed Codes for Data Printer and Tally Printer

Bit: 14	13	12	11	10	9	8	Action	
0	0	0	0	0	0	0	NO LINE FEED	
0	0	0	0	0	0	1	SKIP 1 LINE	DATA PRINTER ONLY
		THRU					THRU	
0	1	1	1	1	1	1	SKIP 63 LINES	
1	X	X	0	0	0	X	SKIP TO CHANNEL 1 of VFU Tape	
1	X	X	0	0	1	0	SKIP TO CHANNEL 2 of VFU Tape	
1	X	X	0	0	1	1	SKIP TO CHANNEL 3 of VFU Tape	
1	X	X	0	1	0	0	SKIP TO CHANNEL 4 of VFU Tape	
1	X	X	0	1	0	1	SKIP TO CHANNEL 5 of VFU Tape	
1	X	X	0	1	1	0	SKIP TO CHANNEL 6 of VFU Tape	
1	X	X	0	1	1	1	SKIP TO CHANNEL 7 of VFU Tape	
1	X	X	1	0	0	0	SKIP TO CHANNEL 8 of VFU Tape	
1	X	X	1	0	0	1	SKIP 1 LINE	
1	X	X	1	0	1	0	SKIP 1 LINE	
1	X	X	1	0	1	1	SKIP 1 LINE	
1	X	X	1	1	0	0	SKIP 1 LINE	
1	X	X	1	1	0	1	SKIP 1 LINE	
1	X	X	1	1	1	0	SKIP 1 LINE	
1	X	X	1	1	1	1	SKIP 1 LINE	

Before using a PRINT command, the program must initialize the pair of memory locations dedicated to the line printer controller's data channel. The first location, the Scan Control Registers (SCR), specifies the number of data words to be read from memory. The second location, the Channel Address Register (CAR), specifies the beginning memory address (minus one) for the data transfer. When the PRINT command is issued, the data word from the memory location addressed by the contents of the CAR+1 is read and transferred to the line printer. Subsequent data words are read from sequential memory locations and transferred to the line printer until the number of data words specified by the SCR have been transferred. TRANSFER COMPLETE then becomes true. When any ongoing paper motion has been completed, OPERATION COMPLETE becomes true.

Each data word includes two characters as shown below:



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 0-5) of the last data word must be USASCII "blank" code (i.e., X'20'). The character set is USASCII, 64 characters. Bits 13 and 5 are the most significant bits.

4.6.2 Instruction Summary

The controller for the line printer has been assigned a device select code of X'C'. The memory locations dedicated to the line printer's SCR and CAR are X'2E' and X'2F', respectively.

Table 4-7 summarizes the XIO instructions that apply to the line printer. The instructions are given in CAP-16 Assembly Language format (see Section 3.9 for their machine language equivalents).

Table 4-7. Instruction Summary for the Line Printer

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

4.6.3 Programming Example

The line printer will operate in either a non-interrupting or an interrupting mode. When the line printer is allowed to interrupt, the output routine normally consists of two parts: an initialization routine that sets up the interrupt vector and transfer parameters (word count, buffer address and/or object word) and initiates the line printer operation; and an interrupt service routine that responds to the interrupt, then returns control to the interrupted program.

A typical line printer output routine is illustrated below. The routine can service either PRINT or PAPER FEED commands. In addition to the "jump to subroutine" statement, the calling sequence for this subroutine must always consist of a three entry data table. The first entry must specify the character count (i.e., the number of characters to be printed in a line) if a PRINT operation (with or without auto line feed) is to be performed, or must specify the paper feed parameter (i.e., the number of lines to be skipped or a VFU tape channel - see discussion below) if a PAPER FEED operation is to be performed. The second entry must specify the buffer address for a PRINT operation; a buffer address, however, is not required for PAPER FEED (e.g., the operator could replace DC BUFAD with DS 1, for a PAPER FEED operation). The third entry must specify the operation to be performed as follows:

Operation Index	Operation
0	PRINT without auto line feed
1	PRINT with auto line feed
2	PAPER FEED - skip X lines*
3	PAPER FEED - skip in channel X of the VFU tape*

* Where X is the paper feed parameter that must be specified in the first entry of the data table.

The calling sequence in this example asks for the printing of one line (132 characters) followed by an automatic line feed. After the transfer and printing of data are initiated, control is returned to the calling program. The example subroutine defines a storage location (LP0) that the programmer can interrogate to determine if the transfer has been completed, the operation has been completed or an error has occurred. For simplicity, the only action that this subroutine takes upon detecting an error is to set an error bit. A programmer might want an error sequence that initiates additional action; for example, a sequence that causes a message to be issued (e.g., at the teletype) stating the nature of the error.

```
        JSR     LPRNT     CALLING SEQUENCE
        DC      132       Character count (or paper feed parameter)
        DC      BUFAD     Buffer address
        DC      1         Operation Index
        (Return when initialization is completed)
        .
        .
        .
LP0     DS      1         { Bit 15 will contain operation complete indicator;
                          { Bit 14 will contain transfer complete indicator;
                          { Bit 13 will contain error indicator.
LPRNT   SARS    *0        } INITIALIZATION ROUTINE
        LDR     B,0       }
        RTR     C,B       } Save registers, etc.
        ADDV    C,9       }
        STR     C,0       }
        RTR     D,E       Set up to fetch parameters
        INE               Enable interrupts
        LDR     X,2,,1    Load operation index
        ANDV    X,X'0003' Mask all bits except operation indices
        LDV     A,ILPNT
        STR     A,X'4C'   Set up interrupt vector
BZY     TEST    5,X'C'    Is controller BUSY?
        JMP     BZY       Yes - try again
        TEST    1,X'C'    Is line printer READY?
        JMP     ERROR     NO - jump to error routine
        SBIT    7,LP0     Set operation not complete
        SBIT    6,LP0     Set transfer not complete
        CTRL    0,X'C'    Enable LP interrupt
        LDR     A,OPTAB,X
        EXIT    A         Go to operation sequence
```

```
OPTAB   DC      PRINT     Code = 0 (PRINT without auto line feed)
        DC      PRTWA            1 (PRINT with auto line feed)
        DC      PFLINE           2 (PAPER FEED skip X lines)
        DC      PFVFU            3 (PAPER FEED skip to ch. X of VFU tape)
PRINT   LDV     C,1       Load object word
SETUP   LDR     A,0,,1    Convert character count into Word count
        INCR    A
        SRA     A,1
        STR     A,X'2E'   Store in SCR location
        LDR     A,1,,1    Fetch buffer address
        DECR    A
        STR     A,X'2F'   Store in CAR location
        DTOR    C,X'C'    Output command
        JMP     RETURN    Jump to return sequence
PRTWA   LDV     C,3       Load object word
        JMP     SETUP     Jump to parameter setup
PFLINE  LDR     C,0,,1
        EXBY    C         Set up object word
        DTOR    C,X'C'    Output command
        JMP     RETURN    Jump to return sequence
PFVFU   LDR     C,0,,1
        ORV     C,X'0040' Set up object word
        EXBY    C
        DTOR    C,X'C'    Output command
RETURN  STR     B,0
        LARS    *0        Restore register and
        ADDV    E,3        return to calling program
        RTRN    E
ERROR   SBIT    5,LP0     Set error bit
        JMP     RETURN    Jump to return sequence

ILPNT   SARS    *0        INTERRUPT SERVICE ROUTINE
        LDR     B,0
        RTR     C,B       Save registers, etc.
        ADDV    C,9
        STR     C,0
        CTRL    1,X'C'    Disable LP interrupt
        INE               Enable interrupts
        TEST    4,X'C'    Transfer Complete?
        JMP     OPCOM     NO - Jump to OPCOM
        RBIT    6,LP0     YES - Set transfer complete
        CTRL    2,X'C'    RESET STATUS (Transfer Complete) bit
        JMP     EXIT      Jump to exit sequence

OPCOM   RBIT    7,LP0     Set operation complete
        CTRL    3,X'C'    RESET OPERATION COMPLETE status bit
        TEST    3,X'C'    Command Reject?
        JMP     REJECT    YES - Jump to error routine
EXIT    INH               Disable master interrupt
        CTRL    0,X'C'    Enable LP interrupt
        STR     B,0
        LARS    *0        Restore registers
        RTRN    E         Return to interrupted program
REJECT  SBIT    5,LP0     Set error bit
        CTRL    2,X'C'    RESET STATUS (Command Reject) bit
        JMP     EXIT      Jump to exit sequence
```

4.7 3341/3343 DISK STORAGE SYSTEM

The 3341/3343 Disk Storage System allows the high speed transfer of data between the SPC-16 and one of four possible disk cartridge drives that can be connected to the disk controller. Two different General Automation disk cartridge drive models may be used: 3341 or 3343.

The 3341 drive has ten recording surfaces; the 3343 drive has twenty recording surfaces. Each recording surface has 203 data tracks and twenty sectors per surface. Additionally, the 3343 is available in an optional sixteen-sector configuration. For 3341 subsystems, each track in a sector has a capacity of 80 sixteen-bit words. For 3343 subsystems, each track in a sector has a capacity of 160 sixteen-bit words (20-sector configuration) or 200 sixteen-bit words (16-sector configuration).

Three types of instructions are used to communicate with the disk controller. A Data Output instruction is used to send an "object word" to the controller, either from memory or from a register. The object word may be one of seven commands to the controller:

- SEEK
- WRITE
- WRITE 1L
- READ
- READ 1L
- RESTORE
- SELECT

A Data Input instruction is sent to the controller when the program wants to interrogate the status of the unit. The controller responds to the Data Input instruction by sending a "status word" to the computer; the status word contains information such as "busy", "operation complete", "error", etc.

Control instructions are sent to perform three functions:

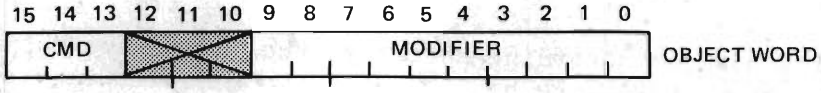
- ENABLE INTERRUPTS
- DISABLE INTERRUPTS
- ABORT (Reset)

The initiation of a data transfer operation is under program control. The program must load the devices dedicated word count (SCR) and channel address (CAR) memory locations in preparation for a DMA transfer, must prepare the proper unit to send/accept the data by executing a series of Data Output and Data Input instructions, and must initiate the transfer (via a READ, READ 1L, WRITE or WRITE 1L command).

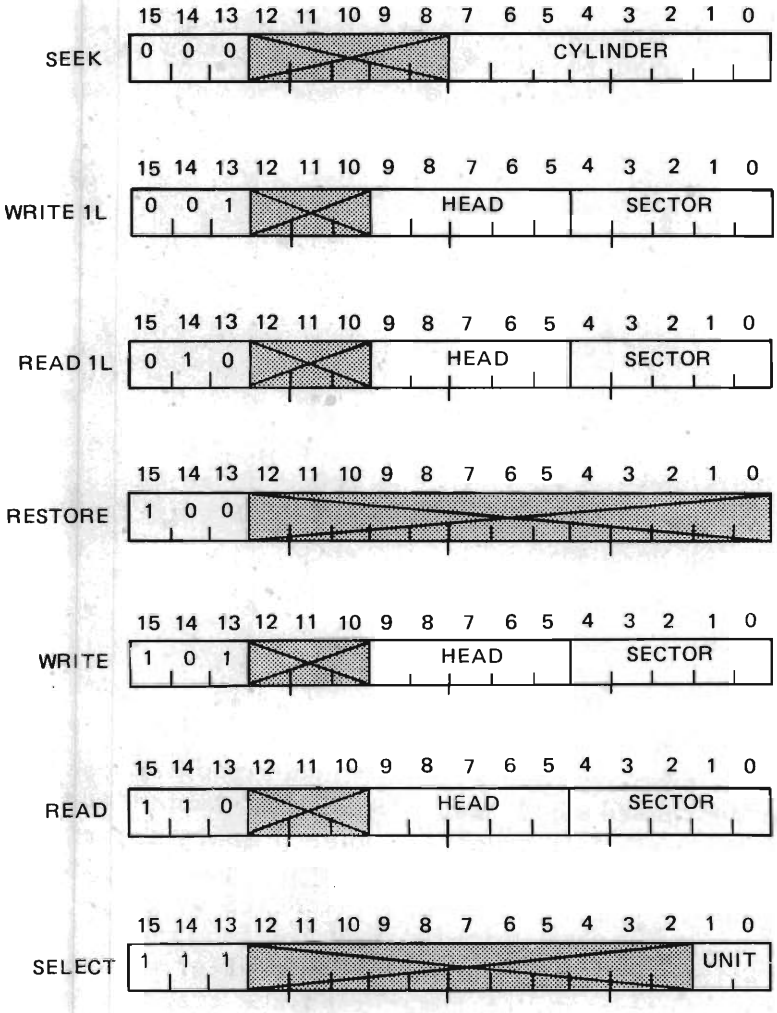
For the actual transfer of data words the disk controller employs a Data Channel (Direct Memory Access); the controller supervises the transfer of each data word between the computer memory and the selected disk. The Disk System can be interrogated by the program to determine if it requires service, or it can be allowed to request a program interrupt when it requires service.

4.7.1 Operational Description

One Data Output instruction is used with the Disk Storage System. This instruction can cause one of seven 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 as follows:

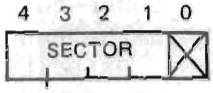


The 3-bit CMD field (bits 13-15) determines the interpretation of the modifier:



(CMD = 011 is undefined and is rejected.)

If the 3343 sixteen-sector optional configuration is used, the SECTOR fields above are formatted as:



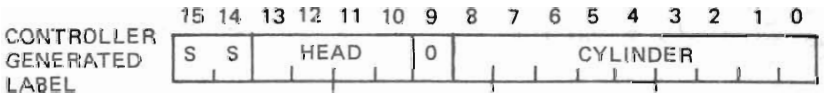
where bit 0 is not used (i.e., ignored).

Execution of the SEEK command causes the selected disk unit to position its heads to the track address specified in the 8-bit Cylinder field. The controller becomes BUSY upon execution of the Data Output command and remains BUSY for approximately 5 μsec. During this period, the controller transfers the 8-bit Cylinder field to the selected disk unit. During its head positioning time, the disk unit exhibits a NOT READY status. The controller is free to perform operations with other disk units which may be selected when the controller is not BUSY. SEEK COMPLETE status interrupt occurs a nominal 70 msec. later, when the disk unit has completed its head positioning. A SEEK command issued to a NOT READY disk unit causes SEEK ERROR and COMMAND REJECT status.

The WRITE 1L and WRITE commands are used to transfer core memory data to the sector specified in the Head and Sector fields at the cylinder address and disk unit current when the Data Output command is issued. WRITE 1L differs from WRITE in the handling of the first word (the label word) of the first sector of the transfer. WRITE 1L writes the first word of the data table as the label of the first sector of the transfer; WRITE labels the first sector with a controller-generated word. Otherwise, execution of the two commands is identical.

The controller becomes BUSY upon execution of the Data Output command and remains BUSY until the WRITE operation is completed or rejected. The controller searches for the specified sector, then writes the number of data words specified by the word count. The tranfer begins at the specified sector and proceeds sequentially, sector by sector and head by head, until the word counter is satisfied or until the head register is incremented to select a non-existent head (End of Cylinder). The first word of the first sector, the label, is written from core for WRITE 1L or is controller-generated for WRITE; all subsequent labels are controller-generated. The program any specify word counts from 1 word to End of Cylinder. With word counts that are not multiples of sector capacity, the controller fills the remaining word slots with zeroes. Zero word transfers are illegal. These, and word counts that exceed cylinder capacity, terminate at End of Cylinder with DATA ERROR status and COMMAND REJECT status/interrupt. OPERATION COMPLETE status/interrupt does not occur. The WRITE 1L and WRITE operations are also rejected by the controller if the disk unit is NOT READY, resulting in DATA ERROR and COMMAND REJECT, and no OPERATION COMPLETE, in approximately 5 μsec.

Controller-generated labels contain the Cylinder Address, Head Address, and the two least significant bits of the Sector Address of the Sector in which the label occurs:



The READ 1L and READ commands are used to transfer disk data to core memory. The transfer begins with the sector specified in the Head and Sector fields. The disk unit and cylinder address used are those current when the Data Output command is issued. READ 1L differs from READ in the handling of the first word (the label word) of the first sector of the transfer. READ 1L treats the label of the first sector of the transfer as data and reads this label into the data table in core; READ edits the labels from disk data records before transferring them to core. Subsequent labels of multiple sector transfers are edited from records by either READ or READ 1L. The controller becomes BUSY upon execution of the Data Output command and remains BUSY until the READ operation is completed or rejected. The controller searches for the specified sector, then transfers the number of words specified by the word count to core. The program may specify word counts from one word to cylinder capacity. The transfer proceeds sequentially across sector boundaries and across head boundaries until the word count is complete or to End of Cylinder (i.e., increments to non-existent head). Zero word transfers are illegal, causing the controller to read to End of Cylinder. Read command issued to a NOT READY disk unit, or attempts to read past End of Cylinder, result in DATA ERROR status and COMMAND REJECT status/interrupt. OPERATION COMPLETE status/interrupt does not occur.

The 3341/3343 controller is capable of data chaining. Chaining permits non-adjacent data tables in core to be linked together to form a continuous data stream. (Chaining is described in Section 5.2.)

The RESTORE operation causes the selected disk unit to position its heads over Track 0. RESTORE initializes the position logic within the disk unit and should be used instead of "SEEK 0" in initialization or recovery routines. The controller becomes BUSY upon execution of the Data Output command and remains BUSY for approximately 5 μsec while transferring the RESTORE command to the disk unit. SEEK COMPLETE status/interrupt occurs when the disk has completed its positioning operation. RESTORE is rejected if sent to a NOT READY unit; rejection causes SEEK ERROR and COMMAND REJECT status/Interrupt.

The SELECT command is used to load the controller's 2-bit unit register, which designates the source/destination of status/commands. The NOT READY and SEEK ERROR status bits are gated by the unit bits; the Next Sector status bits are derived from sector pulses gated by the unit bits. The controller does not become BUSY in response to SELECT commands and status information is available immediately (i.e., with the next instruction). SELECT commands are ignored by the controller while it is BUSY.

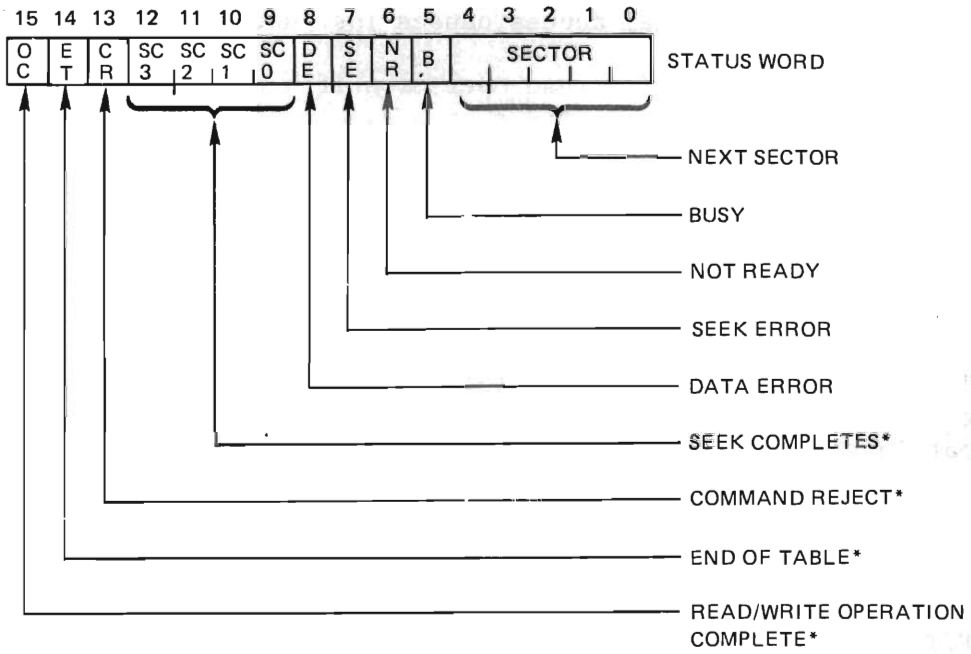
In response to the undefined CMD 011 code, the controller becomes BUSY for approximately 5 μsec and then terminates with COMMAND REJECT status/interrupt.

Eight indicators on the disk controller reflect the status of the Disk Storage System:

- BUSY
- NOT READY
- SEEK ERROR
- DATA ERROR
- SEEK COMPLETE
- COMMAND REJECT
- END OF TABLE
- READ/WRITE OPERATION COMPLETE

These indicators occupy bit positions in a status word. The program interrogates the bits in the status word via a Data Input status access assigned to the Disk Storage System.

The format of the status word is shown below:



* = INTERRUPT CONDITIONS

The Next Sector to become available for reading or writing is reflected in bits 0-4. These bits are derived from the currently selected disk cartridge and are updated as the disk spins. Updating of the Next Sector counter occurs asynchronous to the computer. It is therefore possible to obtain an erroneous value if the program samples the Next Sector counter while it is incrementing. If a system has stringent timing requirements based on the Next Sector bits, it should read these bits at least twice and compare the values to avoid sampling while the counter is changing.

BUSY (bit 5) is true when the controller is performing an input/output operation. While BUSY is true, therefore, the controller ignores all output commands addressed to it (i.e., it treats them as "no-ops").

NOT READY (bit 6) is true when the selected disk drive unit is off-line or is busy seeking a cylinder. NOT READY is also true when the controller is BUSY.

SEEK ERROR (bit 7) is set true when a SEEK or RESTORE command is issued to a NOT READY unit, when a SEEK command specifies a cylinder address greater than 202_{10}, or by a positioning malfunction within the disk unit. SEEK ERROR due to disk NOT READY or illegal cylinder also sets the COMMAND REJECT status/interrupt. This SEEK ERROR is reset by a Data Input status access command or when the controller becomes BUSY in response to a new Data Output command. SEEK ERROR due to positioning malfunction is reset only when a RESTORE command is issued to the unit.

DATA ERROR (bit 8) is set true by the following conditions:

a. At the termination of a read or write command if the word count specified is equal to zero or greater than the cylinder capacity.

b. If a slow data channel access causes lost data (data overrun).

c. At the termination of a read operation if a parity error is detected or if the controller does not detect a sync pattern (uninitialized cartridge).

d. If a read or write is attempted to a NOT READY unit. (This condition also sets COMMAND REJECT.)

DATA ERROR is reset by a Data Input status access or when the controller becomes BUSY in response to a Data Output command.

SEEK COMPLETES (bits 9-12) reflect the status of each disk unit attached to the controller. The SEEK COMPLETE status/interrupt indicator for each disk drive unit is set true when that unit has completed a positioning operation. Such positioning may be in response to a SEEK or RESTORE command or after an initial load sequence (power on or disk cartridge change). All SEEK COMPLETE bits are reset by a Data Input status access, by depressing the SYSTEMS RESET switch on the computer's console, or by issuing the ABORT Control function.

COMMAND REJECT (bit 13) status/interrupt, when set, indicates abnormal termination of an operation by the controller. COMMAND REJECT will be set at termination of any operation attempted with a NOT READY storage unit or at termination of an attempt to execute an undefined command code. At termination of a read or write command, COMMAND REJECT is accompanied by DATA ERROR status; at termination of a SEEK or RESTORE command, it is accompanied by SEEK ERROR status. COMMAND REJECT is reset by a Data Input status access, by depressing the SYSTEMS RESET switch on the computer's console, by issuing the ABORT Control function, or when the controller becomes BUSY in response to a Data Out command.

END OF TABLE (bit 14) status/interrupt occurs during a read or write operation when the final word has been transferred to or from the data table in core if bit 14 in the word count location (SCR) is zero. END OF TABLE is reset by a Data Input status access, by depressing the SYSTEMS RESET switch on the computer's console or by issuing the ABORT Control function.

READ/WRITE OPERATION COMPLETE (bit 15) status/interrupt occurs when the controller terminates an accepted read/write operation. READ/WRITE OPERATION COMPLETE is reset by a Data Input status access, by depressing the SYSTEMS RESET switch, by issuing the ABORT Control function, or when the controller becomes BUSY in response to a Data Out command.

Three Control instruction codes have been assigned to the Disk Storage System. These commands perform the following functions:

- ENABLE INTERRUPT
- DISABLE INTERRUPT
- ABORT

The ENABLE INTERRUPT command sets the interrupt mask on the controller to allow the controller to request an interrupt when SEEK COMPLETE, COMMAND REJECT, END OF TABLE or READ/WRITE OPERATION COMPLETE is true. The DISABLE INTERRUPT command resets the interrupt mask, preventing the disk controller from requesting an interrupt. The state of the mask remains in effect until one of these commands is given or the SYSTEMS RESET switch on the computer's console is depressed (which disables interrupts on all peripheral controllers).

The ABORT function performs the same function as the SYSTEMS RESET switch on the computer's console and forces the controller to an initialized condition: all resettable status bits are reset, the controller is set NOT BUSY, and all disk units are deselected. (The Interrupt Mask is not affected by the ABORT FUNCTION). The ABORT function is intended for use during hardware debugging and should not be used otherwise; its use during a WRITE operation may cause the loss of one sector of data.

4.7.2 Instruction Summary

The controller for the Disk Storage System has been assigned a device select code of X'E'. The memory location dedicated to this device's interrupt vector is X'4E'. The memory locations dedicated to the Disk Storage System's SCR and CAR are X'22' and X'23', respectively.

Table 4-8 summarizes the XIO instructions that apply to the Disk Storage System. The instructions are given in CAP-16 Assembly Language format (see Section 3.9 for their machine language equivalents).

Table 4-8. Instruction Summary for the Disk Storage System

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' or DTOM R,X'E'	OUTPUT object word from register (R) or from memory to disk controller.
DTIR R,X'E' or DTIM R,X'E'	INPUT status word to register (R) or to memory from disk controller.

R = A,X,Y,Z,B,C,D or E

4.7.3 Programming Example

The Disk System operates in either a non-interrupting or an interrupting mode.

The example below illustrates a subroutine that can be called to perform any of five operations:

- WRITE 1L
- READ 1L
- RESTORE
- WRITE
- READ

The example uses interrupting mode and is written for a 3341 subsystem. The calling sequence is as follows:

1. Call the subroutine (JSR DISK)

2. Define word count for Read or Write operations; for RESTORE, code DS 1.

3. Define buffer address for Read or Write operations; for RESTORE, code DS 1.

4. Select drive and operation by coding BYTE (drive index), (operation index).

   The indices for drive and operation selection are given below:

Drive Index	Drive
0	0
1	1
2	2
3	3

Operation Index	Operation
0	WRITE 1L
1	READ 1L
2	RESTORE
3	WRITE
4	READ

   SEEK and SELECT commands are generated by the subroutine.

5. Define sector number for all operations except RESTORE; for RESTORE, code DS 1.

   For 3341 subsystems, sector numbers range from 0 to 40599_{10}; for 3343 subsystems sector numbers range from 0 to 81199_{10}. Large sector numbers can be handled by using double word integers or by revising the sector numbering scheme to include two or more actual disk sectors as one user sector (e.g., a mod_4 version for the 3343 disks, 20-sector configuration, would yield user sector numbers in the range 0 to 20299_{10}, with each "sector" containing 640 words).

The subroutine consists of two parts:

- An initialization routine (DISK)

- An interrupt service routine (IDISK)

The initialization routine sets up the interrupt vector, loads the word count and buffer address in the dedicated SCR and CAR locations, decodes the drive and operation selection codes, decodes the sector number (into head track, and surface sector), and initiates the operation. The interrupt service routine is invoked each time a disk interrupt occurs. The interrupt service routine checks for errors and determines whether the operation is complete, then returns control to the calling program. In read or write operations, the interrupt service routine initiates the actual data transfer following detection of "seek complete".

In addition, the subroutine defines and maintains a memory word (STAT) that the programmer must interrogate in order to check whether a previously-initiated operation has been completed or has encountered an error. Bit 15 of this word is reset when the operation is complete; bit 14 is set when an error occurs.

The example below reads 80 words from Sector 403 of Drive 2 into 80 sequential memory locations beginning at location DBUFFR. A twenty-sector format is assumed.

```
        JSR     DISK        Calling sequence
        DC      80          Word count
        DC      DBUFFR      Data buffer address
        BYTE    2,4         Drive, operation indices
        DC      403         Sector number
        (Return on op code error or SEEK/RESTORE initiated)
        .
        .
STAT    DS      1           Bit 15=op comp, bit 14=error
SAVET   DS      1           Will contain cylinder number
SAVED   DS      1           Will contain drive number
SAVEC   DS      1           Will contain operation command
        .
        .
DISK    SARS    *0          Entry sequence
        LDR     B,0
        RTR     C,B
        ADDV    C,9
        STR     C,0
        RTR     D,E         Set up D to fetch parameters
        INE
        LDR     X,2,,1      Fetch drive, operation indices
        RTR     Y,X
        ANDV    Y,X'7'      Select operation bits
        SUBVC   Y,5
        SKP     ERXIT       Branch if illegal op code
        EXBY    X           Put drive index in lower byte
        ANDV    X,X'3'      Select drive bits
        STR     X,SAVED     Save drive number
DISK1   DTIR    A,X'E'      Input STATUS word
        ANDVC   A,X'0020'   Check BUSY bit
        SKN     DISK1       If BUSY, try again
        RTR     A,X         Get drive number
        ORV     A,X'E000'   Assemble SELECT command
```

```
        DTOR    A,X'E'          Output SELECT command
DISK2   DTIR    A,X'E'          Input STATUS word
        ANDVC   A,X'0040'       Check NOT READY bit
        SKN     DISK2           Try again if NOT READY
        LDV     A,IDISK         Set up interrupt vector
        STR     A,X'4E'
        CTRL    0,X'E'          Enable interrupts from disk
        SUBVC   Y,2
        SKN     DISK3           Branch if not RESTORE
        LDV     A,X'8000'       Load RESTORE command
        SBIT    5,STAT          Set RESTORE flag
        SBIT    7,STAT          Set operation going bit
        DTOR    A,X'E'          Output RESTORE command
        JMP     EXIT
DISK3   LDR     A,0,,1          Fetch word count
        STR     A,X'22'         Store word count in SCR location
        LDR     A,1,,1          Fetch data buffer address
        DECR    A
        STR     A,X'23'         Store address-1 in CAR location
        ZERO    Z               Initialize cylinder counter
        LDR     A,3,,1          Fetch user sector number
        JMP     DISK6           Jump to loop
DISK5   INCR    Z               Decode sector number into cylinder
DISK6   SUBV    A,200            number (0-202)
        SKS     DISK5
        STR     Z,SAVET         Store cylinder no. (SEEK command)
        ADDV    A,200           Restore last positive number (0-199)
        ZERO    C
        JMP     DISK8
DISK7   INCR    C               Decode "head*sector" into head (0-9)
DISK8   SUBV    A,20             and sector (0-19)
        SKS     DISK7
        SRC     C,11            Shift head number left 5 bits
        ADDV    A,20            Restore last positive number (0-19)
        LDR     Z,OPTABL,Y      Fetch op code command
        OR      Z,A             Add sector number
        OR      Z,C             Add head number
        STR     Z,SAVEC         Save command at SAVEC
        LDR     A,SAVET         Load SEEK command
        SBIT    7,STAT          Set operation going bit
        DTOR    A,X'E'          Output SEEK command
        JMP     EXIT
OPTABL  DC      X'1000'         Code = 0 (WRITE 1L)
        DC      X'2000'              = 1 (READ 1L)
        DS      1                    = 2 (RESTORE)
        DC      X'5000'              = 3 (WRITE)
        DC      X'6000'              = 4 (READ)
ERXIT   SBIT    6,STAT          Set error bit
        RBIT    7,STAT          Reset for operation complete
```

```
EXIT    INH                         Exit sequence
        STR       B,0
        LARS      *0
        ADDV      E,5               Increment past data table
        RTRN      E
IDISK   SARS      *0                Entry sequence
        LDR       B,0
        RTR       C,B
        ADDV      C,9
        STR       C,0
        INE
        RBIT      6,STAT            Assume no error
        DTIR      A,X'E'            Input STATUS word
        ANDVC     A,X'2180'         Check error condition bits
        SKN       ERROR             Branch if any error
        LDR       X,SAVED           Fetch drive number
        LDR       C,SCTABL,X        Select SEEK COMPLETE for proper drive
        ANDC      A,C
        SKZ       OPCOM             Branch if interrupt was not SEEK COMPLETE
        RBIT      5,STAT
        SKN       OPCOM             Branch if from RESTORE
        LDR       A,SAVEC           Load saved Read/Write command
        XEC       A                  and execute it
        JMP       CONT
SCTABL  DC        X'0200'           Seek Complete Drive 0
        DC        X'0400'           Seek Complete Drive 1
        DC        X'0800'           Seek Complete Drive 2
        DC        X'1000'           Seek Complete Drive 3
ERROR   SBIT      6,STAT            Error - set error bit
OPCOM   RBIT      7,STAT            Indicate operation complete
CONT    INH                         Exit sequence
        STR       B,0
        LARS      *0
        RTRN      E
```

4.8 3346/3347 DISK STORAGE SYSTEM

The 3346/3347 Disk Storage System allows the high speed transfer of data between the SPC-16 and one of four possible disk cartridge drives that can be connected to the disk controller. Two different General Automation disk cartridge drive models may be used 3346 or 3347.

The 3346 drive is a dual-disk model; one disk is permanently fixed within the drive, the other is removable. The 3347 drive is a single removable disk model. Each disk has two oxide-coated recording surfaces with 204 data tracks and eight sectors per surface. Each sector in a track has a capacity of 401 sixteen-bit words.

Three types of instructions are used to communicate with the disk controller. A Data Output instruction is used to send an "object word" to the controller either from memory or from a register. The object word may be one of six commands to the controller:

- SEEK
- WRITE
- READ
- READ CHECK
- RESTORE
- SELECT

A Data Input instruction is sent to the controller when the program wants to interrogate the status of the unit. The controller responds to the Data Input instruction by sending a "status word" to the computer; the status word contains information such as "busy", "operation complete", "error", etc. Control instructions are sent to perform three functions:

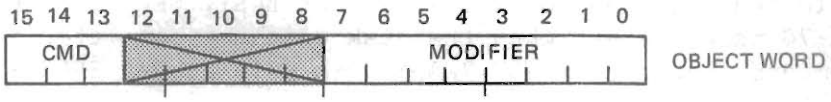
- ENABLE INTERRUPTS
- DISABLE INTERRUPTS
- ABORT (Reset)

The initiation of a data transfer operation is under program control. The program must load the disk's dedicated word count (SCR) and channel address (CAR) memory locations in preparation for a DMA transfer, must prepare the proper unit to send/accept the data by executing a series of Data Output and Data Input instructions, and must initiate the transfer (via a READ, READ CHECK or WRITE command).

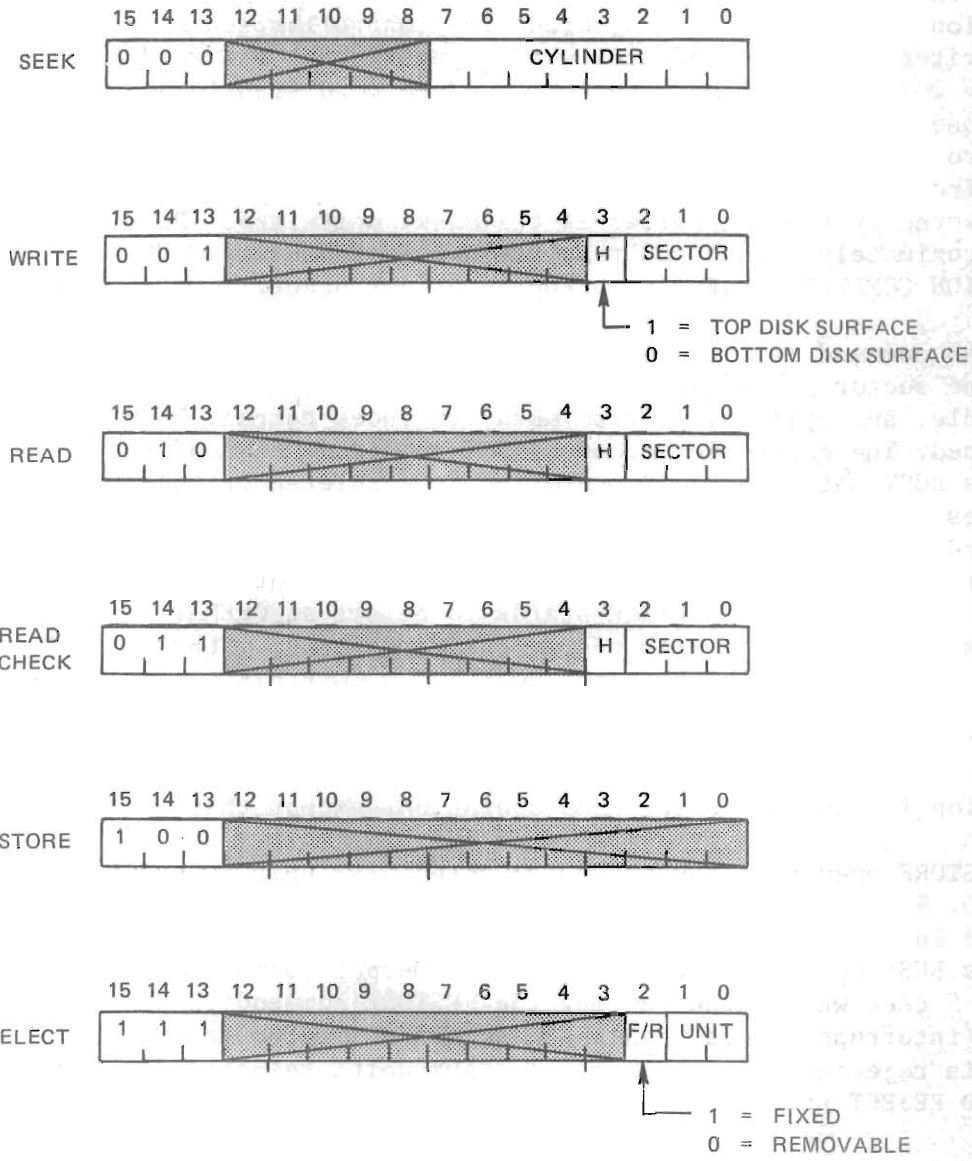
For the actual transfer of data words the disk controller employs a Data Channel (Direct Memory Access); the controller supervises the transfer of each data word between the computer memory and the selected disk. The Disk System can be interrogated by the program to determine if it requires service, or it can be allowed to request a program interrupt when it requires service.

4.8.1 Operational Description

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 as follows:



The 3-bit CMD field (bits 13-15) determines the interpretation of the modifier:



(CMD = 101 and 110 are undefined and are rejected)

Execution of the SEEK command causes the selected disk unit to position its heads to the track address specified in the 8-bit Cylinder field. The controller becomes BUSY upon execution of the Data Output command and remains BUSY for approximately 5 μsec. During this period, the controller transfers the 8-bit Cylinder field to the selected disk unit. During its head positioning time, the disk unit exhibits a NOT READY status. The controller is free to perform operations with other disk units which may be selected when the controller is not BUSY. SEEK COMPLETE status interrupt occurs a nominal 70 msec. later, when the disk unit has completed its head positioning. A SEEK command issued to a NOT READY disk unit causes SEEK ERROR and COMMAND REJECT status.

The WRITE command is used to transfer core memory data to the sector specified in the Head and Sector fields at the cylinder address, disk unit, and fixed/removable cartridge current when the Data Output command is issued. The controller becomes BUSY upon execution of the Data Output command and remains BUSY until the WRITE operation is completed or rejected. The controller searches for the specified sector, then writes the number of data words specified by the word count. The program may specify word counts from 1 to 401_{10}. (Zero word transfers and word counts greater than 401 cause DATA ERROR status at OPERATION COMPLETE.) In WRITE operations specifying non-zero word counts smaller than 401, the controller writes the specified number of words from core and fills the remainder of the sector with zeroes. The WRITE operation is rejected by the controller if the disk unit is NOT READY. Termination can occur in approximately 5 μsec and results in DATA ERROR and COMMAND REJECT status/interrupt. OPERATION COMPLETE status/interrupt does not occur.

The READ command is used to transfer disk data to core memory. The transfer occurs from the sector specified in the Head and Sector fields. The disk unit, fixed/removable, and cylinder address used are those current when the Data Output command is issued. The controller becomes BUSY upon execution of the Data Output command and remains BUSY until the READ operation is completed or rejected. The controller searches for the specified sector, then transfers the number of words specified by the word count to core. The program may specify word counts from 1 to 401_{10}. (Zero word transfers and word counts greater than 401 both cause 401 words to be transferred to core and DATA ERROR status to be set at OPERATION COMPLETE.) If the disk unit is NOT READY, the READ operation is rejected, with DATA ERROR and COMMAND REJECT status/interrupt bits set. OPERATION COMPLETE status/interrupt does not occur.

The READ CHECK command causes 401 words of the specified sector to be transferred from the disk through the controller's validity-checking logic. The READ CHECK operation is similar to the READ operation, except that no data is transferred to core.

The RESTORE operation causes the selected disk unit to position its heads over Track 0. RESTORE initializes the positioning logic within the disk unit and should be used instead of "SEEK 0" in initialization or recovery routines. The controller becomes BUSY upon execution of the Data Output command and remains BUSY for approximately 5 μsec while transferring the RESTORE command to the disk unit. SEEK COMPLETE status/interrupt occurs when the disk has completed its positioning operation. RESTORE is rejected if sent to a NOT READY unit; rejection causes SEEK ERROR and COMMAND REJECT status/interrupt.

The SELECT command is used to load the controller's 3-bit unit and fixed/removable register, which designates the source/destination of status/commands. The NOT READY and SEEK ERROR status bits are gated by the unit bits; the Next Sector status bits originate in the fixed or removable cartridge. The controller does not become BUSY in response to SELECT commands and status information is available immediately (i.e., with the next instruction). SELECT commands are ignored by the controller while it is BUSY.

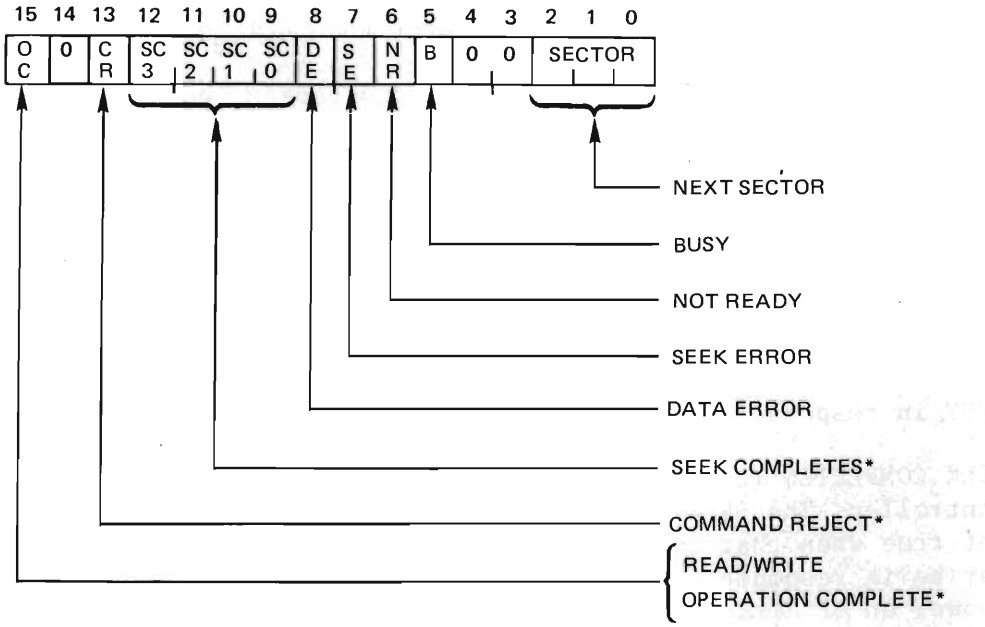
In response to the undefined CMD codes (i.e., 101 and 110), the controller becomes BUSY for approximately 5 μsec and then terminates with COMMAND REJECT status/interrupt.

Seven indicators on the disk controller reflect the status of the Disk Storage System:

- BUSY
- NOT READY
- SEEK ERROR
- DATA ERROR
- SEEK COMPLETES
- COMMAND REJECT
- READ/WRITE OPERATION COMPLETE

These indicators occupy bit positions in a status word. The program interrogates the bits in the status word via a Data Input status access assigned to the Disk Storage System.

The format of the status word is shown below:



* = INTERRUPT CONDITIONS

The Next Sector to become available for reading or writing is reflected in bits 0-2. These bits are derived from the currently selected disk cartridge and are updated as the disk spins. Updating of the Next Sector counter occurs asynchronous to the computer. It is therefore possible to obtain an erroneous value if the program samples the Next Sector counter while it is incrementing. If a system has stringent timing requirements based on the Next Sector bits, it should read these bits at least twice and compare the values to avoid sampling while the counter is changing.

Bits 3 and 4 are always zero.

BUSY (bit 5) is true when the controller is performing an input/output operation. While BUSY is true, therefore, the controller ignores all output commands addressed to it (i.e., it treats them as "no-ops").

NOT READY (bit 6) is true when the selected disk drive unit is off-line or is busy seeking a cylinder. NOT READY is also true when the controller is BUSY.

SEEK ERROR (bit 7) is set true when a SEEK or RESTORE command is issued to a NOT READY unit, when a SEEK command specifies a cylinder address greater than 203_{10}, or by a positioning malfunction within the disk unit. SEEK ERROR due to disk NOT READY or illegal cylinder also sets the COMMAND REJECT status/interrupt. This SEEK ERROR is reset by a Data Input status access command or when the controller becomes BUSY in response to a new Data Output command. SEEK ERROR due to positioning malfunction is reset only when a RESTORE command is issued to the unit.

DATA ERROR (bit 8) is set true by the following conditions:

a. At the termination of a WRITE or READ command if the word count specified is equal to zero or greater than 401_{10}.

b. If a slow data channel access causes lost data (data overrun).

c. At the termination of a READ or READ CHECK operation if a parity error is detected or if the operation is not completed when the next sector boundary occurs.

d. If a WRITE, READ or READ CHECK is attempted to a NOT READY unit. (This condition also sets COMMAND REJECT.)

DATA ERROR is reset by a Data Input status access or when the controller becomes BUSY in response to a Data Output command.

SEEK COMPLETES (bits 9-12) reflect the status of each disk unit attached to the controller. The SEEK COMPLETE status/interrupt indicator for each disk drive unit is set true when that unit has completed a positioning operation. Such positioning may be in response to a SEEK or RESTORE command or after an initial load sequence (power on or disk cartridge change). All SEEK COMPLETE bits are reset by a Data Input status access, by depressing the SYSTEMS RESET switch on the computer console or by issuing the ABORT Control function.

COMMAND REJECT (bit 13) status/interrupt, when set, indicates abnormal termination of an operation by the controller. COMMAND REJECT will be set at termination of any operation attempted with a NOT READY storage unit or at termination of an attempt to execute an undefined command code. At termination of a READ, READ CHECK or WRITE command, COMMAND REJECT is accompanied by DATA ERROR status; at termination of a SEEK or RESTORE command, it is accompanied by SEEK ERROR status. COMMAND REJECT is reset by a Data Input status access, by depressing the SYSTEMS RESET switch on the computer's console, by issuing the ABORT Control function, or when the controller becomes BUSY in response to a Data Out command.

Bit 14 is always zero.

READ/WRITE OPERATION COMPLETE (bit 15) status/interrupt occurs when the controller terminates an accepted read/write operation. READ/WRITE OPERATION COMPLETE is reset by a Data Input status access, by depressing the SYSTEMS RESET switch, by issuing the ABORT Control function, or when the controller becomes BUSY in response to a Data Out command.

Three Control instruction codes have been assigned to the Disk Storage System. These commands perform the following functions:

- ENABLE INTERRUPT
- DISABLE INTERRUPT
- ABORT

The ENABLE INTERRUPT command sets the interrupt mask on the controller to allow the controller to request an interrupt when SEEK COMPLETE, COMMAND REJECT or READ/WRITE OPERATION COMPLETE is true. The DISABLE INTERRUPT command resets the interrupt mask, preventing the disk controller from requesting an interrupt. The state of the mask remains in effect until one of these commands is given or the SYSTEMS RESET switch on the computer's console is depressed (which disables interrupts on all peripheral controllers).

The ABORT function performs the same function as the SYSTEMS RESET switch on the computer's console and forces the controller to an initialized condition: all resettable status bits are reset, the controller is set NOT BUSY, and Disk Unit 0 Removable is selected. (The Interrupt Mask is not affected by the ABORT FUNCTION.) The ABORT function is intended for use during hardware debugging and should not be used otherwise; its use during a WRITE operation may cause the loss of one sector of data.

4.8.2 Instructions Summary

The controller for the Disk Storage System has been assigned a device select code of X'E'. The memory location dedicated to this controller's interrupt vector is X'4E'. The memory locations dedicated to the Disk Storage System's SCR and CAR are X'22' and X'23', respectively.

Table 4-9 summarizes the XIO instructions that apply to the Disk Storage System. The instructions are given in CAP-16 Assembly Language format (see Section 3.9 for their machine language equivalents).

Table 4-9. Instruction Summary for the Disk Storage System

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' or DTOM R,X'E'	OUTPUT object word from register (R) or from memory to disk controller.
DTIR R,X'E' or DTIM R,X'E'	INPUT status word to register (R) or to memory from disk controller

R = A,X,Y,Z,B,C,D or E

4.8.3 Programming Example

The Disk Storage System operates in either a non-interrupting or an interrupting mode.

The example below illustrates a subroutine that can be called to perform any of four operations:

- WRITE
- READ
- READ CHECK
- RESTORE

The calling sequence is as follows:

1. Call the subroutine (JSR DSK)

2. Define word count for Read or Write operations; for RESTORE, code DS 1.

3. Define buffer address for Read or Write operations; for RESTORE, code DS 1.

4. Select drive and operation by coding BYTE (drive index), (operation index).

   The indices for drive and operation selection are given below:

Drive Index	Drive
0	0
1	1
2	2
3	3

Operation Index	Operation
0	WRITE
1	READ
2	READ CHECK
3	RESTORE

SEEK and SELECT commands are generated by the subroutine.

5. Define sector number for all operations except restore. For 3346 subsystems, sector numbers range from 0 to 6527_{10}; for 3347 subsystems, sector numbers range from 0 to 3263_{10}.

The subroutine consists of two parts:

- An initialization routine (DSK)
- An interrupt service routine (IDSK)

The initialization routine sets up the interrupt vector, loads the word count and buffer address in the dedicated SCR and CAR locations, decodes the drive and operation selection codes, decodes the sector number (into head, track and surface sector), and initiates the operation. The interrupt service routine is invoked each time a disk interrupt occurs. The interrupt service routine checks for errors and determines whether the operation is complete, then returns control to the calling program. In read or write operations, the interrupt service routine initiates the actual data transfer following detection of "seek complete".

In addition, the subroutine defines and maintains a memory word (STAT) that the programmer must interrogate in order to check whether a previously-initiated operation has been completed or has encountered an error. Bit 15 of this word is reset when the operation is complete; bit 14 is set when an error occurs.

The example below writes the contents of 40 sequential memory locations, starting at location DBUFR into Sector 1343 of Drive 1.

```
        JSR     DSK         Calling sequence
        DC      40          Word count
        DC      DBUFR       Data buffer address
        BYTE    1,0         Drive, operation indices
        DC      1343        Sector number
        (Return when SEEK or RESTORE has been initiated.)
        .
        .
STAT    DS      1           Bit 15=op complete, bit 14=error
SAVEC   DS      1           Will contain operation command
        .
        .
DSK     SARS    *0          Entry sequence
        LDR     B,0
        RTR     C,B
```

```
        ADDV    C,9
        STR     C,0
        RTR     D,E             Set up D to fetch parameters
        INE                     Enable interrupts
        LDR     X,2,,1          Fetch drive, operation indices
        RTR     Y,X             Operation index will be in Y
        EXBY    X               Put drive index in lower byte of X
        ANDV    X,X'3'          Select drive bits
        ANDV    Y,X'3'          Select operation bits
DSK1    DTIR    A,X'E'          Input STATUS word
        ANDVC   A,X'0020'       Check BUSY bit
        SKN     DSK1            If BUSY, try again
        LDV     A,X'E000'       Load SELECT op code
        OR      A,X             Add drive number
        LDR     C,3,,1          Fetch sector number
        SUBVC   C,3264          Is sector on removable disk
        SKM     DSK2            Yes - branch
        ORV     A,X'4'          No - set fixed bit
        SUBV    C,3264          Reduce sector to single platter number
DSK2    DTOR    A,X'E'          Output SELECT command
DSK3    DTIR    A,X'E'          Input STATUS word
        ANDVC   A,X'0040'       Check NOT READY bit
        SKN     DSK3            Try again if NOT READY
        LDV     A,IDSK          Set up interrupt vector
        STR     A,X'4E'
        CTRL    0,X'E'          Enable interrupts from disk
        SUBVC   Y,3
        SKN     DSK4            Branch if not RESTORE
        LDV     A,X'8000'       Load RESTORE command
        SBIT    5,STAT          Set RESTORE flag
        SBIT    7,STAT          Set operation going bit
        DTOR    A,X'E'          Output RESTORE command
        JMP     EXIT
DSK4    LDR     A,0,,1          Fetch word count
        STR     A,X'22'         Store word count in SCR location
        LDR     A,1,,1          Fetch data buffer address
        DECR    A
        STR     A,X'23'         Store address-1 in CAR location
        LDR     A,OPTABL,Y      Fetch op code command
        OR      A,C             Add surface and sector numbers
        STR     A,SAVEC         Save command at SAVEC
        RTR     A,C             Transfer sector number to A
        SRA     A,4             Isolate cylinder number
        SBIT    7,STAT          Set operation going bit
        DTOR    A,X'E'          Output SEEK command
        JMP     EXIT
OPTABL  DC      X'2000'         Code = 2 (WRITE)
        DC      X'4000'              = 4 (READ)
        DC      X'6000'              = 6 (READ CHECK)
```

```
EXIT    INH                         Exit sequence
        STR         B,O
        LARS        *O
        ADDV        E,5
        RTRN        E
*INTERRUPT RESPONSE ROUTINE
IDSK    SARS        *O
        LDR         B,O
        RTR         C,B
        ADDV        C,9
        STR         C,O
        INE
        RBIT        6,STAT          Assume no error
        DITR        A,X'E'          Input STATUS word
        ANDVC       A,X'2180'       Check error condition bits
        SKN         ERROR           Branch if any error
        RTR         A,A
        SKM         OPCOM           Branch if Read/Write complete
        RBIT        5,STAT
        SKN         OPCOM           Branch if from RESTORE
        LDR         A,SAVEC         Load saved Read/Write Command
        XEC         A                 and execute it
        JMP         CONT
ERROR   SBIT        6,STAT          Error - set error bit
OPCOM   RBIT        7,STAT          Indicate operation complete
CONT    INH                         Exit sequence
        STR         B,O
        LARS        *O
        RTRN        E
```

4.9 MAGNETIC TAPE UNIT (MTU)

The Magnetic Tape Unit allows the high speed transfer of data between the SPC-16 and one of four possible tape drives that can be connected to the MTU controller. Three different General Automation tape drive models may be used 3331, 3332 or 3333.

Model 3331 has a speed of 25 ips; models 3332 and 3333 drive tapes at 37.5 ips and 75 ips, respectively.

Three types of instructions are used to communicate with the Magnetic Tape Unit controller. A Data Output instruction is used to send an "object word" to the controller either from memory or from a register. The object word may be one of seven commands to the controller:

- BACKSPACE
- REWIND
- UNLOAD
- ERASE
- WRITE TAPE MARK
- READ
- WRITE

A Data Input instruction is sent to the controller when the program wants to interrogate the status of the unit. The controller responds to the Data Input instruction by sending a "status word" to the computer; the status word contains information such as "busy", "operation complete", "error", etc. Control instructions are sent to perform four functions:

- ENABLE INTERRUPTS
- DISABLE INTERRUPTS
- RESET CONTROLLER
- SELECT DRIVE (0, 1, 2 or 3)

The initiation of a data transfer operation for the MTU is under program control. The program must load the controller's dedicated word count (SCR) and channel address (CAR) memory locations in preparation for a DMA transfer, must prepare the proper unit to send/accept the data by executing a series of Data Output and Data Input instructions, and must initiate the transfer (via a READ or WRITE command).

For the actual transfer of data words the Magnetic Tape Unit controller employs a Data Channel (Direct Memory Access); the controller supervises the transfer of each data word between the computer memory and the selected tape drive. The MTU can be interrogated by the program to determine if it requires service, or it can be allowed to request a program interrupt when it requires service.

4.9.1 Operational Description

CONTROL COMMANDS

Seven Control instruction codes have been assigned to the MTU. These instructions perform the following operations:

- ENABLE INTERRUPT
- DISABLE INTERRUPT
- RESET CONTROLLER
- SELECT DRIVE 0
- SELECT DRIVE 1
- SELECT DRIVE 2
- SELECT DRIVE 3

The ENABLE INTERRUPT command sets the interrupt mask on the controller to allow the controller to request an interrupt when COMMAND REJECT, END OF DATA TABLE or OPERATION COMPLETE is true. The DISABLE INTERRUPT command resets the interrupt mask, preventing the MTU controller from requesting an interrupt. The state of the mask remains in effect until one of these commands is given or the SYSTEMS RESET switch on the computer's console is depressed (which disables interrupts on all peripheral controllers).

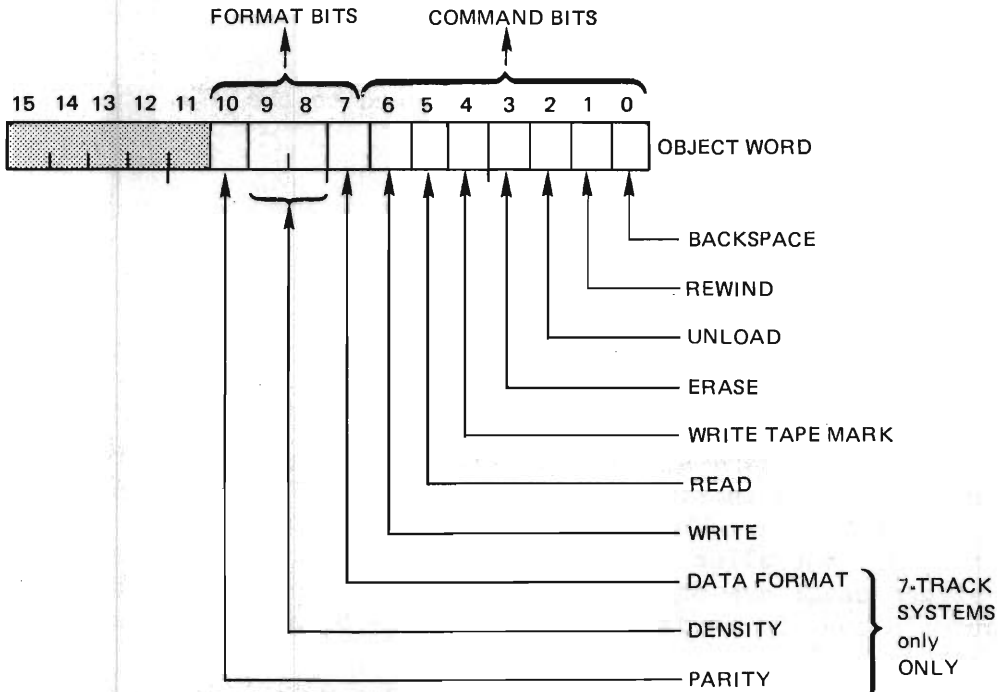
The RESET CONTROLLER command causes a pulse to be issued that is equal to System Reset. This pulse resets the controller to an idle state, stops the tape drive (except for rewind), clears all clearable status bits, disables the MTU interrupt and selects Drive 0.

Each SELECT DRIVE command selects one drive and deselects all other drives.

For proper operation, the desired drive should be selected before any other command is given. Issuing RESET CONTROLLER or depressing the SYSTEMS RESET switch will select Drive 0.

FUNCTIONAL COMMANDS

One Data Output instruction is used with the Magnetic Tape Unit. This instruction can cause one of seven different operations to be performed, depending on the format of the object word it transmits to the controller. The format of the object word is as follows:



NOTE: ONLY ONE COMMAND BIT SHOULD BE SET TRUE AT ANY ONE TIME.

The BACKSPACE command (bit 0=1) causes the selected tape to back up one record. In the event the load point (i.e., the BOT reflective marker) is encountered in this process, the tape will stop on the load point. If BACKSPACE is issued when the tape is at the load point, the command will be rejected and the COMMAND REJECT status/interrupt will be generated.

The REWIND command (bit 1=1) will cause the selected drive to rewind the tape to the tape load point (i.e., beginning of tape). When the drive has started to rewind, OPERATION COMPLETE status/interrupt will be generated, leaving the controller free to perform operations on other drives. If REWIND is issued when the tape is at the BOT, the command will be rejected and the COMMAND REJECT status/interrupt will be generated.

The UNLOAD command (bit 2=1) will place the selected drive in "local" mode and requires the operator to place the drive in "remote" mode for further operation with the drive. This command is normally given after receiving the OPERATION COMPLETE interrupt following a REWIND command. The tape will continue to rewind in an off-line state until it is entirely on the supply reel.

The ERASE command (bit 3=1) causes the tape drive to erase 3.75 inches of tape in the forward direction.

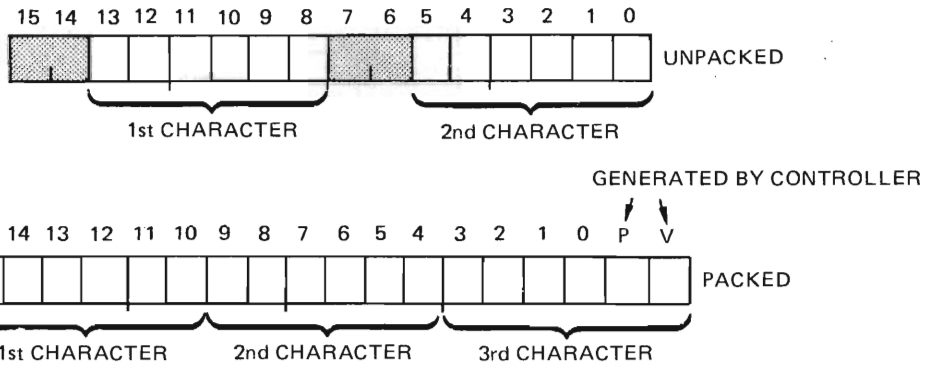
Upon receipt of the WRITE TAPE MARK command (bit 4=1), the controller will erase 3.75 inches of tape in the forward direction, write a tape mark and then stop.

The READ command (bit 5=1) is used to transfer magnetic tape data to core memory. The READ command causes the controller to initiate the Data Channel and read the next tape record from the selected drive.

The WRITE command (bit 6=1) is used to transfer core memory data to magnetic tape. The WRITE command causes the controller to initiate the Data Channel and write one record onto the selected drive.

The Format Bits specify the parity, bit density and data format in transfers to/from a seven-track system. Nine-track systems always use odd parity and a bit density of 800 bits/inch. The Format Bits are defined as follows:

DATA FORMAT may be either "packed" (three characters per data word) if bit 7=1 or "unpacked" (two characters per data word) if bit 7=0. These formats are shown below:



DENSITY is a two-bit format (bits 8 and 9) used to select the tape data density, as given in the table below:

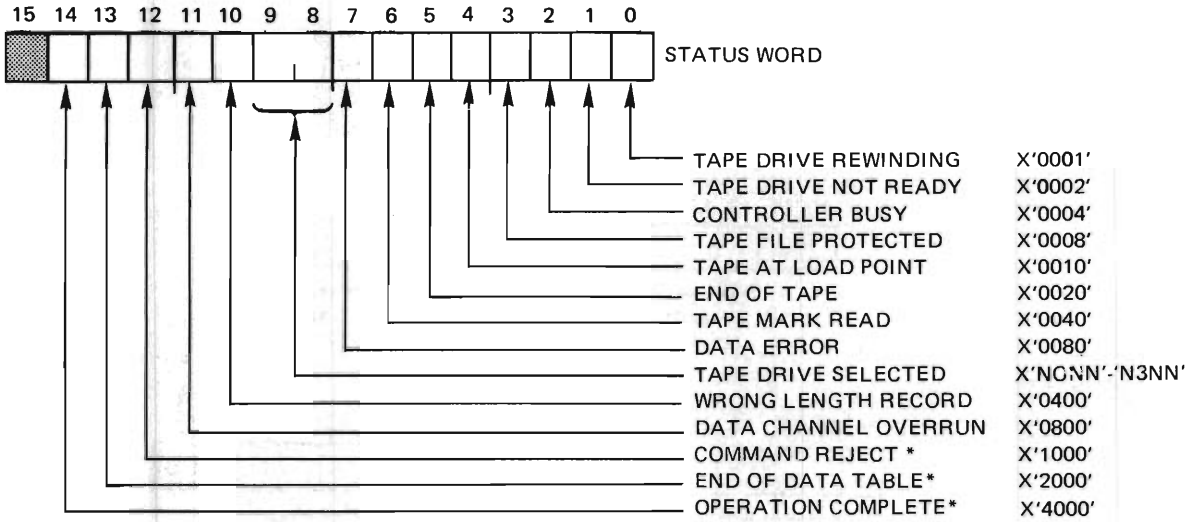
Bit 9	Bit 8	Density
0	0	800 bits/inch
0	1	556 bits/inch
1	0	200 bits/inch
1	1	Not Used

PARITY is specified as odd when bit 10 is set to "zero" and as even when bit 10 is set to "one". It should be noted that zero is stored on tape as "010000" when even parity is used.

STATUS INDICATORS

Fourteen indicators on the MTU controller reflect the status of the MTU. These indicators occupy bit positions in a status word. The program interrogates the status word bits via a Data Input command assigned to the Magnetic Tape Unit.

The format of the status word is shown below:



* = INTERRUPT CONDITIONS

TAPE DRIVE REWINDING (bit 0) is true when the selected drive is in the process of rewinding in the remote mode. Any Data Output command issued to a REWINDING drive will cause COMMAND REJECT status/interrupt to be generated. It will go false when the UNLOAD command is given after a REWIND command.

TAPE DRIVE NOT READY (bit 1) is true when the selected drive is off-line or rewinding. Any Data Output command issued to a drive that is NOT READY will cause COMMAND REJECT status/interrupt to be generated.

CONTROLLER BUSY (bit 2) is true when the controller is busy with the selected drive. All commands sent to a BUSY controller will cause COMMAND REJECT status/interrupt to be generated with the exception of mask enable/disable Data Output commands, a Data Input command and the RESET control command.

TAPE FILE PROTECTED (bit 3) indicates that the write permit ring is not in the reel of tape. If a WRITE, WRITE TAPE MARK or ERASE command is issued to a drive having TAPE FILE PROTECTED true, COMMAND REJECT status/interrupt will be generated.

TAPE AT LOAD POINT (bit 4) is true when the tape is at the load point (i.e., at beginning of the tape). TAPE AT LOAD POINT goes false when the drive is off-line or the tape is advanced past the load point.

END OF TAPE (bit 5) becomes true when the end of tape marker is encountered while the tape is moving in the forward direction.

TAPE MARK READ (bit 6) becomes true when a tape mark is encountered during a READ operation. This bit causes OPERATION COMPLETE status/interrupt to be generated.

DATA ERROR (bit 7) indicates the occurrance of a vertical parity error, longitudinal error or cyclic redundancy error during a READ, WRITE or WRITE TAPE MARK command. DATA ERROR causes OPERATION COMPLETE status/interrupt to be issued.

TAPE DRIVE SELECTED (bits 8 and 9) indicate the binary number of the tape drive selected. The coding is as follows:

00 = Drive 0
01 = Drive 1
10 = Drive 2
11 = Drive 3

These bits are governed by which SELECT DRIVE Control command is issued to the controller. When the SYSTEMS RESET switch is depressed on the computer's console or the RESET Control command is issued, Drive 0 is selected.

WRONG LENGTH RECORD (bit 10) is true if the last record read did not have the exact number of data words requested. This bit causes OPERATION COMPLETE status/interrupt to be generated.

DATA CHANNEL OVERRUN (bit 11) becomes true when a pending data channel access has not been completed before another request is made.

COMMAND REJECT (bit 12) status/interrupt bit becomes true at the completion of an output command issued when any of the following conditions exist:

1. TAPE DRIVE REWINDING is true.
2. TAPE DRIVE NOT READY is true.
3. CONTROLLER BUSY is true.
4. ERASE, WRITE TAPE MARK or WRITE Command when TAPE FILE PROTECTED bit is true.
5. BACKSPACE or REWIND command when TAPE AT LOAD POINT bit is true.

COMMAND REJECT will also become true if an attempt is made to select a drive (the same or a different drive) while the controller is busy.

END OF DATA TABLE (bit 13) status/interrupt becomes true when the DMA Block Complete signal (DCBC-, see Section 5.2.1) is received from the Data Channel if bit 14 in the Scan Control Word is zero. Scan Control Word's bit 14 equal to zero specifies an interrupt after each block transfer.

OPERATION COMPLETE (bit 14) becomes true when the controller finishes a BACKSPACE, UNLOAD, ERASE, WRITE TAPE MARK, READ or WRITE operation. OPERATION COMPLETE also becomes true after a REWIND has been initiated.

Bit 15 is not used in the status word.

Bits 5 through 14, with the exception of bits 8 and 9, are reset when a Data Input, Data Output or RESET Control command is issued or when the SYSTEMS RESET switch on the computer's console is depressed.

4.9.2 Instruction Summary

The controller for the Magnetic Tape Unit has been assigned a device select code of X'F'. The memory location dedicated to this controller's interrupt vector is X'4F'. The memory locations dedicated to the Magnetic Tape Unit's SCR and CAR are X'24' and X'25', respectively.

Table 4-10 summarizes the XIO instructions that apply to the Magnetic Tape Unit. The instructions are given in CAP-16 Assembly Language format (see Section 3.9 for their machine language equivalents).

Table 4-10. Instruction Summary for the Magnetic Tape Unit

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' / or / DTOM R,X'F'	OUTPUT object word from register (R) or from memory to MTU controller.
DTIR R,X'F' / or / DTIM R,X'F'	INPUT status word to register (R) or to memory from MTU controller.

R = A,X,Y,Z,B,C,D or E

4.9.3 Programming Example

The Magnetic Tape Unit operates in either a non-interrupting or an interrupting mode.

The example below illustrates a subroutine that can be called to perform any of the seven operations specified by a Data Output instruction (Backspace, Rewind, etc.). The example uses interrupting mode.

The calling sequence is as follows:

1. Call the subroutine (JSR MAGTAP).
2. Define byte count for READ or WRITE; for all other operations, code DS 1.
3. Define buffer address for READ or WRITE; for all other operations, code DS 1.
4. Select drive and operation by coding BYTE (drive index), (operation index).

   The indices for drive and operation selection are given below:

Drive Index	Drive
0	0
1	1
2	2
3	3

Operation Index	Operation
0	READ
1	WRITE
2	WRITE TAPE MARK
3	ERASE
4	BACKSPACE
5	REWIND
6	UNLOAD

The subroutine consists of two parts:

- An initialization routine (MAGTAP)
- An interrupt service routine (INTRPT)

The initialization routine sets up the interrupt vector, loads the word count (converted from byte count) and buffer address in the MTU's dedicated SCR and CAR locations, decodes the drive and operation selection codes, and initiates the operation. The interrupt service routine is invoked each time an MTU interrupt occurs. The interrupt service routine checks for errors and determines whether the operation is complete, then returns control to the calling program.

In addition, the subroutine defines and maintains a memory word for each unit (TAPE0, TAPE1, TAPE2 and TAPE3) that the programmer must interrogate in order to check whether a previously initiated operation has been completed or has encountered an error. Bit 15 of this word is reset when the operation is complete; bit 14 is set when an error occurs.

The example below reads 100 bytes from Drive 1 into 50 sequential memory locations (2 bytes per word) beginning at location MBUFR plus one.

```
        JSR       MAGTAP        Calling sequence
        DC        100           Byte count
        DC        MBUFFR        Data Buffer address
        BYTE      1,0           Drive, operation indices
        (Return when operation has been initiated)
        .
        .
TAPE0   DS        1             Bit 15 will contain operation
TAPE1   DS        1             complete indicator; bit 14
TAPE2   DS        1             will contain error indicator
TAPE3   DS        1

MAGTAP  SARS      *0
        LDR       B,0
        RTR       C,B           Entry sequence
        ADDV      C,9
        STR       C
        RTR       D,E           Set up D to fetch parameters
        INE                     Enable interrupts
        LDR       X,2,,1        Fetch drive, operation indices
        RTR       Y,X           Load X into Y
        EXBY      X             Swap bytes in X register
        ANDV      X,X'0003'     Select drive bits
        ANDV      Y,X'0007'     Select operation bits
LOOP1   DTIR      A,X           INPUT status word
        ANDVC     A,X'4'        Check CONTROLLER BUSY  bit
        SKN       LOOP1         If BUSY, try again.
        LDR       A,SLTABL,X    Fetch proper SELECT Control command
        XEC       A             and execute
LOOP2   DTIR      A,X'F'        INPUT status word
        ANDVC     A,X'0007'     Check CONTROLLER BUSY, NOT READY
                                and REWINDING bits
        SKN       LOOP2         Try again if any are set
        LDV       A,INTRPT
        STR       A,X'4F'       Set up interrupt vector
        CTRL      0,X'F'        Enable interrupts from MTU
        LDR       A,OPTABL,Y    Fetch operation
        EXIT      A             Go to specified operation
SLTABL  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
OPTABL  DC        READ          CODE = 0
        DC        WRITE              = 1
        DC        WRITTM             = 2
        DC        ERASE              = 3
        DC        BACKSP             = 4
        DC        REWIND             = 5
        DC        UNLOAD             = 6
        DC        ERROR              = 7  (illegal)
```

```
READ    LDV     C,X'20'         Load READ command
SCRCAR  LDR     A,0,,1          Fetch byte count
        INCR    A
        SRA     A,1             Shift count right (divide by 2)
        STR     A,X'24'         Store word count in SCR location
        LDR     A,1,,1          Fetch data buffer address
        STR     A,X'25'         Store buffer address in CAR location
COMAND  ADD     X,X             Double index
        SBIT    7,TAPE0,X       Set operation not complete
        DTOR    C,X'F'          OUTPUT command
        JMP     EXIT
WRITE   LDV     C,X'40'         Load WRITE command
WRTEST  DTIR    A,X'F'          INPUT status word
        ANDV    A,X'0008'       Check TAPE FILE PROTECTED bit
        SKN     ERROR           If file protected, go to error
        JMP     SCRCAR          Go fetch SCR and CAR
WRITTM  LDV     C,X'10'         Load WRITE TAPE MARK command
        JMP     WRTEST
ERASE   LDV     C,X'8'          Load ERASE command
        JMP     WRTEST
BACKSP  LDV     C,1             Load BACKSPACE command
BOTTST  DTIR    A,X'F'          INPUT status word
        ANDV    A,X'0010'       Check TAPE AT LOAD POINT bit
        SKN     EXIT            If bit is true, ignore command
        JMP     COMAND          If bit is false, continue
REWIND  LDV     C,2             Load REWIND command
        JMP     BOTTST
UNLOAD  LDV     C,4             Load UNLOAD command
        JMP     COMAND
ERROR   ADD     X,X             Double index
        SBIT    6,TAPE0,X       Set error bit
        JMP     EXIT

INTRPT  SARS    *0
        LDR     B,0
        RTR     C,B             Entry sequence
        ADDV    C,9
        STR     C,0
        INE                     Enable interrupts
        DTIR    A,X'F'          INPUT status word
        RTR     X,A
        SRC     X,8             Right justify TAPE DRIVE SELECTED
        ANDV    X,X'0003'       Select TAPE DRIVE SELECTED bits
        ADD     X,X             Double index
        RBIT    6,TAPE0,X       Reset error bit
        ANDVC   A,X'1C80'       Check error condition bits
        SKZ     NOERRR          If none are true, no error
        SBIT    6,TAPE0,X       Error - set error bit
NOERRR  RBIT    7,TAPE0,X       Indicate operation complete
        INH                     Disable master interrupt
```

```
        STR     B,0         Restore registers of
        LARS    *0          interrupted program
        RTRN    E           Return

EXIT    INH                 Exit sequence
        STR     B,0
        LARS    *0
        ADDV    E,3         Increment past data table
        RTRN    E
```

SECTION 5
INTERFACING

All peripheral units associated with an SPC-16 computer communicate with the Central Processing Unit (CPU), either directly or indirectly, via the Input/Output Bus (I/O Bus). Three classes of operations take place on the I/O Bus:

- Programmed I/O operations without interrupts.
- Programmed I/O operations with interrupts.
- Direct Memory Access (DMA) operations.

A programmed I/O operation occurs in response to the execution of an I/O instruction by the CPU. An I/O instruction specifies the controller to which it refers by a 6-bit device select code. Each controller interfaced to a particular SPC-16 computer 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 I/O instruction is executed, only the controller whose select code is specified must respond to the indicated operation. A number of device select codes have been assigned to standard GA controllers; these assignments are listed in Table 5-1.

I/O instructions provide four types of operations:

- Data Input
- Data Output
- Control
- 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 the 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.

Programmed I/O may be implemented in controllers that are designed to generate interrupts or in controllers that have no interrupt capability, depending on 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 (see Section 4), but the CPU's attention is required from the beginning to the end of the transfer. This is not necessarily a detrement; in applications requiring only one peripheral unit, or in applications 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.

Table 5-1. Standard Device Select Codes and Interrupt Assignments

Device Select Code	Controller	Interrupt Vector Location
00	Floating Point Processor	046
01		
02		
03		
04		
05		
06		
07	GAARD	---
08	Paper Tape Reader	048
09	Paper Tape Punch	049
0A	Cassette	04A
0B	Card Reader	04B
0C	Line Printer	04C
0D	Card Punch	04D
0E	Disk	04E
0F	Magnetic Tape	04F
10	Selectric	050
11	Plotter	051
12	Fixed Head Disk	052
13	2nd Fixed Head Disk	053
14		
15		
16		
17		
18	2nd Paper Tape Reader	058
19	2nd Paper Tape Punch	059
1A	2nd Cassette	05A
1B	2nd Card Reader	05B
1C	2nd Line Printer	05C
1D	2nd Card Punch	05D
1E	2nd Disk	05E
1F	2nd Magnetic Tape	05F

Device Select Code	Controller	Interrupt Vector Location
20	1590 Comm. MUX or 1561 or 1581 Asynch. Contr.	060
21		061
22		062
23	201 Synch. Contr.	063
24	2nd 1590 Comm. MUX or or 1561 or 1581 Asynch. Contr.	064
25		065
26		066
27	2nd 201 Synch. Contr.	067
28	3rd 1590 Comm. MUX or 1561 or 1581 Asynch. Contr.	068
29		069
2A		06A
2B	3rd 201 Synch. Contr.	06B
2C	4th 1590 Comm. MUX or 1561 or 1581 Asynch. Contr.	06C
2D		06D
2E		06E
2F	4th 201 Synch. Contr.	06F
30	5th 1590 Comm. MUX or 1561 or 1581 Asynch. Contr.	070
31		071
32		072
33		
34	6th 1590 Synch. Contr. or 1561 or 1581 Asynch. Contr.	074
35		075
36		076
37		
38	M.H.S.D.C. CAR/SCR Read Function	---
39	Extended Memory	---
3A	Hi-Speed, Signed MPY-DIV	---
3B		---
3C		---
3D		---
3E	Interrupt Mask/Console	---
3F	Console TTY Switch	045

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 multi-processing 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. 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 (interrupting 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 or "status-bit" associated with a single controller can initiate an interrupt request, the program must 'sense' the status of the controller (usually via Test instructions or an input status word) to determine the interrupting condition. A number of interrupt vector locations have been assigned to standard GA controllers; these assignments are summarized in Table 5-1.

There are two situations in which programmed I/O may prove insufficient:

1. When the peripheral unit is a block of a record-orineted 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.

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 amy transfer data at a rate of 10^7 bits/sec., which means that even the fastest SPC-16 processors, Models 80 and 85, 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 is to use a controller unit 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. A memory address is supplied to the CPU for each data item that is transferred; a cycle is "stolen" from the running program to transfer each data item to/from the supplied memory address. When the 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.

A controller may contain all the logic required to implement the 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.

As an alternative to implementing all the nexessary 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. A single Data Channel Module provides eight high-speed channels and will service up to eight peripheral units. It maintains a block count register and address register for each active channel. It performs cycle stealing requests 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, bit 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.

Figure 5-1 depicts typical input/output systems for the SPC-16/40, 60 and 80 (integral I/O package) and the SPC-16/45, 65 and 85 (External I/O package). The diagram illustrates the relationship between the CPU, the I/O Bus and various peripheral controller cards for both packaging configurations.

As illustrated, some controllers require one subunit, others require two or more; in some cases, two peripheral controllers are implemented on a single subunit. (The location of controller cards on the diagram is arbitrary.)

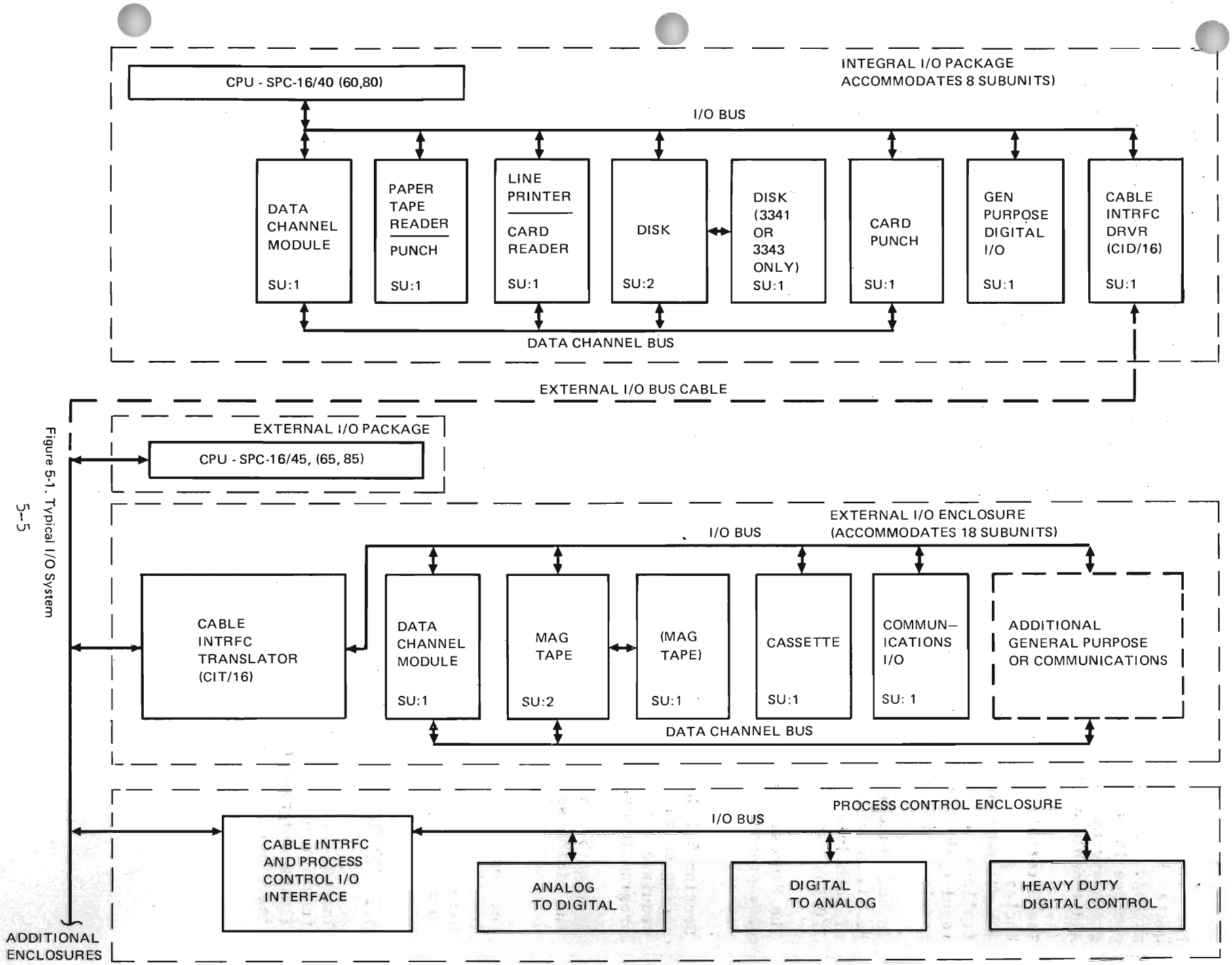


Figure 5-1. Typical I/O System

83A00243A

The Integral I/O package (SPC-16/40 60 and 80) accommodates up to eight I/O subunits. If controllers that require the Data Channel Bus are to be housed in this package, one subunit must be devoted to a Data Channel Module card. If the Integral package is not large enough to accommodate all the controller cards required for a given system, one subunit slot must be devoted to a Cable Interface Driver card. Additional controllers are housed in one or more External I/O Enclosures.

The External I/O package (SPC-16/45, 65 and 85) always requires an External I/O Enclosure. The External I/O Enclosure accommodates up to nineteen subunits; one subunit 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 subunit slot must be devoted to a Data Channel Module card (the Data Channel Bus is constrained to its own physical enclosure).

In both Integral I/O and External I/O, vacant slots between controller boards must be filled with GA - supplied "Shorter Boards". These board maintain the continuity of both interrupt priority and DMA priority signals, which daisy-chain down the I/O Bus (as explained later in this section).

The Process Control Enclosure and I/O Interface is for General Automation heavy-duty process control interfaces. It is a self-contained package that connect directly to the I/O Bus.

Section 5.1 describes the signals on the I/O Bus and the requirements for interfacing to the I/O Bus. The section includes a discussion of the timing and logic required for the implementation of device selection, programmed I/O Operations, program interrupt and DMA operations (without the Data Channel Bus).

Section 5.2 describes the Data Channel Bus. The section includes a discussion of the timing and logic required for the implementation of DMA operations using the Data Channel Module approach.

Section 5.4 includes a physical description of the layout of printed circuit boards for the SPC-16/40, 60 and 80 mainframe, the SPC-16/45, 65 and 85 mainframe and the External I/O Enclosure. The section describes the physical connection of controller cards to the I/O Bus, to the Data Channel Bus and to the peripheral units themselves.

5.1 INTERFACING TO THE I/O BUS

This section describes the I/O Bus and the signals that occur on the I/O Bus as a function of I/O instruction execution, interrupt requests and DMA data transfers. Timing diagrams for I/O Bus signal sequences as well as logic design examples for interfacing to the I/O Bus are given.

5.1.1 I/O Bus Description

The Input/Output Bus is the communication path between the SPC-16 computer and any peripheral equipment. The bus consists of 16 data output lines, 16 data input lines, 19 control lines to the controllers and 9 control lines from the controllers.

With one exception, all bus signals are binary. A minus sign following a signal mnemonic indicates that the signal is between 0 and .4 volts 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 on the external I/O Cable.

Signal	Direction	Definition
INB00-I to INB15	To CPU	IN-Bus. These sixteen data input lines are used to transfer data from any peripheral unit to the CPU. For programmed input, the CPU generates READ- to place data from the selected controllers onto the IN-Bus. 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 0-6). 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.
OTB00-I to OTB15-I	To controller	OUT-Bus. These sixteen data output lines are used to transfer data and control information from the CPU to any peripheral unit. 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 operation. 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.

Signal	Direction	Definition
		For DMA outputs, the data is placed onto these lines and DDTP- is generated telling the controller with priority to accept the data.
FAP-I	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	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. If a cycle steal occurs between FAP and DTP cycles of a READ command, READ-I will be disabled for the duration of the cycle steal.
WRIT-I	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	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	To controller	Control Signal. This signal is used by the selected controller to perform a specified control function. The signal CNTL-I is the simple logical decode of bits 6 and 7 of the OUT-Bus. CNTL-I may be true (low) at times other than that associated with the specified CONTROL instruction. Users of CNTL-I should logically "and" this signal with FAP.

Signal	Direction	Definition
		CNTL- is energized before FAP- and FAP- is used to "time" the control pulse.
TEST-I	To CPU	Common Test Line. This lines is used by the selected controller to return the result of a previously specified test function. When FAP- occurs, the selected controller accepts and 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	To controller	General purpose periodic clock pulse running synchronously when the CPU is in the RUN mode. The period is one memory cycle or one-half the instruction being executed. It always occurs with a fixed skew with respect to the I/O timing pulses FAP-, DTP- and DDTP-.
POLL-I	To controller	Periodic pulse that is identical to D3-, 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	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	To controller	Interrupt Hold. This common signal is used by controller interrupt networks to freeze any further interrupt requests.
IPSO-I	To controller	Interrupt Priority Status Out. 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 receivers its series priority on its

Signal	Direction	Definition
		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	To CPU	Interrupt Priority Return Status. This common signal to the CPU is enabled by IHLD- and IPSO- 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	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 0-6). Although this signal is common to all controllers, it is sued only by the controller that has accepted IPSO- (i.e., the highest priority controller requesting an interrupt).
DREQ-I	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-.
DPSO-I	To controller	DMA Priority Status Out. 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 wired energized at the CPU. 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 DSCI-) is implemented between

Signal	Direction	Definition
		controllers associated with a given Data Channel Module. This technique allows a much higher 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	To controller	DMA Acknowldge. 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.
DDTP-I	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	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. RQIN- is only energized by the controller that has priority. RQIN- must be energized for any of the 'Add', 'Increment' or 'Or' modes (see below).
DMF0-I and DMF1-I		DMA Function Lines. These two lines are used by the controller that has priority to indicate to the CPU the type of DMA cycle as follows:

DMF1-	DMF0-	
0	0	Transfer Data In-Bus→M or M→OUT-Bus
0	1	Increment Memory In-Bus + M + 1→M
1	0	Add to Memory IN-Bus + M→M
1	1	OR to Memory IN-Bus V M→M

NOTE

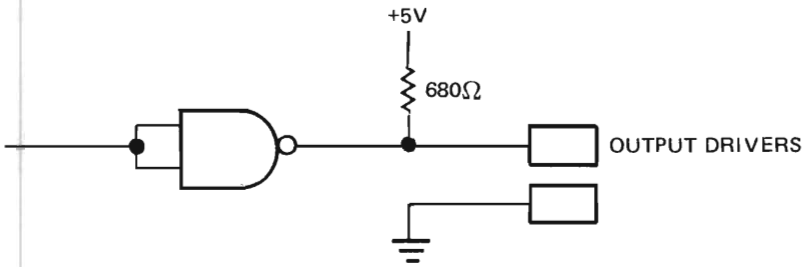
In this table, 0 means that the signal is left high (which is the quiescent state). 1 means that the signal is pulled low.

Signal	Direction	Definition
RDCL-I	To CPU	Read-Clear Memory. This line is energized by the controller that has priority to indicate a Read-Clear memory cycle, as opposed to a Read-Restore memory cycle.
CACK-I	To controller	Carry Acknowledge. This common signal indicates that the result of the last "Operate-On-Memory' operation resulted in a carry out of bit 15.
DCHN-	Common to all Data Channel Modules and DMA Controllers	DMA Chain. This signal is common to all Data Channel Modules and DMA Controllers. This signal when true indicates reverse channel priority has been established due to a Data Channel Module or a DMA controllers going into the "initiate" or "chaining" mode. All other Data Channel modules and DMA controllers should hold any further DMA requests until the DCHN- signal goes false (i.e., Data Channel Module or DMA controller presently initiating or chaining has finished).
SYRT-I	To controller	System Reset. This signal is used to initiate the I/O system. SYRT- occurs during power on/off or when the SYSTEMS RESET console switch (see Section 6) is pressed.
SFEC-I	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 (see PMA, Section 3.8.5).
SYNC-I	To controller	Synchronization Pulse. This pulse is generated by the CPU during the 'SYNC' instruction (see Section 3.8.8). SYNC is provided to allow program debugging technique requiring synchronization when observing the analog representation of the contents of the Console Display Register (VU+).
VU+I	To controller	Digital-to-Analog conversion of the first four bits (0-3) of the Console Display Register (K register). Each increment in the register represents 0.5 volts. VU ranges from 0 to +7.5 volts.

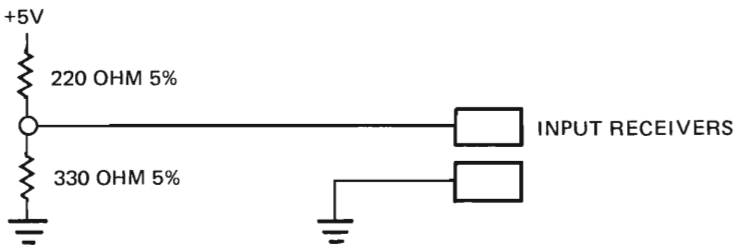
Signal	Direction	Definition
CLDS-I	To CPU	Cold Start. This signal is used to change the Auto-Restart Interrupt Vector to a ROM for cold start operation. When CLDS-I is energized, the power-fail and auto-restart vectors are changed from X'40' and X'41' to X'C0' and X'C1', respectively. A 32 or 64 word ROM cold start loader may then be located at X'C1' which is out of the normal interrupt vector area. In addition, when CLDS-I is energized all memory protect modules are disabled (i.e., memory is not protected).

I/O BUS TERMINATIONS

All output drivers from the SPC-16 processors, except for VU+I are implemented using an open collector TTL gate capable of sinking 48 ma to ground with a saturation voltage of 0.4 volt. VU+I is an analog output and can be terminated to ground with 2.7KΩ or greater.



All input lines are terminated as follows:



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/O bus signal with any peripheral controller.

5.1.2 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.

PROGRAMMED I/O OPERATIONS

The execution of any instruction in the Input/Output group requires two memory cycles. With respect to a controller, the first memory cycle is termed the "device select phase" and the second memory cycle 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

(See Section 3.9 for I/O instruction format information.)

Figure 5-2 is a flowchart of the signal sequence that occurs for I/O instructions.

Figure 5-3 includes a timing diagram for the execution of a TEST instruction and a CTRL (Control)instruction. The logic diagram, also in Figure 5-3, illustrates a typical scheme for implementing the device selection network, the test function and control function on any peripheral controller.

An eight input NAND gate performs the device select decoding; this gate continuously monitors the 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 OTF10+. (The state of function code flip-flops is irrelevant on a controller which has not been select.)

The test multiplexer decodes 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 IN-Bus. 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, CNTRL- 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.

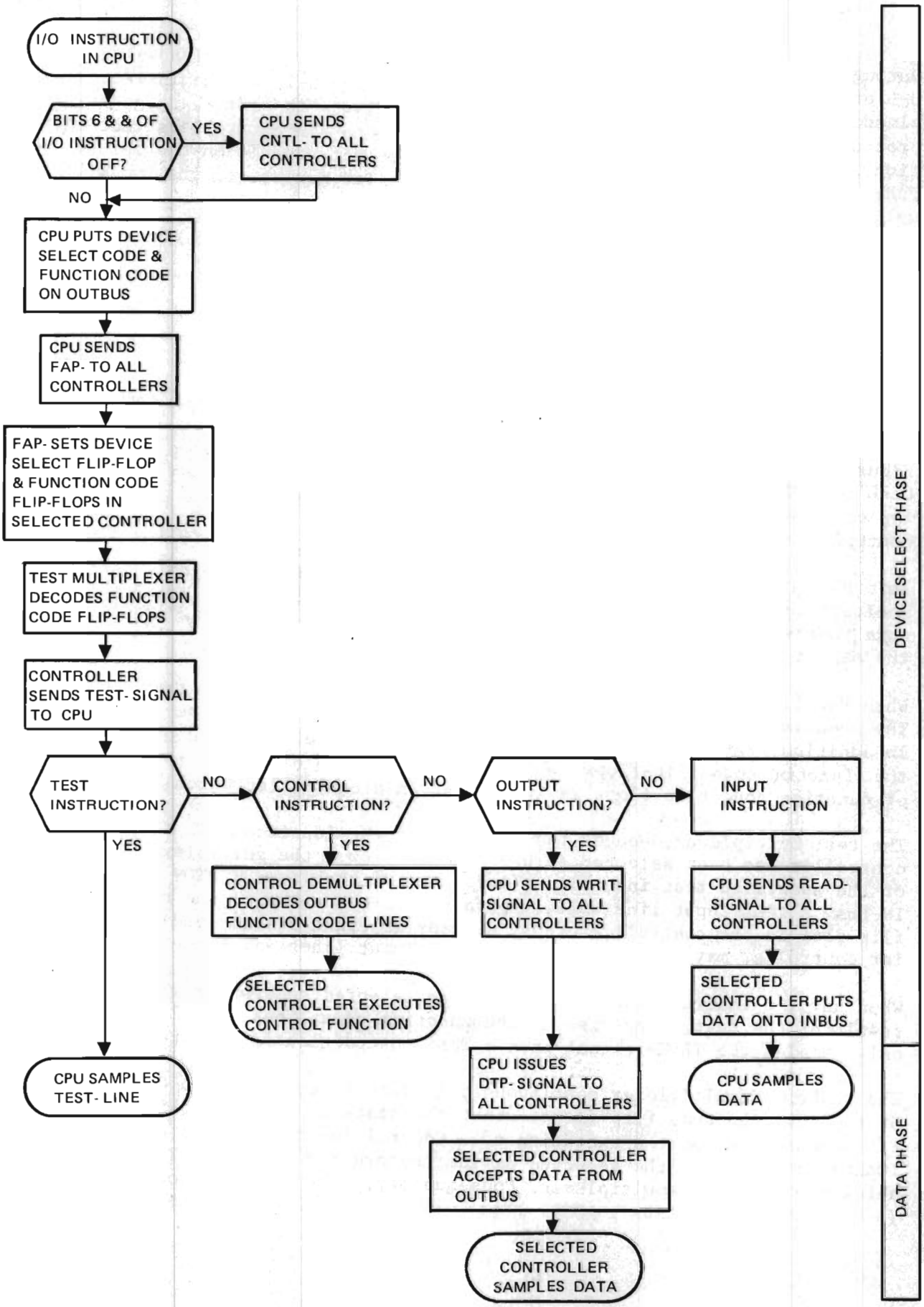
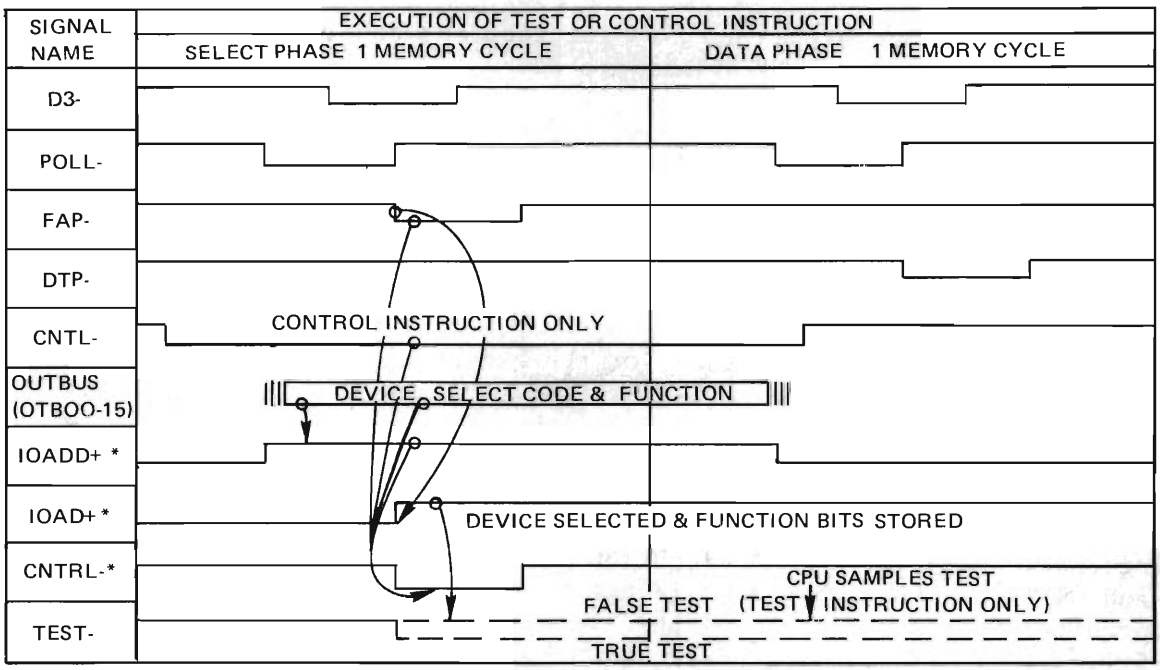


Figure 5-2. Flowchart of I/O Instructions



*NOTE: THESE SIGNALS ARE INTERNAL TO THE CONTROLLER. ALL OTHER SIGNAL ARE ON I/O BUS.

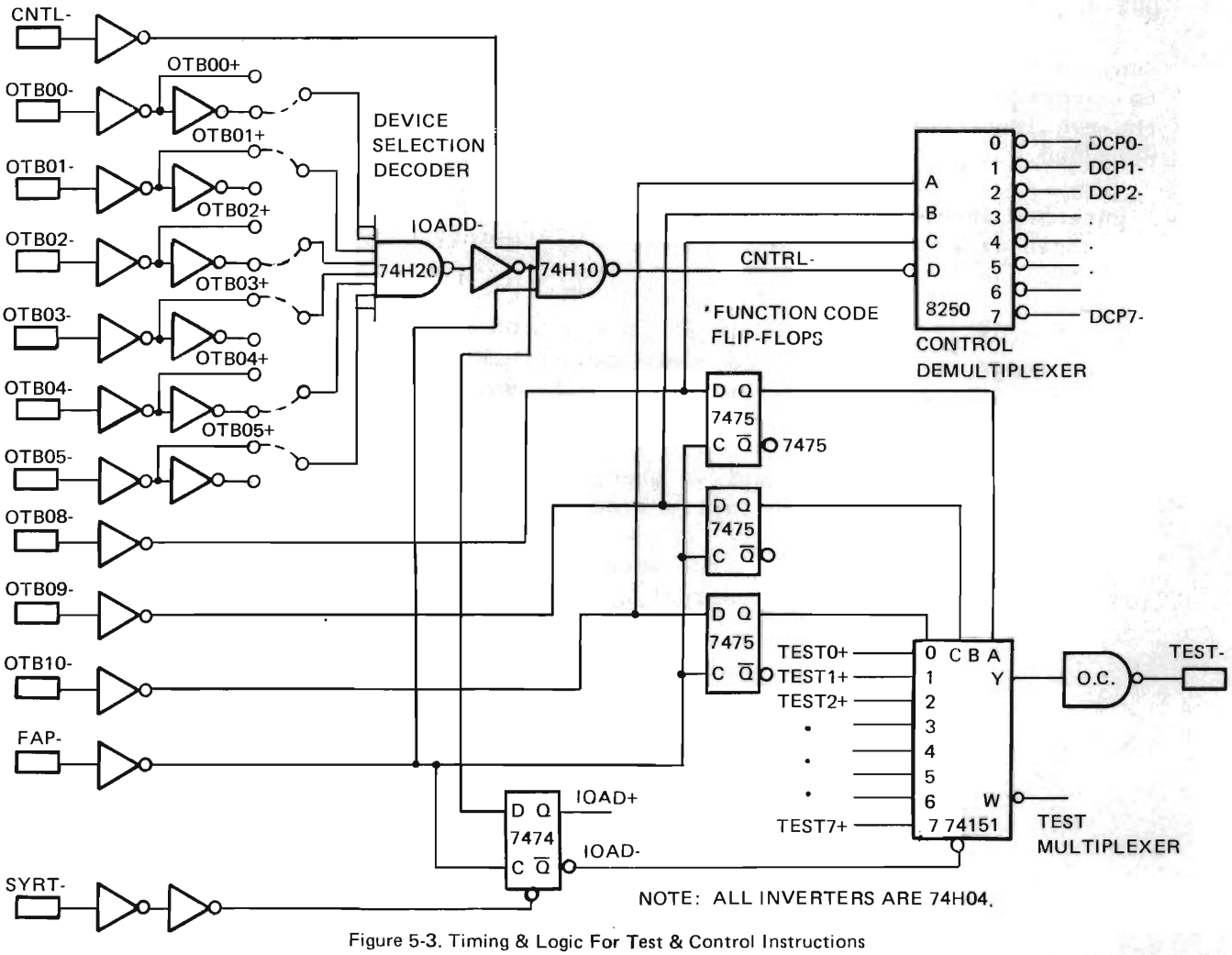


Figure 5-3. Timing & Logic For Test & Control Instructions

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.

Figure 5-4 shows what transpires when a DMA cycle occurs between the FAP and DTP cycle of a control or list I/O instruction.

Figure 5-5 includes a timing diagram for the execution of a DATA OUTPUT instruction. The logic diagram, also in Figure 5-5, 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. Also during STRY-, a WSTRB+ is generated quiescently thereby clearing (D.C. trigger on buffer flip-flops) the data output buffer since the OUT-Bus is unenergized during SYRT-.

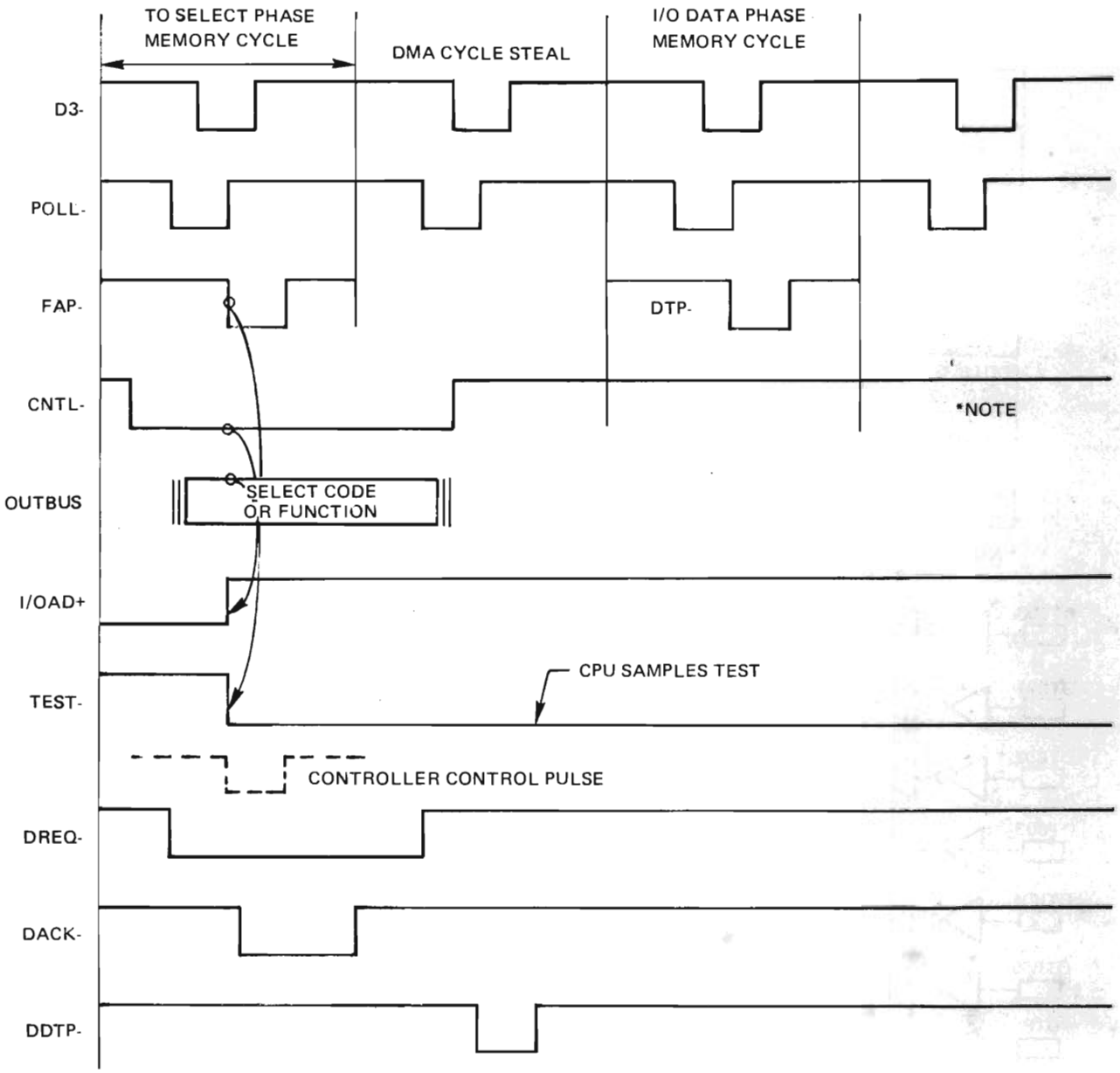
Many controllers do not use all sixteen data bits; only the OUT-Bus lines relevant to a controller need be sampled and stored. If one gate is not sufficient to drive the required number of storage flip-flops, an extra, shown with dashed lines, may be added.

Figure 5-6 includes a timing diagram for the execution of a DATA INPUT instruction. The logic diagram, also in Figure 5-6, illustrates the logic required to implement the data input function.

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).

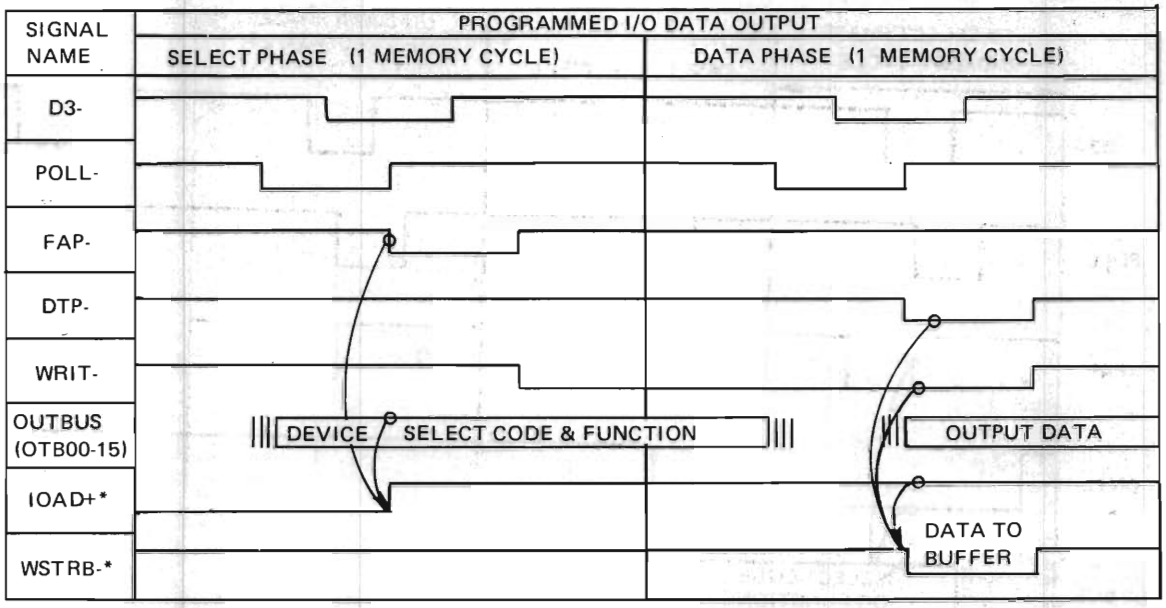
Figure 5-7 shows what transpires when a DMA cycle steal occurs between the FAP and DTP cycle of an Output I/O instruction.

Figure 5-8 shows what transpires when a DMA cycle steal occurs between the FAP and DTP cycle of an Input I/O instruction.



NOTE: The signal CNTL- is the simple logical decode of bits 6 & 7 of the OUT-Bus. CNTL- may be true (low) at times other than that shown during the FAP cycle.

Figure 5-4. XI/O Control or Test



* NOTE: THESE SIGNALS ARE INTERNAL TO THE CONTROLLER. ALL OTHER SIGNALS ARE ON I/O BUS
ALL INVERTERS ARE 74H04

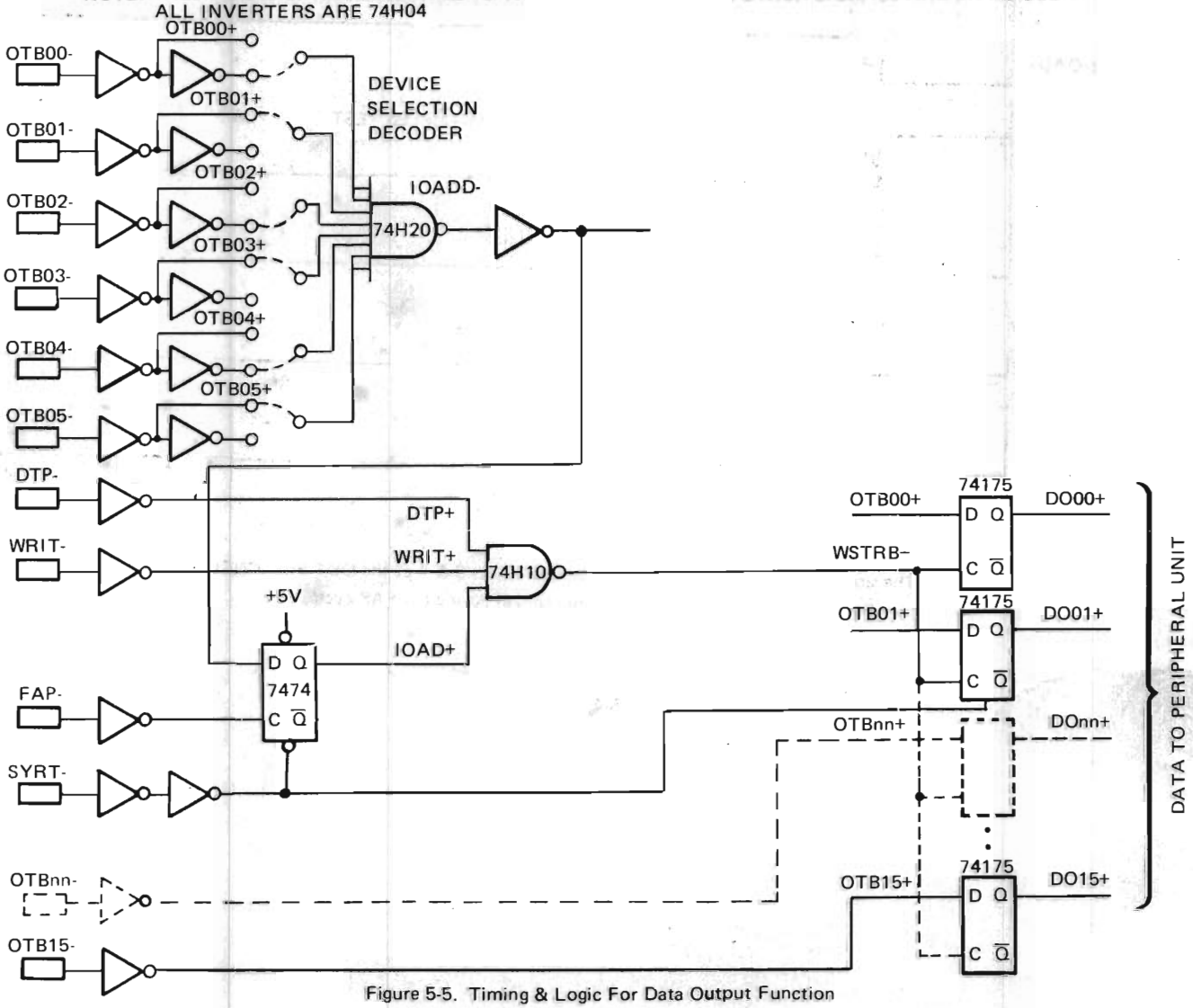
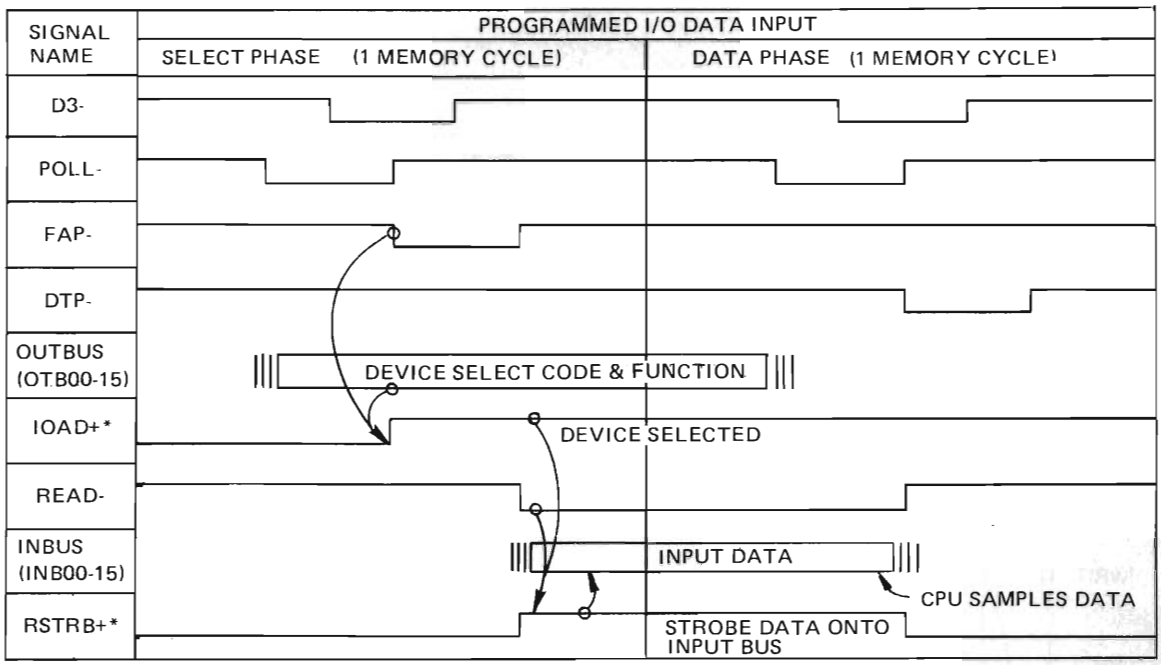
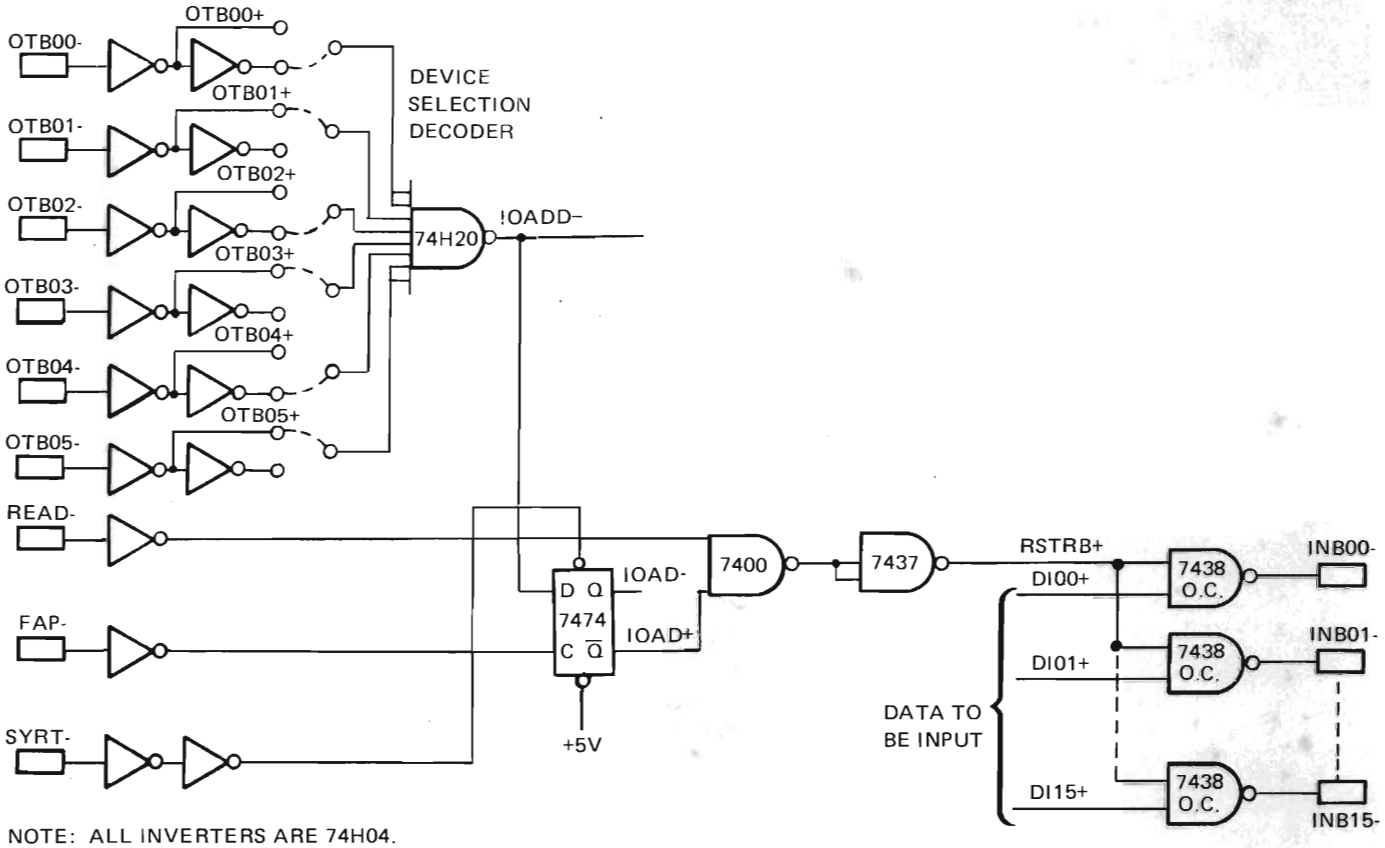


Figure 5-5. Timing & Logic For Data Output Function



* NOTE: THESE SIGNALS ARE INTERNAL TO THE CONTROLLER. ALL OTHER SIGNALS ARE ON I/O BUS



NOTE: ALL INVERTERS ARE 74H04.

Figure 5-6. Timing & Logic For Data Input Function

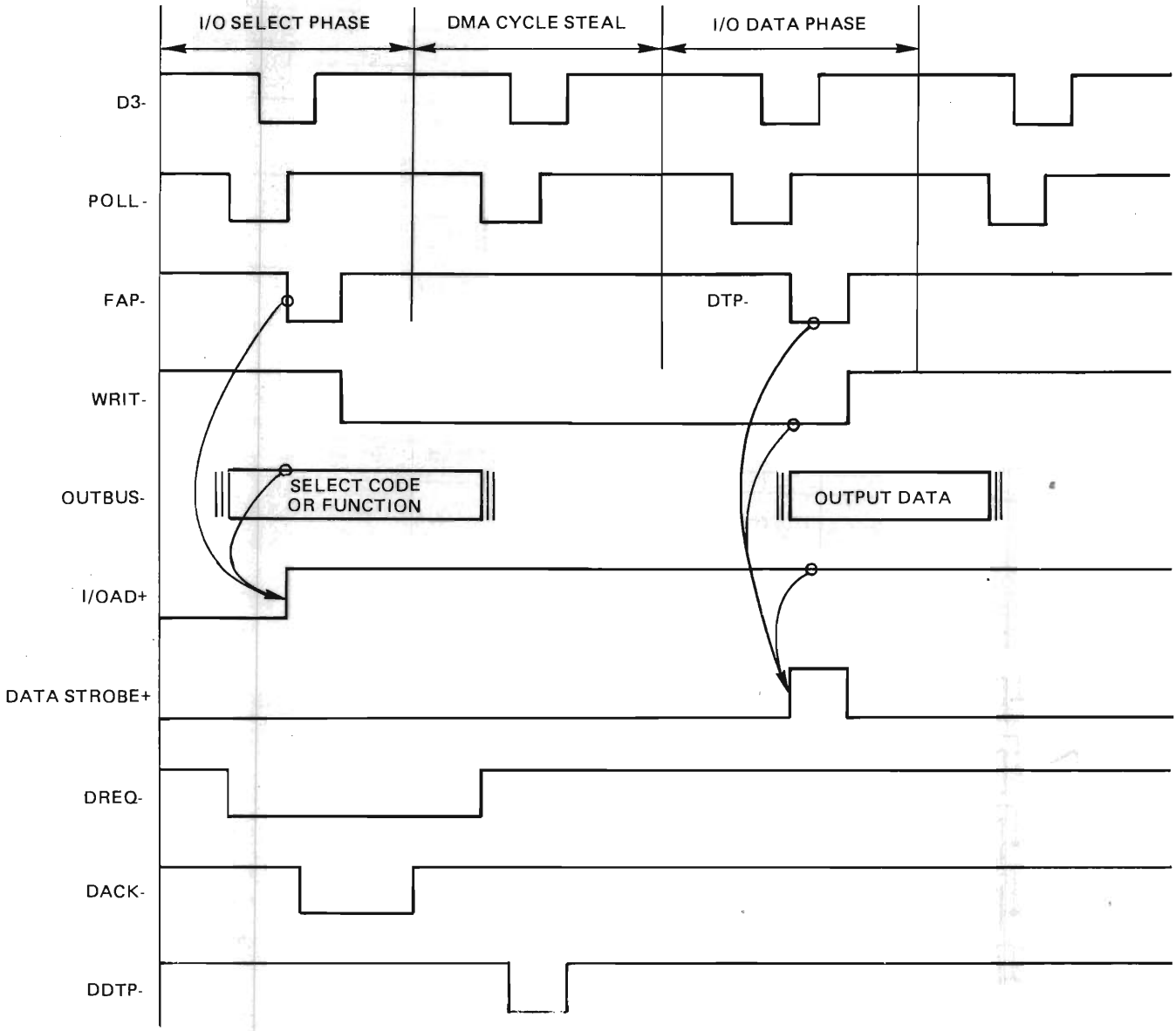


Figure 5-7. XIO Data Output

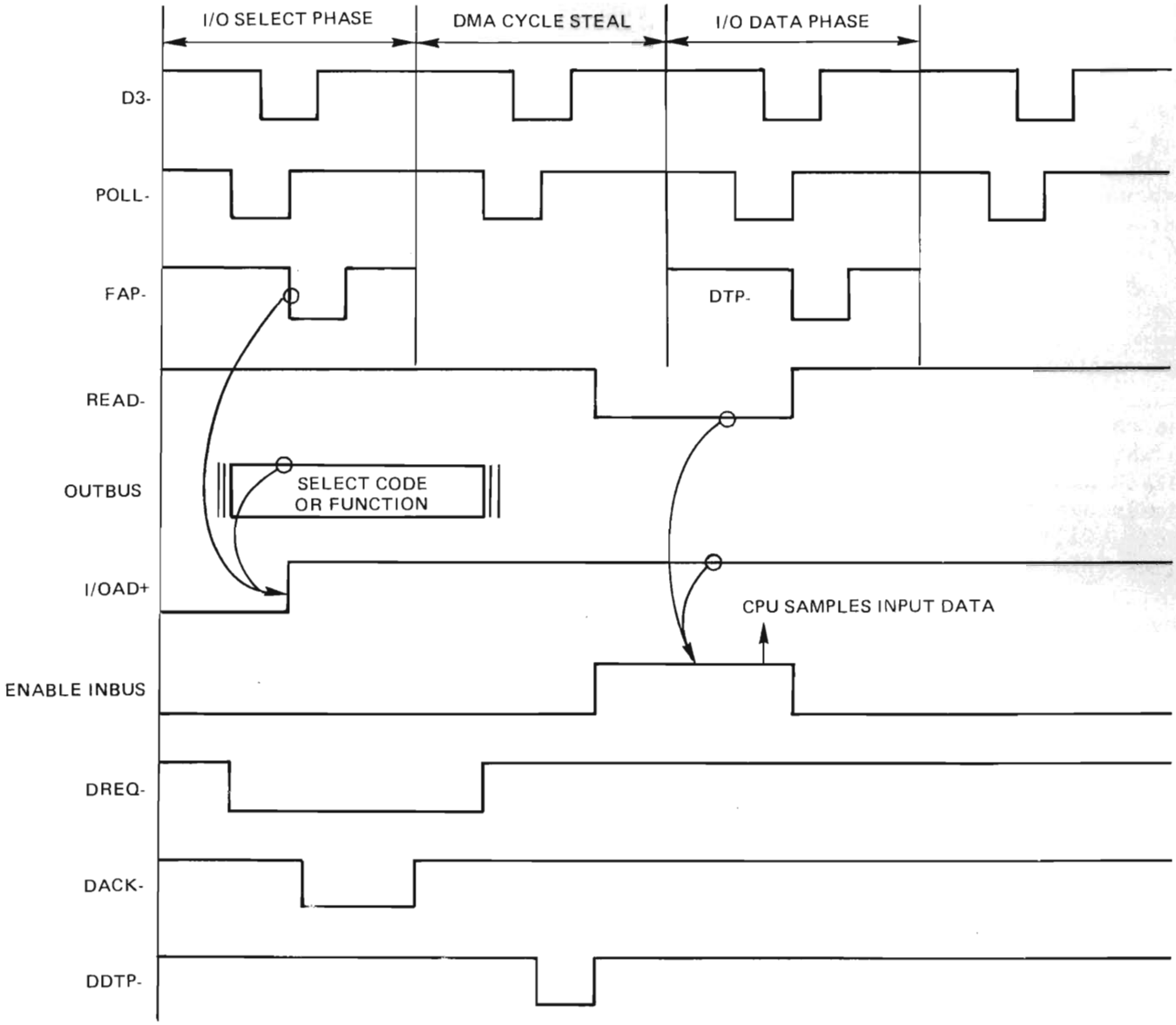


Figure 5-8. XIO Data Input

INTERRUPT SEQUENCE

There are basically three phases that occur in an interrupt sequence:

- The Request Phase
- The Priority Determination Phase
- The 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 5-9 includes a timing diagram for an interrupt sequence. The logic diagram, also in Figure 5-9 illustrates a typical scheme for implementing the interrupt logic on a controller.

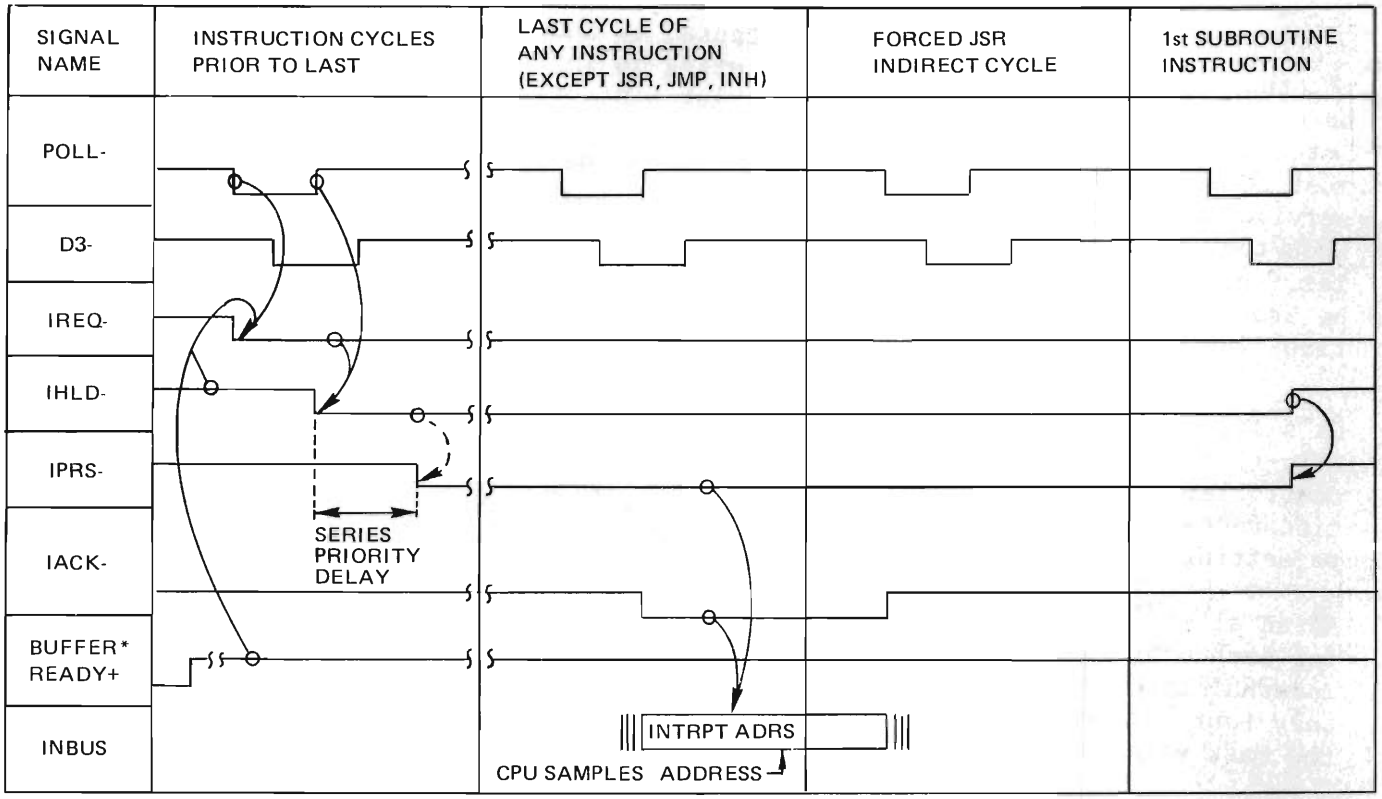
The request phase initiates an interrupt sequence and may occur in any memory cycle in which IHLD- is not energized. When a controller requires service, it brings BUFFER READY+ true. Provided the controller's interrupt mask flip-flop is set (described below), the negative-going edge of POLL-, which occurs in every memory cycle, 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.

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 this signal to the controller of next highest priority (the next signal is transmitted to the next controller in the series via its IPSO-output). 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 uses this signal in conjunction with IHLD- to energize IPRS-.

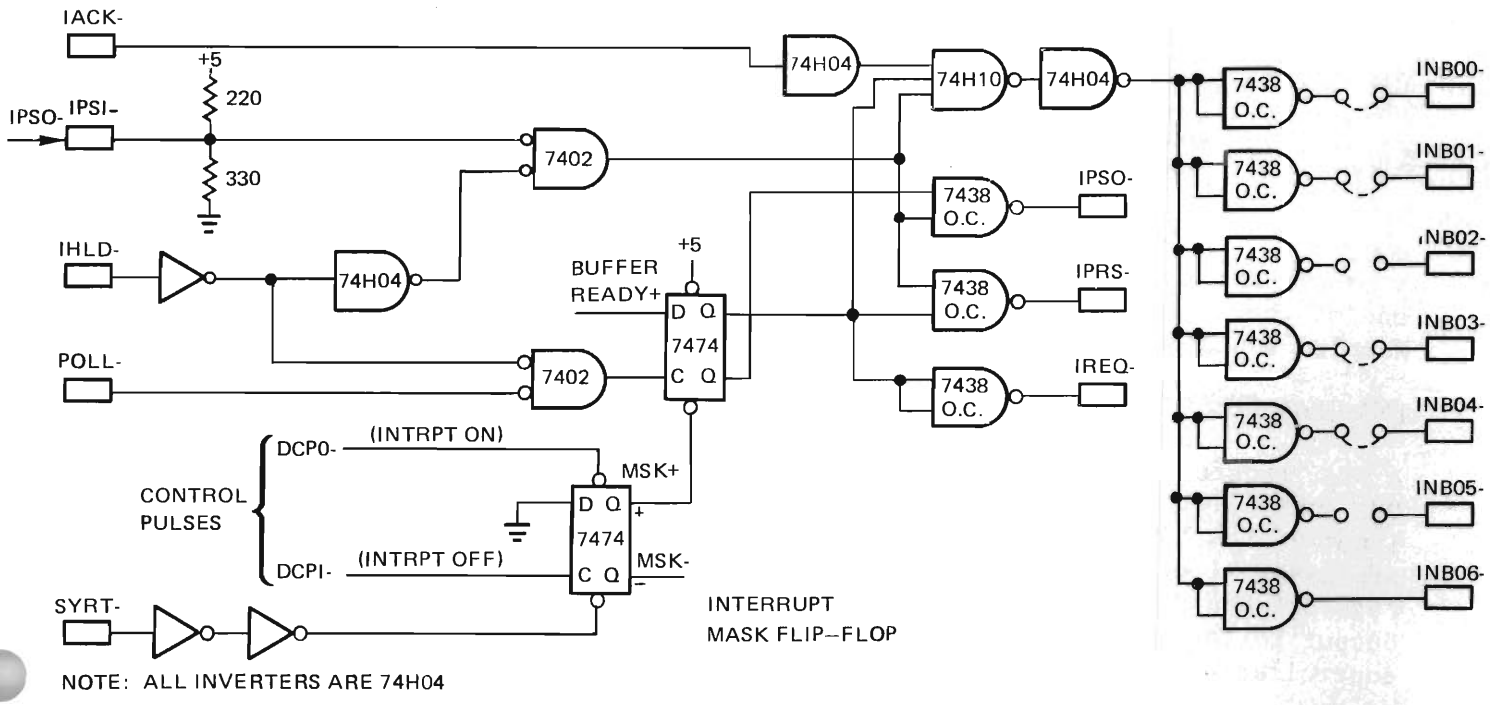
Although IPSO is not explicitly shown in the timing diagram, it's effect is shown in the delay between IHLD- and IPRS-. The delay may, conceivably, run over several memory cycles. The IPRS- signal indicates to the processor that the priority determination phase is complete.

In the last memory cycle 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 memory cycle, 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 5-9 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.



*NOTE: THIS SIGNAL IS INTERNAL TO THE CONTROLLER. ALL OTHER SIGNALS ARE ON THE I/O BUS



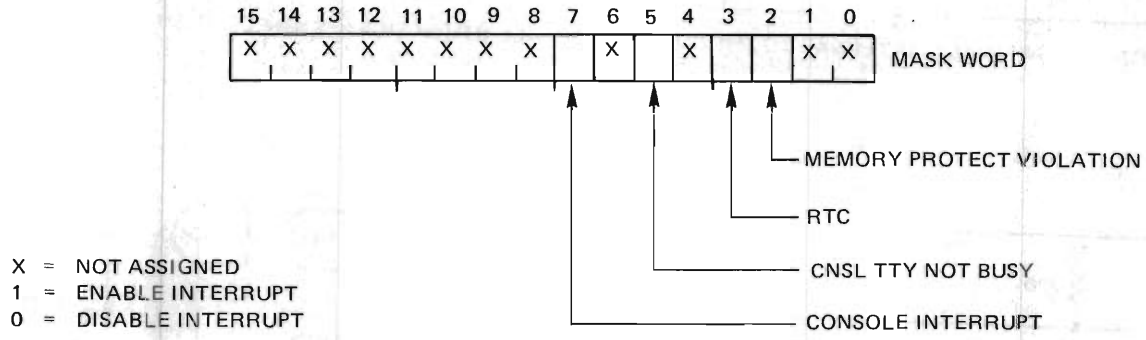
NOTE: ALL INVERTERS ARE 74H04

Figure 5-9. Logic and Timing For Program Interrupt Sequence

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 occurrance 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. As discussed in Section 4.2 internal interrupts are masked by outputting a mask-word to device X'3E'. There are only four internal interrupts and, consequently, only four of the sixteen bits of the mask word have been assigned.



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.

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. DPSO- is wired energized at the processor (i.e., DPSO- is always low). 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. The DPSO signal must be present at a controller before it may make a DMA request; when a controller makes a DMA request, it must not transmit DPSO- to the next controller in the DMA series. Each DMA device must also hold all DMA requesting if the reverse channel priority signal DCHN- is true.

The timing diagram in Figure 5-10 illustrates the signal sequence for a DMA data output. The logic diagram also in Figure 5-10 illustrates a typical scheme for implementing the DMA data output function on any controller. Figure 5-11 illustrates the timing of consecutive DMA cycles for the DMA data output function.

A DMA controller must supply the CPU with a memory address for the data item 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.

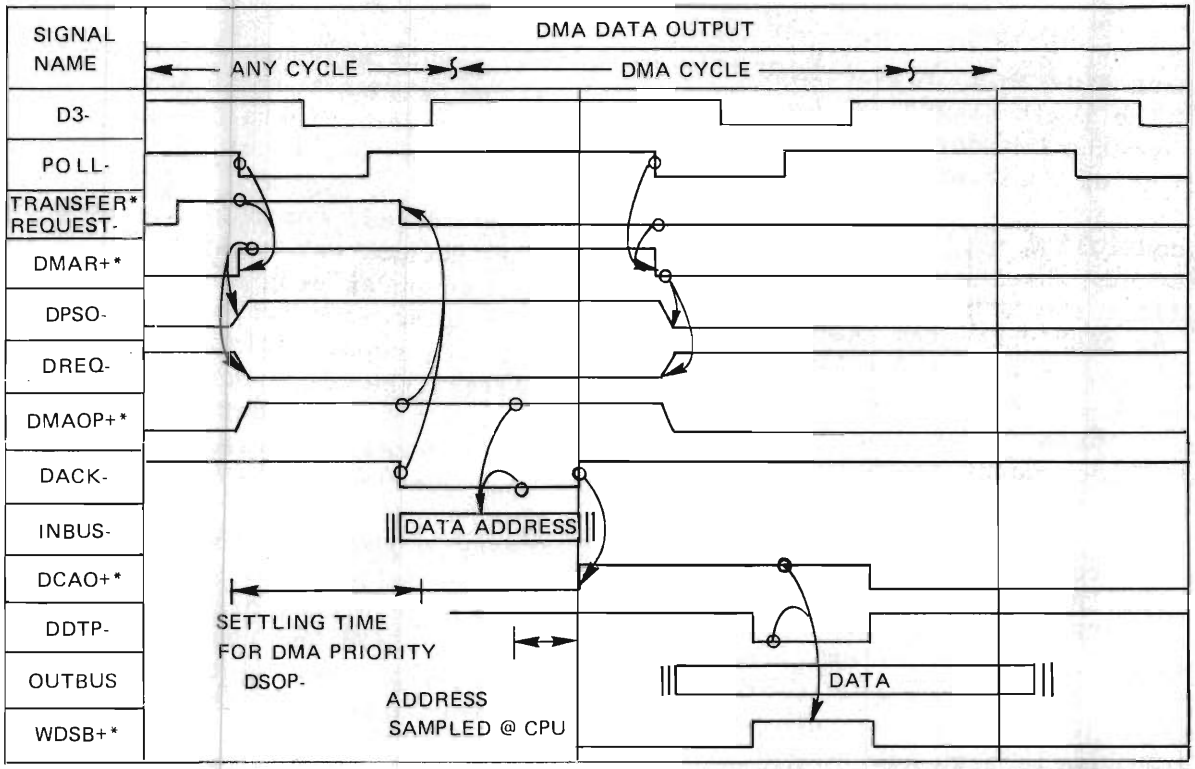
When a controller requires a DMA transfer, TRANSFER REQUEST+ on the controller becomes true. At the next occurrence of POLL-, the controller's DMA Request flip-flop is set. If DPSO- is present at the controller (i.e., no higher priority DMA controller is requesting a DMA transfer), the DREQ- line is 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 the 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 Figure 5-9).

Figure 5-12 includes a timing diagram for a DMA input of operate-on-memory operation. The logic diagram, also in Figure 5-12, illustrates the logic required, in addition to that shown in Figure 5-10, to implement the DMA output or operate-on-memory function. Figure 5-13 illustrates the timing of consecutive DMA cycles for these functions.

The DMA input request and acknowledge sequence 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:



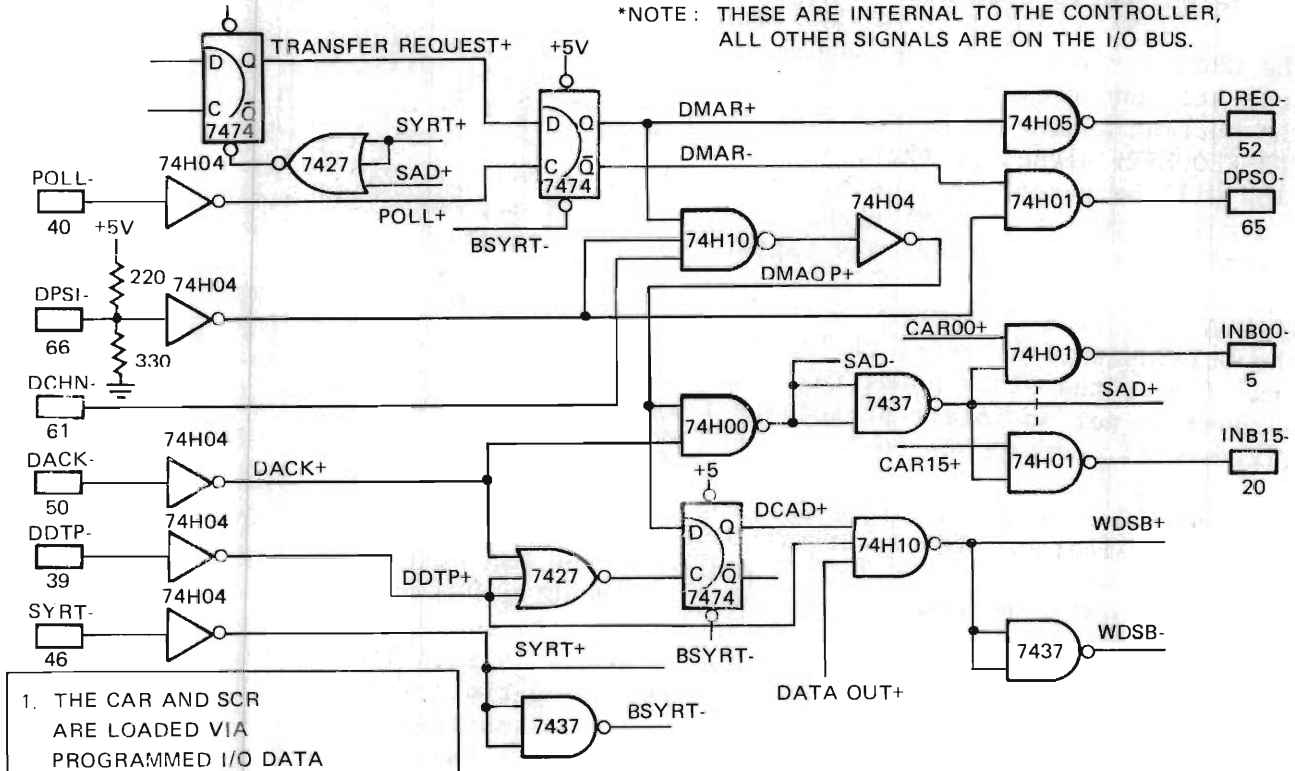


Figure 5-10. Timing and Logic for DMA Data Output.

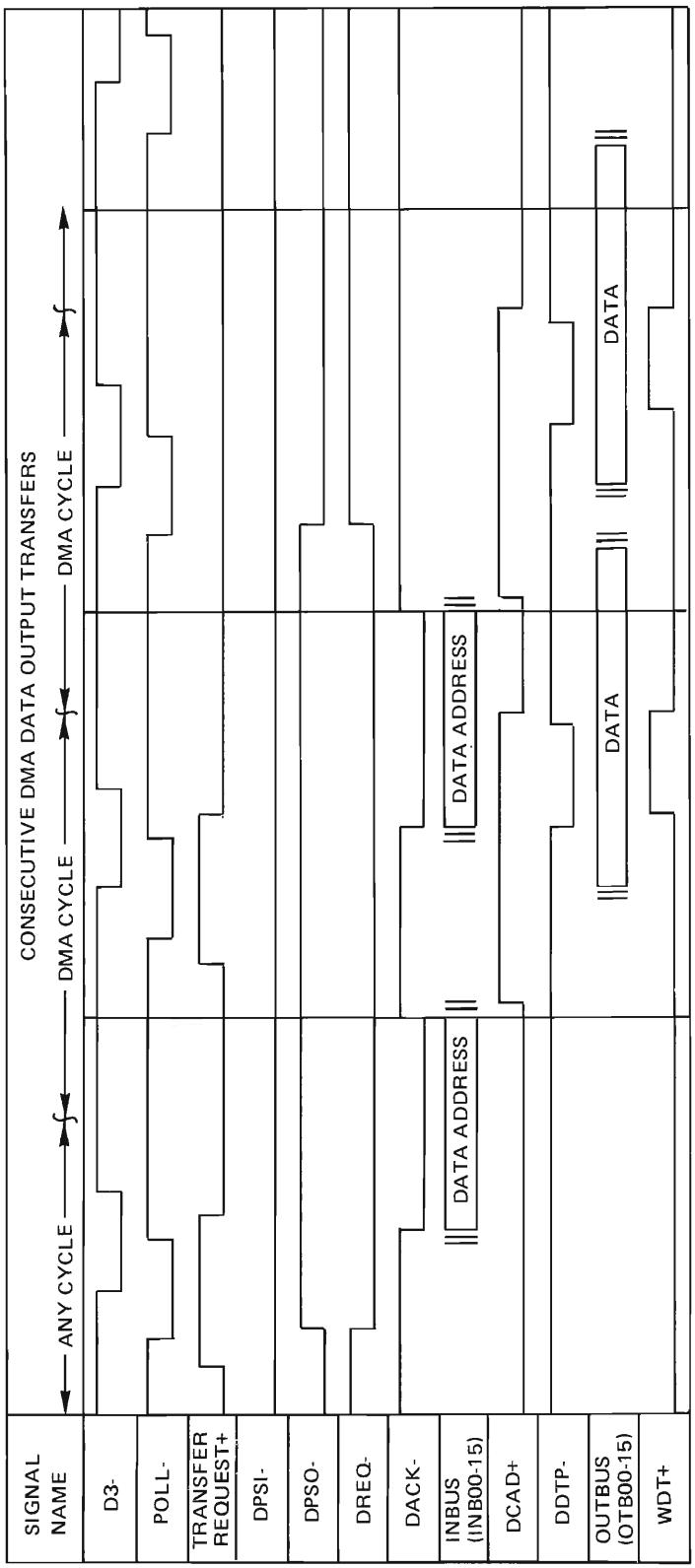
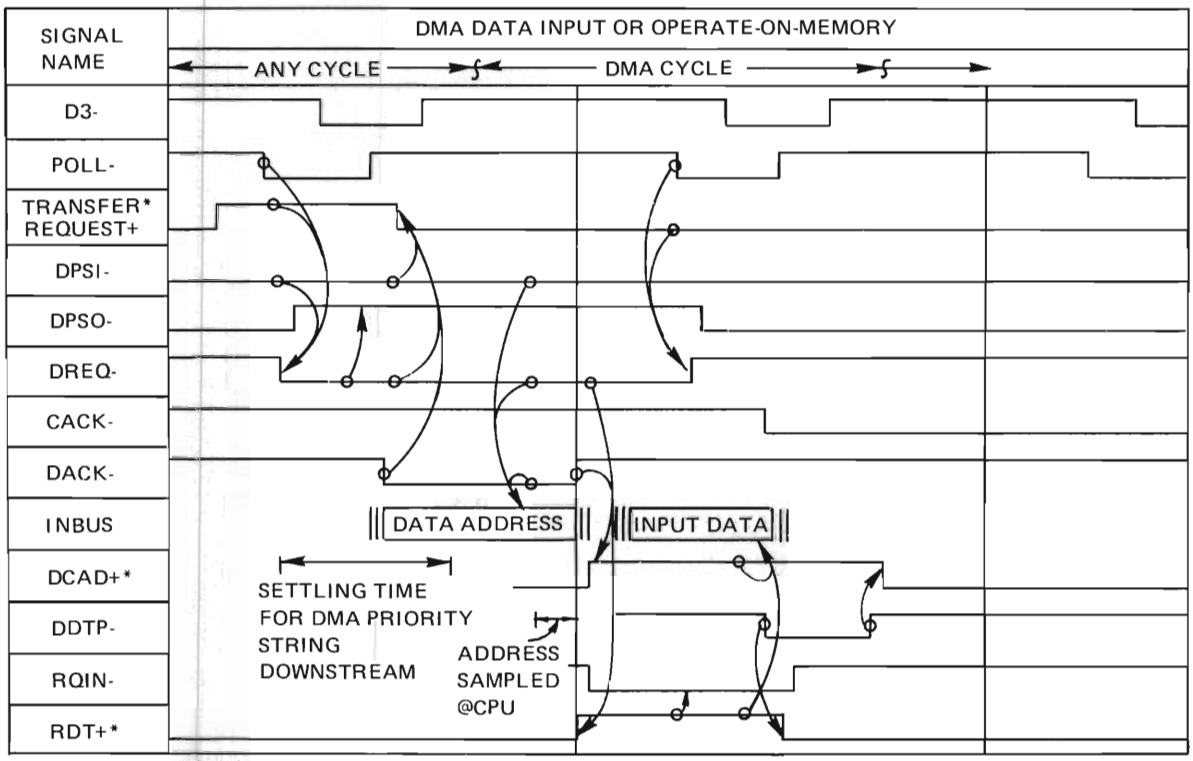


Figure 5-11. Timing for Consecutive DMA Data Output Transfers

- STROBE - which strobes the data from the controller's data input buffer onto the IN-Bus;
- RQIN - which signals the CPU that the DMA transfer is a data input operation;
- DMF0- and/or DMF1- (or neither - see the description of these signals in Section 5.1.1) which determine the type of input operation to be performed.



NOTE: DMF0- AND DMF1- HAVE SAME TIMING AS RQIN-
NOTE: * INDICATES SIGNAL IS INTERNAL TO CONTROLLER. ALL OTHER SIGNALS ARE ON I/O BUS.

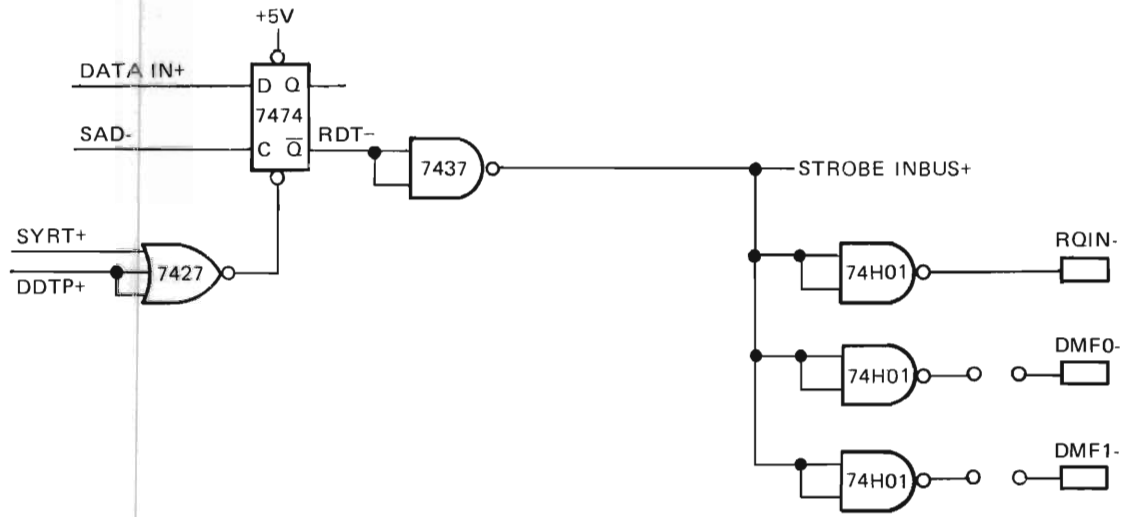


Figure 5-12. Timing and Logic for DMA Input or Operate-On-Memory.

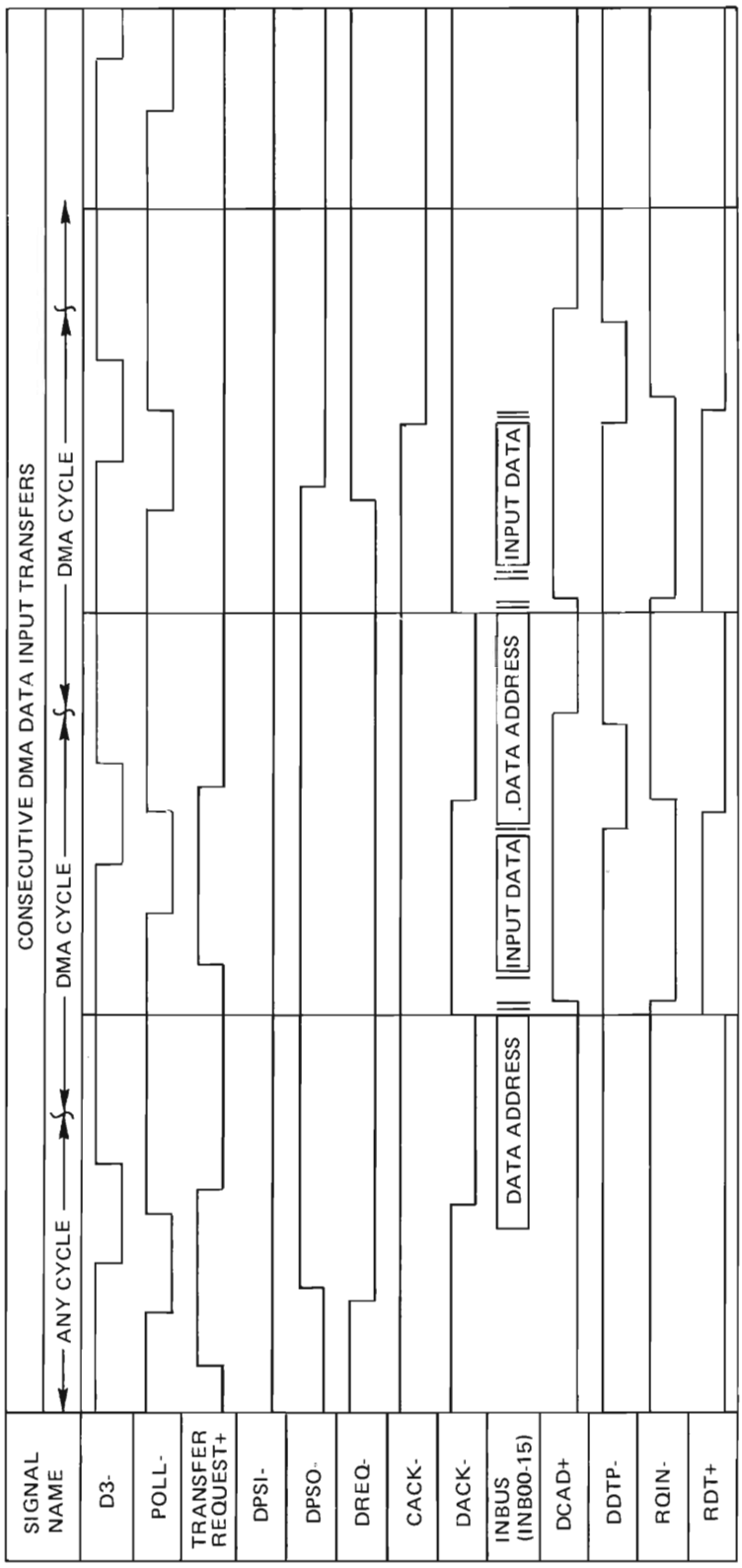


Figure 5-13. Timing for Consecutive DMA Data Input Transfers

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).

DETAILED TIMING OF I/O BUS SIGNALS

Figure 5-14 summarizes the detailed timing of the major control signals on the I/O Bus. Times are given in nanoseconds for the SPC-16 40/45 (1440 NS full memory cycle time), the SPC-16/60/65 (960 NS cycle time) and the SPC-16/80/85 (800 NS cycle time), respectively.

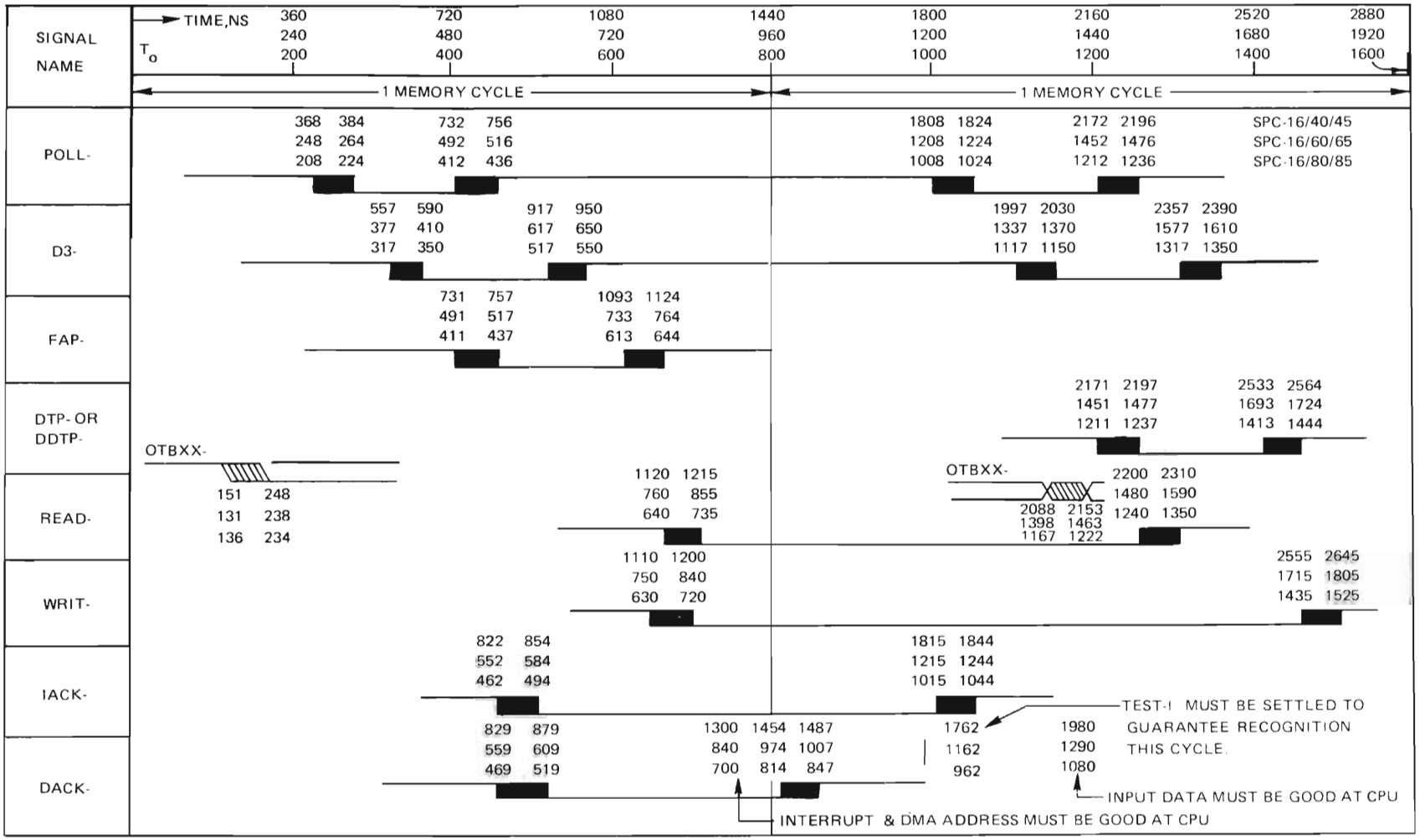
5.2 INTERFACING TO THE DATA CHANNEL BUS

As mentioned previously, General Automation implements its SPC-16/40/45 family data-channel system via a 'lumped' philosophy whereby 8-high-speed channels are contained in 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 Dedicated Memory Map, Figure 2-5).

Figure 5-15 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, or different data blocks that will appear to the controller as a continuous stream of data. The example shown in Figure 5-15 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 intelligence at the controller to chain, and controllers which chain must be designed to do so.

To start a peripheral controller block transfer, a program must initialize the dedicated memory SCR and CAR (and also any subsequent SCR's and CAR's if chaining is used) and execute the I/O control instruction that is assigned to that controller to initialize a Read or Write operation. From this point, the rest of the operation is automatic until an interrupt occurs requiring program intervention.

Aside from the fact that a "zero-word" block transfer does not exist logically, an SCR word containing all zeroes is prohibited because the decrement occurs before the chain and interrupt bits are sensed, thereby indicating a false chain.



NOTES:
1. Shaded area indicates minimum and maximum logic delays.
2. Above times are at the CPU (includes Internal I/O on the SPC-16/40/60/80).
   a. For an SPC-16/45/65/85 in the External I/O after the CIT/16: add 9 ns minimum and 27 ns maximum for each positive transition; and add 7 ns minimum and 20 ns maximum for each negative transition.
   b. For an External I/O on the SPC-16/40/60/80: add 18 ns minimum and 54 ns maximum for each positive transition; and add 14 ns minimum and 40 ns maximum for each negative transition.
3. POLL and D3 occur every cycle. During Shift and MPY/DIV instructions, they occur at twice the repetition rate shown above.
4. FAP, DTP, READ and WRIT occur only during a Programmed I/O instruction.
5. IACK occurs only when the CPU acknowledges an interrupt.
6. DACK and DDTP occur only during DMA cycles.

Figure 5-14. Detailed Timing of Signals On I/O Bus

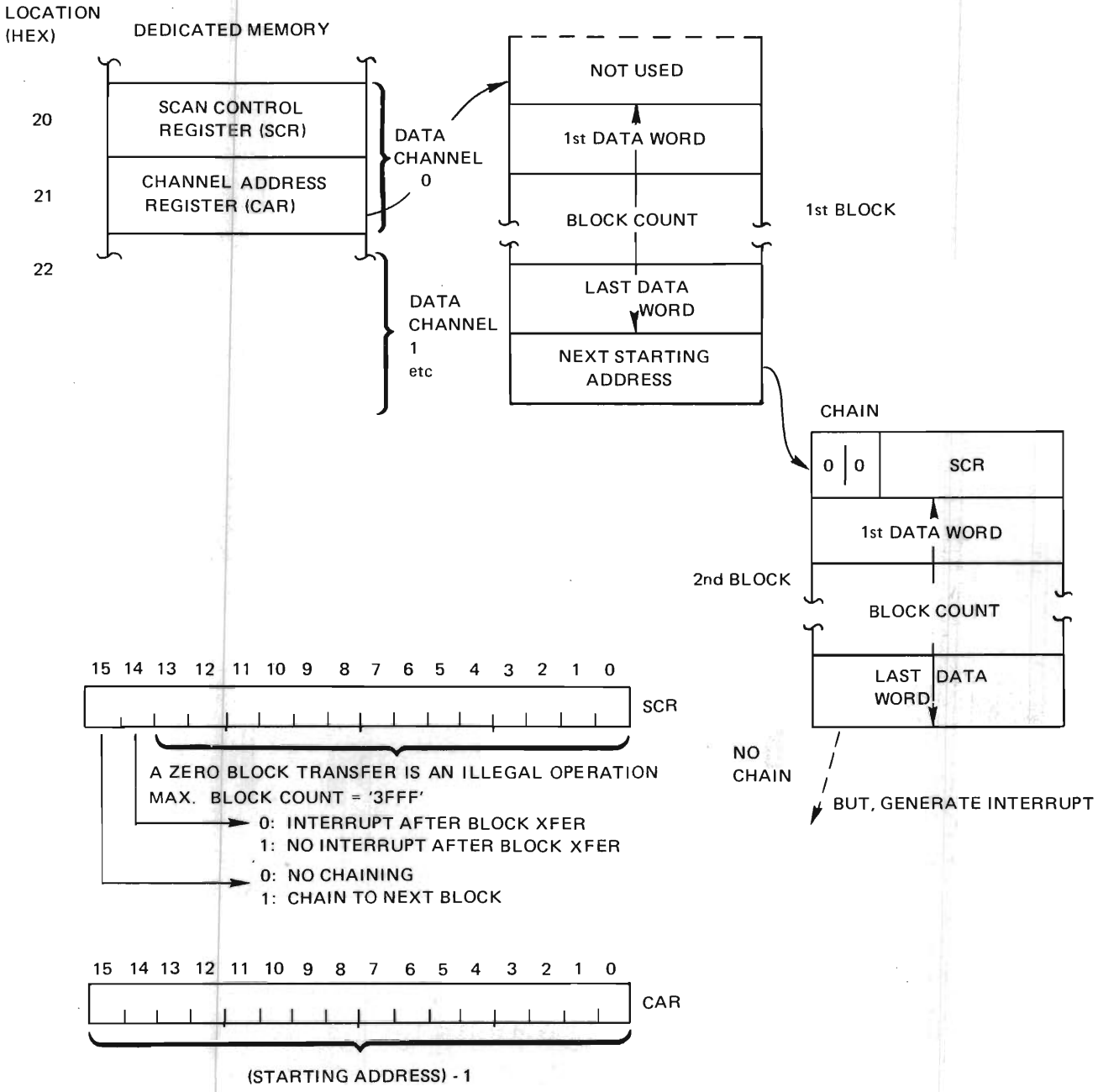


Figure 5-15. Example Block Transfer Memory Map

5.2.1 Data Channel Bus Description

DATA CHANNEL BUS SIGNALS

A Data Channel Module communicates with the CPU via the I/O Bus, bit it communicates with controllers that use its data channels via the Data Channel Bus. The controllers that use the data channels on a particular Data Channel Module are packaged in the same housing (I/O chassis) as the Data Channel Module itself; the Data Channel Bus associated with a particular module is restricted to that module's housing (I/O chassis). For example, there may be one Data Channel Bus (and module) in the mainframe (Models 40, 60 and 80) and another one in the External I/O Enclosure; the buses would be totally independent of each other. However, the Data Channel Modules have a relative priority, since they must use the same I/O Bus.

The Data Channel Bus signals are summarized below:

Signal	Direction	Definition
DCEN-	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.
DAD0- to DAD2-	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-	To controller	DMA Block Complete. Common signal returned to the controllers from the Data Channel Module indicating completion of the current block transfer.
DHLD-	To controllers	Data Channel Hold. Common signal from the Data Channel Module to the controllers that use DMA logic to freeze any further DMA requests and to enable the Data Channel Address lines (DAD0- thru DAD2-) and other control signals.
DINT-	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

Signal	Direction	Definition
DINT- (continued)		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-	To Data Channel	Data Channel Request. Common signal from the controllers to the data channel requesting a cycle steal operation.
DCSO-	To controller	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-	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	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. Also, the DCIN- line must be energized, when the controller is in any DMA mode other than transfer, i.e., in 'Add', 'Increment', or 'OR'. DCIN- is only energized by the controller with priority.

Signal	Direction	Definition
DCTP-	To Controller	Data Channel Transfer Pulse. Common signal to the controllers used to accept output data or to terminate their operation.

5.2.2 Data Channel Timing and Interfacing

A Data Channel Module can operate in three modes with respect to each of its 8 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 an I/O 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 Module retrieves the Channel Address Register (CAR) and the Scan Control Register (SCR) from the memory locations dedicated to that controller's data channel. 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 request. 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.

DATA CHANNEL INITIALIZATION SEQUENCE

Figure 5-16 illustrates the timing relationships for a controller initialization sequence. The initialization normally starts immediately following the programmed I/O (XIO) instruction that is designed to initialize the controller. The logic diagram in Figure 5-17 illustrates a typical data channel/controller interface logic needed to implement initialization as well as the data input function.

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.

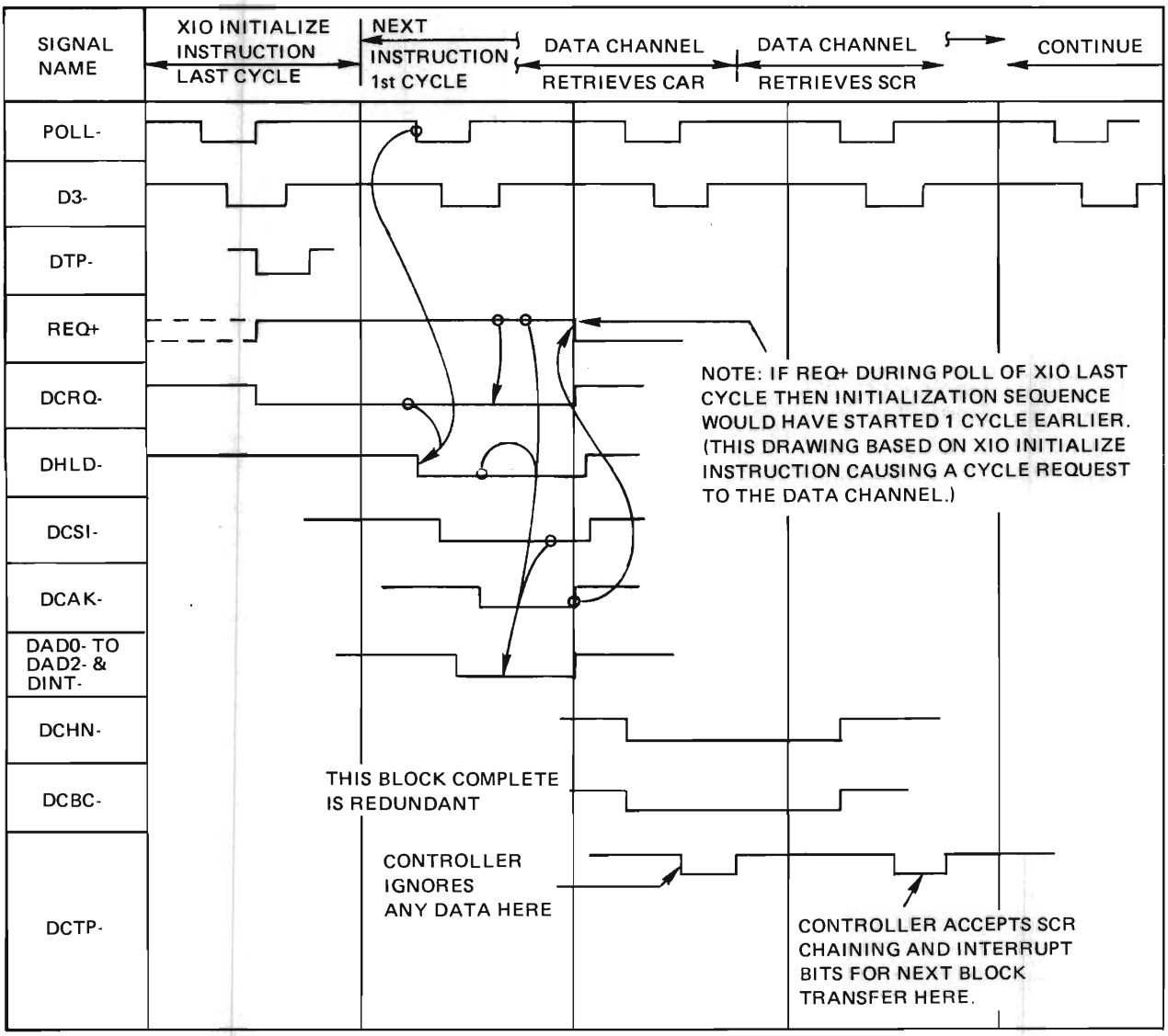


Figure 5-16. Timing for Data Channel Initialization Sequence

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 DSCI- input. If this controller is not requesting a data transfer, it retransmits the signal on its DSCO- output (DCSO- is always energized at the D.C.) to the controller of second highest priority. The highest priority controller that is requesting service traps the DCSO signal, selects 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 than 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.

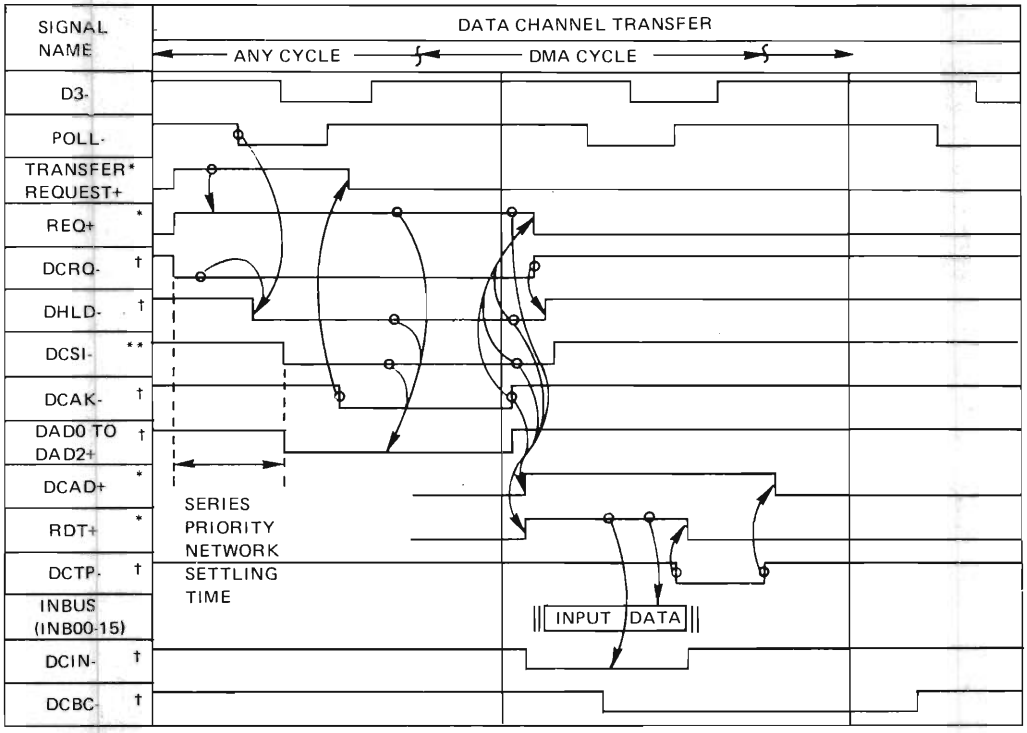
DATA CHANNEL INPUT BLOCK TRANSFER OPERATIONS

The timing diagram in Figure 5-17 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 5-17, illustrates a typical data channel/controller interface logic. Figure 5-18 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. When energizing DCIN-, the controller must also energize RDCL-, DMF0- and/or DMF1- to the CPU (see the definition of these signals in Section 5.1.1) if any of these signals are required (i.e., if an operation on memory is required).



NOTE: DMF0- AND DMF1- HAVE SAME TIMING AS DCIN-.
NOTE: * INDICATES SIGNAL IS INTERNAL TO CONTROLLER
† INDICATES SIGNAL IS ON DATA CHANNEL BUS
** SEE DCSO- SECTION 5.2.1

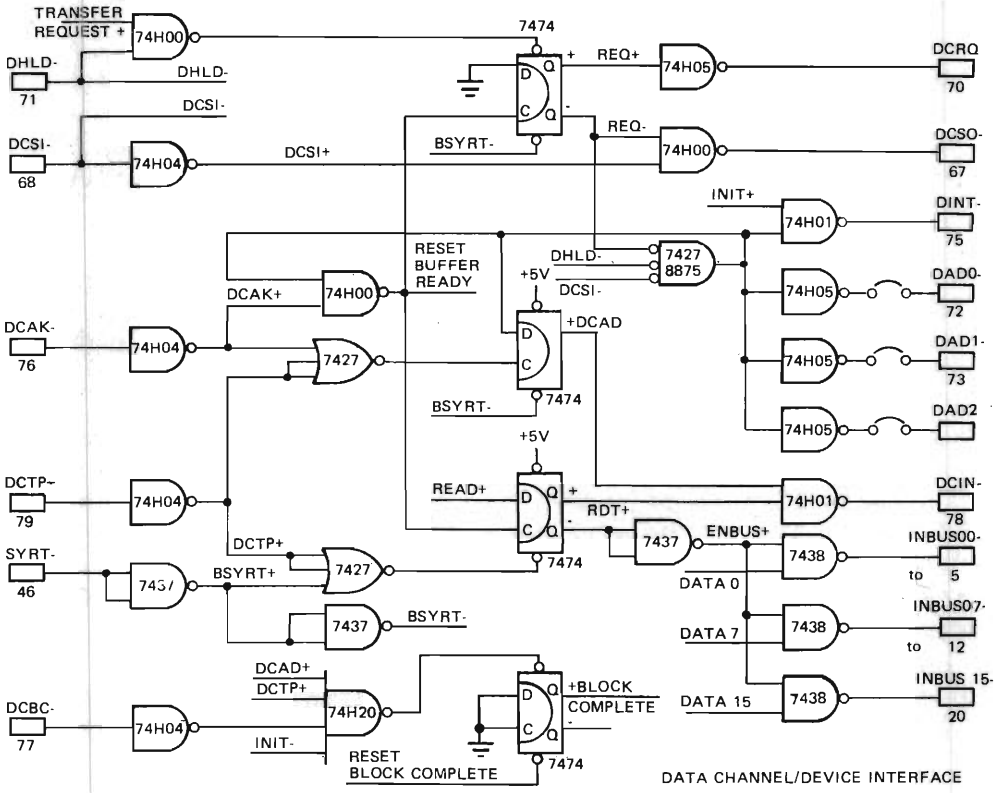


Figure 5-17. Timing and Logic for Data Channel Data Transfer (Input)

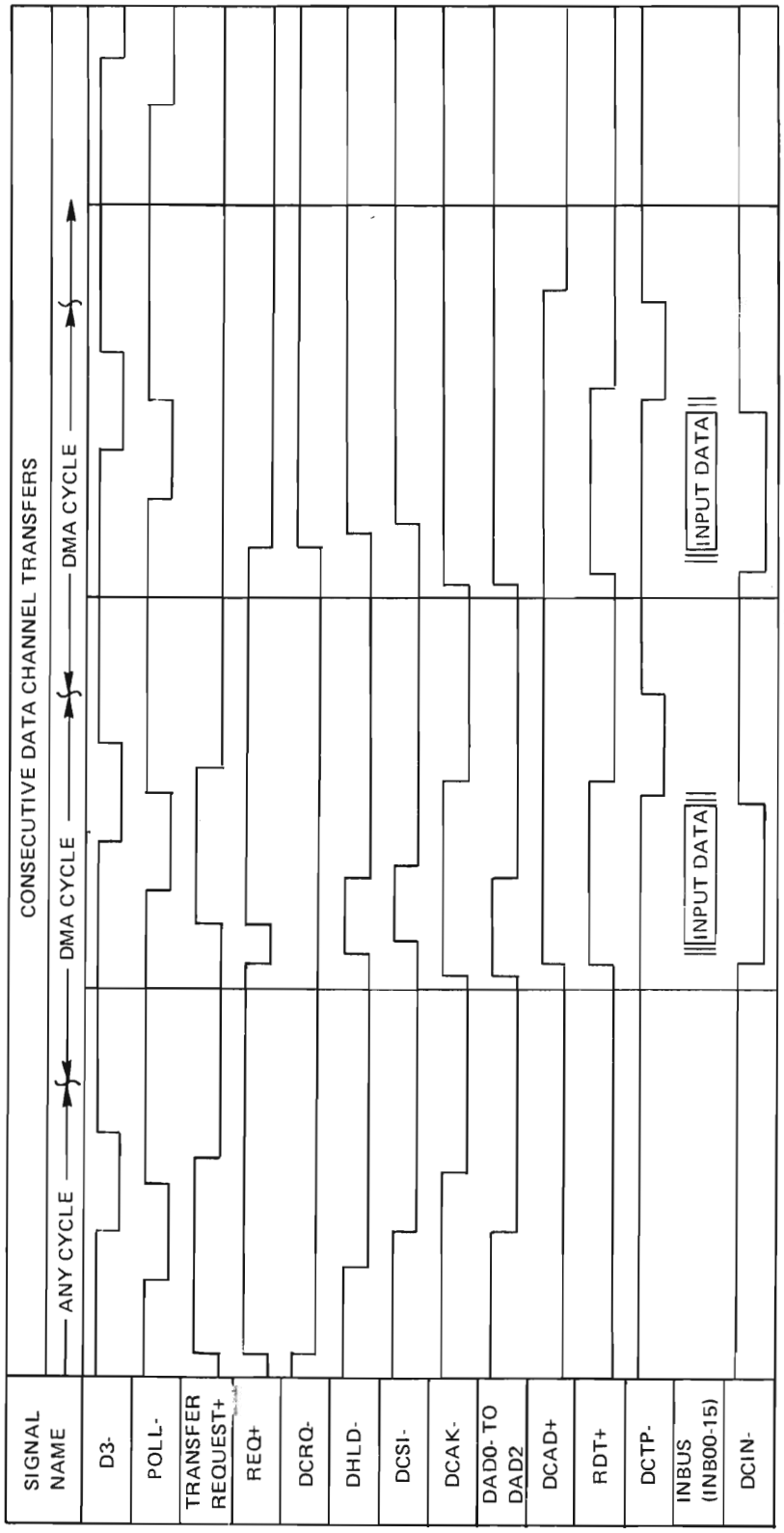


Figure 5-18. Timing for Consecutive Data Channel Data Transfers (Input)

The Data Channel Module energizes DCTP-. On the negative going edge of this signal, the controller clears RDT+, thereby disabling DCIN- (as well as DMF0 and DMF1) 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 becomes zero, then 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 (see Data Channel/ Controller Chaining Sequence below for exceptions).

DATA CHANNEL OUTPUT BLOCK TRANSFER OPERATIONS

The timing diagram in Figure 5-19 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 5-19, illustrates the logic required, in addition to that shown in Figure 5-17, needed to implement the data output function. Figure 5-10 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.

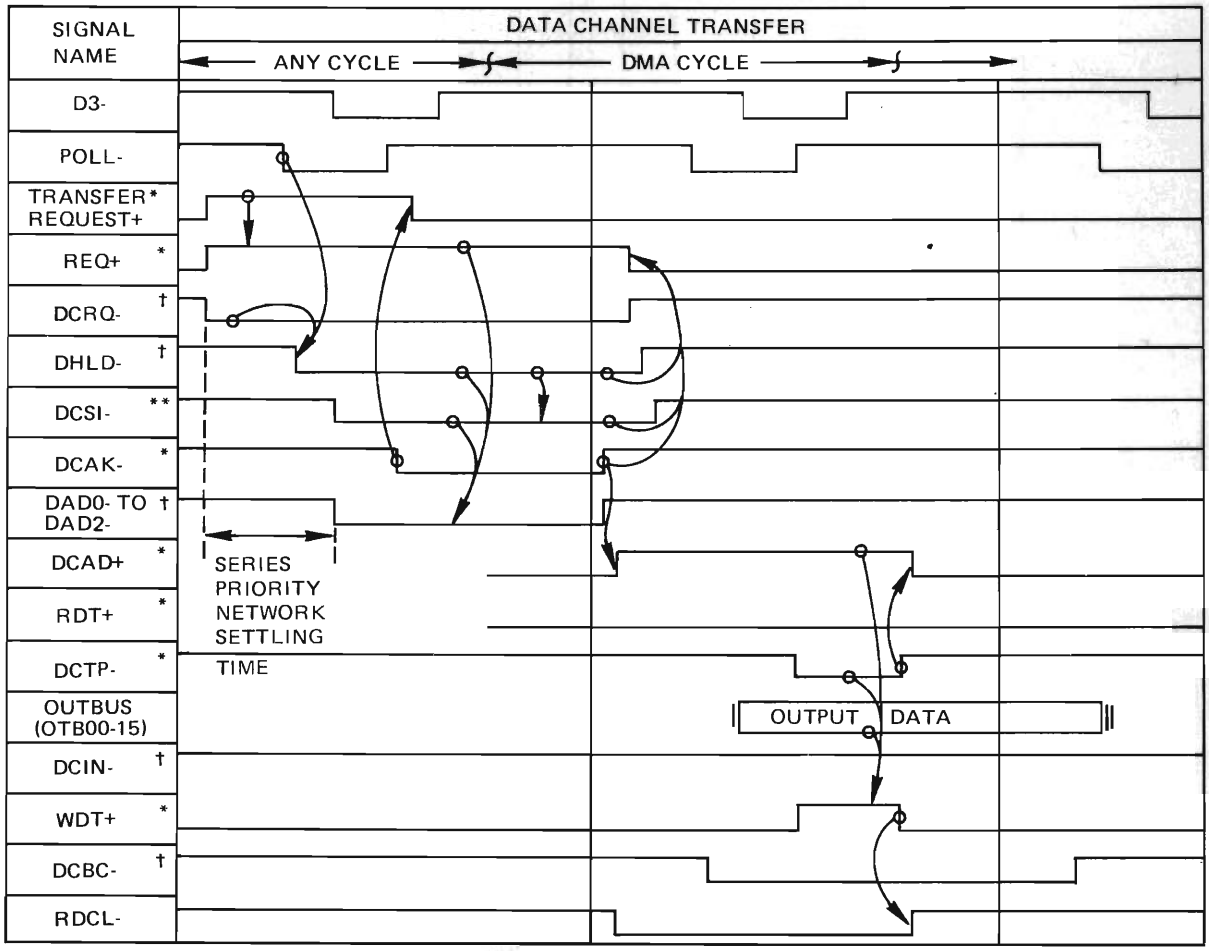
The Data Channel Module then generates DCAK-. The controller should use the negative-going edge of DCAK- to reset TRANSFER REQUEST+ and the positive-going edge of DCAK- to set it "select" flip-flop (DCAD+) and reset the REQ+ flip-flop.

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 becomes 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 (see Data Channel/Controller Chaining Sequence below for exceptions).

DATA CHANNEL/CONTROLLER CHAINING SEQUENCE

Figure 5-21 illustrates the timing relationships for a device chaining sequence. 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.



NOTE: * INDICATES SIGNAL IS INTERNAL TO CONTROLLER
† INDICATES SIGNAL IS ON DATA CHANNEL BUS
** SEE DCSO-I, SECTION 5.2.1

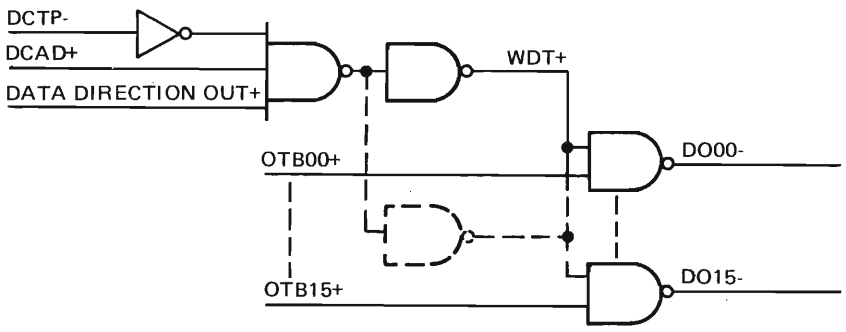


Figure 5-19. Timing and Logic for Data Channel Data Transfer (Output)

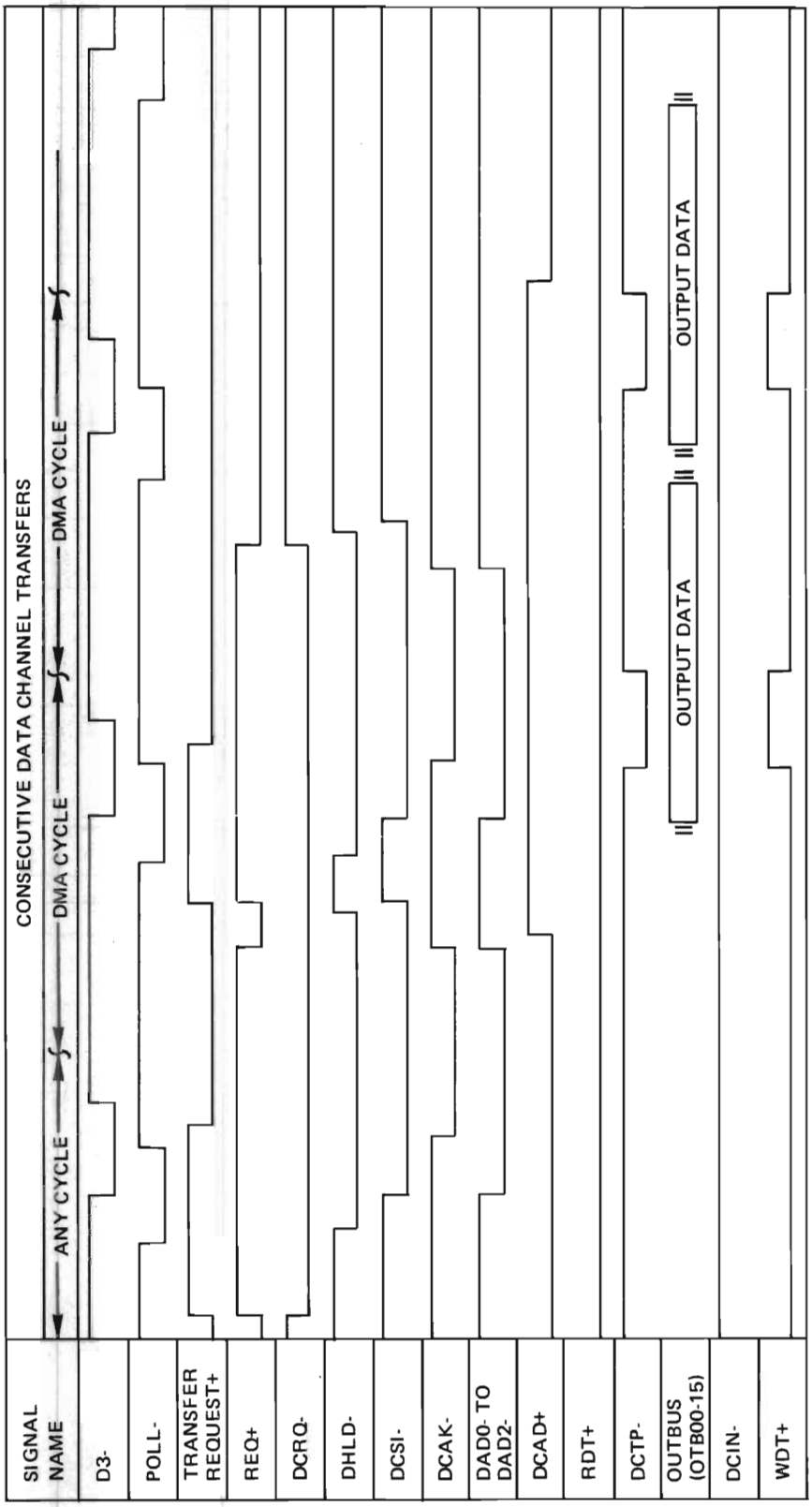


Figure 5-20. Timing for Consecutive Data Channel Data Transfers (Output)

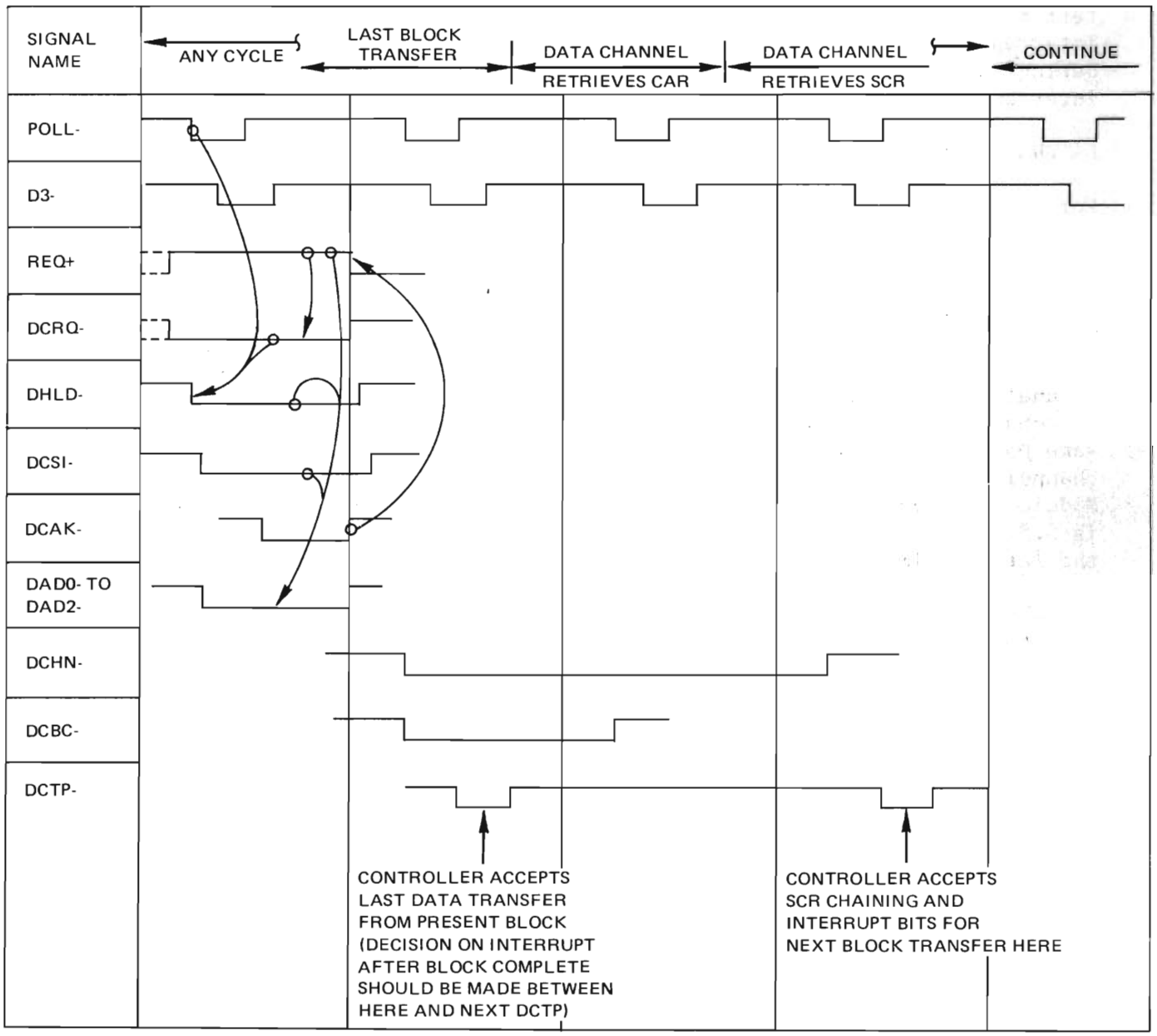


Figure 5-21. Data Channel/Controller Chaining Sequence

The left part of the timing diagram illustrates the signals which occur during the last data transfer from the present block. After the last data 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 interrupt bit (stored by the controller during the last chaining or initiate operation), the controller must initiate this interrupt before it stores the new SCR chaining and interrupt bits.

LATENCY TIME

For a block transfer DMA cycle steal, the minimum latency time (time between TRANSFER REQUEST going true and the trail edge of DDTP for the desired cycle steal) is 0.5 memory cycles and the maximum latency time is 1.5 memory cycles.

During an initialization or chaining sequence the active channel "locks out" all other channels until the sequence is completed. Also, a channel cannot steal a cycle immediately following an initialization or chaining sequence of another channel on the same Data Channel Module. Therefore, the maximum latency time when one channel could be initializing and another channel (of higher priority on the same Data Channel Module) is requesting is 4.5 memory cycles. Also, when one channel is chaining and another channel (of higher priority on the same Data Channel Module) is requesting, the maximum latency time (for the higher priority channel) is 5.5 memory cycles. If the two channels are on different Data Channel Modules, the latency time is reduced by 1 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, it takes one memory cycle in addition to the latency time.

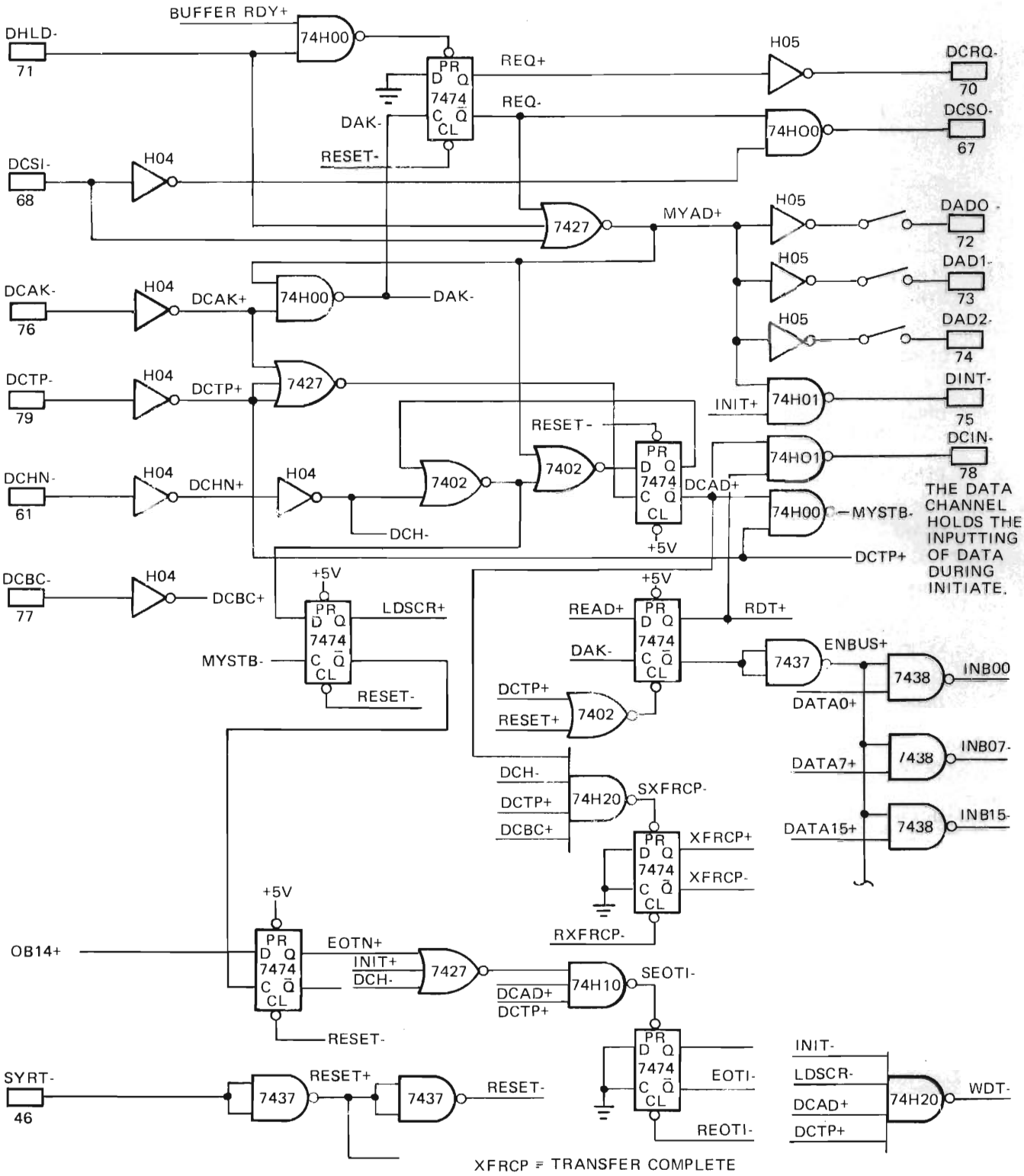


Figure 5-22. Standard Controller to Data Channel Interface Featuring Logic to Accommodate All Cases

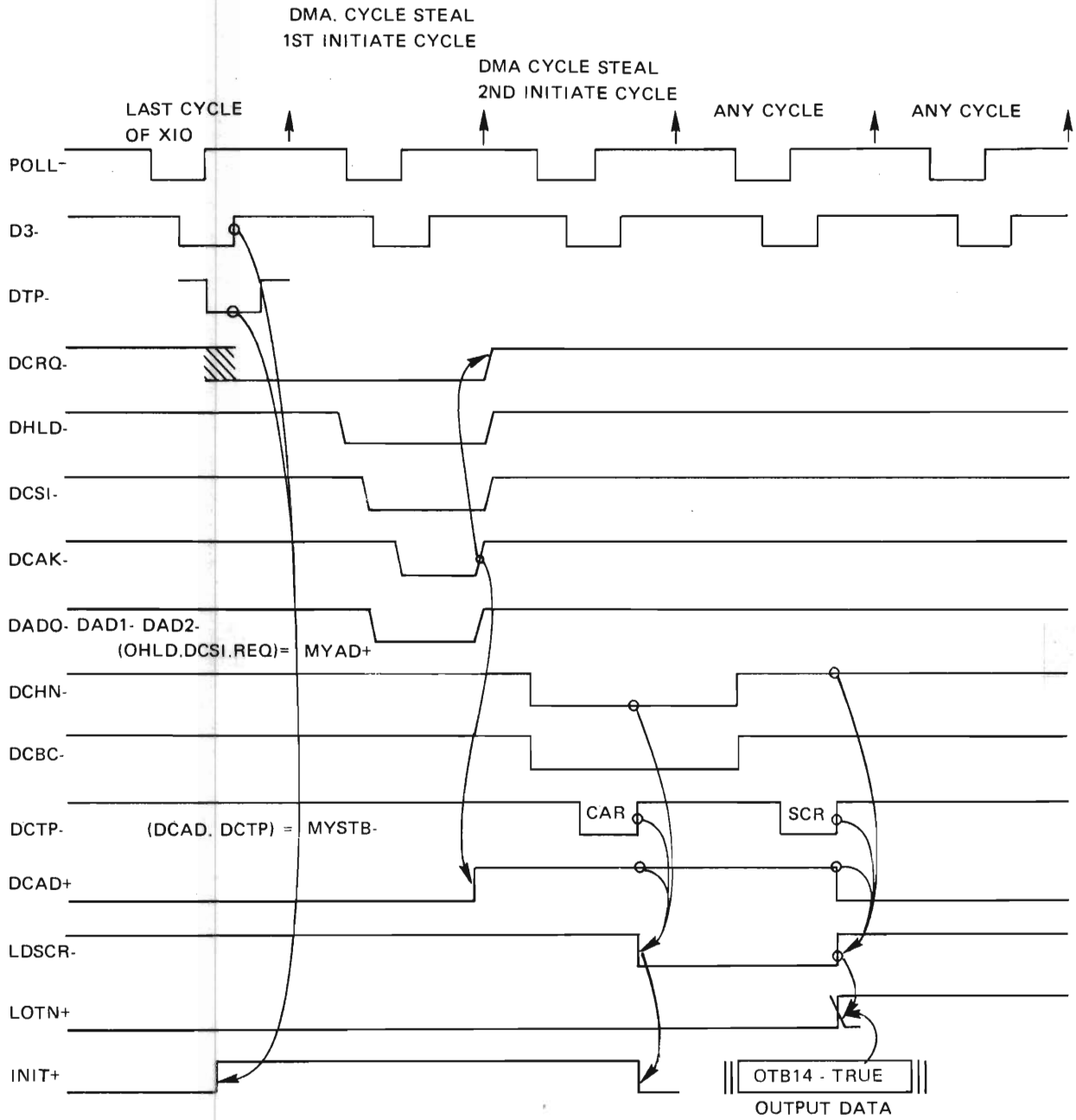


Figure 5-23. Initiate Timing

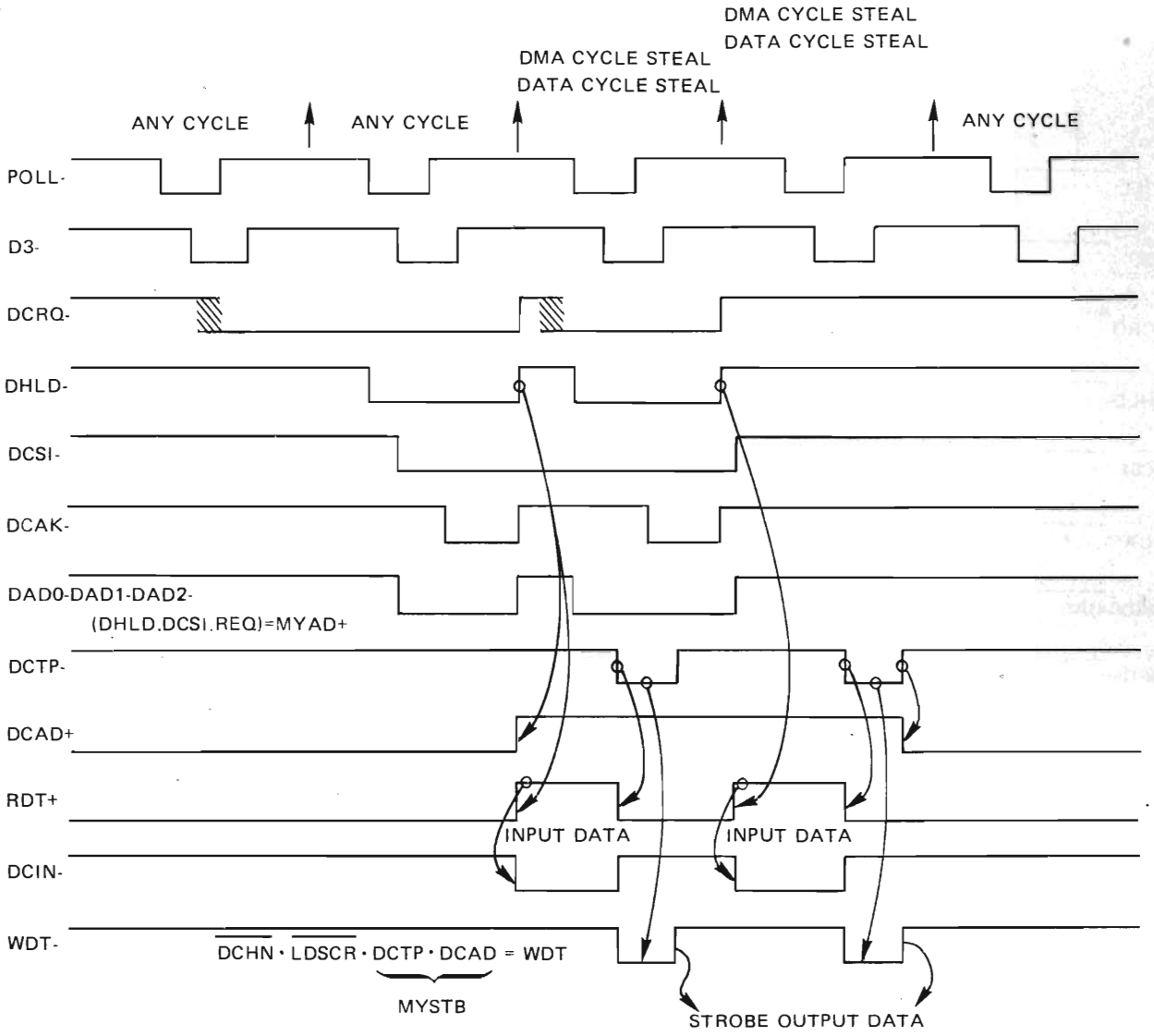


Figure 5-24. Normal Cycle Stealing.

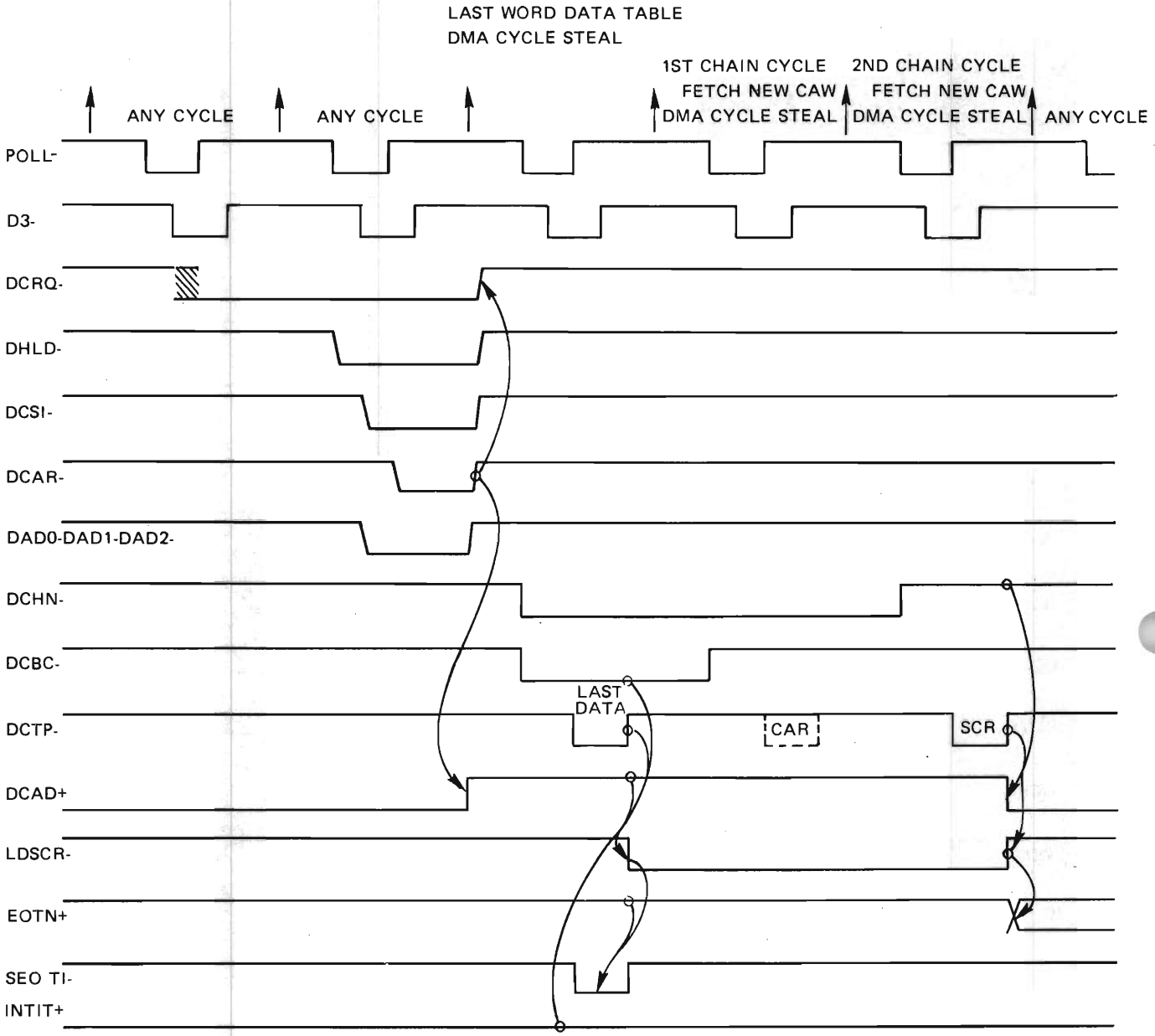


Figure 5-25. Chain Timing

5.3 PHYSICAL DESCRIPTION OF I/O SYSTEM

Card slots for eight I/O controller boards are provided in the mainframe of SPC-16/40, 60 and 80 models. For systems requiring additional card slots, one or more External I/O Enclosures may be added. With SPC-16/45, 65 and 85 models, all controller cards are accommodated in one or more External I/O Enclosures. Each External I/O Enclosure provides card slots for up to eighteen I/O Controller boards.

All computer models have a swing-away hinged front console; CPU and memory boards (and I/O controller boards on Models 40, 60 and 80) may be inserted or removed from the front. Figure 5-26 illustrates the mainframe circuit board layout for Models 40, 60 and 80 and Models 45, 65 and 85 and the I/O controller board layout in an External I/O Enclosure.

As illustrated for the Model 40, 60 and 80 enclosures, as many as four memory boards, four CPU boards and up to eight I/O controller boards may plug into the edge connectors provided on the front surface of the Master Interconnect Board (MIB). The I/O Bus and the Data Channel Bus lines are etched on the MIB. Cables from peripheral units have paddle-boards that plug into 56-pin edge connectors at the rear of the MIB at a position corresponding to their respective controller card. When an External I/O Enclosure is required, one internal card slot must be devoted to a Cable Interface Driver (CID) card; a special 72-pin edge connector must be installed on the rear of the MIB, at a position corresponding to the CID, to accommodate the External I/O Bus cable paddle board.

As illustrated for Models 45, 65 and 85 enclosures, as many as eight memory boards and four CPU boards plug into the edge connectors of the front surface of the MIB board(s) (for expansion beyond four memory boards, an extension to the main MIB is required.) The External I/O Bus cable paddle board plugs into the 72-pin edge connector provided on the rear surface of the main MIB.

An External I/O Enclosure will accommodate up to eighteen I/O controller boards. 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 External I/O Bus cable enters the External I/O enclosure via one 72-pin edge connector and leaves the enclosure (if an additional enclosure is required) via another 72-pin edge connector, both of which are provided on the rear surface of the MIB.

Figure 5-27 illustrates the relationship between pins on the 140-pin edge connector that accepts the controller board and pins on the 56-pin edge connector that accepts the paddle board via which the cable to the a-sociated peripheral unit is connected. This diagram applies equally to the Integral I/O package (Model 40, 60 and 80) and the External I/O Enclosures.

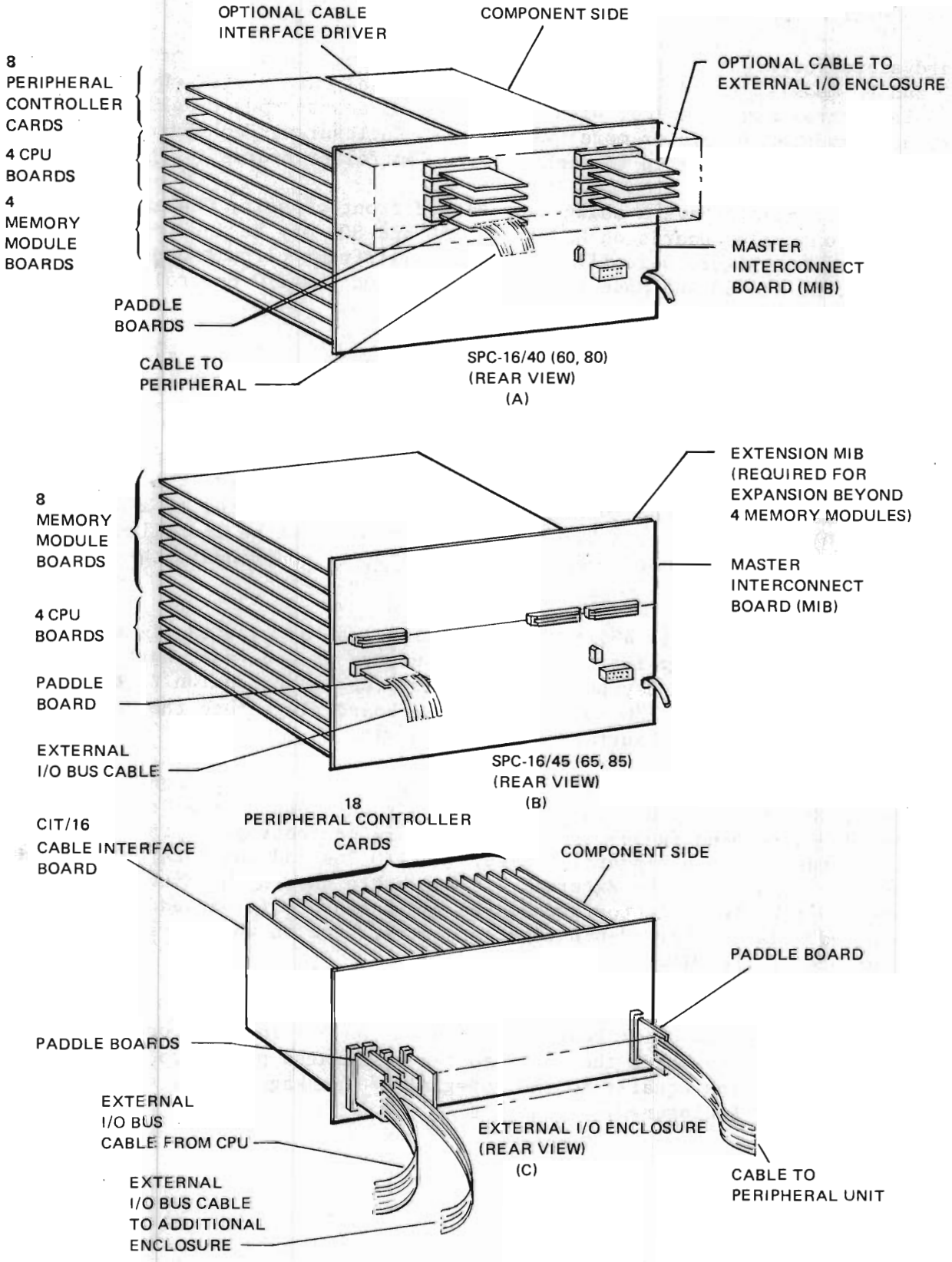


Figure 5-26. Circuit Board Layout Models 40, 60, 80; Models 45, 65, 85; External I/O Enclosure

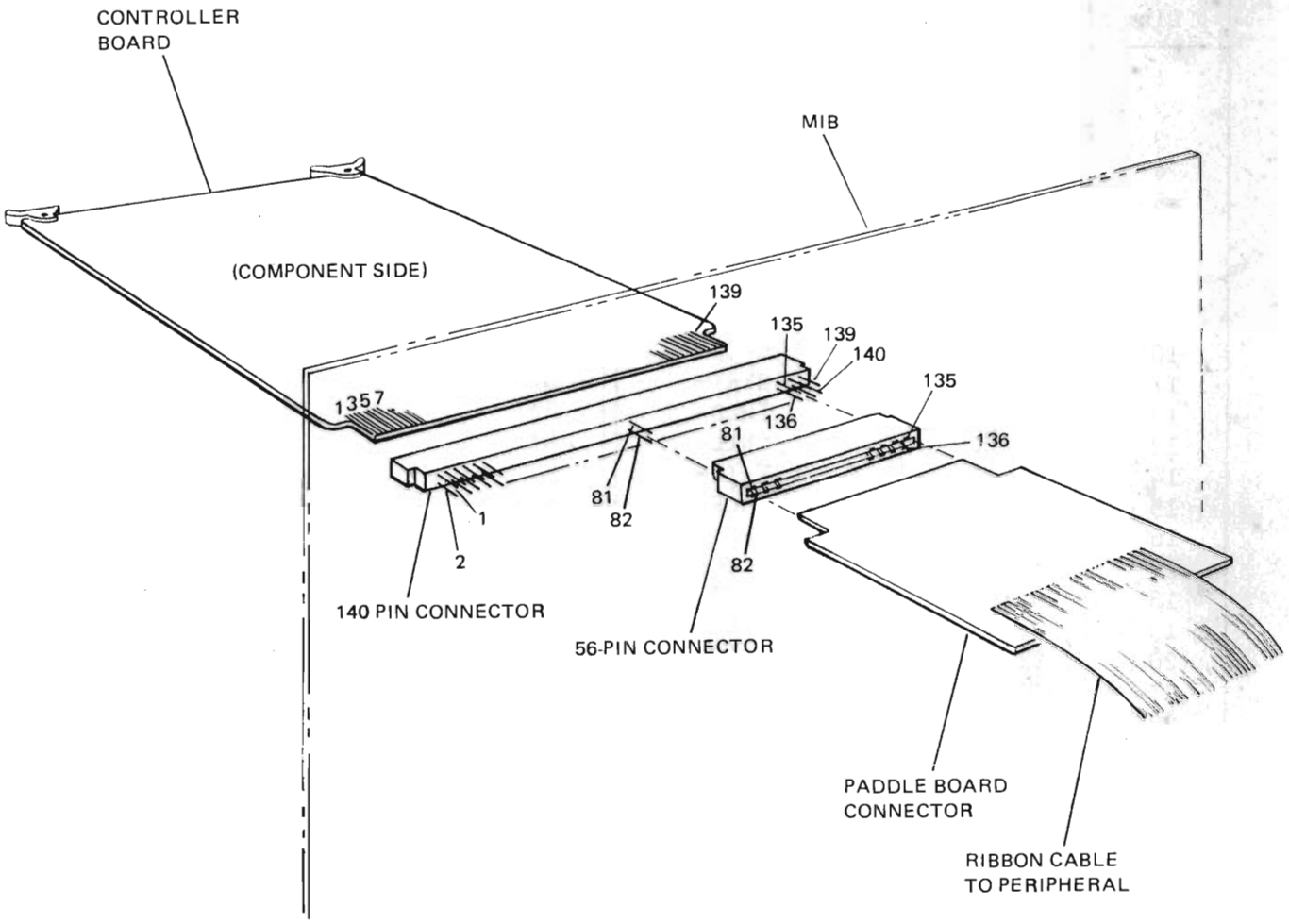


Figure 5-27. Pin Relationship Between Controller Card and Paddle Board

Table 5-2 summarizes signal pin assignments. 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 5-2. I/O Signal Pin Assignments

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	+5	36	OTB15-	71	DHLD-	106	*
2	-15	37	FAP-	72	DAD0-	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	VU+	79	DCTP-	114	*
10	INB05-	45	CACK-	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	DMF0-	91	*	126	*
22	OTB01-	57	DMF1-	92	*	127	*
23	OTB02-	58	RQIN-	93	*	128	*
24	OTB03-	59	IPRS-	94	*	129	*
25	OTB04-	60	RDCL-	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	OTB14-	70	DCRQ-	105	*	140	+5

* Controller card/Peripheral cable interface

Figure 5-28 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.

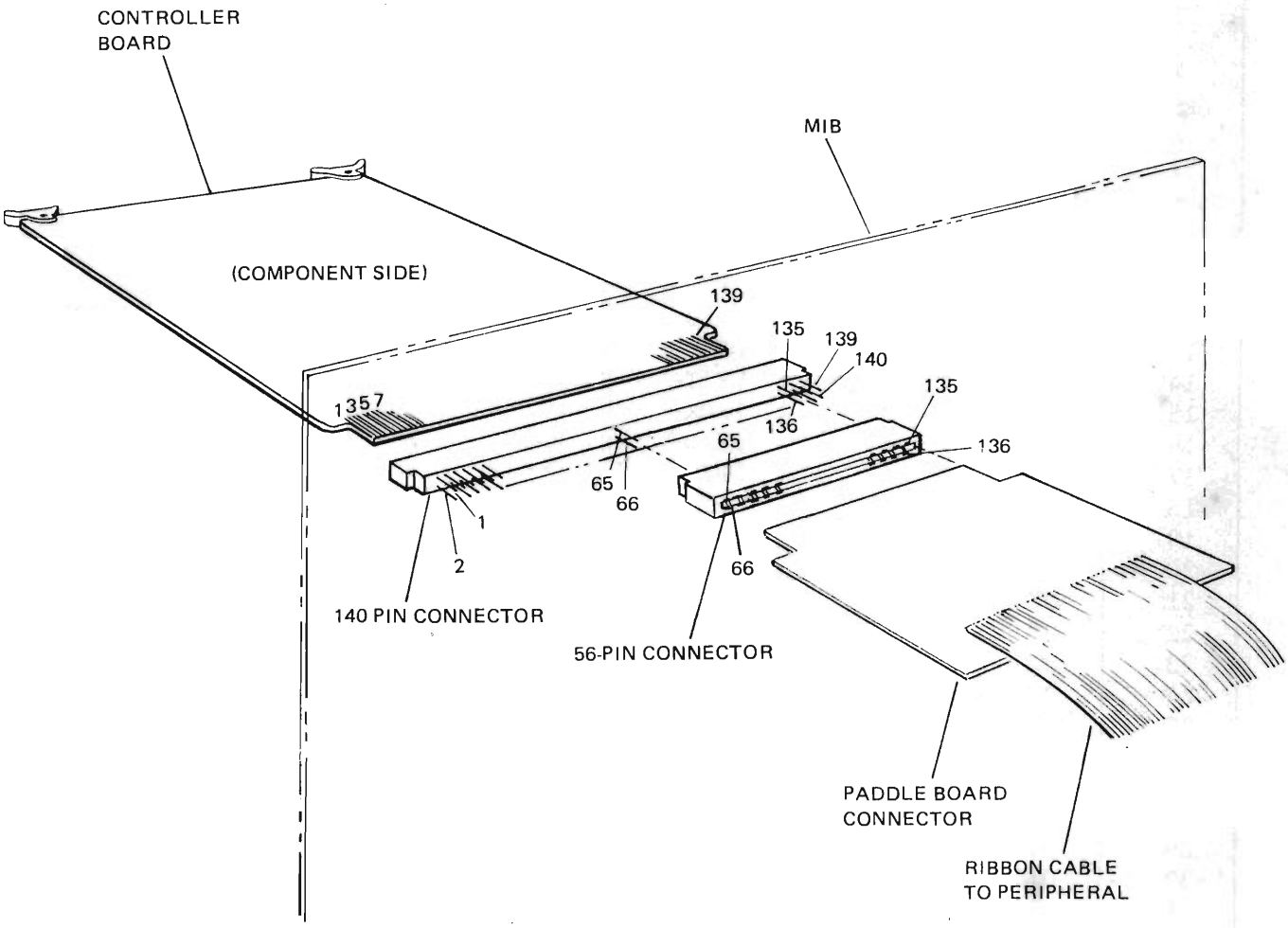


Figure 5-28. Pin Relationship Between CIT-16 Board and External I/O Bus Cable

Table 5-3 shows connection pin assignments for the External I/O Bus cable. Pins 65 - 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.

Table 5-3. I/O Cable & 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	VU+I
9	INB04-	44	VU+	79	INB09-I	114	CACK-I
10	INB05-	45	CACK	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	IACK-	90	OTB03-I	125	IREQ-I
21	OTB00-	56	DMF0-	91	GND	126	IACK-I
22	OTB01-	57	DMF1-	92	OTB04-I	127	DMF0-I
23	OTB02-	58	RQIN-	93	OTB05-I	128	DMF1-I
24	OTB03-	59	IPRS-	94	OTB06-I	129	RQIN-I
25	OTB04-	60	RDCL-	95	OTB07-I	130	IPRS-I
26	OTB06-	61	DCHN-	96	OTB08-I	131	GND
27	OTB06-	62	CLDS-	97	OTB09-I	132	RDCL-I
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	Not Used
31	OTB10-	66	DPSO-I	101	OTB12-I	136	Not Used
32	OTB11-	67	GND	102	OTB13-I	137	GND
33	OTB12-	68	Not Used	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

Figures 5-29 and 5-30 illustrate I/O signal connections for the Integral I/O Package (SPC-16/40, 60 and 80) and the External I/O Enclosure (as used with Models 40, 60 and 80 and Models 45, 65 and 85).

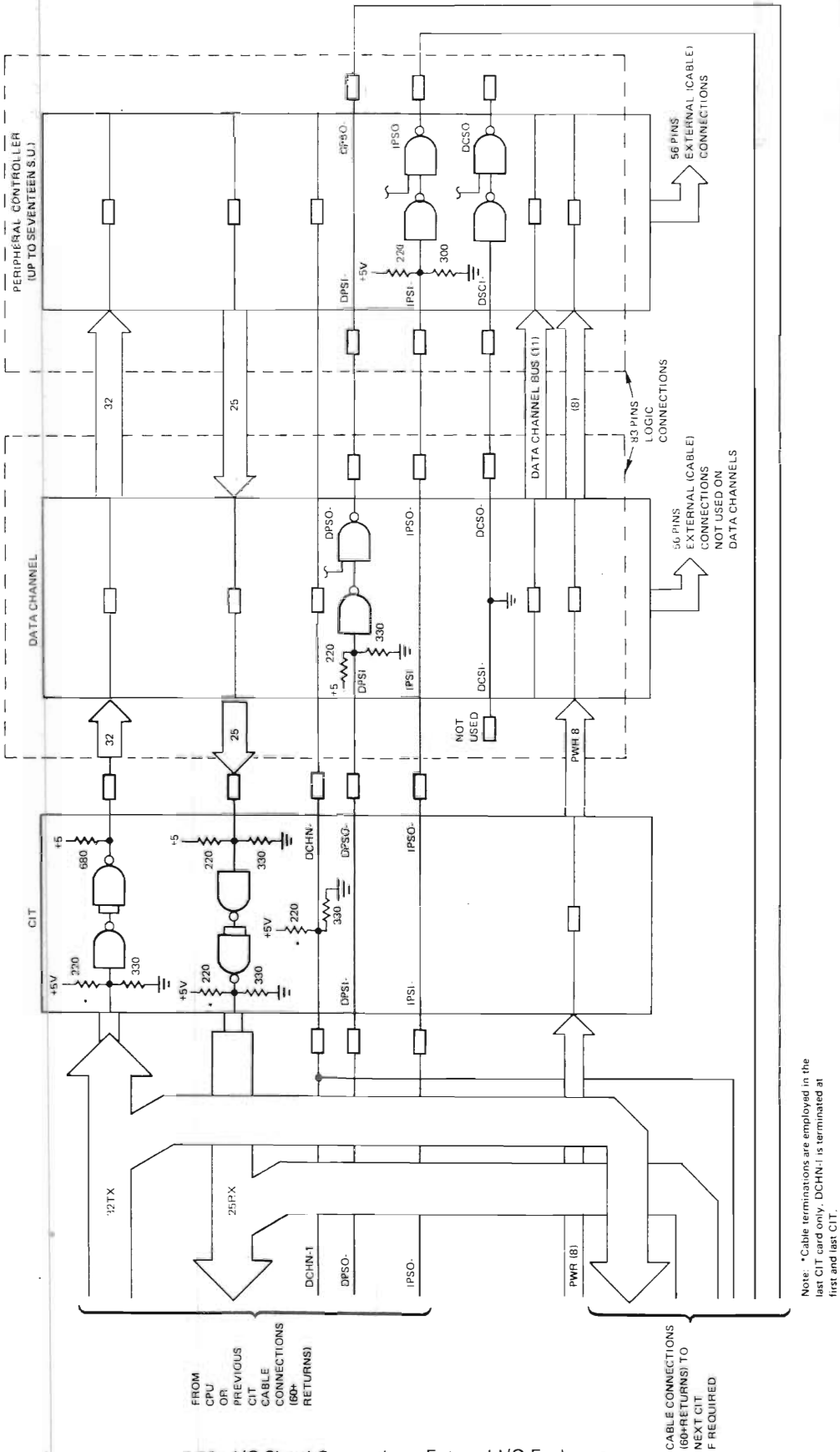


Figure 5-29. I/O Signal Connections, External I/O Enclosure

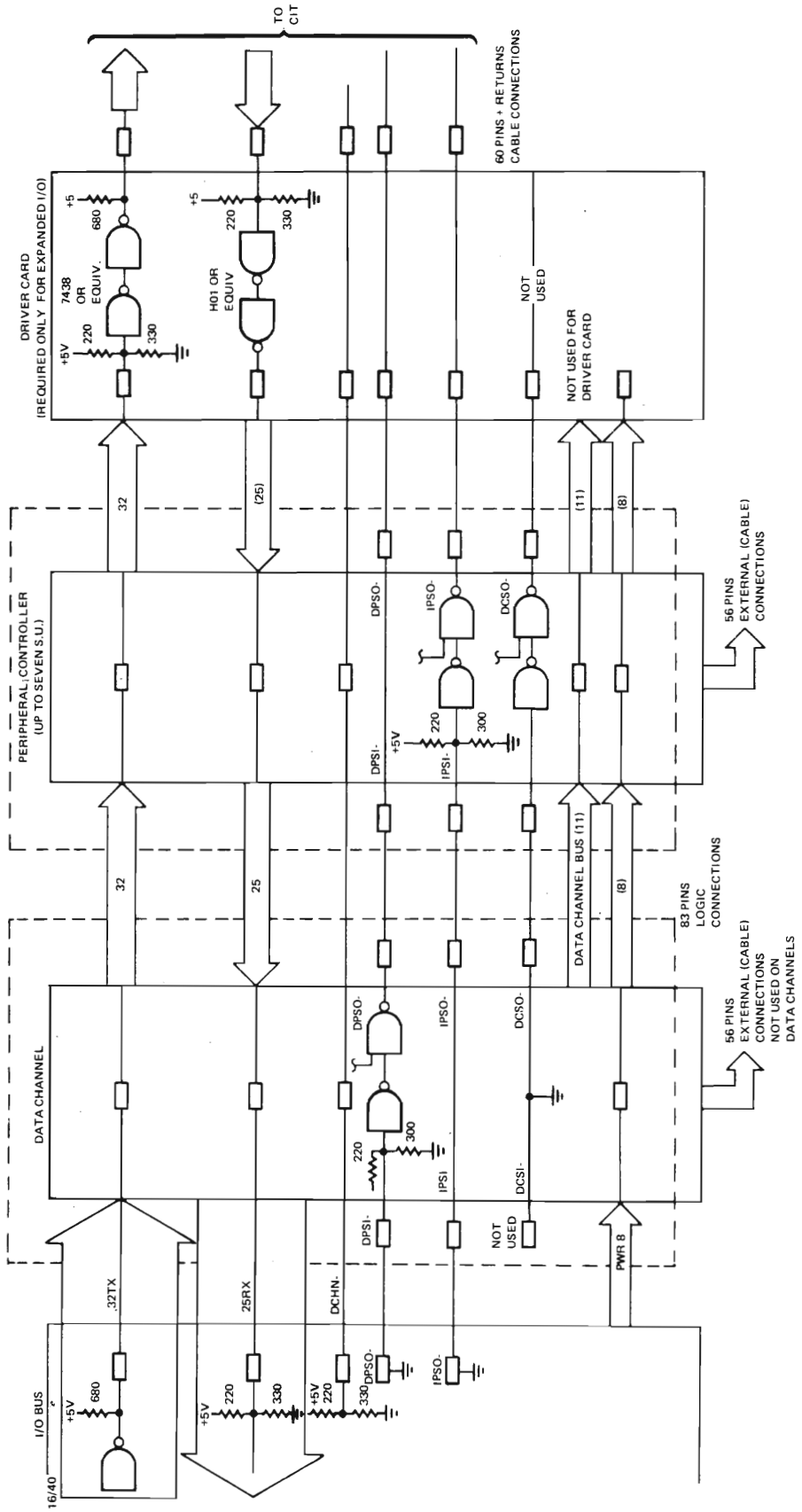


Figure 5-30. I/O Signal Connections, Integral I/O Package (Models 40, 60 and 80)

SECTION 6
SPC-16 CONSOLE

The controls and indicators, which are located on the front panel of the computer, allow manual data entry, examination of register and memory contents and initiation of automatic operation.

A key-operated console lock switch on the front panel provides a safety feature to prevent inadvertent operation of console controls when the system is on-line and operating.

The console is illustrated in Figure 6-1. The console panel contains indicators (lighted/unlighted) and switches (up = 0 or not set, down = 1 or set). The ENTER (8), STEP (10), SYSTEMS RESET (12), and CONSOLE INTERRUPT (14) switches return to the up (off) position after being manually depressed. All other switches may be manually set either up or down. Switches and indicators are discussed individually in Section 6.1.

Step-by-step instructions for performing the following operations are given in Section 6.2.

a. Display/modify the contents of registers and indicators.
b. Display/modify the contents of memory locations.
c. Manual step-execute through a program.
d. Start automatic program execution.

6.1 SWITCHES AND INDICATORS

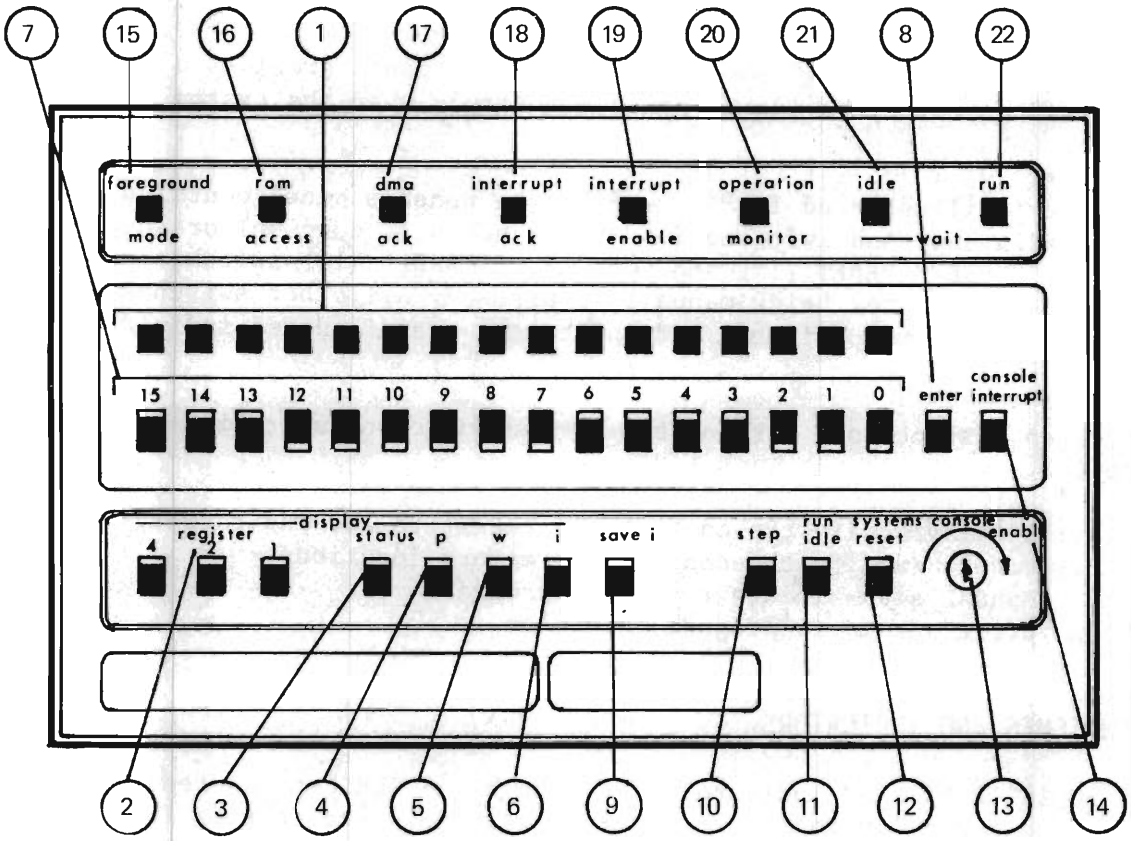
(1) REGISTER DISPLAY indicators (0 through 15) indicate contents of the System Console Display register (K register). The K register is loaded under program control by the DSPL instruction in the RUN mode; in the IDLE mode, it is loaded with the contents of the selected register (see Register Select switches below).

REGISTER SELECT SWITCHES (2 through 6)

These switches are used to select a register for display or data entry. They have a left-to-right priority; for example, if switch (6) is on (down), the other Register Select switches are disabled and the I register is selected.

(2) GENERAL PURPOSE REGISTER select switches are used to select one of the eight operational registers to receive data subsequently keyed into the Data Entry switches (7). The other Register Select switches ((3), (4), (5) and (6)) must be in the up (off) position in order to select a register with switches (2). After keying in a register code in the IDLE mode, the register contents are displayed on the Register Display indicator lights (1).

The register codes are the same as those used in the instruction set. (Section 3.0).



1. 16 REGISTER DISPLAY indicators
2. 3 GENERAL PURPOSE REGISTER select switches
3. STATUS Register select switch
4. P Register select switch
5. W Register select switch
6. I Register select switch
7. 16 DATA Entry switches
8. ENTER switch
9. SAVE I switch
10. STEP switch
11. RUN/IDLE switch
12. SYSTEMS RESET switch
13. CONSOLE KEY LOCK
14. CONSOLE INTERRUPT switch
15. FOREGROUND MODE indicator
16. READ ONLY MEMORY ACCESS indicator
17. DIRECT MEMORY ACCESS/AUTOMATIC DATA CHANNEL ACKNOWLEDGE indicator
18. INTERRUPT ACKNOWLEDGE indicator
19. INTERRUPT ENABLE indicator
20. OPERATIONS MONITOR ALARM indicator
21. IDLE indicator
22. RUN indicator

Figure 6-1. The SPC-16 Computer Console

Register	Select Switches
A	000
X	001
Y	010
Z	011
B	100
C	101
D	110
E	111

(3) STATUS REGISTER select switch places the contents of the indicators and the shift counter (S register) into the Console Display register. The Register Display indicators (1) will show the status as follows:

Shift Counter → $DSPL_{0-3}$

Link → $DSPL_4$

Overflow → $DSPL_5$

Plus → $DSPL_6$

Zero → $DSPL_7$

Foreground → $DSPL_8$ (same as (15))

0 → $DSPL_{9-15}$

The indicators may be changed by the Data Entry switches (7) .

(4) "P" REGISTER select switch displays the contents of the Program Counter register on the Register Display indicators (1) , bits 0-14. The interrupt status (same as (19)) is displayed in bit 15. When manually stopping an executing program, the current instruction is completed before execution stops. Since the program counter is incremented by the current instruction, the P register will contain the address of the next instruction to be executed.

The program counter may be changed via the Data Entry switches (7) and the P register select switch (4) . The interrupt status enable (bit 15) is not affected when the program counter is changed.

(5) "W" REGISTER select switch displays the contents of the Working register. The W register contains the effective operand address for the last instruction executed. The W register may be changed by the Data Entry switches (7) .

(6) "I" REGISTER select switch displays the contents of the Instruction register.

When manually stopping an executing program, execution of the current instruction is completed and then stops, so that the Instruction register will contain the instruction just completed. The Instruction register may be altered by the Data Entry switches (7).

(7) DATA ENTRY switches (0 through 15) are used to enter data or instructions into the selected register in the IDLE mode. These switches may be tested as sense switches or used as data input switches during program execution. Data switches in down position = 1, in up position = 0.

(8) ENTER switch. In the IDLE mode, this switch replaces the contents of the selected register with the contents of the Data Entry switches.

(9) SAVE I switch preserves the contents of the Instruction register for repeated execution (e.g., referencing memory locations using the same load/store instruction) in the IDLE mode. SAVE I is effective only in the IDLE mode.

(10) STEP switch. When this switch is depressed, the computer fetches an instruction from the location specified by the P register, executes the instruction and increments the P register. In the IDLE mode, only a single instruction cycle executes. (If the SAVE I switch is down, the current contents of the I register are executed and preserved; the P register is still incremented, however.) In the RUN mode automatic program execution is initiated by pressing STEP. A WAIT condition may be terminated by pressing the STEP switch.

(11) RUN/IDLE switch is used to place the computer in either the RUN mode (switch up) or the IDLE mode (switch down).

(12) SYSTEMS RESET switch is used to initialize the computer and peripheral controllers. This switch sets foreground mode, if the foreground option is installed, (shown by FOREGROUND MODE indicator (15)) and disables all interrupts.

(13) CONSOLE ENABLE Key Lock. This is a two position lock, as follows:



Turning the key counterclockwise disables the console (i.e., changing any of the console switches has no effect). Turning the key clockwise will enable the console switches. The console may be enabled/disabled in either the RUN or IDLE mode.

The SPC-16 console was intended to be able to be used as a "Systems-Operating-Console". In line with this, the program has complete control over the Console Display register (K) and it can "sense" the console data switches under program control for manual input commands. The console lock-out feature is built in as part of the fail-safe features required for an on-line real-time system in which only qualified operators can affect the operation of the system. Rather than "lock-out" the ability to sense the console switches, General Automation has provided the means whereby either the "locked-out" or the "unlocked-out" approach could be used in sensing the console switches.

The "unlocked-out" approach means that the program senses the switches at will for manual input commands and anybody can change the switches at any time to implement these commands, independently of whether or not the console is locked.

The "locked-out" approach is accomplished by a software philosophy that requires a console-interrupt prior to sensing the console data switches for manual input commands, i.e., the console-interrupt is dedicated to initiating the sensing of the console data switches. Since the console-interrupt cannot be initiated unless the console is unlocked, then the operation is a "locked-out" one.

(14) CONSOLE INTERRUPT switch. Pressing this switch causes a console-interrupt request when the console is enabled.

(15) FOREGROUND MODE indicator indicates which set of eight General Purpose registers is being used. The indicator is on for foreground, off for background. If the foreground/background option is not installed, this indicator is always off.

(16) READ ONLY MEMORY ACCESS indicator indicates when Read Only Memory is being accessed for instructions or data.

(17) DIRECT MEMORY ACCESS/AUTOMATIC DATA CHANNEL ACKNOWLEDGE indicator indicates that a DMA cycle "steal" is taking place.

(18) INTERRUPT ACKNOWLEDGE indicator indicates when an interrupt is being serviced. Manually stepping through an interrupt will result in the address of the branch vector being displayed in the indicators. Exit must be made by pressing STEP (to execute the interrupt) or SYSTEMS RESET (to reinitialize the system).

(19) INTERRUPT ENABLE indicator indicates the system interrupt status: on = enabled, off = disabled.

(20) OPERATIONS MONITOR ALARM indicator indicates when the alarm has been activated.

(21) IDLE indicator indicates when the computer is in the Idle mode.

(22) RUN indicator indicates when the computer is in the Run mode.

NOTE

When executing a program in the Run mode and a WAIT instruction is encountered, both the RUN and IDLE indicators come on.

6.2 BASIC OPERATIONS

6.2.1 Displaying/Modifying Registers

1. Turn key lock (13) clockwise to enable console switches.
2. Set the RUN/IDLE switch (11) to IDLE. (Indicator (21) will be illuminated). Press SYSTEMS RESET switch (12) .
3. Select register from Register Select switches (2) through (6) . The contents of the selected register will be displayed on the Register Display indicators (1) .
4. To enter data into the selected register, key the data into the Data Entry switches (7) and press the ENTER switch (8).

6.2.2 Displaying/Modifying Memory Locations

1. Set RUN/IDLE switch (11) to IDLE. Press SYSTEMS RESET switch (12).
2. Load the value (ADDRESS-1) into the P register, where ADDRESS is the address of the first memory location to be displayed/modified.
3. Load the "LDA 0,0" instruction (X'4000') or "STA 0,0" instruction (X'5000') into the I register for displaying or modifying the memory location of ADDRESS, respectively.
4. Preserve the contents of the I register by pressing the SAVE I switch (9) down.
5. Select the A register for display.

6a. To display the contents of the memory location at ADDRESS, press STEP switch (10) .

6b. To enter data into the memory location at ADDRESS, key the data into the Data Entry switches (7) , press ENTER switch (8) and then press STEP.

7a. The contents of sequential memory locations are displayed each time STEP is pressed.

7b. To enter data into sequential memory locations, repeat step 6b.

6.2.3 Manual Execution of Stored Program

1. Set the RUN/IDLE switch (11) to IDLE. Press SYSTEMS RESET switch (12).
2. Load beginning address of program into the P register.
3. Press STEP switch (10). The program will execute one instruction. Relevant registers may then be selected for display.
4. Repeat step 3. to execute the program one instruction at a time.

NOTE

When stepping through an instruction and an interrupt is acknowledged (INTERRUPT ACK indicator (18) is lighted), the interrupt vector location is displayed on the Register Display indicators and all console enter/display switches are disabled. Pressing STEP will execute the interrupt cycle and re-enable the console.

6.2.4 Initialization of Automatic Execution

1. Set the RUN/IDLE switch (11) to IDLE. Press SYSTEMS RESET switch (12).
2. Load beginning address of program into the P register.
3. Set the RUN/IDLE switch (11) to RUN.
4. Press STEP switch (10). The program will execute.
5. To stop the execution of a program, set the RUN/IDLE switch to IDLE.

6.3 BOOTSTRAPPING

Programs may always be loaded into the computer's memory manually by entering the individual instruction codes and data via the console. However, if a program that reads data from a peripheral and places it in memory in a sequential fashion is already in memory, it may be executed and used to load another program automatically.

Two data formats are used for object programs on paper tape. Most system software (the CAP-16 Assembler, the FORTRAN compiler, operating systems, etc.) as well as the object tapes of user programs that have been assembled or compiled are in PGS format. In addition to the actual program instruction codes and data, this format includes linking and relocation information that can be used by the loading program to build an executable program unit from separately assembled or compiled program modules. The smallest loading program that will accept programs in PGS format is the PGS loader; it consists of more than 250 instructions and it is, therefore, unpractical to load it manually. (PGS data format and the loading programs that will accept this format are described in detail in the FSOS manual and the DBOS manual.)

A few basic system programs, including the PGS loader program, are supplied in simple binary format on paper tape. A binary tape consists of a sequence of data bytes (each representing half an instruction or data word). These bytes may be input via an appropriate peripheral (teletype paper tape reader or high speed paper tape reader) and stored in sequential words (two data bytes per word) in memory. The program that performs this operation is known as a bootstrap loader.

The bootstrap loader that loads binary tape format from the teletype (Teletype Bootstrap) consists of only eight instructions; thus, it may be easily keyed in via the computer console. Another bootstrap loader, which loads binary tape format from the high-speed paper tape reader (High Speed Paper Tape Reader Bootstrap) consists of 20 instructions; it may be keyed in manually (see listing in Section 7.2.1) or loaded from the teletype using the Teletype Bootstrap (the High-Speed Paper Tape Reader Bootstrap program is supplied on paper tape in binary format). Either bootstrap loader may be used to load the more complex PGS loader, which is also supplied in binary format.

6.3.1 Loading the Teletype Bootstrap

If the system does not include a high speed paper tape reader, then the Teletype Bootstrap will normally be loaded into the eight highest memory locations (e.g., for a 4K machine, the beginning address would be X'0FF8').

If the Teletype Bootstrap is used for loading the High Speed Paper Tape Reader Bootstrap, the Teletype bootstrap may be loaded into the eight lowest memory locations (i.e., beginning at X'0000'); the High Speed Paper Tape Reader Bootstrap may then be loaded into the highest locations.

1. Load the teletype bootstrap into sequential memory locations as described in Section 6.2.2 for modifying memory locations:

Teletype Bootstrap

Hexadecimal	Instruction	Operation
0620	ZERO	X
0524	LOOP DSPL	X
10FF	TEST	RDY,TY
73FE	JMP	$-1
18BF	DTIR	A,TY
9900	STBY	A,0,X,1
072E	INCR	X
73F9	JMP	LOOP

2. Verify that the bootstrap was keyed in correctly by displaying the eight memory locations, following the procedure described in Section 6.2.2 for displaying memory locations.

6.3.2 Executing the Teletype Bootstrap

With the bootstrap in memory, the following steps are used to load programs using the bootstrap:

1. Set the D register to the starting address at which the program or data is to be loaded.
2. Set the P register to the starting address of the bootstrap.
3. Place the program paper tape into the teletype paper tape reader. The first binary frame must be over the read station.
4. Press SYSTEMS RESET.
5. Set the RUN/IDLE switch to RUN.
6. Press STEP.
7. Turn the reader on. If the program/data is loading sequentially, the REGISTER DISPLAY indicators (contents of the X register) will increment as the tape is being read.
8. When program/data has been loaded, turn the reader off. The loading process does not stop after the last data byte has been read but continues until the physical end of the tape is reached or the reader is turned off.
9. Set the RUN/IDLE switch to IDLE.
10. Press SYSTEMS RESET.

SECTION 7
SPECIAL FEATURES, OPTIONS AND CAPABILITIES

7.1 FAIL-SAFE GROUP

The fail-safe group comprises the following features:

- Relative Time Clock Interrupt
- Operations Monitor Alarm
- Power-Fail Interrupt
- Automatic Restart Interrupt

These features are standard on the SPC-16/40, 60 and 80 and optional on the SPC-16/45, 65 and 85.

(When the Power-Fail/Auto-Restart interrupt option is not present, memory protection still exists; when power fails this feature forces the system to an IDLE condition shortly followed by the system being initialized (system reset) prior to the power supply going into a non-regulated state. When the system is powering up and the power supply reaches its stable state, the system is brought out of the initialized state and into an IDLE condition regardless of the RUN/IDLE switch. The system must then be started manually.)

7.1.1 Relative Time Clock (RTC)

This feature provides a periodic interrupt (every 1,000 memory cycles) when the interrupt level from the RTC is enabled. A program may use this interrupt to simulate a real-time clock for multiprogramming applications. In addition, the interrupt may be used to protect against unusual conditions by periodically returning control to a monitor program.

The RTC interrupt may be enabled or disabled under program control. The state of the interrupt mask for the RTC is controlled by the internal interrupt mask word (device select code X'3E'), which is described in Section 4.2.

The RTC derives its time base from the internal crystal controlled master clock. Every 1,000 memory cycles 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. Such a routine (TIME) is illustrated in the example below:

Example

The following program maintains the time of day in hours (HRS), minutes (MINS) and seconds (SECS) by counting RTC interrupts. The program is designed for the SPC-16/80 or 85 where an RTC interrupt occurs every 800 microseconds, (1,250 interrupts = 1 second). It is assumed that location X'43' contains the address of location TIME.

```
MASK    DS      1           Internal interrupt mask word
HRS     DC      0           Hours
MINS    DC      0           Minutes
SECS    DC      0           Seconds
CNT     DC      0           No. of RTC interrupts
TIME    PMA                 Reset operations monitor alarm
        SARS    *3,,0       Save registers
        LDR     A,MASK      } Disable RTC interrupt
        XORV    A,X'0008'   }  by modifying
        STR     A,MASK      }  and
        DTOR    A,X'3E'     }  outputting mask word
        RISE    E           } Restore interrupt enable status
        INCM    CNT         Increment interrupt count
        LDV     A,1250      } Has one second
        CMR     A,CNT       }  elapsed?
        SKN     EXIT        } EXIT if not
        ZERO    A           } Yes - reset counter
        STR     A,CNT       }
        INCM    SECS        Increment second count
        LDV     B,60        } Has one second
        CMR     B,SECS      }  elapsed?
        SKN     EXIT        } EXIT if not
        STR     A,SECS      Yes - reset seconds
        INCM    MINS        Increment minutes count
        CMR     B,MINS      } Has one minute elapsed?
        SKN     EXIT        } EXIT if not
        STR     A,MINS      Yes - reset minutes
        INCM    HRS         Increment hour count
        LDV     B,24        } Has one day
        CMR     B,HRS       }  elapsed?
        SKN     EXIT        } EXIT if not
        STR     A,HRS       Yes - reset hours
EXIT    INH
        LDR     A,MASK      } Re-enable RTC
        XORV    A,X'0008'   }  interrupt by
        STR     A,MASK      }  modifying
        DTOR    A,X'3E'     }  and outputting MASK
        LARS    *3,,0       Restore registers
        RTRN    E           Return
```

7.1.2 Operations Monitor Alarm

The Operations Monitor Alarm (OMA) feature can be used to provide automatic processing shutdown in the event that program control fails to return periodically to a system program.

The OMA feature comprises an alarm counter that is initially enabled by the execution of the 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 the PMA instruction. Failure to issue a PMA instruction within 200,000 memory cycles from the last resetting will activate the alarm.

If the alarm is activated, the computer automatically switches from the RUN mode to the IDLE mode (the switch is internal - the physical position of the RUN/IDLE switch on the console is not affected). 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 SYSTEMS RESET switch on the console with the RUN/IDLE switch in the IDLE position. When cleared manually with SYRT, the program must be restarted manually in the normal manner.

The example program for the Relative Time Clock (Section 7.1.1) illustrates a typical application of the OMA feature. The first instruction of the RTC interrupt service routine resets the OMA counter. Under normal conditions, program control will be transferred to this routine every 1,000 memory cycles (i.e., every RTC interrupt). However, another program may disable the interrupt level for the RTC or disable the master interrupt system and thereby prevent the RTC interrupt service routine from being executed at its regular interval. If the interrupt system is not re-enabled soon enough, the OMA timer will be activated and the system will automatically shut-down. Having interrupts disabled for a period of time on the order of 200,000 memory cycles is usually indicative of a program failure (in real-time and process control applications).

7.1.3 Power-Fail Interrupt

When the AC primary power input to the SPC-16 drops below a predetermined value (less than 105 vac under full load), the Power-Fail Detect circuit requests a Power-Fail Interrupt. The regulated direct current voltages are guaranteed to be good at this time. This interrupt is the highest priority and cannot be inhibited or masked off.

At the same time the interrupt is requested, the OMA timer is initialized to time-out in 100 memory cycles and any further PMA instructions are ignored. Also, any further data-channel requests are ignored. Therefore, after the Power-Fail interrupt request the programmer has 100 memory cycles (minus the time of the longest instruction in his program) to execute whatever shutdown routines he requires. After 100 memory cycles, the OMA times out and the machine goes through a normal alarm sequence, except that the SFEC line remains in the "safe" state. This allows one to differentiate between a normal power-fail OMA and a system malfunction OMA.

The Power-Fail interrupt has been assigned an interrupt vector location of X'40'. Location X'40' must contain the address of the interrupt service routine responsible for shutting down the system in an orderly manner. Such a routine (FAIL) is illustrated below:

```
SAVE    DS      9       Register storage
FAIL    SARS    SAVE    Save registers
        WAIT            Wait for power to
                        fail completely
```

7.1.4 Automatic-Restart Interrupt

When the computer is powered up with the RUN/IDLE switch in the RUN position (e.g., after a power failure) and the AC primary power input goes above a predetermined value (guaranteed not to be greater than 105 vac), the Auto-Restart circuit requests an interrupt on its preassigned level. During the powering-up sequence the entire system is initialized (equivalent to pressing SYSTEMS RESET on the computer's console) so that when the Auto-Restart interrupt occurs all interrupts are disabled via ISE being off. However, the internal interrupt mask register is not initialized and it is the responsibility of the program to initialize this register prior to turning ISE on.

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. A typical Auto-Restart routine is outlined below:

```
SAVE    DS      9       Registers were saved
                        here by power-fail routine
.
.
START   .               Print operator message
        .               or sense status of the
        .               system, etc.
        .
        .
        .
JMP     PROGRAM START   Return to program
                        start where system
                        start procedure is
                        determined.
```

NOTE

Power up sequencing causes a system reset, which among other functions sets Foreground register mode. If a program is operating in Background mode at power fail interrupt time, a single SARS instruction will store the Background registers only. Power up, setting Foreground mode, followed by a single LARS instruction will load Foreground registers then return to Background mode when the last register (STATUS) is loaded, thus restoring only the information from previous Background registers into the Foreground registers. It is therefore the programmer's responsibility, if using Background mode, to save all registers upon power fail and restore all registers via the auto/restart subroutine.

7.2 ENTENDED MEMORY SYSTEM (64K)

Models 40, 60, and 80 of the SPC-16 will accommodate four memory modules and can, therfore, provide up to 32K words of memory if 8K or 16K modules are used. (The 16K modules occupy two memory slots due to their extra thickness, permitting only two modules to be used in a mainframe.)

The mainframe for Models 45, 64 and 85 will accommodate eight memory modules, thereby providing up to 64K if 8K or 16K modules are used. Extending the memory from 32K to 64K requires implementation of the 64K extended memory option.

The Extended Memory System, available for Models 45, 65 and 85 provides two additional card slots to accommodate logic boards that govern the extended memory modules.

If, for any SPC-16 model, the CPU is configured to support 8K or 16K modules, it can additionaly support only one 4K module. In order to maintain continuous addresses, this module must be placed in the highest address position.

The Extended Memory System was designed using a "mapping" philosophy where a 4K block of extended memory can be switched in, in place of any of the four 4K blocks (two blocks per 8K module or four blocks per 16K) residing in the normal 16 to 32K expansion area. The switch is instantaneous (XIO instruction).

The extended memory system, like the standard GA memory system, is comprised of random-access, three-wire, lithium ferrite core memory, arranged in 64-by-64 matrix planes.

7.2.1 System Description

The SPC-16 addressing scheme allows for addresses in the range of 0_{16} to $7FFF_{16}$ (i.e., a total of 32K words of addressable memory). Addressable memory is divided into eight 4K blocks of addresses:

- 0 - 4K
- 4 - 8K
- 8 - 12K
- 12 - 16K
- 16 - 20K
- 20 - 24K
- 24 - 28K
- 28 - 32K

In a fully-expanded 32K system, the eight 4K address blocks are assigned to four 8K or two 16K memory modules.

The two 8K memory modules or one 16K memory module housed in the MEM1 and MEM2 slots of the SPC-16/45 (65, 85) enclosure are permanently assigned to the first four address blocks (0-16K). If 8K modules are used, the first module is assigned to blocks 0-4K and 4-8K; the second module is assigned to blocks 8-12K and 12-16K. If a 16K module is used, it contains all four blocks (0-4K, 4-8K, 8-12K and 12-16K). Any address in the range 0_{16} to $3FFF_{16}$ refers to a memory location in the above module(s).

The second two 8K memory modules, or the second 16K memory module, housed in the bottom two memory slots (MEM3 and MEM4) of the SPC-16/45 (65, 85) enclosure, are initially assigned to the second four address blocks (16-32K). (The system is initialized when the Systems Reset switch on the computer's console is pressed or when an auto-restart occurs following a power-fail.) The module in the third memory slot is assigned to blocks 16-20K and 20-24K; the module in the fourth memory slot is assigned to blocks 24-28K and 28-32K. These initial assignments remain in effect until an appropriate I/O instruction is executed (explained below) which deactivates a 4K portion of one of these modules.

The Extended Memory System adds up to four additional 8K memory modules or two additional 16K memory modules to the memory configuration, making a total of eight 8K memory modules or four 16K memory modules.

The extended memory modules reside in the MIB's four upper-most memory slots (MEM5-8). Extended memory modules are divided into two groups of two 8K modules or one 16K module each (in a fully-extended system). Each group is controlled by a Memory Logic Display Board in the top sub-chassis. All extended memory modules are initially inactive (i.e., unaddressable). In order to make a 4K block in an extended memory module addressable, the program must first issue an XIO instruction. This instruction maps one of the 4K blocks in an extended memory module into one of the addressable 4K address blocks in the range of 16-32K.

Figure 7-1 illustrates the memory allocation in a fully-extended memory system. Both the physical 4K block assignments and the corresponding address assignments are shown.

The XIO instruction used for extended memory mapping is a Data Output instruction with a device select code of X'39' (e.g., DTOR A, X'39' or DTOM A, X'39'). The output data word has the following format:

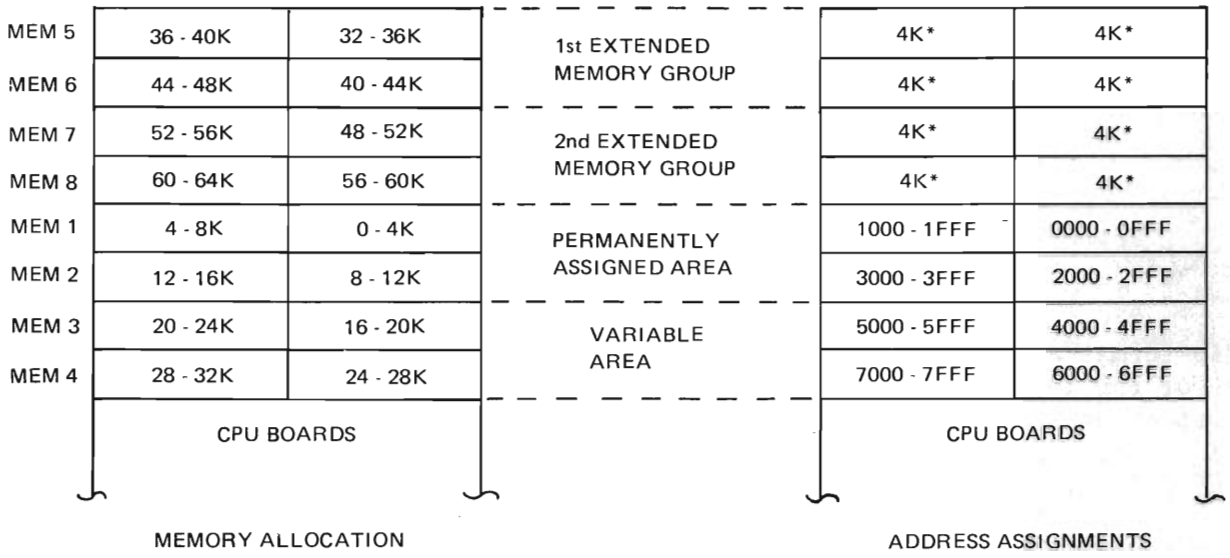
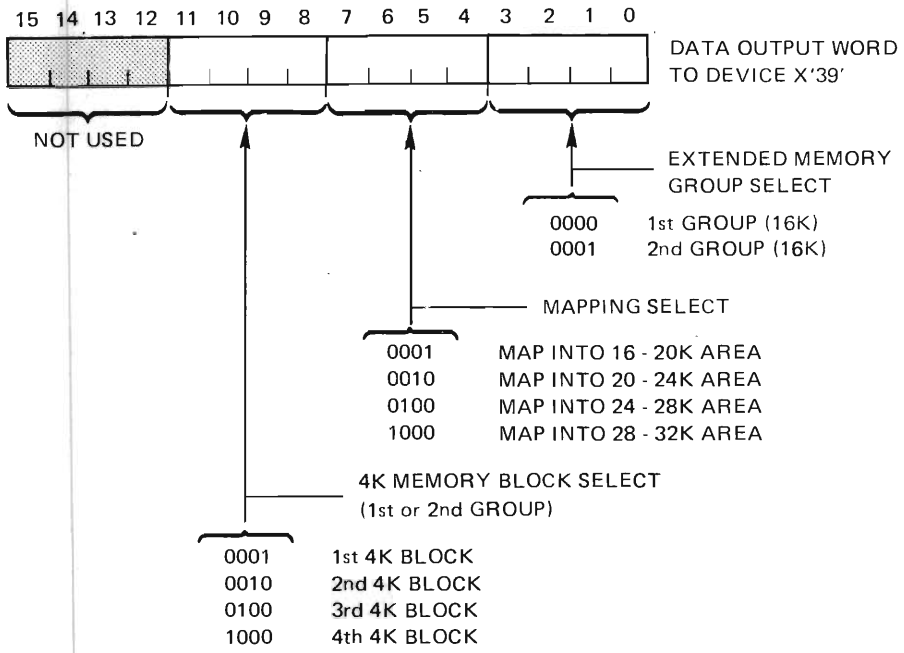
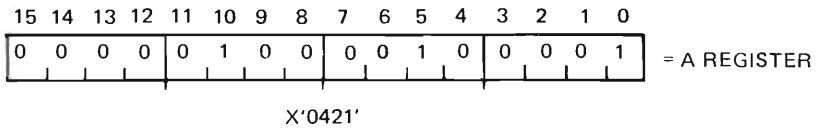


Figure 7-1. SPC-16/45 (65, 85) Memory Expansion



This format allows any one of the eight extended memory blocks to be mapped into any one of the four variable address blocks (16-20K, 20-24K, 24-28K, 28-32K), thereby deselecting the specified variable address block. Alternatively, the format allows the normal variable address block to be reinstated, thereby deselecting the extended memory block occupying that position.

When mapping an extended memory block into one of the variable address blocks, a combination of bits 0-3 and 8-11 of the data output word specify the extended memory block: bits 0-3 select the extended 16K group and bits 8-11 select one of the four 4K blocks within that group. The selected extended memory block is mapped into the variable address block specified by bits 4-7 of the data output word. For example, if the A register contains:

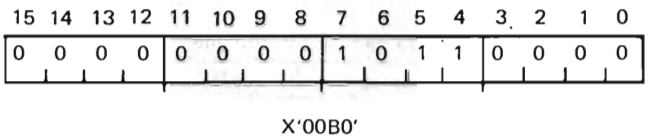


and the following instruction is executed:

```
DTOR A, X'39'
```

the third memory block in the second extended group will be mapped into the 20-24K address block, and the second normal variable address block will be de-activated. Any subsequent reference to a memory location in the range 5000_{16} to $5FFF_{16}$ will access the extended memory module just assigned to this address block.

In order to reinstate one or more of the normal variable address blocks, bits 8-11 of the data output word must all contain zero. Bits 4-7 select which address block (or blocks) is to be re-initialized. For example, if the A register contains:



and the following instruction is executed:

```
DTOR A, X'39'
```

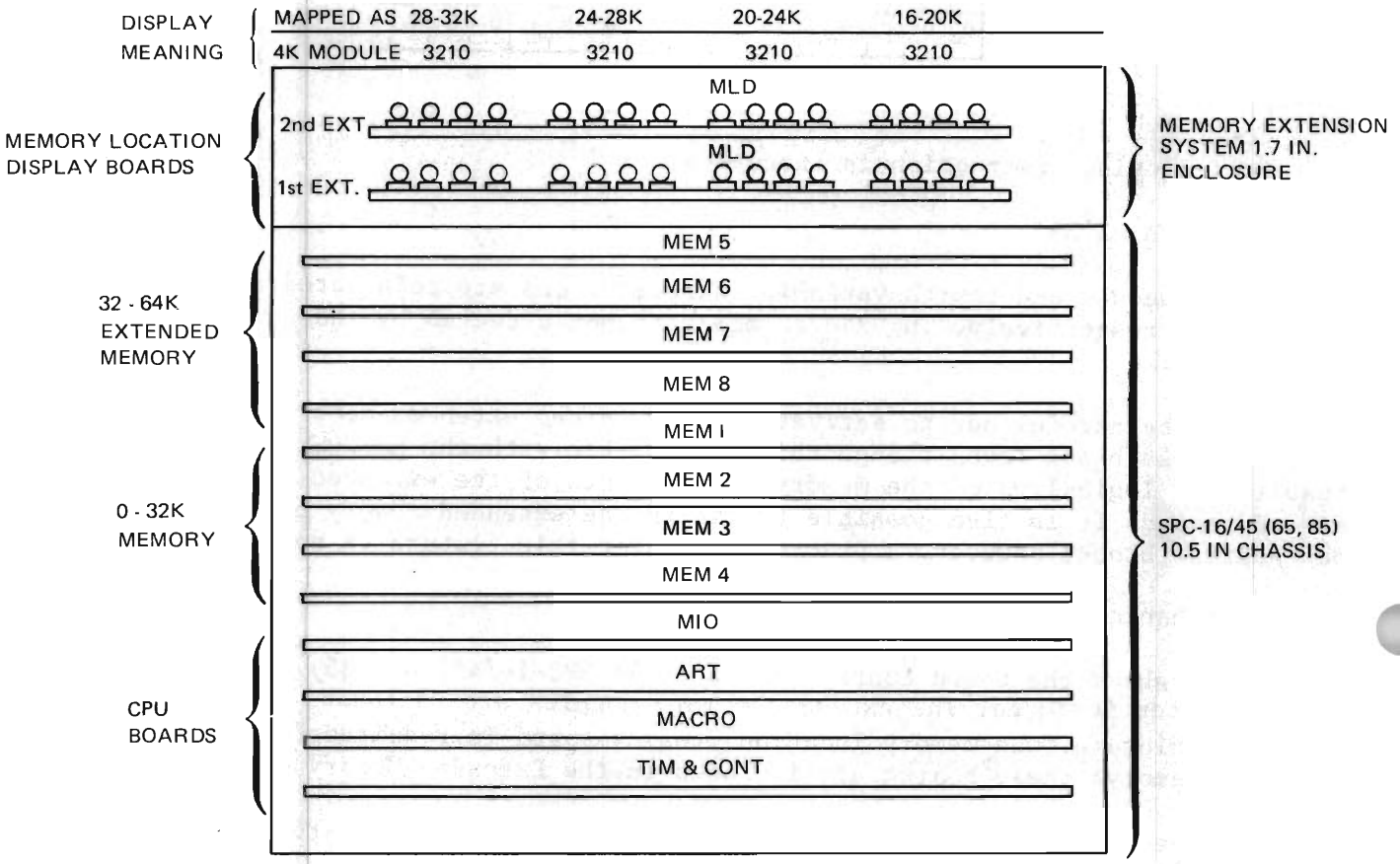
the first, second and fourth variable address blocks are reinstated as 16-20K, 20-24K and 28-32K, respectively. The 24-28K block is not affected by the execution of this instruction.

One should be careful not to activate more than one extended memory block for any given address block (even though this is possible with the hardware); this would result in a logical OR of the memory data outputs of the extended blocks when they are addressed. It is also possible to assign one extended memory block to more than one address block; however, a practical use for this feature is not anticipated.

7.2.2 Mechanical Description

Figure 7-2 shows the board configuration of an SPC-16/45, 65, 85) with the Extended Memory System included. The extended memory modules are included in the computer's enclosure. One memory location display board is required for each 16K of extended memory; these boards are included in the Extended Memory System enclosure.

Lamps on the Extended Memory System front panel indicate the status of each of the four variable address blocks: 16-20K, 20-24K, 24-28K and 28-32K. If none of the eight lamps (one lamp for each extended memory block) which correspond to an address block are illuminated, then the normal memory block is currently assigned to that address block. An illuminated lamp indicates that the respective extended memory block is assigned to the corresponding address block.



NOTES: 1) THE MLD BOARD IN THE 1ST SLOT CONTROLS THE MODULES IN THE MEM 5 AND MEM 6 POSITIONS. THE MLD BOARD IN THE 2ND EXT SLOT CONTROLS THE MODULES IN THE MEM 7 AND MEM 8 POSITIONS.

2) 8K MODULE = ONE MEM x SLOT
16K MODULE = TWO MEM x SLOTS

Figure 7-2. SPC-16/45 (65, 85) With Extended Memory System

7.3 EXTENDED MEMORY SYSTEM (128K)

Models SPC-16/45 and SPC-16/65 can be expanded to support up to 128K words of memory if the 128K extended memory option is used. The additional eight 8K modules or four 16K modules are housed in an Extended Memory Chassis. (The 16K modules occupy two memory slots due to their extra thickness, permitting only four modules to be used in the Extended Memory Chassis.)

Two modes of operation are used with the 128K extended memory systems: map mode (see Section 7.3.1) and direct addressing mode (see Section 7.3.2). The direct addressing mode is an additional option on the system.

7.3.1 Map Mode

The map mode accesses extended memory by switching in a 4K block of extended memory (two blocks per 8K module or four blocks per 16K module) in place of any of the 4K blocks in the normal 16K to 32K memory area (called the variable area). This switching is instantaneous, accomplished via a programmed I/O data output instruction (XIO instruction).

Decimal Address	Hexadecimal Address	
0 - 4K	0000 - 0FFF	
4 - 8K	1000 - 1FFF	Permanent
8 - 12K	2000 - 2FFF	(Non-Mappable Memory)
12 - 16K	3000 - 3FFF	
16 - 12K	4000 - 4FFF	
20 - 24K	5000 - 5FFF	Variable
24 - 28K	6000 - 6FFF	(Mappable Memory)
28 - 32K	7000 - 7FFF	

These first 32K words are assigned to the first four 8K memory modules, or the first two 16K memory modules, MEM1 through MEM4 (see Figure 7-3). When the system is in map mode, XIO instructions selectively map extended memory through the 16 to 32K area.

All 96K of extended memory (modules in slots MEM5 through MEM16) is initially inactive (unaddressable). The execution of a mapping XIO instruction permits addressing of a 4K extended memory block, deactivating a 4K block in the variable area (16K to 32K).

SLOT	MEMORY ALLOCATION			ADDRESS ASSIGNMENTS	
MEM 13	100-104K	96-100K	5TH EXTENDED MEMORY GROUP	4K*	4K*
MEM 14	108-112K	104-108K		4K*	4K*
MEM 15	116-120K	112-116K	6TH EXTENDED MEMORY GROUP	4K*	4K*
MEM 16	124-128K	120-124K		4K*	4K*
EMLD	EXTENDED MEMORY LOGIC DISPLAY BOARD			EXTENDED MEMORY LOGIC DISPLAY BOARD	
MEM 9	68-72K	64-68K	3RD EXTENDED MEMORY GROUP	4K*	4K*
MEM 10	76-80K	72-76K		4K*	4K*
MEM 11	84-88K	80-84K	4TH EXTENDED MEMORY GROUP	4K*	4K*
MEM 12	92-96K	88-92K		4K*	4K*
MEM 5	36-40K	32-36K	1ST EXTENDED MEMORY GROUP	4K*	4K*
MEM 6	44-48K	40-44K		4K*	4K*
MEM 7	52-56K	48-52K	2ND EXTENDED MEMORY GROUP	4K*	4K*
MEM 8	60-64K	56-60K		4K*	4K*
MEM 1	4-8K	0-4K	PERMANENTLY ASSIGNED AREA	1000-1FFF	0000-0FFF
MEM 2	12-16K	8-12K		3000-3FFF	2000-2FFF
MEM 3	20-24K	16-20K	VARIABLE AREA	5000-5FFF	4000-4FFF
MEM 4	28-32K	24-28K		7000-7FFF	6000-6FFF

8K MODULE = ONE MEM x SLOT
16K MODULE = TWO MEM x SLOTS

*ANY ONE 4K MAY BE MAPPED (PROGRAMMED) INTO THOSE ADDRESSES NORMALLY ASSIGNED TO THE BOARDS IN SLOTS MEM 3 OR MEM 4 (VARIABLE AREA).

Figure 7-3. SPC-16/45, 65 Memory Expansion.

PROGRAMMING FOR MAP MODE

The 128K extended memory system uses programmed data output XIO instructions with device select code X'39' to select the extended memory area. There are two types of data output instructions: Data Out of Register (DTOR) and Data Out of Memory (DTOM). When a DTOR instruction is issued (e.g., DTOR A,X'39'), the data word located in the specified register is output. When a DTOM is issued (e.g., DTOM Z,X'39'), the indicated register supplies the memory address from which the data word will be output. Figure 7-4 illustrates format of the data output instructions.

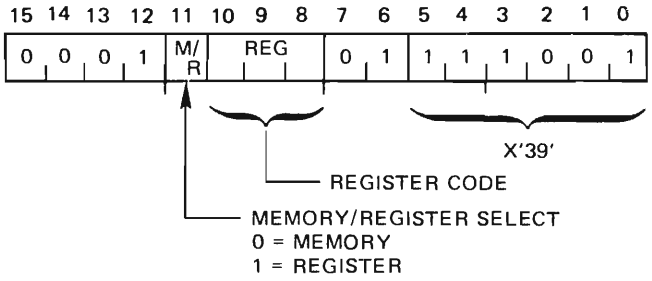


Figure 7-4. Programmed Input/Output Instructions.

The data words output by the DTOR/DTOM instructions follow the format illustrated in Figure 7-5. Refer back to Figure 7-3 for the physical location of the memory areas listed below.

The output data word format allows any one of twenty-four 4K extended memory blocks to be mapped into any one of the four variable address blocks (16-20K, 20-24K, 24-28K, 28-32K), at the same time de-selecting a previously specified variable address block. The data output word also allows the normal variable address block to be reinstated, de-selecting the previously specified extended memory block in that position.

As shown in Figure 7-5, a combination of bits 0-3 and 8-11 of the output data word specify the extended memory block: bits 0-3 select one of the six 16K groups in the extended area; bits 8-11 select one of the four 4K blocks within that group. Bits 4-7 specify the variable address block the 4K extended memory block will be mapped into.

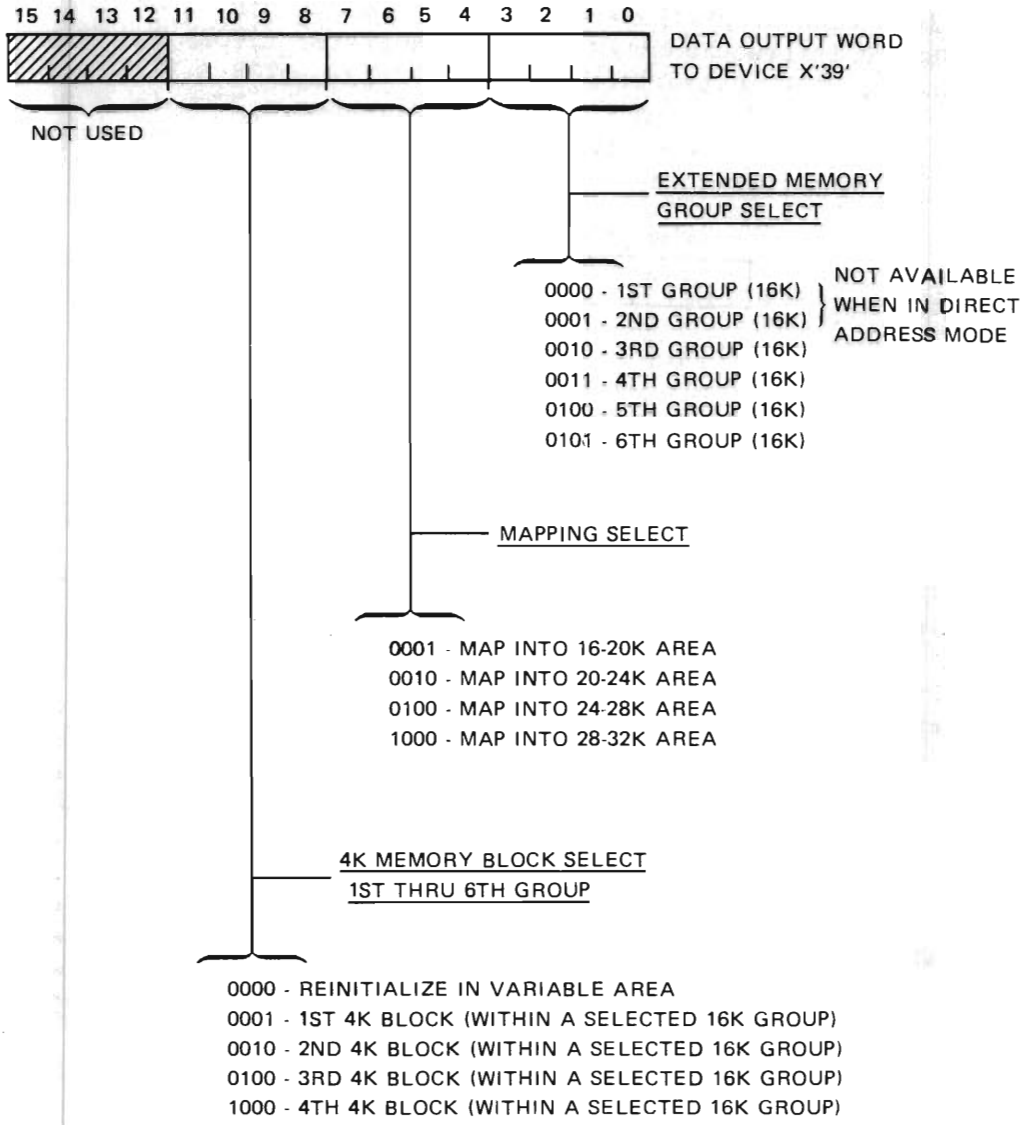
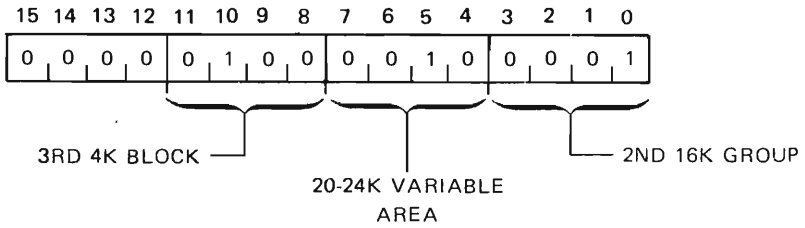


Figure 7-5. Format of Output Data Word.

If the instruction DTOR A,X'39' is executed while the A register contains X'0421' (the data word):

A REGISTER (X'0421')

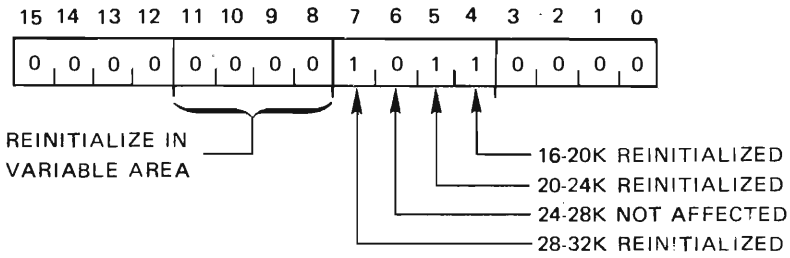


The third memory block in the second extended group will be mapped into the 20-24K address block, de-activating that block. Subsequent references to memory locations 5000_{16} $5FFF_{16}$ will access this third block of the second group until a further data output instruction restores the area to its initial state.

To reinstate one or more of the four address blocks in the variable area, the data output word must have all zeroes in bits 8-11 (see Figure 7-5).

If the data output instruction DTOR A,X'39' is executed while the A register contains X'00B0':

A REGISTER (X'00B0')



The first, second and fourth variable address blocks are restored to the initial status as memory locations 16-20K, 20-24K and 28-32K, respectively. The 42-28K block is not affected by the execution of this instruction.

7.3.2 Direct Addressing Mode

The direct addressing mode permits access of the first 32K of extended memory via base relative, indexed or indirect addressing instructions. (See Section 3 for a detailed discussion of address generation.) Under the first addressing mode, these two groups of extended memory (groups 1 and 2, MEM5 through MEM8) are addressed by generating effective operand addresses in the range 8000_{16} through $FFFF_{16}$.

Because of these generated addresses, groups 1 and 2 of extended memory are also accessible via direct memory access (DMA) under the direct memory addressing mode. This availability effectively doubles the memory available for DMA.

Because the direct addressing mode permits generation of assigned addresses for the first and second extended memory groups (32K to 64K), they are not available for mapping. The rest of extended memory (64K to 128K) remains unchanged; this area may still be mapped into the variable 16K to 32K area.

A switch added on the left side of the extended memory logic display board (viewed from front of machine) indicates that the direct addressing option has been added to a system. This switch is accessed only by opening the front panel of the extended memory chassis. The switch has two positions: MAP and MAP/DA/ Setting the switch to MAP/DA activates the direct addressing mode.

Figure 7-6 illustrates the physical 4K block assignments and the corresponding address assignments under direct addressing mode.

PROGRAMMING FOR DIRECT ADDRESS MODE

To operate in the direct addressing mode, the system with this option must have the switch on the memory logic display board moved to the MAP/DA position.

Under direct addressing mode, all instructions (including the instructions for map mode) operate as they normally would, however, only 64K of memory is available for mapping into the variable area. The 32K of extended memory is reserved for data storage and retrieval. This area is also available for direct memory access.

Programs are still limited to the lower 32K of memory (i.e., addresses up to $7FFF_{16}$), however, the direct addressing mode permits generation of effective operand addresses in the range 8000_{16} to $FFFF_{16}$ using indexed, indirect, or base relative instructions.

	MEMORY ALLOCATION			ADDRESS ASSIGNMENTS	
MEM 13	100-104K	96-100K	5TH EXTENDED MEMORY GROUP	4K*	4K*
MEM 14	108-112K	104-108K		4K*	4K*
MEM 15	116-120K	112-116K	6TH EXTENDED MEMORY GROUP	4K*	4K*
MEM 16	124-128K	120-124K		4K*	4K*
EMLD	EXTENDED MEMORY LOGIC DISPLAY BOARD			EXTENDED MEMORY LOGIC DISPLAY BOARD	
MEM 9	68-72K	64-68K	3RD EXTENDED MEMORY GROUP	4K*	4K*
MEM 10	76-80K	72-76K		4K*	4K*
MEM 11	84-88K	80-84K	4TH EXTENDED MEMORY GROUP	4K*	4K*
MEM 12	92-96K	88-92K		4K*	4K*
MEM 5	36-40K	32-36K	1ST EXTENDED MEMORY GROUP	9000-9FFF	8000-8FFF
MEM 6	44-48K	40-44K		B000-BFFF	A000-AFFF
MEM 7	52-56K	48-52K	2ND EXTENDED MEMORY GROUP	D000-DFFF	C000-CFFF
MEM 8	60-64K	56-60K		F000-FFFF	E000-EFFF
MEM 1	4-8K	0-4K	PERMANENTLY ASSIGNED AREA	1000-1FFF	0000-0FFF
MEM 2	12-16K	8-12K		3000-3FFF	2000-2FFF
MEM 3	20-24K	16-20K	VARIABLE AREA	5000-5FFF	4000-4FFF
MEM 4	28-32K	24-28K		7000-7FFF	6000-6FFF

*ANY ONE 4K BLOCK MAY BE MAPPED (PROGRAMMED) INTO THOSE ADDRESSES NORMALLY ASSIGNED TO THE BOARDS IN SLOTS MEM 3 OR MEM 4 (VARIABLE AREA).

Figure 7-6. SPC-16/45, 65 Memory Expansion With Direct Addressing Mode

7.3.3 Console Indicators

Lamps, coded in octal on the 128K extended Memory System front panel, indicate the status of each of the four 4K blocks in the variable area: 16-20K, 20-24K, 24-28K and 28-32K. Figure 7-7 illustrates the array of lamps on the front panel. The top row of lamps indicates which 16K extended memory group is selected, and the bottom row indicates which of the four 4K blocks is mapped into the variable area.

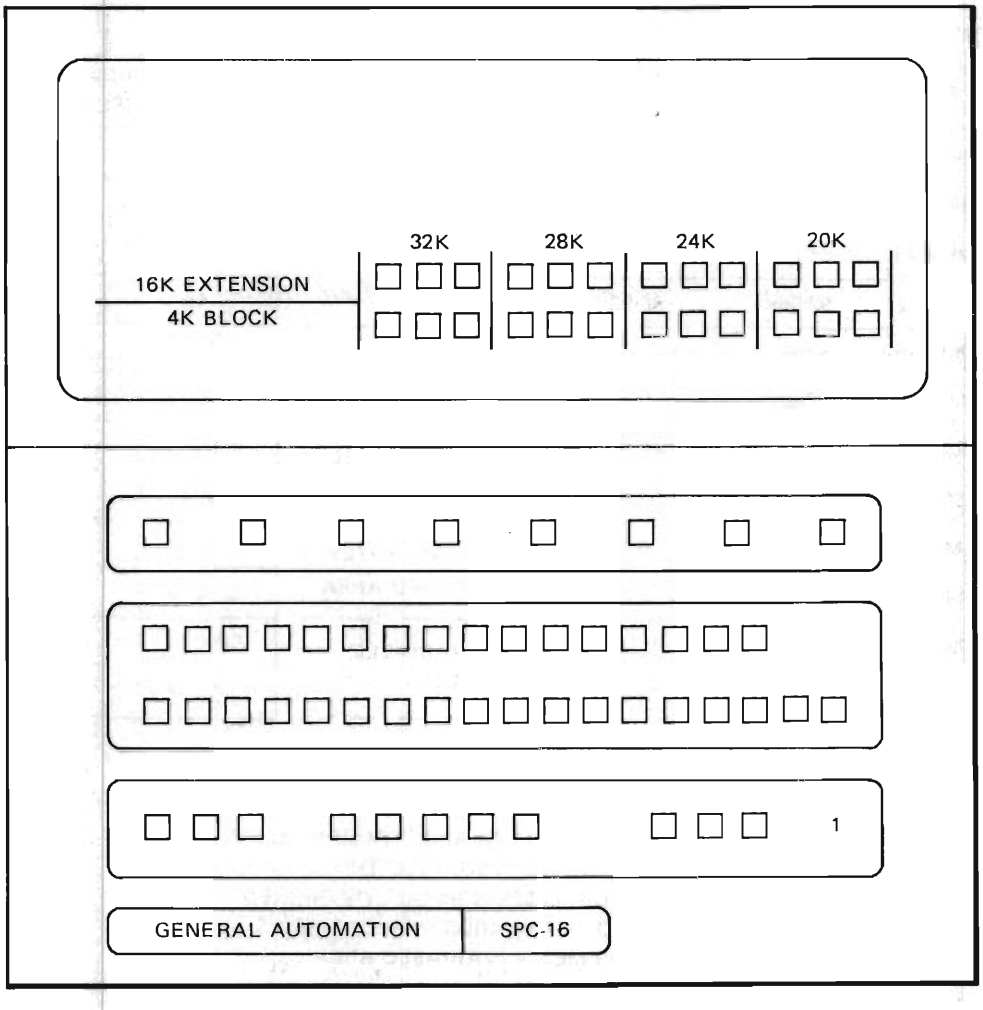
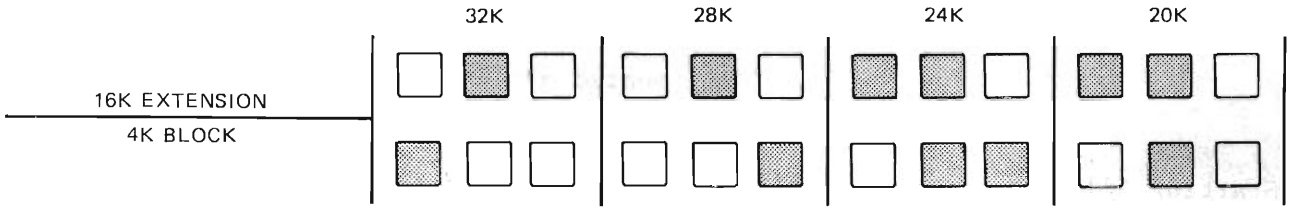


Figure 7-7. Lamp Array on Console Panel of SPC-16 With Extended Memory.

Figure 7-8 illustrates a sample array on the extended memory indicators.



(SHADED INDICATORS ARE LIT)

Figure 7-8. Sample Array on Extended Memory Console Lamps.

The array in Figure 7-8 would indicate:

- The second 4K block of the sixth memory extension group is mapped into the 16-20K variable area.
- The third 4K block of the sixth memory extension group is mapped into the 20-24K variable area.
- The first 4K block of the second memory extension group is mapped into the 24-28K variable area.
- The fourth 4K block of the second memory extension group is mapped into the 28-32K area.

7.4 MEMORY PROTECT OPTION

The Memory Protect option is available for the SPC-16/40, 45, 60, 65, 80 and 85. The unit may be physically plugged into any 4K, 8K or 16K Read/Write Memory module (more than one memory module on a given system may include a Memory Protect unit). No module can support a Memory Protect unit and a ROM at the same time.

On a given module, the Memory Protect unit is hardwired to protect a user-specified area of memory within that module. Within a 4K module, the size of the protected area can range between 8 words and 4K words, in powers of 2 (i.e., 8, 16, 32, 64, ...4K). Within an 8K or 16K module, an 8- to 4K-word block of memory can be protected in any one or all 4K portions of the module. (The same area of core is protected in each 4K memory section.)

The Memory Protect option includes a Memory Protect Violation interrupt that is activated should the program attempt to store information into a protected area. The block protect function is disabled by a switch on each memory module (the switch only disables the protect function of that module) or by a common line on the I/O bus (CLDS-I, which disables all memory protect modules).

SPECIFICATIONS

Mechanical:	Piggy-back pluggable module mounting to a 4K, 8K, or 16K memory module.
Block Size:	2^N $3 \leq N \leq 12$ (Implemented by jumper wires to user's specification.)
Starting Address:	Equal to zero or any integral multiple of the block size (address is relative to a 4K module). (Implemented by jumper wires to user's specification.)
Memory Protect Violation Interrupt:	Interrupt is maskable (see below) with an interrupt vector location of X'42'.

The Memory Protect Violation, interrupt mask is not initialized when the Systems Reset switch on the computer's console is pressed or following a Power-Fail and Auto-Restart sequence. This is an internal interrupt and is controlled via the internal interrupt masking word (device select code X'3E', see Section 4.2).

When the Memory Protect Violation interrupt is disabled, an attempt to write into a protected area will still be unsuccessful. However, the program will not 'know' that an illegal write operation has been attempted.

It is the programmer's responsibility to ensure that memory location X'42' contains the address of the routine responsible for servicing a Memory Protect Violation interrupt. An attempt to write into a protected memory location usually indicates a programming error. The interrupt service routine should perform the operations appropriate to the particular application.

As with any interrupt, the E register will contain the address of the memory location following the one containing the last instruction executed. In the case of the Memory Protect Violation interrupt, it was this instruction that caused the interrupt (i.e., the instruction tried to write into a protected area).

Jumper wire programming information is contained in General Automation operational manual 88A00707A, "SPC-16 Memory Protect Option".

7.5 HIGH-SPEED SIGNED MPY/DIV

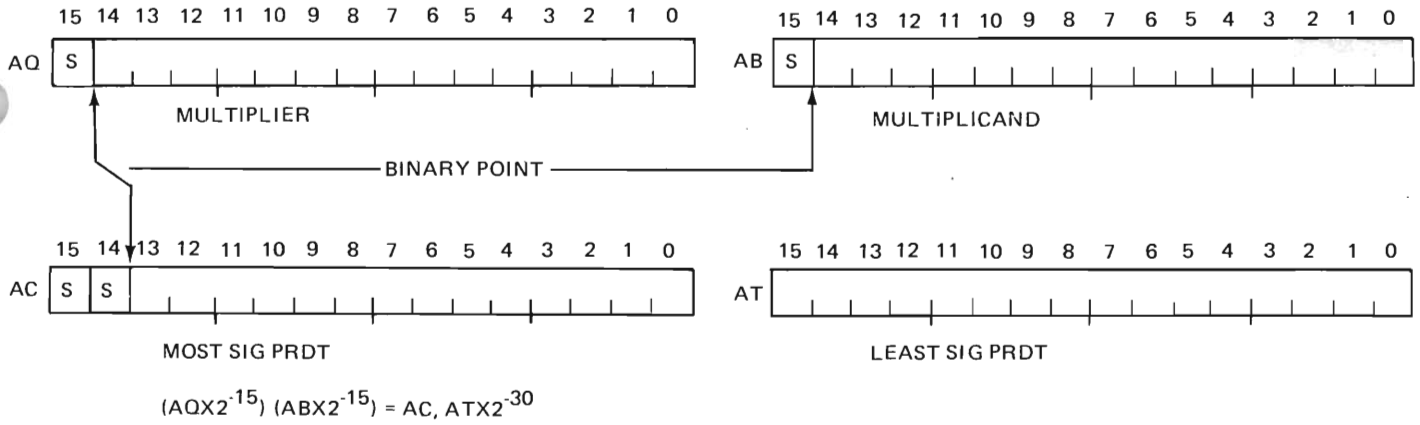
The High-Speed Signed (HSS) MPY/DIV option performs signed, single-precision multiplication and division. A total of ten memory cycles is normally required for a complete multiplication or division operation.

From both a hardware and software standpoint, the HSS MPY/DIV option is an input/output device. The option is implemented on two I/O controller-size printed circuit boards and mounts in any two adjacent I/O card slots. Four I/O device select codes: (X'3A', X'3B', X'3C', X'3D') have been assigned to the device to input and output operands and results and initiate the multiplication or division operation.

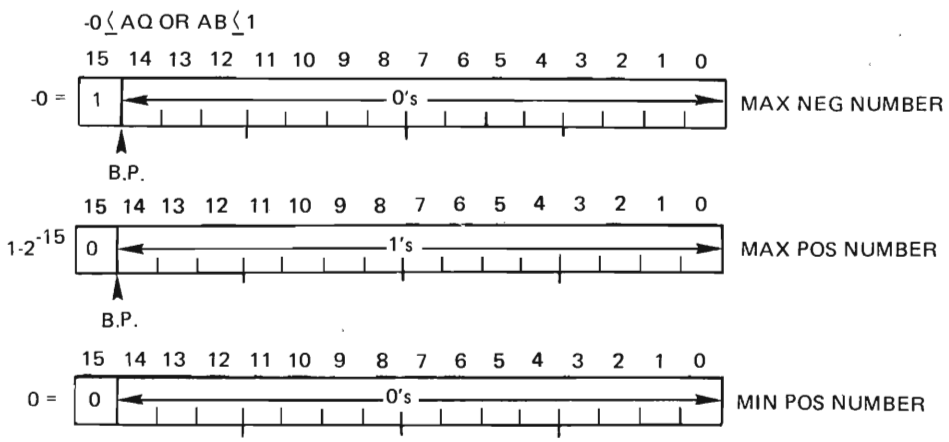
7.5.1 Multiplication

ALGORITHM

The HSS MPY/DIV algorithm operates on two 16-bit (15 bits plus sign) fractional numbers and results in a 30-bit plus sign fractional product as follows: (AQ, AB, AC and AT are operational registers in the option boards:



Negative numbers are presented in twos complement format. The range of numbers that may be handled is defined as follows:



PROGRAMMING

As mentioned earlier, the HSS MPY/DIV appears as another input/output device to the central processor. The Multiplier and Multiplicand may, therefore, be sent to the device using either the DTOR (if the operands are in General Purpose registers) or the DTOM (if the operands are in memory) XIO instruction. Likewise, the product may be input from the device using either DTIR (data to registers) or DTIM (data to memory) XIO instruction.

The programming sequence to perform the MPY function is as follows:

TIME		
2 Cyc	(1)	Output Multiplier to AQ register, dev. select code = X'3A' DTCR R, X'3A' or DTOM R, X'3A' where R is any General Purpose register.
2 Cyc	(2)	Output Multiplicand to AB register, dev. select code = X'3B' This instruction indicates multiplication. DTOR R, X'3B' or DTOM R X'3B'
2 Cyc	(3)	Allow 2 memory cycles for operation. Program two no-ops if product is to be input to registers. Program one no-op and set up address if product is to be input to memory.
TIME		
2 Cyc	(4)	Input most significant PRDT from AC register, dev. select code = X'3C'. DTIR R, X'3C' or DTIM R, X'3C'
2 Cyc	(5)	Input least significant PRDT from AT register, dev. select code = X'3A' DTIR R, X'3A' or DTIM R, X'3A'

Steps (4) and (5) may be reversed.

EXAMPLE:

The following example uses the HSS MPY/DIV unit to multiply the contents of location M1 by the contents of location M2 and store the two word product in locations P1 and P2:

```
                DSECT
M1              DC      X'7FFF'     First argument (multiplier)
M2              DC      X'7FFF'     Second argument (multiplicand)
P1              DC      0           Product - LS 16 bits
P2              DC      0                     MS 14 bits + sign
                PSECT
                LDV     Z,M1        Set up address of arguments
                DTOM    Z,X'3A'     Output multiplier (M1)
                INCR    Z           Increment argument address
                DTOM    Z, X'3B'    Output multiplicand (M2) and
                                    initiate multiply
                SKZ     $+1   }     Two memory cycles
                INCR    Z     }     for operation
                DTIM    Z, X'3A'    Input LS 16 bits (P1)
                INCR    Z
                DTIM    Z, X'3C'    Input MS 14 bits (P2)
```

7.5.2 Division

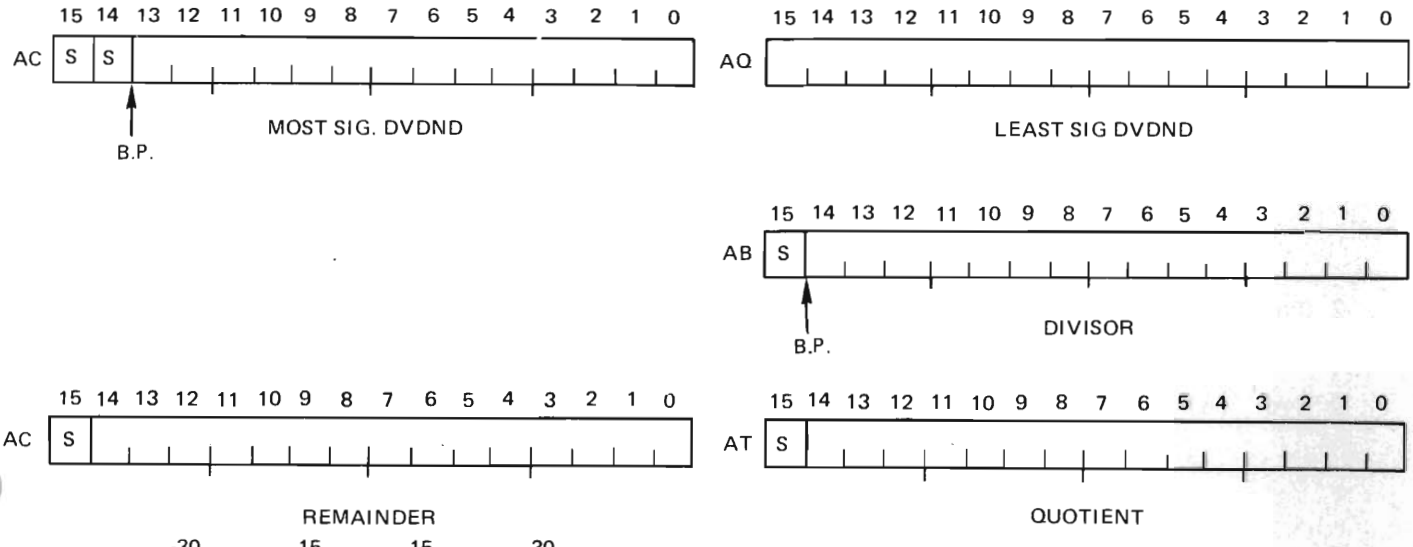
ALGORITHM

The division algorithm is a non-restoring type with appropriate corrections to the quotient and remainder. This division is defined such that

(DIVIDEND) = (QUOTIENT) (DIVISOR) + REMAINDER
AND $|\text{REMAINDER}| \leq |\text{DIVISOR}|$
AND SIGN OF REMAINDER = SIGN OF DIVIDEND

i.e., (AC,AQ) = (AT) (AB) + AC
AND $|\text{AC}| \leq |\text{AB}|$
AND SIGN AC BEFORE = SIGN OF AC AFTER

The operands and results are as follows:



$(AC,AQ) \times 2^{-30} = (ATX2^{-15})\ (ABX2^{-15}) + ACX2^{-30}$

Negative numbers are presented in 2's complement format and the range and definition of numbers are equivalent to those explained under the MPY section.

OVERFLOW

In general, an overflow condition will be detected if the results of the division process cannot be contained in the format listed previously. An overflow will exist under the following conditions:

1. SIGN of 1/2 (AC, AQ) ≠ SIGN of (AC,AQ), or
2.

AC	AB	Condition
+	+	(AC,AQ)≥AB
-	+	|AC,AQ| > |AB|
+	-	|AC,AQ| ≥ |AB|
-	-	|AC,AQ| ≥ |AB|

If one is concerned about an overflow condition, the program must sense the condition by executing an XIO input using device select code X'3D'. If an overflow condition exists, a '1' bit will be input on bit 15, otherwise a '0' bit will be input.

PROGRAMMING

The programming sequence to perform the DIV function is as follows:

TIME		
2 Cyc	(1)	Output least significant DVDND to AQ register, dev. select code = X'3A'. DTOR r, X'3A' or DTOM R, X'3A' where R is any General Purpose register.
2 Cyc	(2)	Output most significant DVDND to AB register, dev. select code = X'3C'. (Hardware moves AB→ AC) DTOR R, X'3C' or DTOM R,X'3C'
2 Cyc	(3)	Output DIVISOR to AB register, dev. select code = X'3D'. This instruction initiates division. DTOR R, X'3D' or DTOM R, X'3D'
2 Cyc	(4)	Allow 2 memory cycles for operation. Program two no-ops if quotient and remainder are to be input to registers. Program one no-op and set-up address if product is to be input to memory.
2 Cyc	(5)	Overflow can be input, dev. select code = X'3D' DTIR R, X'3D' or DTIM R, X'3D'
2 Cyc	(6)	Input QUOTIENT from AT register, dev. select code = X'3A'. DTIR R, X'3A' or DTIM R, X'3A'

2 Cyc (7) Input REMAINDER from AC register, dev. select code = X'3C'.
DTIR R, X'3C' or DTIM R, X'3C'

EXAMPLE

The following example uses HSS MPY/DIV to divide the contents of locations D1 and D2 by D3 and store the quotient in Q1 and the remainder in R1. The routine also tests for an overflow condition before inputting the answer.

```
            DSECT
D1          DC      X'5555'     LS 16 bits of dividend
D2          DC      X'2AAA'     MS 14 bits of dividend + sign
D3          DC      X'5555'     Divisor + sign
Q1          DC      0           Quotient + sign
R1          DC      0           Remainder + sign
            PSECT
            LDV     Z,D1        Set up parameter address
            DTOM    Z,X'3A'     Output LS 16 bits of dividend (D1)
            INCR    Z           Increment parameter address
            DTOM    Z,X'3C'     Output MS 14 bits dividend (D2)
            INCR    Z
            DTOM    Z,X'3D'     Output divisor (D3) and initiate divide
            SKZ     $+1  }      Two memory
            INCR    Z    }      cycles for operation
            DTIR    A,X'3D'     Test for overflow
            ANDV    A,X'8000'
            SKZ     $+2         Skip next instruction if no OVRFLW
            JSR     OVRFLW
            DTIM    Z,X'3A'     Input quotient (Q1)
            INCR    Z
            DTIM    Z,X'3C'     Input remainder (R1)
```

7.6 ARITHMETIC PROCESSING UNIT

General Automation offers an arithmetic processing unit that provides the user extremely fast arithmetic operations with a high degree of accuracy.

The features of the Arithmetic Processing Unit (APU) can be summarized as follows:

- Single and Double Precision Floating Point Arithmetic
- Double Precision Integer Arithmetic
- Interrupt generated on any error

NOTE

Errors are: Overflow (result equals max. possible value), underflow (results equals zero), divide by unnormalized number (original value retained) and fix overflow (result equals max. possible value).

- Truncate or Round Mode
- Automatic Argument Transfer
- Concurrent operation with CPU

The Arithmetic Processing Unit is implemented on five I/O controller-size printed circuit boards and occupies five contiguous card slots in a Model 1615 I/O Enclosure.

Data formats used by the APU are:

- Double Precision Integer

  31 bits plus sign -2,147,483,648 to +2,147,483,647

- Single Precision Floating Point

Mantissa	24 bits plus sign	7.1 decimal digits
Exponent	6 bits plus sign	$2.7 * 10^{-20}$ to $9.2 * 10^{18}$

- Double Precision Floating Point

Mantissa	56 bits plus sign	16.7 decimal digits
Exponent	6 bits plus sign	$2.7 * 10^{-20}$ to $1.2 * 10^{18}$

The average arithmetic calculation takes about six microseconds. The fastest take about 3 microseconds and the slowest (a worst-case double-precision floating-point divide) takes about 17 microseconds.

SECTION 8
PHYSICAL PLANNING INFORMATION

Several factors are important in planning the installation of a computer system. The planner must be aware of the physical dimensions of the system so that the necessary space can be allocated before the installation procedure begins. He must also know the lengths of cabling required in order to plan the placement of the system's various elements. Further, AC power requirements must be known in order to determine whether the existing source is adequate. Finally, the planner must know the system's temperature tolerances and heat output in order to determine whether the site will provide adequate heating and cooling for the system.

This section contains information on each of the physical planning factors described above. Since specific user applications are beyond the scope of this manual, discussion is limited to the processor-power supply-I/O enclosure level. Section 8.1 covers physical space considerations, including dimensions and system cabling. Section 8.2 covers power requirements. Section 8.3 covers environmental specifications.

For a more detailed description of this material, the reader should consult the SPC-16 System Installation and Physical Planning manual (88A00263A).

8.1 PHYSICAL SPACE CONSIDERATIONS

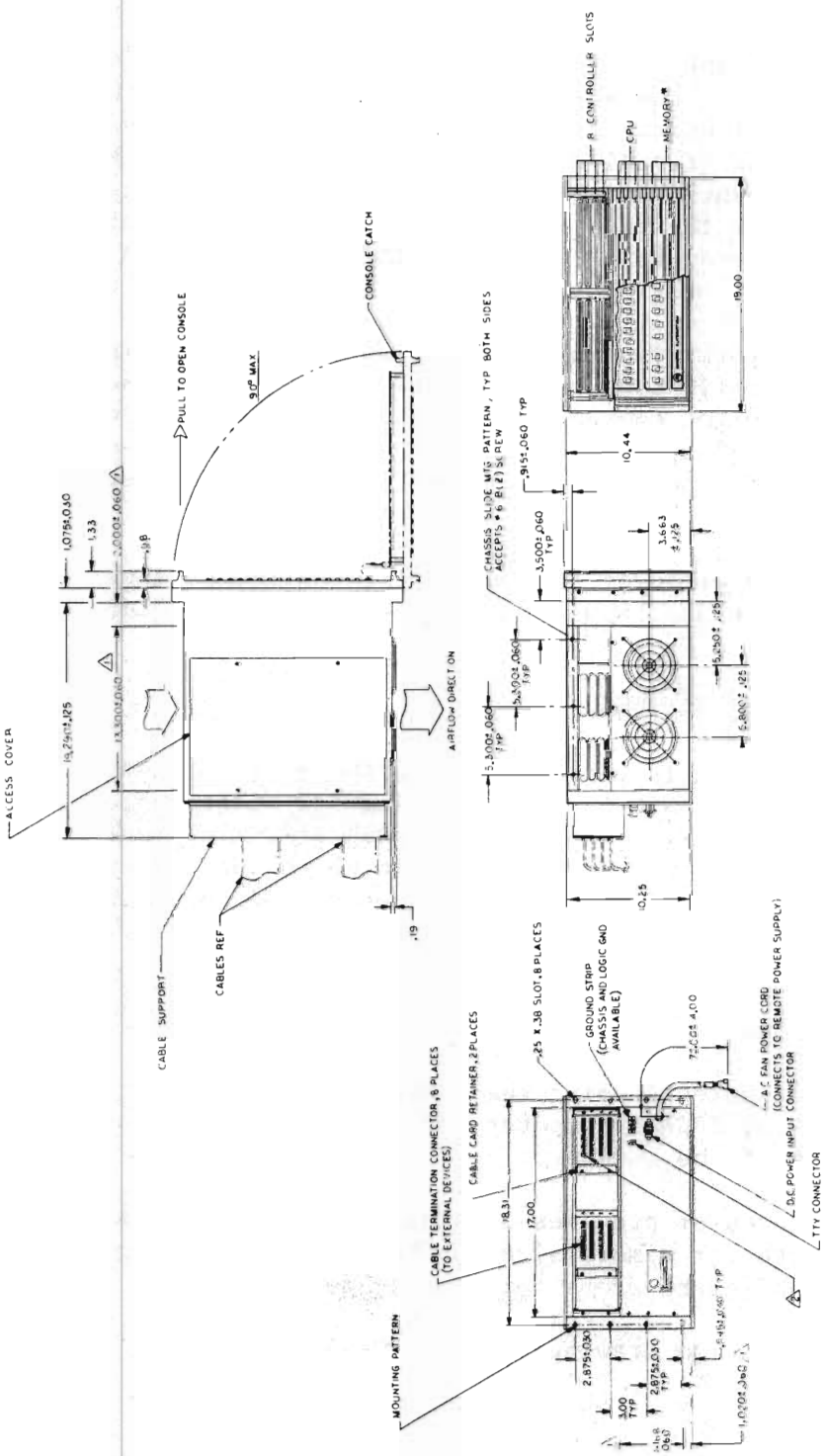
The material in this section is intended to aid the installation planner in preparing the site to accept an SPC-16 processor and its basic supporting equipment. Section 8.1.1 presents overall dimensions for each of the elements of a basic system. Section 8.1.2 presents configurations in which the system can be arranged in a GA System Enclosure. Section 8.1.3 describes cabling between each of the elements of the system.

8.1.1 Overall Dimensions

This section contains figures showing the overall dimensions of the various models and configurations of the SPC-16 computer and the General Automation products pertinent to installation of the computer.

The SPC-16/40/60/80 processor provides a mainframe chassis containing the CPU boards, four memory module slots for memory sizes up to 32K memory using 4K, 8K, or 16K memory module sizes, and eight slots for I/O controllers.

Figure 8-1 is an installation drawing of the SPC-16/40/60/80.



*NOTE
FOUR MEMORY SLOTS

Figure 8-1. Installation Drawing, SPC-16/40/60/80

The SPC-16/45/65/85 is provided in three basic configurations. Provision for I/O is made external to the computer mainframe.

1. The basic configuration provides the CPU chassis containing the CPU boards, and slots for 4 memory modules. 4K, 8K, or 16K memory module sizes can be accommodated by this configuration with total memory capacity up to 32K of memory.

2. The 64K memory expansion configuration provides the CPU chassis containing the CPU boards, an extended Master Interconnect Board (MIB), and extended chassis for additional memory capacity up to 64K of memory and one or two Memory Logic and Display (MLD) boards for mapping of memory beyond 32K. 4K, 8K, or 16K memory module sizes can be accommodated by this configuration with total memory capacity up to 64K of memory.

3. The 128K memory expansion configuration provides the CPU chassis containing the CPU boards, an extended MIB and extended chassis for additional memory capacity up to 128K of memory and a single MLD board for mapping of memory beyond 32K. 4K, 8K, pr 16K memory module sizes can be accommodated by this configuration with total memory capacity up to 128K of memory.

Figure 8-2 is an installation drawing of the SPC-16/45/65/85 without memory expansion. Figure 8-3 is an installation drawing of the SPC-16/45/65/85 with memory expansion to 64K. Figure 8-4 is an installation drawing of the SPC-16/45/65 with memory expansion to 128K.

Figure 8-5 is an installation drawing of the Processor Power Supply, which is used with all models of the SPC-16 processor family.

Figure 8-6 is an installation drawing of the Model 1615 I/O Enclosure, which is optional for the SPC-16/40/60/80 series and is required for the SPC-16/45/65/85 series. The power supply for the I/O Enclosure is identical in appearance to the Processor Power Supply (Figure 8-5) and is not, therefore, illustrated separately.

Figure 8-7 is an installation drawing of the optional Model 1995 Mounting Slides. These slides are required to mount the 3322 Paper Tape Punch, the 3325 Paper Tape Reader/Punch, the 3342 Head per Track Storage Drive and the 3346 or 3347 Disk Storage Drive. Some users may also wish to mount their SPC-16 processor and/or 1615 I/O Enclosure or mounting slides, as shown, but this is not a general practice.

Figure 8-8 is an installation drawing of the Model 1910 System Enclosure. The model 1901 System Enclosure is identical to the Model 1910 except that the AC outlet strip is replaced by a power distribution panel in the Model 1901.

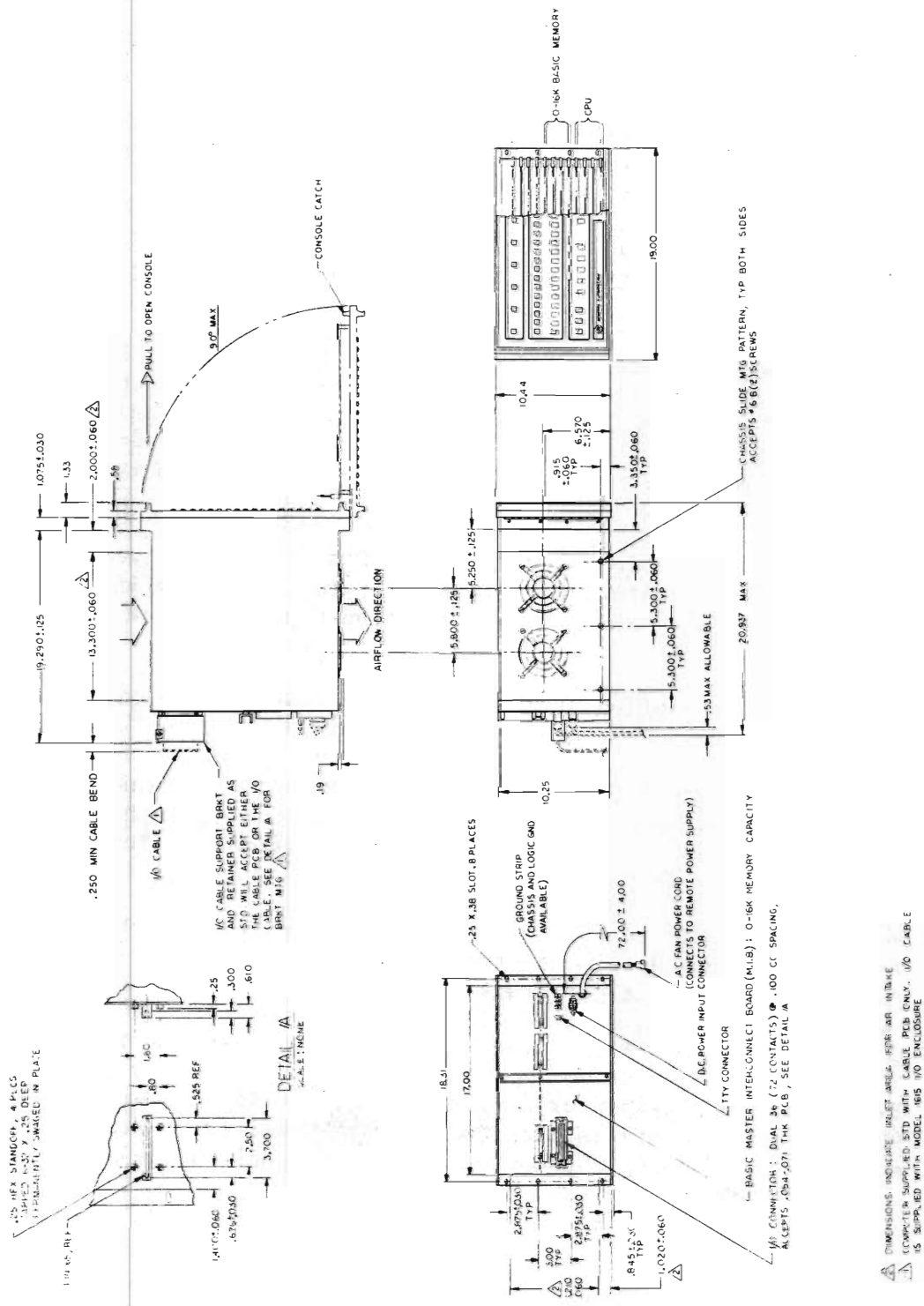


Figure 8-2. Installation Drawing, SPC-16/45/65/85 With 0 - 32K Memory Capability

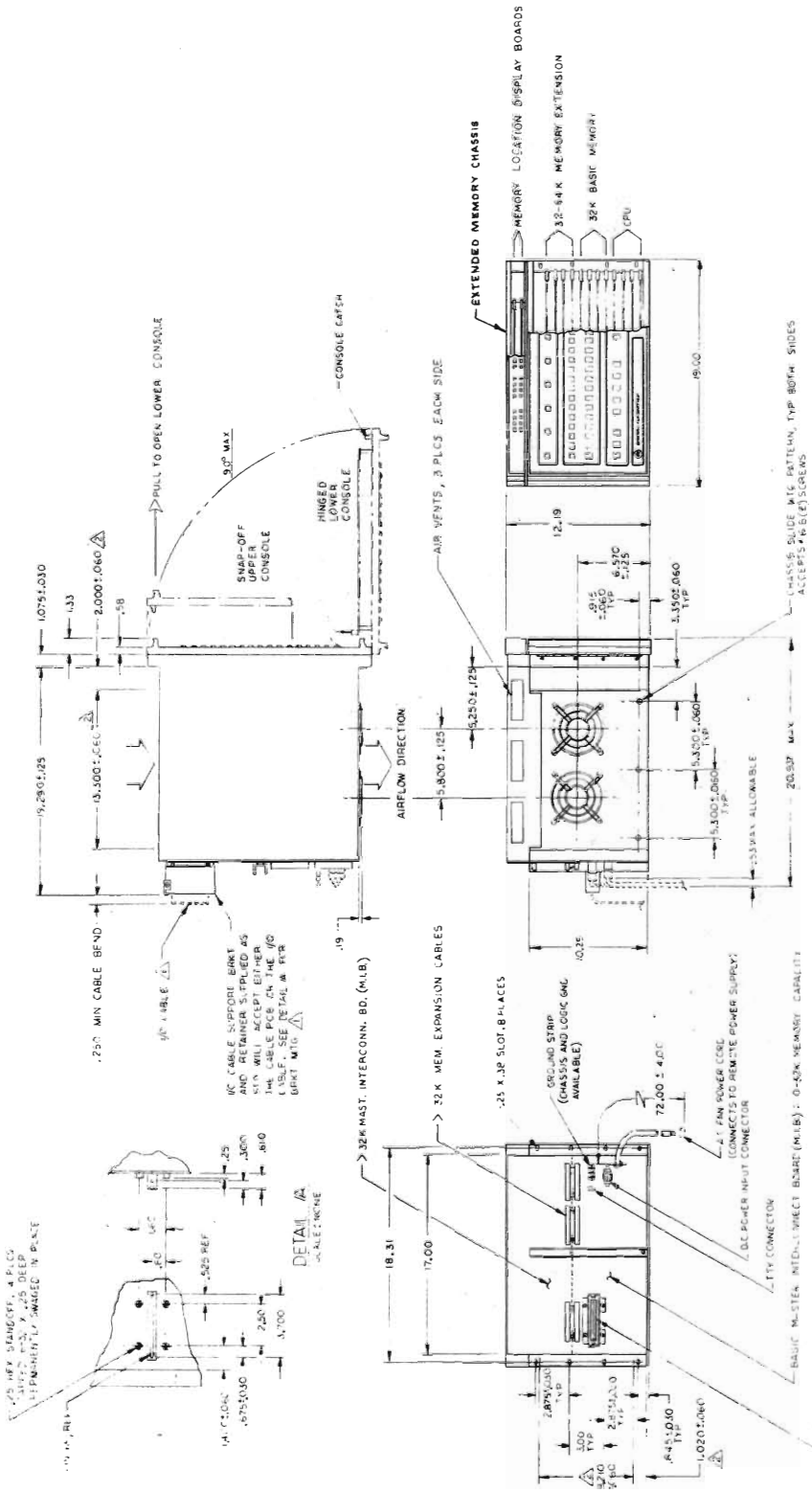


Figure 8-3. Installation Drawing, SPC-16/45/65/85 With Extended 32 - 64K Memory Option

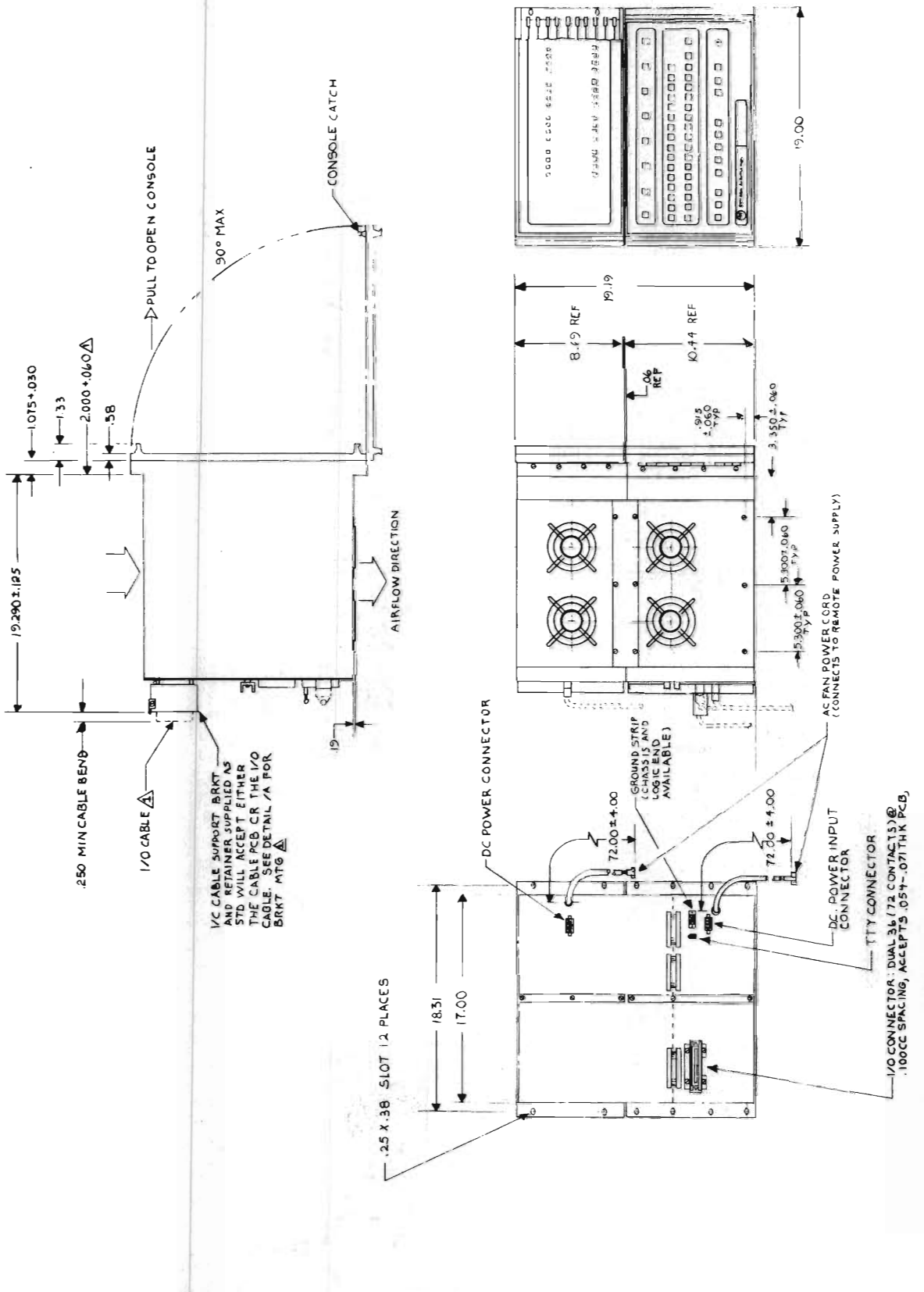


Figure 8-4. Installation Drawing, SPC-16 45/65 with Extended 64 - 128K Memory Option.

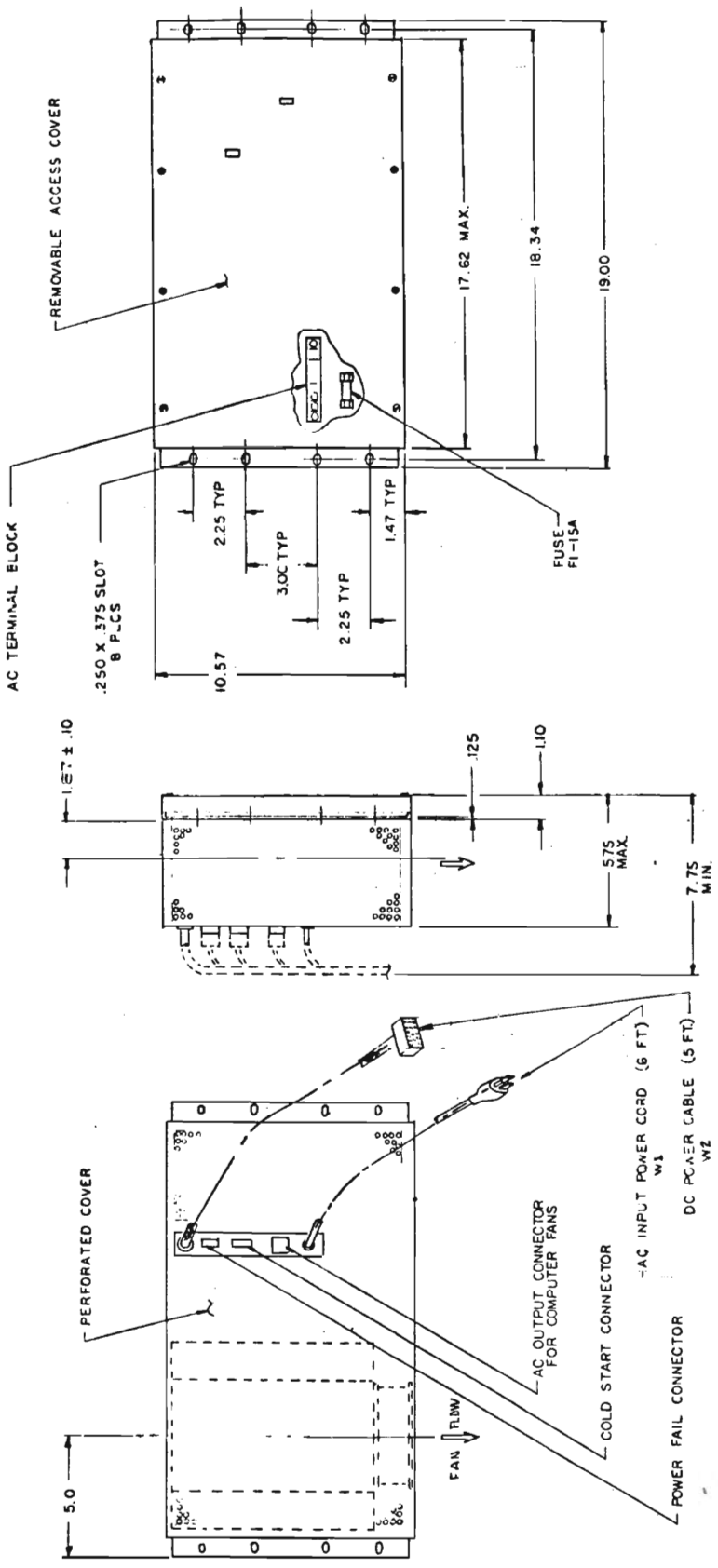


Figure 8-5. Installation Drawing 375-Watt Processor Power Supply.
(Typical of CPU and I/O Power Supply)

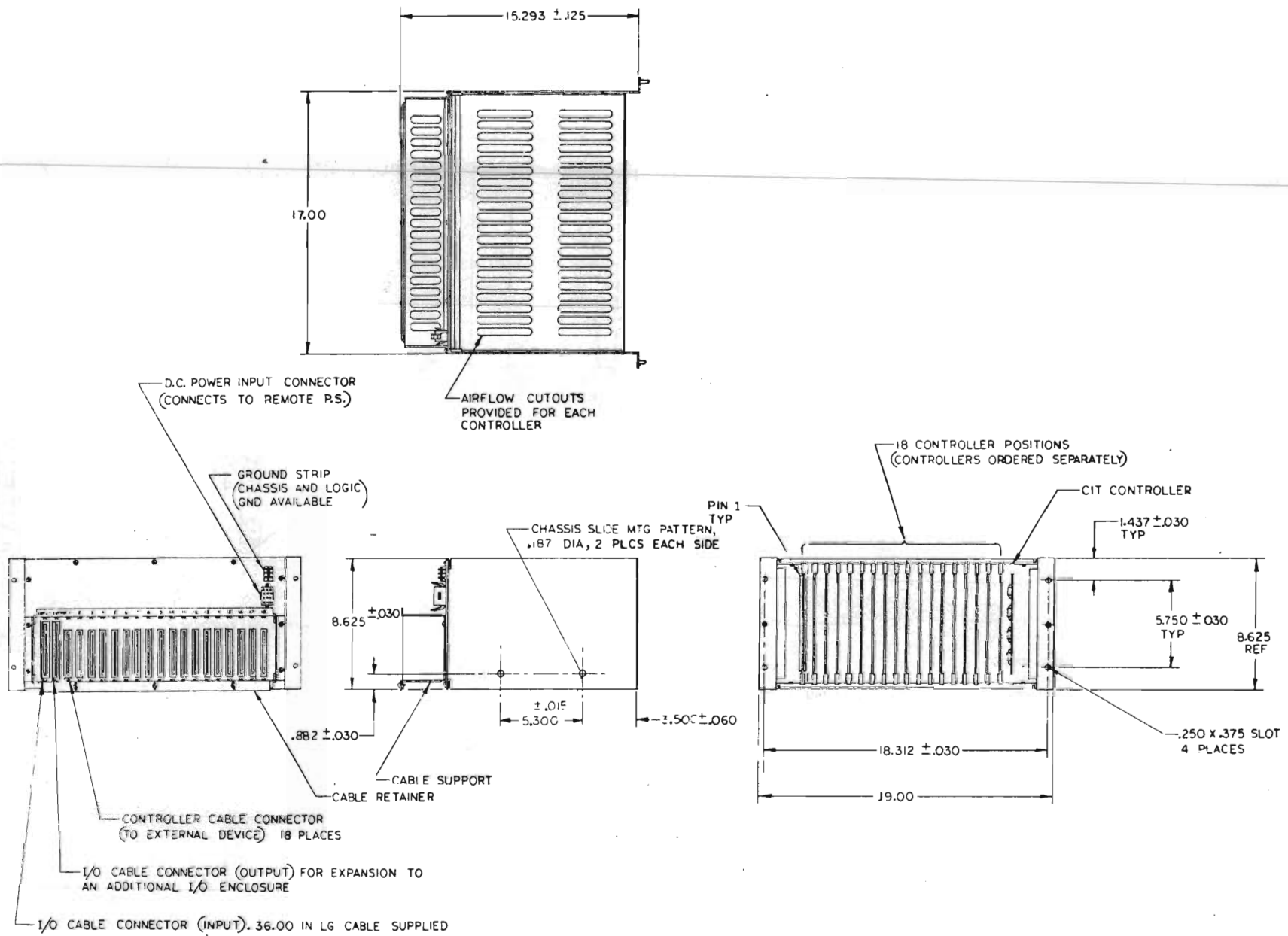


Figure 8-6. Installation Drawing, I/O Enclosure Model 1615.

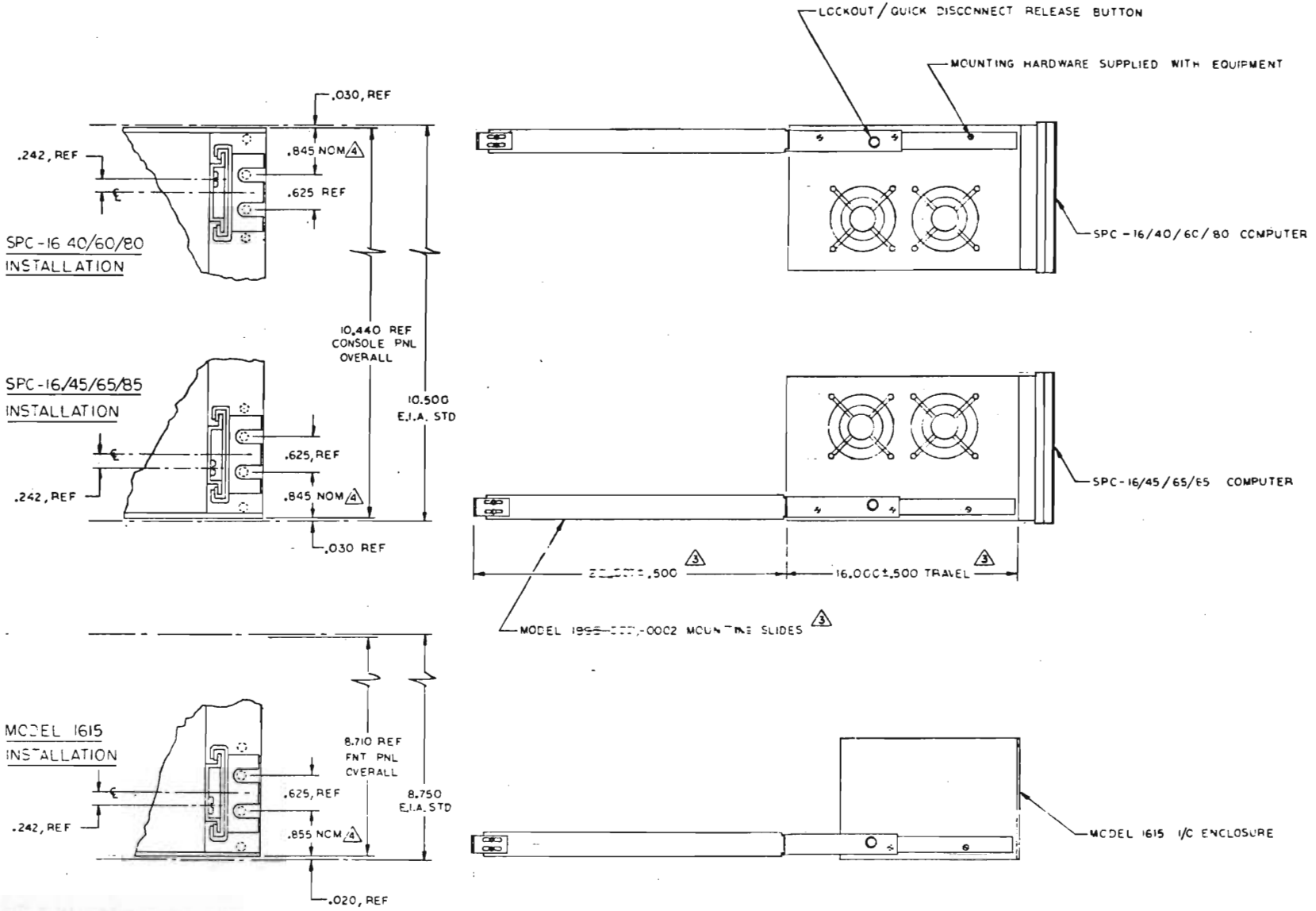


Figure 8-7. Installation Drawing, Optional Mounting Slides Model 1995.

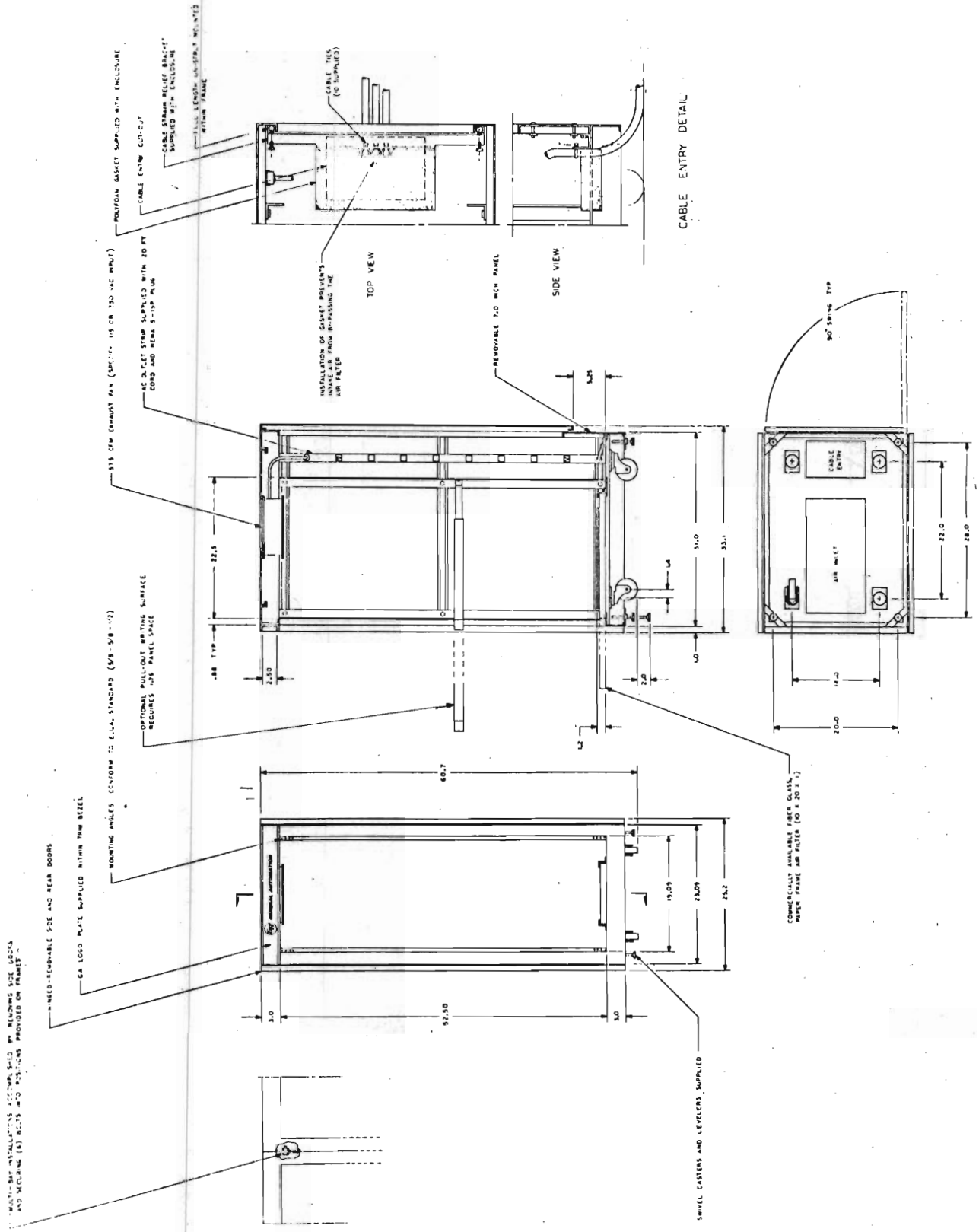


Figure 8-8. Installation Drawing, System Enclosure Model 1910

8.1.2 Racking Arrangements

This section illustrates recommended arrangements for various SPC-16 system configurations, using the General Automation 1901 System Enclosure. The following configuration examples are shown:

- Figure 8-9 shows an SPC-16/40/60/80 mounted with a 1615 I/O Expansion Chassis, a Paper Tape Reader and a Paper Tape Punch.
- Figure 8-10 shows an SPC-16/40/60/80 mounted with a 3346 or 3347 Disk unit, a Paper Tape Reader and a Paper Tape Punch.
- Figure 8-11 shows an SPC-16/45/65/85 mounted with a 1615 I/O Expansion Chassis.
- Figure 8-12 shows an SPC-16/45/65/85 mounted with a 1615 I/O Expansion Chassis and optional Paper Tape Reader and Paper Tape Punch.
- Figure 8-13 shows an SPC-16/45/65/85 mounted with a 1615 I/O Expansion Chassis, a 3346 or 3347 Disk unit, and optional Paper Tape Reader and Paper Tape Punch.
- Figure 8-14 shows an SPC-16/45/65/85 mounted with two 1615 I/O Expansion Chassis and optional Paper Tape Reader.

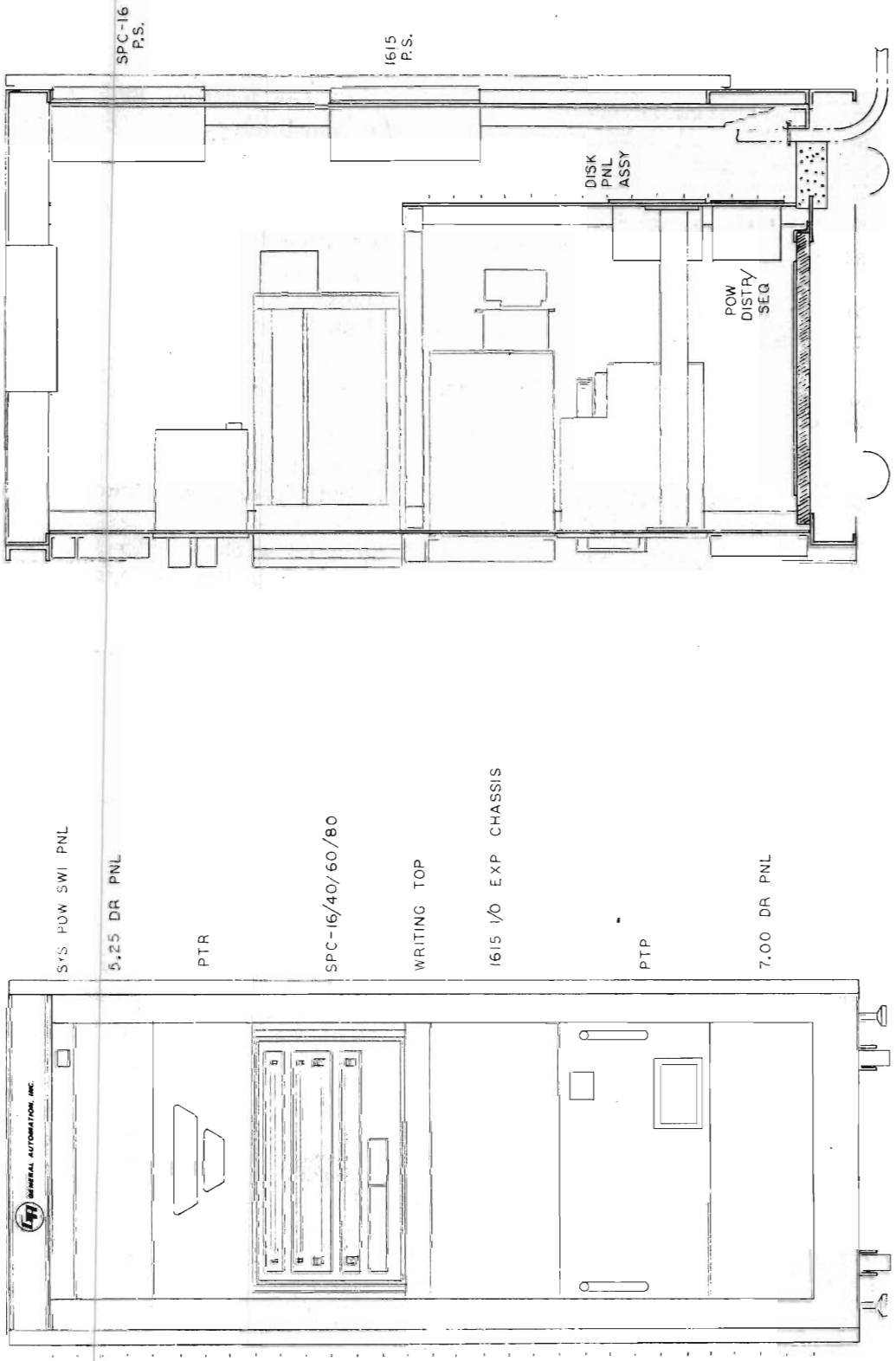


Figure 8-9. 1901 System Enclosure with SPC-16/40/60/80, 1615 I/O Expansion Chassis, Paper Tape Reader and Paper Tape Punch

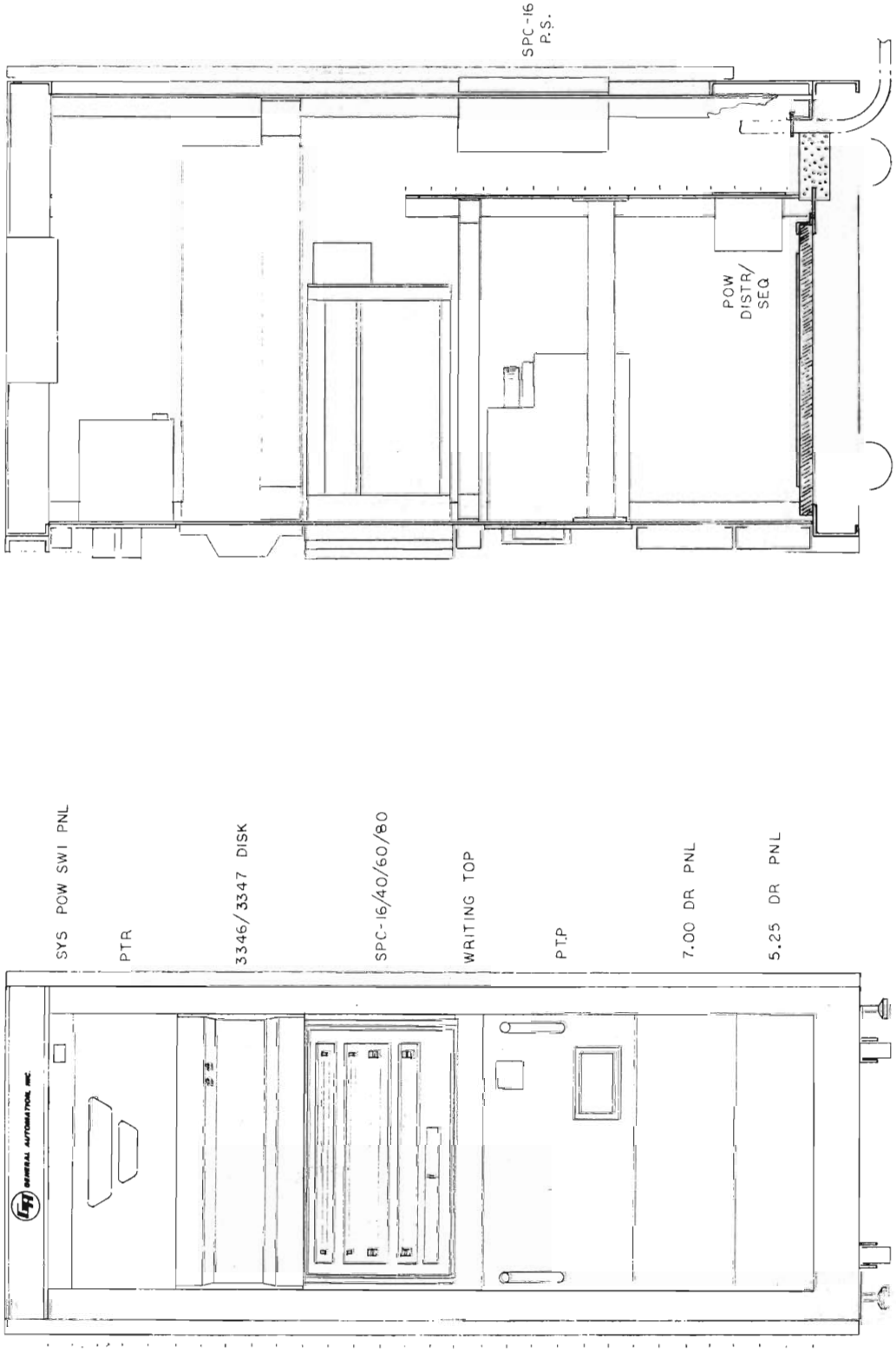


Figure 8-10. 1901 System Enclosure with SPC-16/40/60/80, 3346/3346 Disk, Paper Tape Reader and Paper Tape Punch

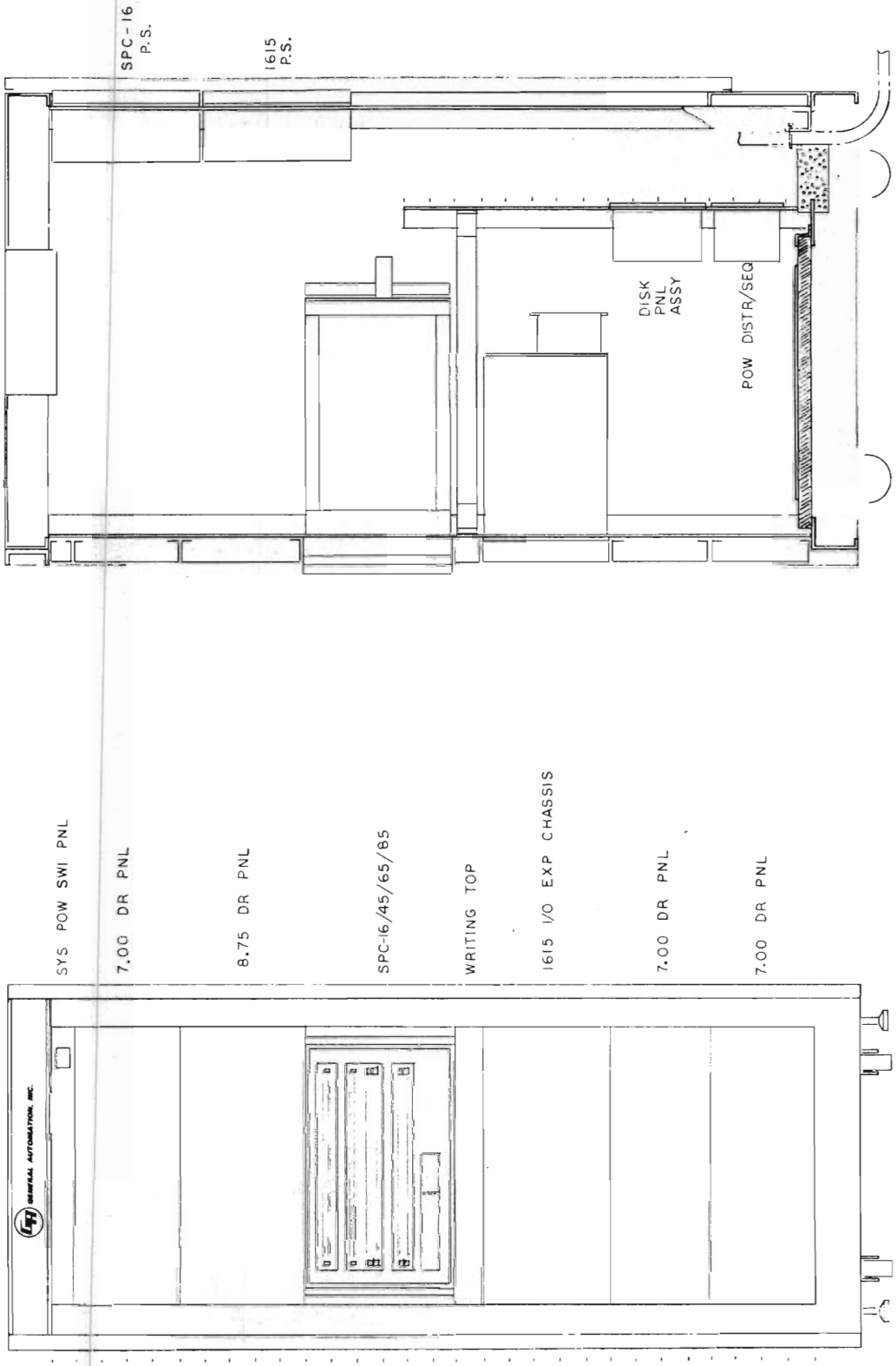


Figure 8-11. 1901 System Enclosure with SPC-16/45/65/85 and 1615 I/O Expansion Chassis

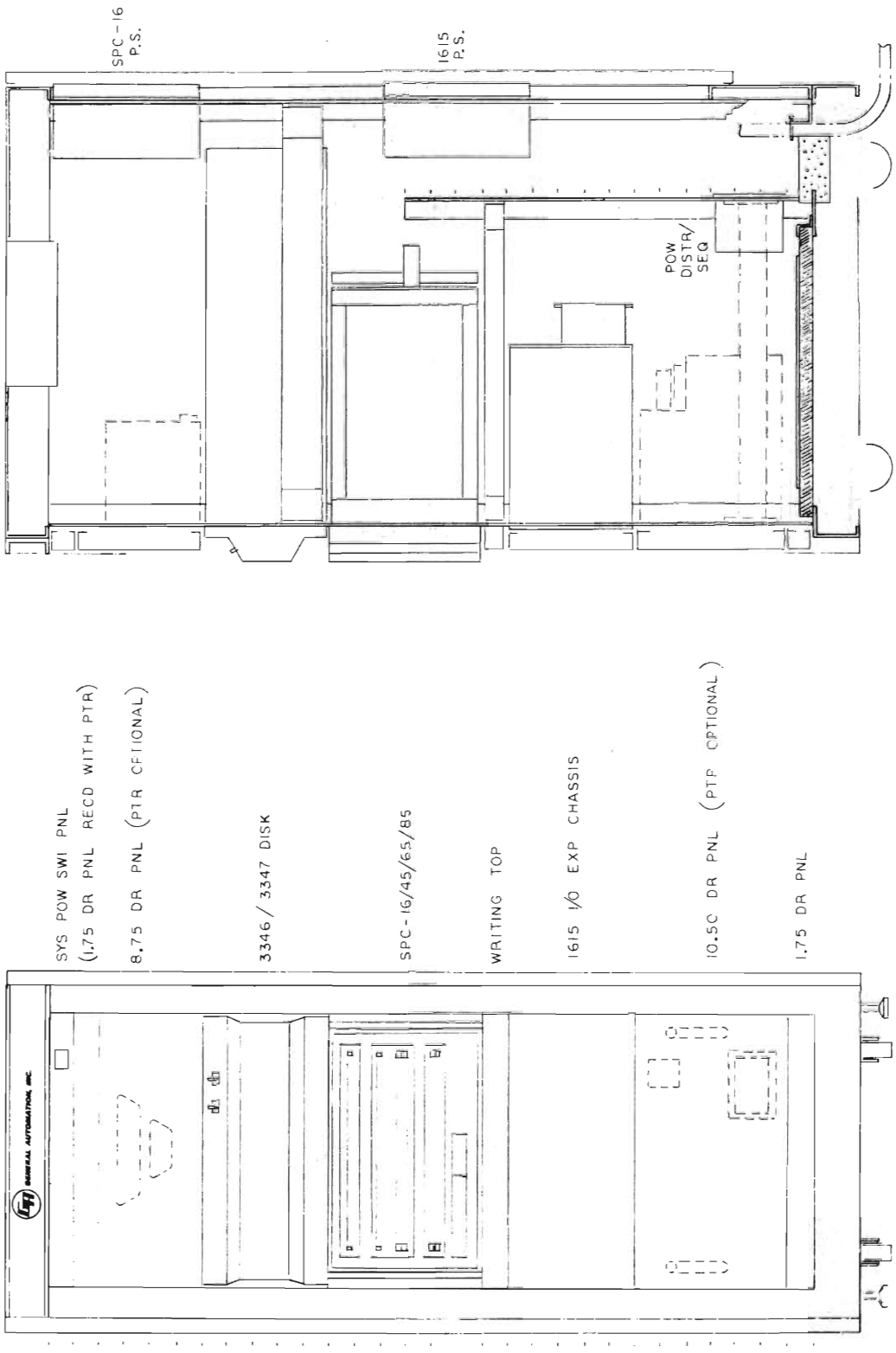


Figure 8-12. 1901 System Enclosure with SPC-16/45/65/85, 1615 I/O Expansion Chassis, Paper Tape Reader (Optional) and Paper Tape Punch (Optional)

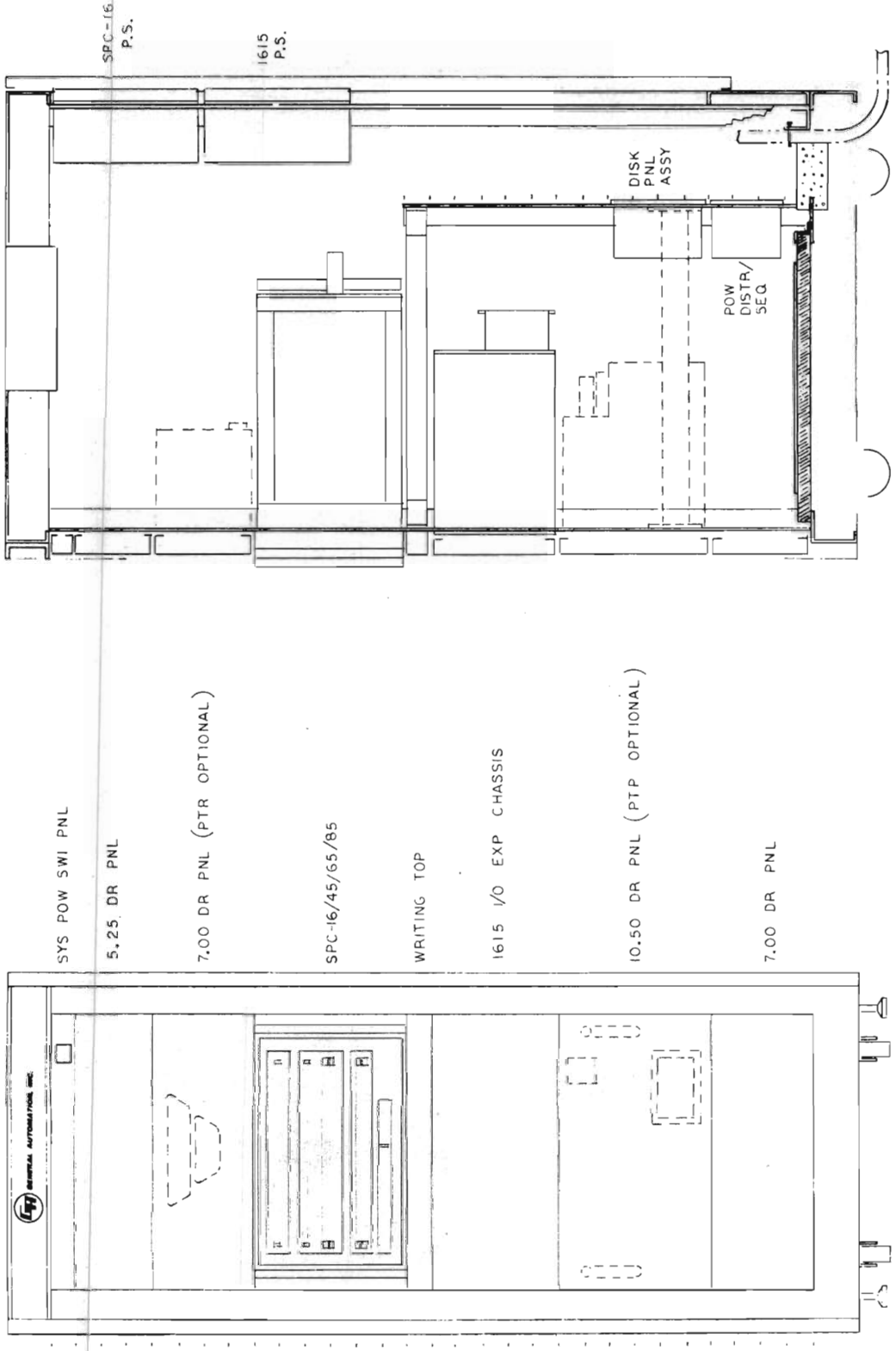


Figure 8-13. 1901 System Enclosure with SPC-16/45/65/85, 1615 I/O Expansion Chassis, 3346/3347 Disk, Paper Tape Reader and Paper Tape Punch

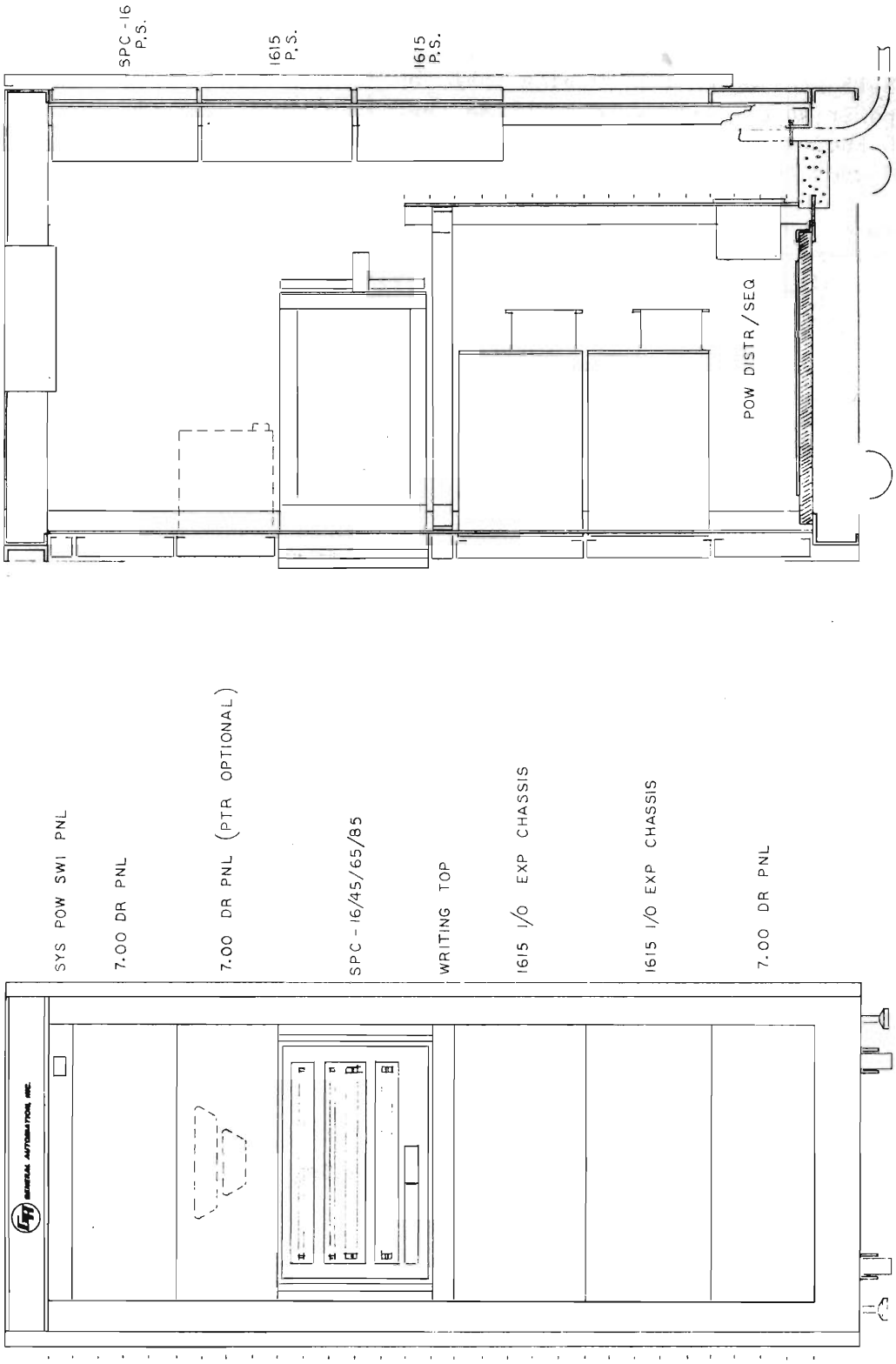
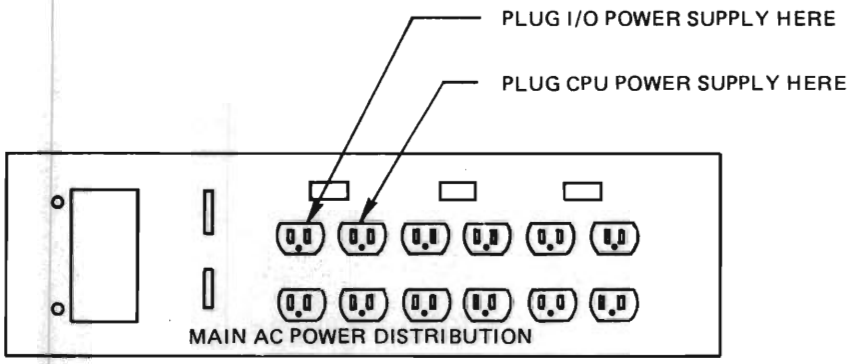


Figure 8-14. 1901 System Enclosure with SPC-16/45/65/85, Two 1615 I/O Expansion Chasses and Paper Tape Reader (Optional)

8.1.3 System Cabling

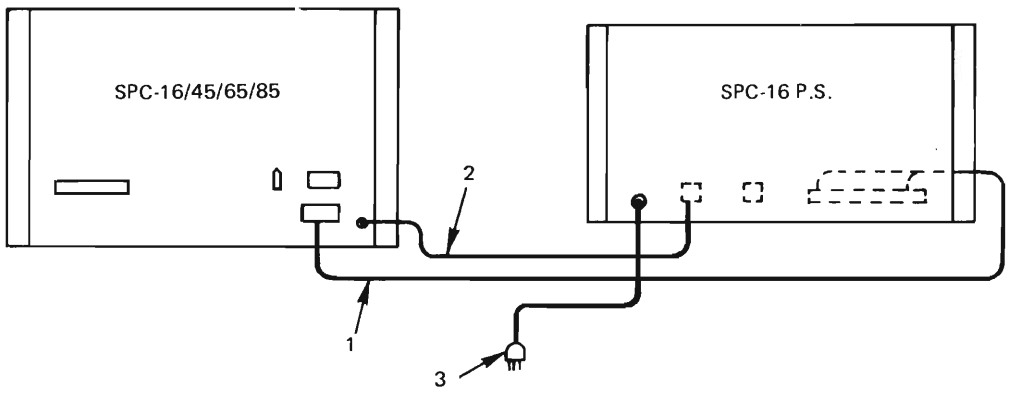
Figure 8-15 is a schematic drawing that shows how an SPC-16 processor is connected to its power supply. Figure 8-16 shows the addition connections required when the 1615 I/O Enclosure is included in the system. Figure 8-17 illustrates the cabling required when the 128K memory expansion option is included in the system.

The three-prong plugs in each of these drawings are plugged into the Main AC Power Distribution panel (as shown below), if a GA Model 1901 System Enclosure is used or plugged into one of the AC outlets on the AC outlet strip if a Model 1910 System Enclosure is used.



	Part Number	Std. Length	Max. Length	Function
1	Attached to P.S.	5 ft	15 ft	DC power
2	Attached to computer	6 ft	-	AC fan power
3	Attached to P.S.	6 ft	-	AC power

BASIC CONFIGURATION



BASIC CONFIGURATION

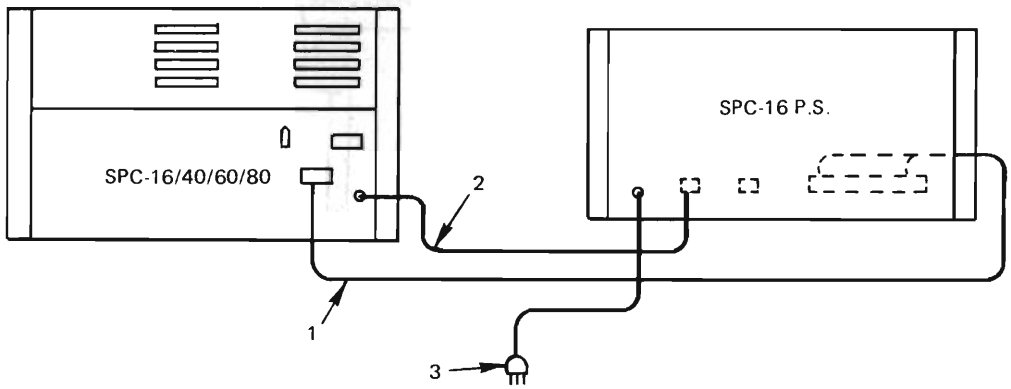


Figure 8-15. Cabling of SPC-16 Processor and Power Supply

	Part Number	Std. Length	Max. Length	Function
1	Attached to P.S.	5 ft	15 ft	DC power
2	Attached to computer	6 ft	-	AC fan power
3	Attached to P.S.	6 ft	-	AC power
4	21D00265A31	3 ft	Δ	I/O signal
5	Attached to P.S.	5 ft	15 ft	DC power
6	Attached to P.S.	6 ft	-	AC power
7	21C00529A01	3 ft	-	PFD control

Δ See Table 8-1

BASIC CONFIGURATION

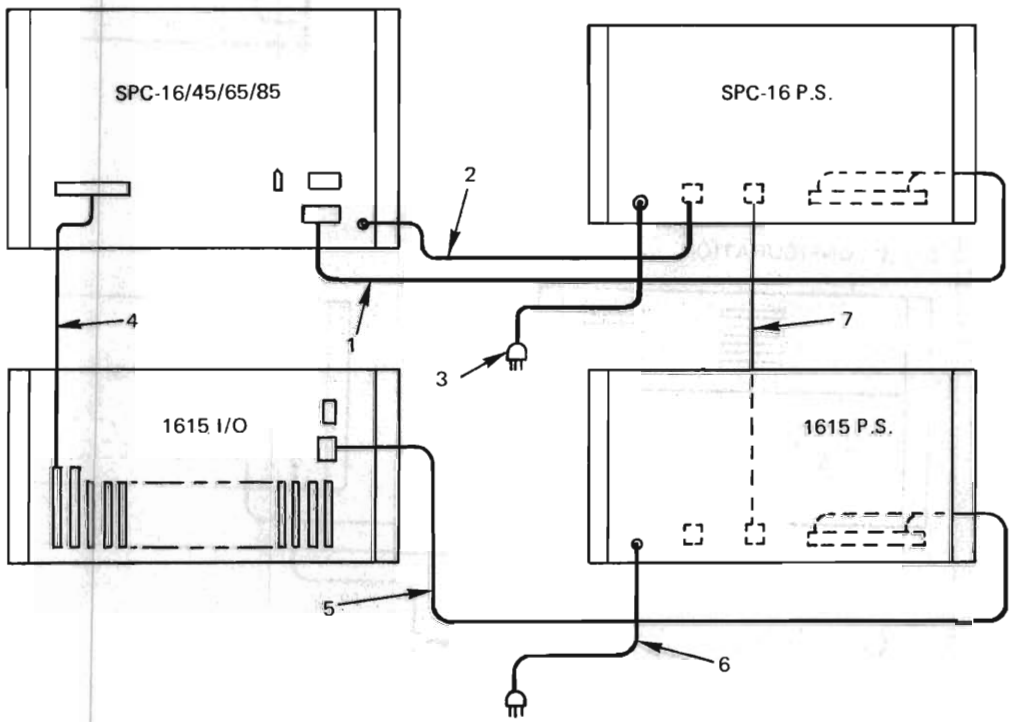


Figure 8-16. Cabling of SPC-16 System with Extended I/O Included

EXTENDED I/O CONFIGURATION

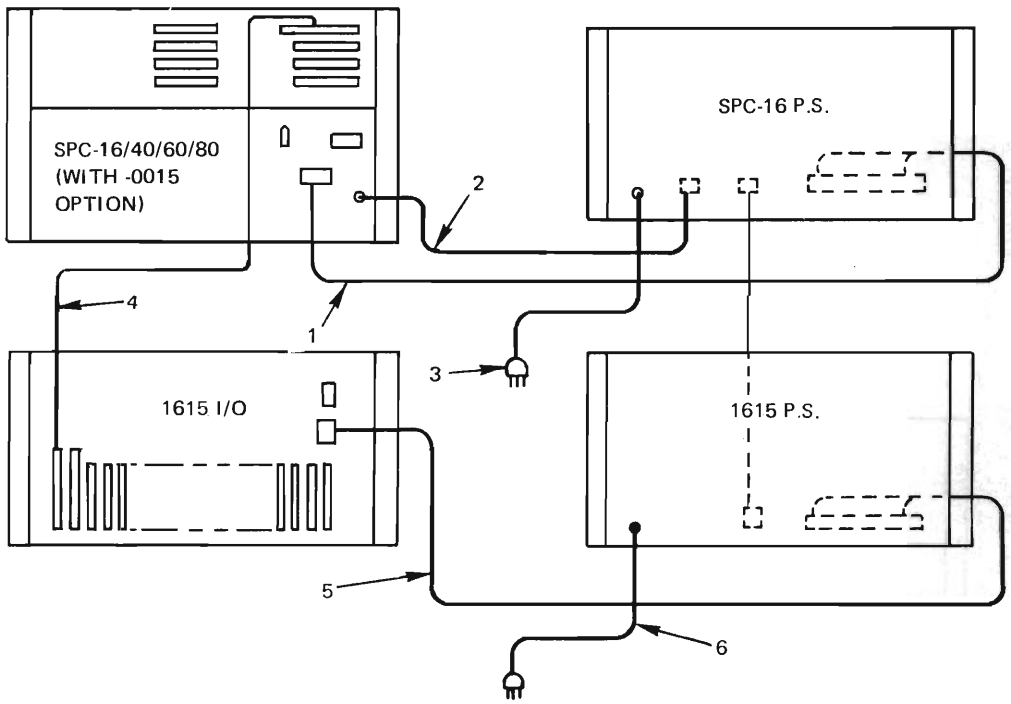


Figure 8-17. Cabling of SPC-16-40/60/80 System with Extended I/O Included /Continued)

Table 8-1. Maximum Combined I/O Cable Length Between I/O Enclosure and SPC-16

Model	Maximum I/O Cable Length (Total Length to Last I/O Enclosure)
SPC-16/40	80 feet
SPC-16/45	90 feet
SPC-16/60	20 feet
SPC-16/65	30 feet
SPC-16/80	6 feet
SPC-16/85	18 feet

Part Number	Physical Configuration	Std. Length	Max. Length	Function
1	Attached to P.S.	5 ft.	15 ft.	DC Power
2	Attached to CPU	6 ft.	-	AC fan power
3	Attached to P.S.	6 ft.	-	AC power
4	Separate cable 21C00326A	6 ft.	-	DC power
5	Attached to CPU	6 ft.	-	AC fan power
6	Attached to P.S.	6 ft.	-	AC power
7	21C00529A	3 ft.	-	PFD Control

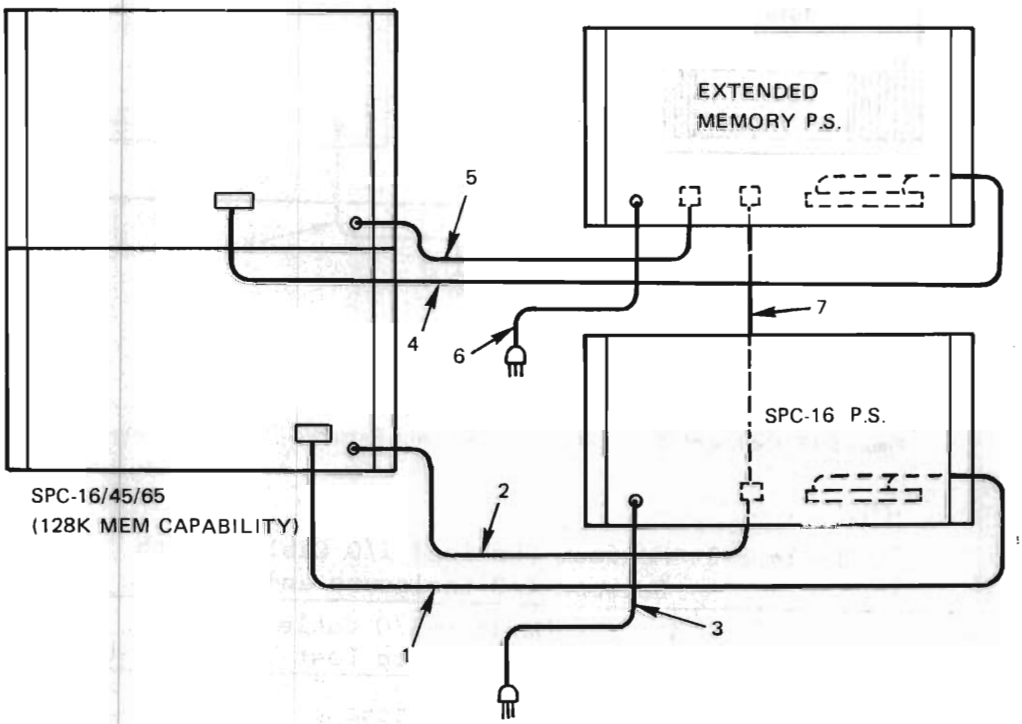


Figure 8-17. Cabling for SPC-16/45/65 with 128K Memory Expansion

8.2 POWER REQUIREMENTS

Sections 8.2.1 and 8.2.2 describe the various AC power sources that should be used to meet the requirements of a particular SPC-16 system. Section 8.2.1 describes the AC power sources for domestic use; Section 8.2.2 describes the AC power sources for European use.

8.2.1 AC Power Sources for Domestic Use

Three different AC power sources are considered for domestic applications:

1. 110 VAC, single-phase, 60 A source
2. 220 VAC, single-phase, 50 A source
3. 208 VAC, three-phase, 50 A source

The 110 VAC, single-phase 60 AMP source provides 6.6 KW of power and should be used only with small systems that do not include either a 3341 or 3343 disk drive. Figure 8-18 illustrates the power line attachment to the Main AC Power Distribution Panel.

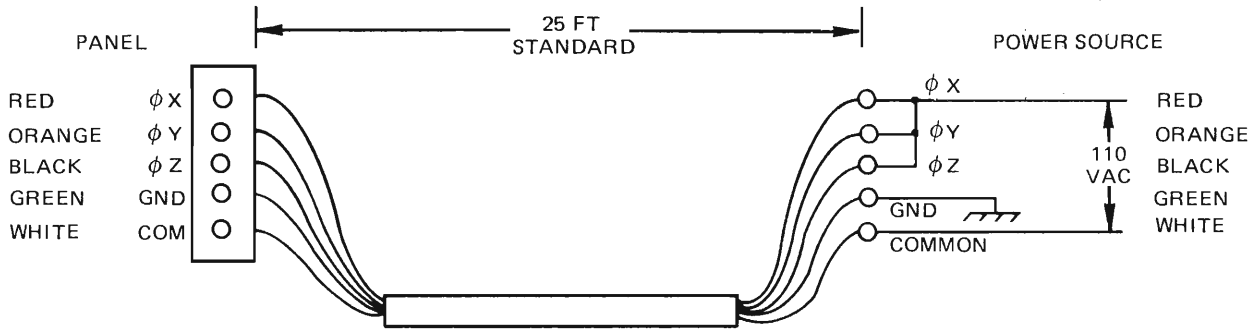


Figure 8-18. Power Line Attachment, 110 VAC Single-Phase 60 A

The 220 VAC, single-phase, 50 AMP source provides about 8.8 KW of power and should be used with systems that include only one 3341 or 3343 disk drive. Figure 8-19 illustrates the power line attachment to the Main AC Power Distribution Panel.

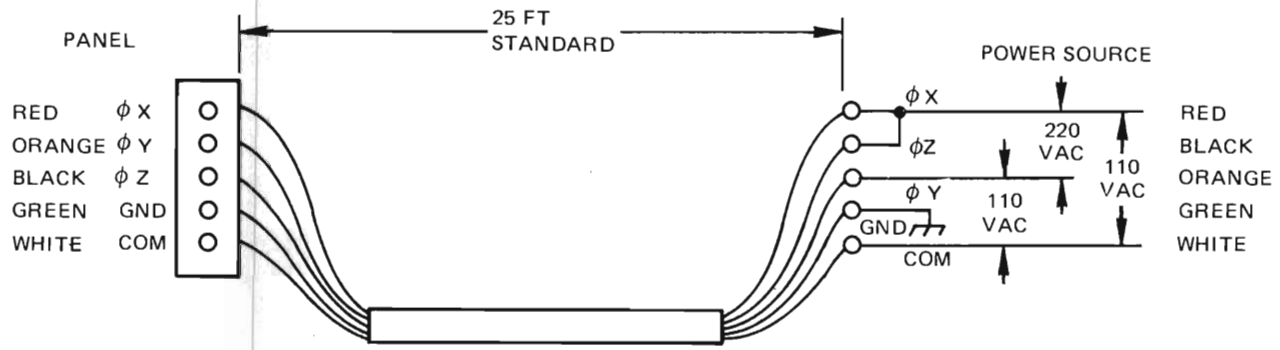


Figure 8-19. Power Line Attachment, 220 VAC Single-Phase 50A

The 208 VAC, three-phase, 50 AMP source provides about 16.5 KW of power and is required when the system includes two or more 3341 or 3343 disk drives. Figure 8-20 illustrates the power line attachment to the Main AC Power Distribution Panel.

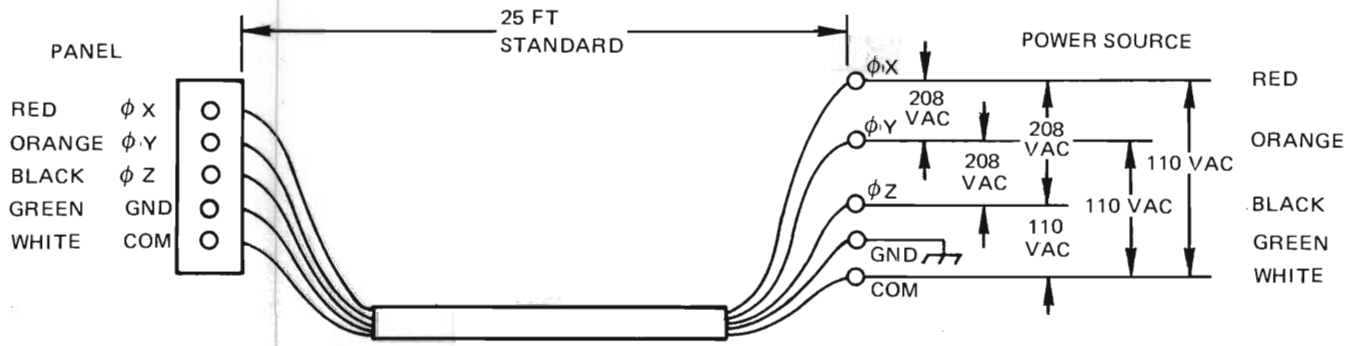


Figure 8-20. Power Line Attachment, 208 VAC Three-Phase 50A

8.2.2 AC Power Sources for European Use

Three different AC power sources are considered for European applications:

- 220 VAC, single-phase, 30 A source
- 440 VAC, single-phase, 30 A source
- 382 VAC, three-phase, 40 A source

The 220 VAC, single-phase, 30 AMP source provides 6.6 KW of power and should be used only with small systems that do not include either a 3341 or 3343 disk drive. Figure 8-21 illustrates the power line attachment to the Main AC Power Distribution Panel.

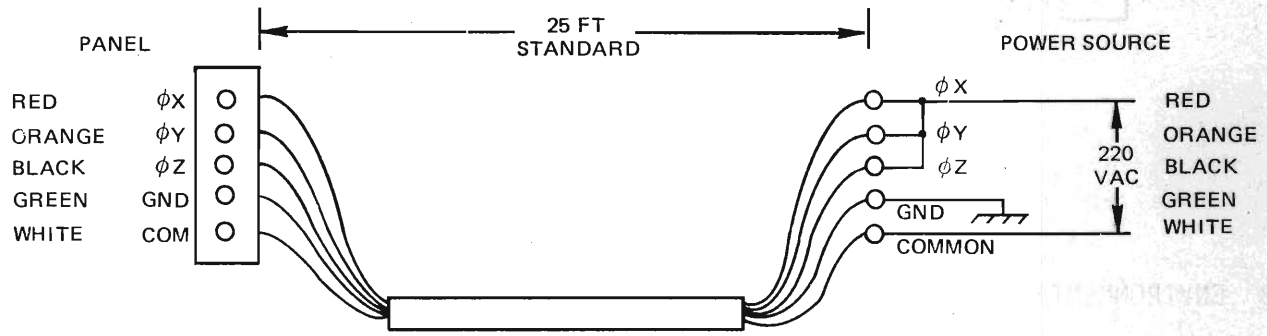


Figure 8-21. Power Line Attachment, 220 VAC Single-Phase 30A

The 440 VAC, single-phase, 30 AMP source provides about 8.8 KW of power and should be used with systems that include only one 3341 or 3343 disk drive. Figure 8-22 illustrates the power line attachment to the Main AC Power Distribution Panel.

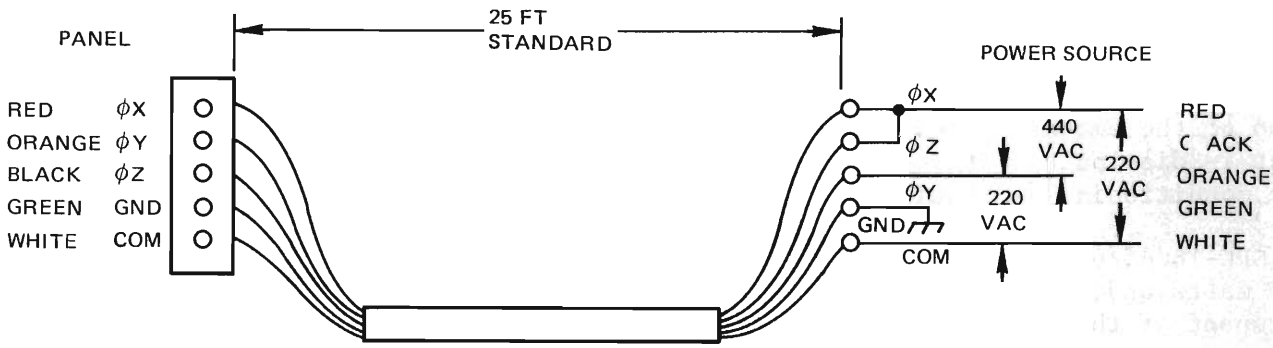


Figure 8-22. Power Line Attachment, 440 VAC Single-Phase 30A

The 382 VAC, three-phase, 40 AMP source provides about 16.5 KW of power and is required when the system includes two or more 3341 or 3343 disk drives. Figure 8-23 illustrates the power line attachment to the Main AC Power Distribution Panel.

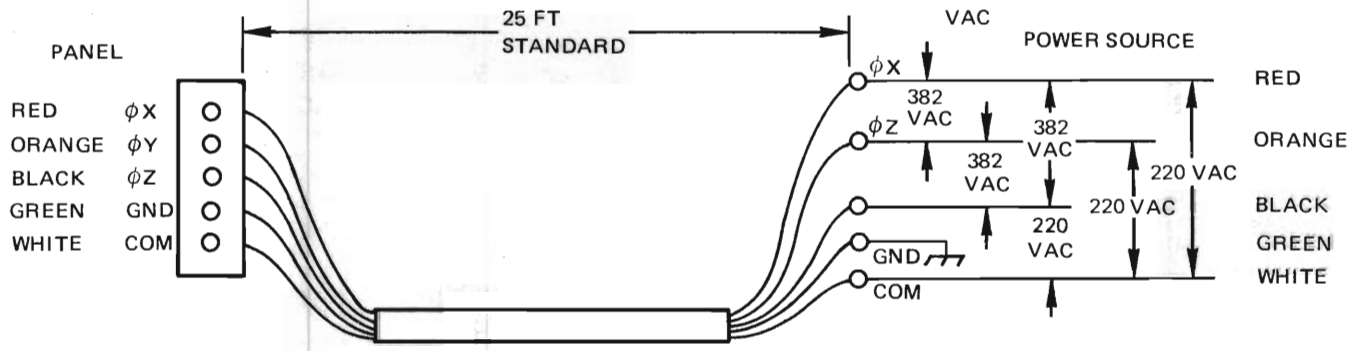


Figure 8-23. Power Line Attachment, 382 VAC Three-Phase 40A

8.3 ENVIRONMENTAL SPECIFICATIONS

This section presents the environmental conditions that are required for proper operation of the SPC-16 Processor, the extended I/O system and their associated power supply units.

8.3.1 Operating and Storage Temperatures

All models of the SPC-16 are designed to operate properly within the temperature range -0°C to 50°C (32°F to 122°F) at 10-90% relative humidity without condensation.

All models of the SPC-16 may be stored in environments within the temperature range -20°C to +70°C (-4°F to 140°F) at 0-95% relative humidity without condensation.

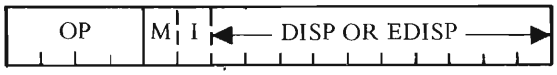
8.3.2 Cooling Considerations

The amount of cooling needed in the SPC-16 system's operating environment is a function of the amount of heat generated by the system. Specifically, each watt of power that is dissipated into the air stream generates 3.4 BTU's per hour and one ton of air conditioning is required to cool 12,000 BTU's per hour.

An SPC-16/45/65/85 operating with 64K of core memory using 8K memories requires 910 watts and, therefore, generates about 3094 BTU's per hour. In addition, each occupant of the room is estimated to generate 500 BTU's per hour.

APPENDIX A
SPC-16 INSTRUCTION SUMMARY HEXADECIMAL CODING

MEMORY REFERENCE

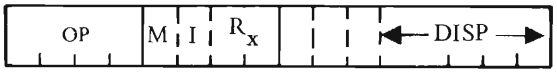


LDA	4
STA	5
JSR	6
JMP	7

DISP for M=1 (Base Relative)

EDISP for M=0 (Program Relative)

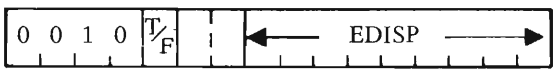
MEMORY REFERENCE WITH INDEX



LDR	C	REG
STR	D	REG
CMR	E	REG
LDBY	8	REG
STBY	9	REG
SBIT	B	Bit ID
RBIT	3	Bit ID
TBIT	A	Bit ID
LARS	F	1 0 0
SARS	F	1 1 0
INCM	F	0 0 0
DECM	F	0 1 0

M=1 (Base)
M=0 (Direct)

SKIP

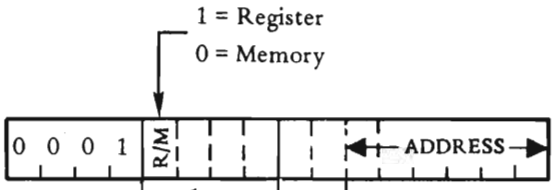


0 0	OVFL
0 1	LINK
1 0	ZERO
1 1	PLUS

	SKIP FWD		SKIP REV	
SKOT	28	XX	29	XX
SKOF	20	XX	21	XX
SKS	2A	XX	2B	XX
SKR	22	XX	23	XX
SKZ	2C	XX	2D	XX
SKN	24	XX	25	XX
SKP	2E	XX	2F	XX
SKM	26	XX	27	XX

NOTE: SKIP REV XX is in 2's complement form.

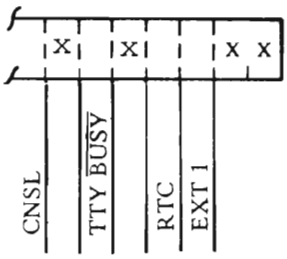
XIO



CTRL	1	REG	0 0	CONTROL
DTOM/DTOR		$\overline{FUN}$	0 1	OUTPUT
DTIM/DTIR			1 0	INPUT
TEST			1 1	TEST

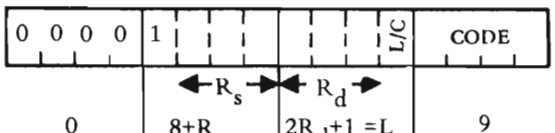
INTERNAL I/O (REF. SECTION 4.2)

ENABLE INTERRUPTS	1	0/1	R	7	E
TTY OUTPUT	1	0/1	R	7	F
TTY INPUT	1	0/1	R	B	F
TTY TEST	1		0	F	F
TTY TRANSMIT	1		0	3	F
TTY RCV ONLY	1		2	3	F
TTY RCV/ECHO	1		4	3	F
TTY BREAK	1		6	3	F
RCSR/RCSM	1	0/1	R	B	E



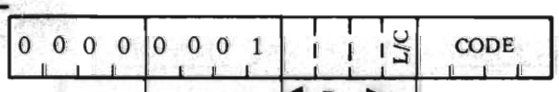
INTERRUPT MASK BITS

REGISTER OPERATE



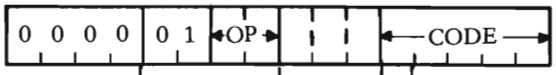
ADD, ADDC	0	$8+R_s$	$2R_d+1 = L$	9	ADD
SUB, SUBC			$2R_d = C$	6	SUB
AND, ANDC				7	AND
XOR, XORC				8	XOR
OR, ORC				D	OR
RTR				5	XFER

REGISTER OPERATE LITERAL



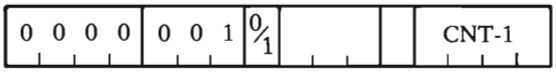
ADDV, ADDVC	0	1	$2R_d+1 = L$	9	ADD
SUBV, SUBVC			$2R_d = C$	6	SUB
ANDV, ANDVC				7	AND
XORV, XORVC				8	XOR
ORV, ORVC				D	OR
LDV				5	LDV

REGISTER CHANGE



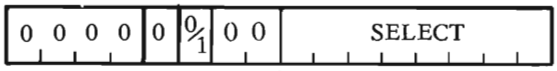
XEC	0	5	$2R_d+1$	0
RISE	0	5	$2R_d$	1
RTRN	0	5	$2R_d$	3
DSPL	0	5	$2R_d$	4
TRS	0	5	$2R_d$	8
ZERO	0	6	$2R_d$	0
ZRBY	0	6	$2R_d$	1
ZLBY	0	6	$2R_d$	2
EXBY	0	6	$2R_d$	4
TSR	0	6	$2R_d$	8
RCSW	0	6	$2R_d+1$	0
CMPL	0	7	$2R_d$	0
RLK	0	7	$2R_d$	1
DECR	0	7	$2R_d$	2
ADDS	0	7	$2R_d$	B
INCR	0	7	$2R_d$	E

SHIFTS



SRLC	0	2	$2R_d$	CNT-1
SRA	0	2	$2R_d+1$	CNT-1
SRC	0	3	$2R_d$	CNT-1
SRCL	0	3	$2R_d+1$	CNT-1

CONTROL



INE	0	4	0	3
INH	0	4	0	2
FMS	0	4	0	C
BMS	0	4	0	8
LKS	0	4	3	0
LKR	0	4	2	0
PMA	0	4	4	0
SYNC	0	4	8	0
WAIT	0	0	0	X
MPY	0	0	8	n
DIV	0	0	A	n

APPENDIX B
STANDARD CHARACTER CODES

Graphic or Control	System ASCII See Note 1 (Hexadecimal)	Card Code See Note 2		Printer See Note 3
		IBM 029	IBM 026	
NULL	80			
SOM	81			
EOA	82			
EOM	83			
EOT	84			
WRU	85			
RU	86			
BELL	87			
FE	88			
H. Tab	89			
Line Feed	8A			
V. Tab	8B			
Form	8C	No Equivalent Codes		No Equivalent Codes
Return†	8D			
SO	8E			
SI	8F			
DCO	90			
X-On	91			
Tape Aux. On	92			
X-Off	93			
Tape Aux. Off	94			
Error	95			
Sync	96			
LEM	97			
S0	98			
S1	99			
S2	9A			
S3	9B			
S4	9C			
S5	9D			
S6	9E			
S7	9F			

† When output is to a Teletype the Return code should precede the Line Feed to prevent overprinting.

Graphic or Control	System ASCII See Note 1 (Hexadecimal)	Card Code See Note 2		Printer See Note 3
		IBM 029	IBM 026	
blank	A0	No Punch		
! exclamation	A1	11-2-8 ‡		
" quote	A2	7-8	0-5-8	
#	A3	3-8	0-7-8	
$	A4	11-3-8		
%	A5	0-4-8	11-7-8	
&	A6	12	12-7-8	
' apostrophe	A7	5-8	4-8	
(open paren.	A8	12-5-8	0-4-8	
) close paren.	A9	11-5-8	12-4-8	
*	AA	11-4-8		
+	AB	12-6-8	12	
, comma	AC	0-3-8		
- minus	AD	11		
. period	AE	12-3-8		
/ slash	AF	0-1		
0 zero	B0	0		All characters are valid
1	B1	1		
2	B2	2		
3	B3	3		
4	B4	4		
5	B5	5		
6	B6	6		
7	B7	7		
8	B8	8		
9	B9	9		
: colon	BA	2-8	5-8	
; semicolon	BB	11-6-8		
< less	BC	12-4-8	12-6-8	
= equal	BD	6-8	3-8	
> greater	BE	0-6-8	6-8	
?	BF	0-7-8	12-2-8	
@	C0	4-8	0-2-8	

Graphic or Control	System ASCII See Note 1 (Hexadecimal)	Card Code See Note 2 IBM 029	Card Code See Note 2 IBM 026	Printer See Note 3
A	C1	12-1		All characters valid
B	C2	12-2		
C	C3	12-3		
D	C4	12-4		
E	C5	12-5		
F	C6	12-6		
G	C7	12-7		
H	C8	12-8		
I	C9	12-9		
J	CA	11-1		
K	CB	11-2		
L	CC	11-3		
M	CD	11-4		
N	CE	11-5		
O	CF	11-6		
P	D0	11-7		
Q	D1	11-8		
R	D2	11-9		
S	D3	0-2		
T	D4	0-3		
U	D5	0-4		
V	D6	0-5		
W	D7	0-6		
X	D8	0-7		
Y	D9	0-8		
Z	DA	0-9		
[open bracket	DB	12-2-8		
\ back slash	DC	0-2-8		
] close bracket	DD	11-2-8 ‡		
↑ Λ ††	DE	11-7-8		
← ___ ††	DF	0-5-8		
ACK	FC	12-11-0-4		No equivalent code
Alt. Mode	FD	12-11-0-5		
Rubout	FF	12-11-0-7		

†† ASCII codes DE and DF produce up arrow and left arrow for Teletype; circumplex and underscore for the Line Printer. See note 3.

‡ Depends upon type of font and usage.

NOTES FOR STANDARD CHARACTER CODES

Note 1. ASCII is a seven bit data code represented by the seven least significant bits of an eight bit data byte. The eighth bit of an ASCII data byte may be used for a parity bit. The parity scheme used may be odd or even or not used at all, depending on the particular device and model. GA I/O systems software ignore this parity bit when received. When the data byte is stored in memory the eighth bit will be set to a one. The data is then in System ASCII form. On output the I/O software systems always set the eighth bit to a one.

Note 2. All GA I/O systems are set up to recognize IBM 029 keypunch codes. IBM 026 codes are given only where they are different from the IBM 029 codes.

Note 3. The line printers use a 64-character subset of the ASCII code. The seventh and eighth bit are completely ignored and may be in any state. The System ASCII codes given in column one are fully usable as printer codes for all legal characters. Codes DE_{16} and DF_{16} produce unique characters on the printer which differ from the Teletype. See graphic column.

APPENDIX C
SUMMARY OF SYSTEM SOFTWARE

General Automation provides SPC-16 users with a powerful and comprehensive set of programs and programming systems. This software is specifically designed to relieve the user of much of the detailed programming effort and provide powerful tools to assist in the preparation of the balance.

Programming aids and systems are geared to the level of hardware system capability. Minimal systems are supported by special versions of GA software designed to cope with the unique problems these systems create for the user. More sophisticated hardware configurations are supported by operating systems and system program modules designed to utilize the additional flexibility and power the hardware provides.

A number of fundamental design concepts are consistently used throughout the SPC-16 standard software. An appreciation of these concepts will enable the SPC-16 user to better understand the use his software. The more important concepts are described briefly in the following paragraphs. More extensive treatments of these subjects will be found in the applicable publications.

The idea of device independence is not new to software systems. The FORTRAN user has written essentially device independent code for years. The shortcoming of previous systems was that at execution time the choices offered for device selections were limited. Further, the assembly language programmer rarely benefited from the FORTRAN support software. The software supplied with the SPC-16 is largely device independent. That is, not only can the FORTRAN programmer write device independent code but the compiler itself is device independent. The user can connect any combination of compatible input/output devices to the compiler. It is possible, therefore, to input source data to the compiler from a paper tape reader, card reader, magnetic tape, disk file or any other system supported device capable of symbolic input.

The freedom of device selection is provided by the SPC-16 Input/Output System (IOS), which is at the heart of the FSOS, DBOS, RTX, and RTOS higher level systems (described below). Correlation of logical units referenced by a program (referred to as a Processor by GA) and the desired physical unit is accomplished by cross reference tables contained in IOS. The user adjusts the contents of these tables using logical unit assignment commands via the operating system executive.

Such assignments frequently are made from the Teletype Keyboard. However, since the Executives are also device independent, logical unit selection can be made via any input device capable of symbolic input. A logical unit referred to symbolically as 'CC' is standardly assigned (by system generation) to the teletype. The simple procedure of typing in the command $CC=CR (card reader symbolic name) will cause the system to switch to the card reader for command input, providing the user with an efficient batch processing capability. Commands include many directives to the Executive, including logical/physical unit assignments.

System generation procedures enable the user to modify (delete and add to) IOS to fit his particular hardware system. When a mass storage device is available, the user may establish file areas to be referenced as logically separate devices.

Virtually, all General Automation software is relocatable and interruptible. The subroutine library is comprised of relocatable, re-entrant and recursive subroutines. The flexibility provided by such programming techniques is widely known and understood. It is likewise an accepted premise that these features are expensive both in terms of speed and core requirements; however, the SPC-16 hardware makes such coding techniques both practical and efficient. Base relative addressing is used extensively to access data base areas. This scheme permits the subroutine (or program) to be written as pure procedure. No variable data is stored within the subroutine itself. The timing overhead associated with these subroutines amounts to an average of 18 cycle times while the core storage is totally volatile. This means that the core is used temporarily while a subroutine is in execution and then relinquished. The concept used to implement this variable data storage is referred to as Dynamic Storage Allocation. System macros and subroutines are provided to assist the user in preparing his own re-entrant subroutines.

C.1 OPERATING SYSTEMS

General Automation presently supports four basic levels of hardware/software environments: a Basic Systems level, a Free Standing Device Independent level, a Disk Based Device Independent level and a Real Time Multiprogramming Device Independent level. In addition, a number of special purpose systems are available that include the following:

- Remote Job Entry (RJE) interface system.
- File Management System (FMS) for building and maintaining large data files.
- Commercial FORTRAN system for standard business applications.

The following subsections summarize the basic GA operating systems.

TELETYPE ONLY SYSTEM (TTY ONLY)

TTY ONLY programs are designed to support a system with the following hardware configuration:

- SPC-16 computer with 4K words of memory
- Teletype with associated paper tape reader and punch

The TTY ONLY software package is a set of stand alone program modules to aid the user in all aspects of application program preparation. The package includes a variety of utility programs for loading and punching object programs, preparing and editing program source tapes and debugging object programs. The TTY ONLY assembler and Linking Loader programs provide the user of a minimum system with maximum flexibility in configuring his application programs.

For complete details consult the SPC-16 TTY-ONLY System manual, GA document number 88A00231A.

FREE STANDING OPERATING SYSTEM (FSOS)

The Free Standing Operating System (FSOS) is an interrelated group of programs intended to support a SPC-16 computer system whose peripheral configuration does not include a disk or drum storage unit. FSOS operates with the following minimum hardware configuration:

- SPC-16 computer with 8K words of memory.
- Teletype with associated paper tape reader and punch.
- High speed input device (usually a high speed paper tape reader or a card reader).
- High speed output device (usually a high speed paper tape punch or a card punch).

The hardware configuration may also include:

- Line Printer
- Magnetic tape units

The FSOS package includes a number of stand alone utility programs for basic operations such as loading and punching, editing and debugging programs.

The central element of FSOS is the Stand-Alone Monitor (SAM) that is tailored to the user's hardware configuration. SAM provides an operating environment for device independent programs (both system and user written). SAM includes a Keyboard Executive that allows the user to enter control commands to modify I/O device assignments, initiate program loading and linking and initiate program execution.

The following device independent programs are supplied with FSOS to operate in the SAM environment:

- CAP-16 Assembler
- FORTRAN-16 Compiler (requires 12K)
- Core Load Builder

- Source Text Editor
- BASIC Language Processor
- Debug Utility

For full details consult the Free Standing Operating System Manual, GA document number 88A00139A.

DISK/DRUM BASED OPERATING SYSTEM (DBOS)

The SPC-16 Disk/Drum Based Operating System (DBOS) is a software package for hardware systems which include a disk or drum. DBOS may be used with the following minimum hardware configuration:

- SPC-16 computer with 8K words of memory if FORTRAN operations is not required; with 12K core if FORTRAN is to be used, or with 16K core if Extended FORTRAN is to be used.
- Teletype with associated paper tape reader and punch.
- Model 3341, 3343, 3346, 3347 or 3349 Disk or 3342 Head Per Track Disk.
- High speed input device (high speed paper tape reader or card reader).

The following peripherals are supported and recommended for optimum system use:

- High speed paper tape punch
- Card punch
- Line printer
- Graphic printer
- Magnetic tape units
- Keyboard/CRT terminals
- Additional disk units (up to 3 additional).

The SPC-16 Disk/Drum Based Operating System (DBOS) is a software package which provides the user with a comprehensive and easy-to-use program generation environment. DBOS includes a Bulk Monitor, a System Executive, an Input/Output System and System Processors for program compilation, assembly, debugging, editing and construction. All DBOS operations are initiated by simple control commands input by the operator or in the form of a prebuilt batch processing sequence.

DBOS is controlled by a core resident Monitor. The Monitor contains control logic for bulk/core management, the input/output system and a device independent loader. The Monitor also resides on the bulk unit in its initial generated configuration. If, any time, the core resident Monitor should be destroyed, the user may utilize the bulk bootstrap loader to reload the Bulk Monitor.

When initially loaded from the bulk unit, or whenever control passes to the Monitor in the course of job processing, the Monitor calls the System Executive into core. The Executive is a system processor which is loaded by the Monitor to process control commands. The control commands define logical unit assignments, program assembly or compilation, program execution and bulk utility functions.

DBOS incorporates the GA Input/Output System (IOS). This I/O system provides the capability for generating totally device independent programs. Programs reference I/O operations by logical unit numbers. These logical unit numbers are equaled to physical devices by IOS. This system enables any program to use any peripheral on the system. The user simply directs the Executive to equate the desired peripheral to the referenced logical unit and selects standard logical/physical relationships during system generation.

DBOS supports multiple bulk files. These files may be used to store ASCII or binary data or programs. Many of the files can be directoried such that data blocks can be named. Such named files are specified by name when equating a logical unit the the named file block.

DBOS as supplied by GA includes the following device independent processors (programs);

- CAP-16 Assembler
- FORTRAN or Extended FORTRAN Compiler
- Core Load Builder
- Core Load Overlay Builder and associated Symbol Map Sort Program
- Debug Utility
- Source Text Editors
- BASIC Language Processor
- Concordance Generator
- Device Independent PGS Loader
- Sequence and Compare Utility

- Backup on Magnetic Tape Utility
- Source Record Insertion (PRE-PASS)
- Compress PGS Records

For complete operating procedures, consult the Disk/Drum Based Operating System manual GA document number 88A00115A.

REAL TIME EXECUTIVE (RTX-16)

The Real Time Executive System (RTX-16) is a set of modular programs which provides the operating environment for real time system applications. An RTX system can be used with the following minimum hardware configuration:

- SPC-16 computer with 8K words of memory to build an RTX system; a generated system can be run in 4K of memory.
- Teletype with associated paper tape reader and punch.

The system will normally include a number of process control devices specific to each application.

RTX system elements include a System Initialization Program, Executive Programs and and Executive Options Library. When linked to user-written applications programs, RTX forms a functioning real time system which provides for:

- Real-time scheduling of up to 155 functional programs on a full priority basis.
- An input/output system which includes drivers to support most standard peripherals and can incorporate special purpose, user-programmed drivers.
- Modification of dynamic scheduling via user program calls.
- Bulk management capabilities for systems which include a bulk storage unit (disk or drum).

For a complete description, the reader is referred to the Real Time Executive (RTX-16) System manual, GA document number 88A00114A.

REAL-TIME OPERATING SYSTEM (RTOS-16)

The SPC-16 Real Time Operating System (RTOS-16) combines the real time system capabilities of RTX-16 with expanded program generation facilities available under DBOS-16 to provide a multiprogramming foreground/background system for dynamic real time operation.

RTOS may be used with the following hardware configuration:

- SPC-16 computer with 16K words of memory (24K words or more is recommended).
- Teletype with associated paper tape reader and punch.
- High speed input device (card reader or high speed paper tape reader).
- Model 3341, 3343, 3346, 3347 or 3349 Disk or 3342 Head Per Track Disk.

The following peripheral units are supported and enhance the utility of RTOS-16.

- High speed paper tape punch
- Card punch
- Line printer
- Graphic printer
- Magnetic tape units
- Keyboard/CRT terminals
- Additional disk units (up to 3 additional)
- Process control devices

The extensive set of RTOS Executive control commands is a convenient and efficient means of generating application programs in the background processing environment and incorporating them into the real time foreground system. Foreground control provides program scheduling and automatic execution, memory allocation, input/output control and other facilities for supporting the RTOS multiprogramming environment. Background processing tasks such as program development, batch processing and disk file management can be interleaved with concurrent foreground operations. A monitoring processor is supplied to prevent background processing from disturbing foreground operations.

A comprehensive set of support software is provided, including a library of real time system subroutines to facilitate dynamic scheduling and timing. The following system processors are supplied with RTOS:

- CAP-16 Assembler
- FORTRAN or Extended FORTRAN Compiler
- Core Load Builder

- Core Load Overlay Builder and associated Symbol Map Sort Program
- Debug Utility
- Source Text Editors
- Execution Monitor
- BASIC Language Processor
- Concordance Generator
- Device Independent PGS Loader
- Sequence and Compare Utility
- Backup on Magnetic Tape Utility
- Source Record Insertion (PRE-PASS)
- Compress PGS Records

For a complete description, the reader is referred to the Real Time Operating System (RTOS-16) Manual, GA document number 88A00230A.

C.2 SYSTEM PROCESSORS

General Automation operating systems are supplied with various system processors to aid in all aspects of application program development: source tape editing, assembling or compiling, debugging and program linking. System processors are device independent; they operate under the I/O environment of the operating system. Commonly used system processors are described below.

ASSEMBLER (CAP-16)

The CAP-16 Assembler allows the programmer to code program using mnemonic operation codes and symbols to define machine instructions. A set of directives facilitates memory allocation, register utilization, definition of constant and literal data and conditional assembly.

Programs can be either relocatable or absolute. Actual instruction addressing (base relative, program relative or absolute) is generated automatically through usage of data and program selection directives. Directives allow program linking by means of the Core Load Builder program.

The 4K Assembler permits users with a basic system configuration to generate programs with a minimum of effort. Expanding the system to include 8K or more of memory and high speed Input/Output devices provides for operation of CAP-16 in its macro form, which substantially increases the assembler's facilities. While retaining all the advantages of the 4K Assembler, it also allows program-defined macros and use of system macros to facilitate coding of input/output calls, subroutine entry and exit sequences, and other standard code generation.

The macro assembler is also supplied with an Assembler Kit and Procedures Deck that provides the additional capability of tailoring the assembler language to a user's psecial requirements.

FORTRAN AND EXTENDED FORTRAN COMPILERS

The GA FORTRAN or Extended FORTRAN Compiler enables the programmer to solve problems using a higher-level language.

GA FORTRAN contains execptional capabilities for generated code optimization. For example, redundant arithmetic expressions are noted and illuminated across statement boundaries. Extensive use of the computation is prepared at compile time (no subscript subroutine) as is the reduction of all integer expressions. A special feature of GA FORTRAN is the generation of source code listings. This feature, combined with the in-line assembly feature, makes it possible for the user to further optimize the code generated by the compiler.

The FORTRAN Compiler produces object programs from source statements written in the FORTRAN lanaguage. The SPC-16 FORTRAN meets the American Standard BASIC FORTRAN X3, 10-1966, and offers a wide range of extensions including:

- In-Line assembly language
- Bit and Byte arrays
- Logical operations
- Relational expressions
- Two named COMMON areas
- Free Form Input/Output (no Format Specifications)
- Executable END statement
- H descriptor for read-in Format Strings
- A subscript may be any legal integer expression, including subscripted elements.

SPC-16 Extended FORTRAN exceeds ANSI standards for FORTRAN IV (X3.9 - 1966) and provides all of the above features plus additional capabilities including the following:

- BLOCKDATA, ASSIGN, EXTERNAL statements
- Labeled COMMON
- Double precision and complex arithmetic
- Mixed mode arithmetic
- Random access files
- TRACE capability

GA FORTRAN object program output is completely compatible with CAP-16 assembler output, thus providing the user with the choice of the most convenient language for any program module.

BASIC

BASIC is a simple mathematical language resembling elementary algebra. Requiring only 4K words of memory, the BASIC compiler offered by General Automation makes a possible general computation from a dedicated computer when it is not "on-line".

GA BASIC gives the user the equivalent of years of experience with the language. GA BASIC is available in two levels - a standard BASIC and an extended BASIC. Both compilers use GA device-independent I/O systems and are therefore usable under control of DBOS, RTOS and RTX as well as stand-alone programs. The stand-alone and DBOS versions are single user conversational compilers. The RTX version will handle multiple users in a real-time conversational mode. When used with DBOS and RTX all peripherals, including magnetic tape and disk are available. This versatility is unique with GA minicomputers. All versions of BASIC permit immediate execution of compiled programs.

CORE LOAD BUILDER (CLB)

The Core Load Builder processes user object programs. CLB performs all program linkages, selectively links programs from the library and produces an absolute or relocatable object program output in a form suitable for loading and execution by the host operating system. A core load map of the built program may be produced as directed by the operator.

CORE LOAD OVERLAY BUILDER (CLOB)

The Core Load Overlay Builder performs the linking and building functions of CLB with the added capability of generating program overlays for systems with a bulk storage unit. Program overlays allow program segments to be dynamically read into core, thus minimizing the core requirements of a functional program at execution time. CLOB cal also program automatic extended memory module selection.

Expanded map capabilities are available with CLOB and the associated CLOB Map Sort Program, the latter providing multiple listings of references sorted as requested by the user.

TEXT EDITORS

Text Editors provide the system user with a convenient and flexible text generation system. Source statements are entered via any source input device/file. The entered source text may be output after statements are added, deleted or modified.

The EDIT processor provides basic source line generation and modification. The Source Text Editor is a more powerful version of EDIT that has additional text handling capabilities.

DEBUG

The Debug processor assists the user in testing programs during the debugging stage. With Debug commands the user can control program execution by a multiple trap function. Debug also provides data manipulation capabilities, specified in convenient symbolic form, to:

- Selectively examine and modify memory locations, registers and bulk sectors.
- Move, search and output memory contents.

OTHER SYSTEM PROCESSORS

Other system processors are supplied to perform various services such as cross reference mapping, sequencing card decks, source record insertions and copy and verify functions. A Compress utility is available that minimizes storage requirements of programs to be maintained in relocatable form.

The reader is referred to the SPC-16 System Processor Manual, GA document number 88A00422A, for further details.

C.3 SUBROUTINE LIBRARY

The GA subroutine library consists of the USASI Basic External and Intrinsic Functions together with routines for performing floating point and extended precision integer arithmetic operations. A set of double precision arithmetic routines is provided in the library. Also included is a group of error processing and monitor interfacing routines. All subroutines are equally usable by the assembler language and FORTRAN user.

APPENDIX D
CONVERSION TABLES

This appendix contains the following reference tables:

Title	Page
Hexadecimal Arithmetic	D-2
Addition Table	D-2
Multiplication Table	D-2
Powers of 16_{10}	D-3
Powers of 10_{16}	D-3
Hexadecimal-Decimal Integer Conversion	D-4
Hexadecimal-Decimal Fraction Conversion	D-10
Powers of Two	D-14
Mathematical Constants	D-14
Decimal/Binary Position Table	D-15

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

TABLE OF POWERS OF SIXTEEN$_{10}$

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}
918 4E72 A000	13	0.1C25 C268 4976 81C2 x 16^{-10}
5AF3 107A 4000	14	0.2D09 370D 4257 3604 x 16^{-11}
3 8D7E A4C6 8000	15	0.480E BE7B 9D58 566D x 16^{-12}
23 8652 6FC1 0000	16	0.734A CA5F 6226 F0AE x 16^{-13}
163 4578 5D8A 0000	17	0.B877 AA32 36A4 B449 x 16^{-14}
DE0 B6B3 A764 0000	18	0.1272 5DD1 D243 ABA1 x 16^{-14}
8AC7 2304 89E8 0000	19	0.1D83 C94F B6D2 AC35 x 16^{-15}

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	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

Hexadecimal fractions may be converted to decimal fractions as follows:

1. 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.

$$0.\,CA9BF3_{16} = CA9\,BF3_{16} \times 16^{-6}$$

2. Find the decimal equivalent of the hexadecimal integer

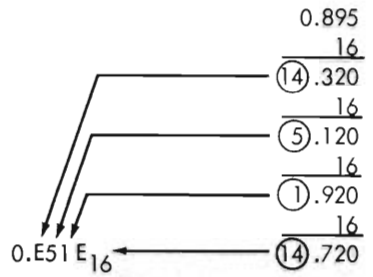
$$CA9\,BF3_{16} = 13\,278\,195_{10}$$

3. Multiply the decimal equivalent by 16^{-n}

$$\begin{array}{l} 13\,278\,195 \\ \times\ 596\,046\,448 \times 10^{-16} \\ \hline 0.791\,442\,096_{10} \end{array}$$

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



	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

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

HEXADECIMAL-DECIMAL FRACTION CONVERSION (Cont.)

Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.00 00 00 40	.00000 00149	.00 00 00 80	.00000 00298	.00 00 00 C0	.00000 00447
.00 00 00 01	.00000 00002	.00 00 00 41	.00000 00151	.00 00 00 81	.00000 00300	.00 00 00 C1	.00000 00449
.00 00 00 02	.00000 00004	.00 00 00 42	.00000 00153	.00 00 00 82	.00000 00302	.00 00 00 C2	.00000 00451
.00 00 00 03	.00000 00006	.00 00 00 43	.00000 00155	.00 00 00 83	.00000 00305	.00 00 00 C3	.00000 00454
.00 00 00 04	.00000 00009	.00 00 00 44	.00000 00158	.00 00 00 84	.00000 00307	.00 00 00 C4	.00000 00456
.00 00 00 05	.00000 00011	.00 00 00 45	.00000 00160	.00 00 00 85	.00000 00309	.00 00 00 C5	.00000 00458
.00 00 00 06	.00000 00013	.00 00 00 46	.00000 00162	.00 00 00 86	.00000 00311	.00 00 00 C6	.00000 00461
.00 00 00 07	.00000 00016	.00 00 00 47	.00000 00165	.00 00 00 87	.00000 00314	.00 00 00 C7	.00000 00463
.00 00 00 08	.00000 00018	.00 00 00 48	.00000 00167	.00 00 00 88	.00000 00316	.00 00 00 C8	.00000 00465
.00 00 00 09	.00000 00020	.00 00 00 49	.00000 00169	.00 00 00 89	.00000 00318	.00 00 00 C9	.00000 00467
.00 00 00 0A	.00000 00023	.00 00 00 4A	.00000 00172	.00 00 00 8A	.00000 00321	.00 00 00 CA	.00000 00470
.00 00 00 0B	.00000 00025	.00 00 00 4B	.00000 00174	.00 00 00 8B	.00000 00323	.00 00 00 CB	.00000 00472
.00 00 00 0C	.00000 00027	.00 00 00 4C	.00000 00176	.00 00 00 8C	.00000 00325	.00 00 00 CC	.00000 00474
.00 00 00 0D	.00000 00030	.00 00 00 4D	.00000 00179	.00 00 00 8D	.00000 00328	.00 00 00 CD	.00000 00477
.00 00 00 0E	.00000 00032	.00 00 00 4E	.00000 00181	.00 00 00 8E	.00000 00330	.00 00 00 CE	.00000 00479
.00 00 00 0F	.00000 00034	.00 00 00 4F	.00000 00183	.00 00 00 8F	.00000 00332	.00 00 00 CF	.00000 00481
.00 00 00 10	.00000 00037	.00 00 00 50	.00000 00186	.00 00 00 90	.00000 00335	.00 00 00 D0	.00000 00484
.00 00 00 11	.00000 00039	.00 00 00 51	.00000 00188	.00 00 00 91	.00000 00337	.00 00 00 D1	.00000 00486
.00 00 00 12	.00000 00041	.00 00 00 52	.00000 00190	.00 00 00 92	.00000 00339	.00 00 00 D2	.00000 00488
.00 00 00 13	.00000 00044	.00 00 00 53	.00000 00193	.00 00 00 93	.00000 00342	.00 00 00 D3	.00000 00491
.00 00 00 14	.00000 00046	.00 00 00 54	.00000 00195	.00 00 00 94	.00000 00344	.00 00 00 D4	.00000 00493
.00 00 00 15	.00000 00048	.00 00 00 55	.00000 00197	.00 00 00 95	.00000 00346	.00 00 00 D5	.00000 00495
.00 00 00 16	.00000 00051	.00 00 00 56	.00000 00200	.00 00 00 96	.00000 00349	.00 00 00 D6	.00000 00498
.00 00 00 17	.00000 00053	.00 00 00 57	.00000 00202	.00 00 00 97	.00000 00351	.00 00 00 D7	.00000 00500
.00 00 00 18	.00000 00055	.00 00 00 58	.00000 00204	.00 00 00 98	.00000 00353	.00 00 00 D8	.00000 00502
.00 00 00 19	.00000 00058	.00 00 00 59	.00000 00207	.00 00 00 99	.00000 00356	.00 00 00 D9	.00000 00505
.00 00 00 1A	.00000 00060	.00 00 00 5A	.00000 00209	.00 00 00 9A	.00000 00358	.00 00 00 DA	.00000 00507
.00 00 00 1B	.00000 00062	.00 00 00 5B	.00000 00211	.00 00 00 9B	.00000 00360	.00 00 00 DB	.00000 00509
.00 00 00 1C	.00000 00065	.00 00 00 5C	.00000 00214	.00 00 00 9C	.00000 00363	.00 00 00 DC	.00000 00512
.00 00 00 1D	.00000 00067	.00 00 00 5D	.00000 00216	.00 00 00 9D	.00000 00365	.00 00 00 DD	.00000 00514
.00 00 00 1E	.00000 00069	.00 00 00 5E	.00000 00218	.00 00 00 9E	.00000 00367	.00 00 00 DE	.00000 00516
.00 00 00 1F	.00000 00072	.00 00 00 5F	.00000 00221	.00 00 00 9F	.00000 00370	.00 00 00 DF	.00000 00519
.00 00 00 20	.00000 00074	.00 00 00 60	.00000 00223	.00 00 00 A0	.00000 00372	.00 00 00 E0	.00000 00521
.00 00 00 21	.00000 00076	.00 00 00 61	.00000 00225	.00 00 00 A1	.00000 00374	.00 00 00 E1	.00000 00523
.00 00 00 22	.00000 00079	.00 00 00 62	.00000 00228	.00 00 00 A2	.00000 00377	.00 00 00 E2	.00000 00526
.00 00 00 23	.00000 00081	.00 00 00 63	.00000 00230	.00 00 00 A3	.00000 00379	.00 00 00 E3	.00000 00528
.00 00 00 24	.00000 00083	.00 00 00 64	.00000 00232	.00 00 00 A4	.00000 00381	.00 00 00 E4	.00000 00530
.00 00 00 25	.00000 00086	.00 00 00 65	.00000 00235	.00 00 00 A5	.00000 00384	.00 00 00 E5	.00000 00533
.00 00 00 26	.00000 00088	.00 00 00 66	.00000 00237	.00 00 00 A6	.00000 00386	.00 00 00 E6	.00000 00535
.00 00 00 27	.00000 00090	.00 00 00 67	.00000 00239	.00 00 00 A7	.00000 00388	.00 00 00 E7	.00000 00537
.00 00 00 28	.00000 00093	.00 00 00 68	.00000 00242	.00 00 00 A8	.00000 00391	.00 00 00 E8	.00000 00540
.00 00 00 29	.00000 00095	.00 00 00 69	.00000 00244	.00 00 00 A9	.00000 00393	.00 00 00 E9	.00000 00542
.00 00 00 2A	.00000 00097	.00 00 00 6A	.00000 00246	.00 00 00 AA	.00000 00395	.00 00 00 EA	.00000 00544
.00 00 00 2B	.00000 00100	.00 00 00 6B	.00000 00249	.00 00 00 AB	.00000 00398	.00 00 00 EB	.00000 00547
.00 00 00 2C	.00000 00102	.00 00 00 6C	.00000 00251	.00 00 00 AC	.00000 00400	.00 00 00 EC	.00000 00549
.00 00 00 2D	.00000 00104	.00 00 00 6D	.00000 00253	.00 00 00 AD	.00000 00402	.00 00 00 ED	.00000 00551
.00 00 00 2E	.00000 00107	.00 00 00 6E	.00000 00256	.00 00 00 AE	.00000 00405	.00 00 00 EE	.00000 00554
.00 00 00 2F	.00000 00109	.00 00 00 6F	.00000 00258	.00 00 00 AF	.00000 00407	.00 00 00 EF	.00000 00556
.00 00 00 30	.00000 00111	.00 00 00 70	.00000 00260	.00 00 00 B0	.00000 00409	.00 00 00 F0	.00000 00558
.00 00 00 31	.00000 00114	.00 00 00 71	.00000 00263	.00 00 00 B1	.00000 00412	.00 00 00 F1	.00000 00561
.00 00 00 32	.00000 00116	.00 00 00 72	.00000 00265	.00 00 00 B2	.00000 00414	.00 00 00 F2	.00000 00563
.00 00 00 33	.00000 00118	.00 00 00 73	.00000 00267	.00 00 00 B3	.00000 00416	.00 00 00 F3	.00000 00565
.00 00 00 34	.00000 00121	.00 00 00 74	.00000 00270	.00 00 00 B4	.00000 00419	.00 00 00 F4	.00000 00568
.00 00 00 35	.00000 00123	.00 00 00 75	.00000 00272	.00 00 00 B5	.00000 00421	.00 00 00 F5	.00000 00570
.00 00 00 36	.00000 00125	.00 00 00 76	.00000 00274	.00 00 00 B6	.00000 00423	.00 00 00 F6	.00000 00572
.00 00 00 37	.00000 00128	.00 00 00 77	.00000 00277	.00 00 00 B7	.00000 00426	.00 00 00 F7	.00000 00575
.00 00 00 38	.00000 00130	.00 00 00 78	.00000 00279	.00 00 00 B8	.00000 00428	.00 00 00 F8	.00000 00577
.00 00 00 39	.00000 00132	.00 00 00 79	.00000 00281	.00 00 00 B9	.00000 00430	.00 00 00 F9	.00000 00579
.00 00 00 3A	.00000 00135	.00 00 00 7A	.00000 00284	.00 00 00 BA	.00000 00433	.00 00 00 FA	.00000 00582
.00 00 00 3B	.00000 00137	.00 00 00 7B	.00000 00286	.00 00 00 BB	.00000 00435	.00 00 00 FB	.00000 00584
.00 00 00 3C	.00000 00139	.00 00 00 7C	.00000 00288	.00 00 00 BC	.00000 00437	.00 00 00 FC	.00000 00586
.00 00 00 3D	.00000 00142	.00 00 00 7D	.00000 00291	.00 00 00 BD	.00000 00440	.00 00 00 FD	.00000 00589
.00 00 00 3E	.00000 00144	.00 00 00 7E	.00000 00293	.00 00 00 BE	.00000 00442	.00 00 00 FE	.00000 00591
.00 00 00 3F	.00000 00146	.00 00 00 7F	.00000 00295	.00 00 00 BF	.00000 00444	.00 00 00 FF	.00000 00593

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 13113	.00 00 56 00	.00000 51259	.00 00 96 00	.00000 89406	.00 00 D6 00	.00001 27553
.00 00 17 00	.00000 13709	.00 00 57 00	.00000 51856	.00 00 97 00	.00000 90003	.00 00 D7 00	.00001 28149
.00 00 18 00	.00000 14305	.00 00 58 00	.00000 52452	.00 00 98 00	.00000 90599	.00 00 D8 00	.00001 28746
.00 00 19 00	.00000 14901	.00 00 59 00	.00000 53048	.00 00 99 00	.00000 91195	.00 00 D9 00	.00001 29342
.00 00 1A 00	.00000 15497	.00 00 5A 00	.00000 53644	.00 00 9A 00	.00000 91791	.00 00 DA 00	.00001 29938
.00 00 1B 00	.00000 16093	.00 00 5B 00	.00000 54240	.00 00 9B 00	.00000 92387	.00 00 DB 00	.00001 30534
.00 00 1C 00	.00000 16689	.00 00 5C 00	.00000 54836	.00 00 9C 00	.00000 92983	.00 00 DC 00	.00001 31130
.00 00 1D 00	.00000 17285	.00 00 5D 00	.00000 55432	.00 00 9D 00	.00000 93579	.00 00 DD 00	.00001 31726
.00 00 1E 00	.00000 17881	.00 00 5E 00	.00000 56028	.00 00 9E 00	.00000 94175	.00 00 DE 00	.00001 32322
.00 00 1F 00	.00000 18477	.00 00 5F 00	.00000 56624	.00 00 9F 00	.00000 94771	.00 00 DF 00	.00001 32918
.00 00 20 00	.00000 19073	.00 00 60 00	.00000 57220	.00 00 A0 00	.00000 95367	.00 00 E0 00	.00001 33514
.00 00 21 00	.00000 19669	.00 00 61 00	.00000 57816	.00 00 A1 00	.00000 95963	.00 00 E1 00	.00001 34110
.00 00 22 00	.00000 20265	.00 00 62 00	.00000 58412	.00 00 A2 00	.00000 96559	.00 00 E2 00	.00001 34706
.00 00 23 00	.00000 20861	.00 00 63 00	.00000 59008	.00 00 A3 00	.00000 97155	.00 00 E3 00	.00001 35302
.00 00 24 00	.00000 21457	.00 00 64 00	.00000 59604	.00 00 A4 00	.00000 97751	.00 00 E4 00	.00001 35898
.00 00 25 00	.00000 22053	.00 00 65 00	.00000 60200	.00 00 A5 00	.00000 98347	.00 00 E5 00	.00001 36494
.00 00 26 00	.00000 22649	.00 00 66 00	.00000 60796	.00 00 A6 00	.00000 98943	.00 00 E6 00	.00001 37090
.00 00 27 00	.00000 23245	.00 00 67 00	.00000 61392	.00 00 A7 00	.00000 99539	.00 00 E7 00	.00001 37686
.00 00 28 00	.00000 23841	.00 00 68 00	.00000 61988	.00 00 A8 00	.00001 00135	.00 00 E8 00	.00001 38282
.00 00 29 00	.00000 24437	.00 00 69 00	.00000 62584	.00 00 A9 00	.00001 00731	.00 00 E9 00	.00001 38878
.00 00 2A 00	.00000 25033	.00 00 6A 00	.00000 63180	.00 00 AA 00	.00001 01327	.00 00 EA 00	.00001 39474
.00 00 2B 00	.00000 25629	.00 00 6B 00	.00000 63776	.00 00 AB 00	.00001 01923	.00 00 EB 00	.00001 40070
.00 00 2C 00	.00000 26226	.00 00 6C 00	.00000 64373	.00 00 AC 00	.00001 02519	.00 00 EC 00	.00001 40666
.00 00 2D 00	.00000 26822	.00 00 6D 00	.00000 64969	.00 00 AD 00	.00001 03116	.00 00 ED 00	.00001 41263
.00 00 2E 00	.00000 27418	.00 00 6E 00	.00000 65565	.00 00 AE 00	.00001 03712	.00 00 EE 00	.00001 41859
.00 00 2F 00	.00000 28014	.00 00 6F 00	.00000 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

POWERS OF TWO

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5
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	0.000 000 000 000 000 027 755 575 615 628 913 510 590 791 702 270 507 812 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 C1B7
$\sqrt{\pi}$	1.77245 38509 05516	1.C5BF 891C
$\ln \pi$	1.14472 98858 49400	1.250D 048F
e	2.71828 18284 59045	2.B7E1 5163
e^{-1}	0.36787 94411 71442	0.5E2D 58D9
$\sqrt{e}$	1.64872 12707 00128	1.A612 98E2
$\log_{10} e$	0.43429 44819 03252	0.6F2D EC55
$\log_2 e$	1.44269 50408 88963	1.7154 7653
γ	0.57721 56649 01533	0.93C4 67E4
$\ln \gamma$	-0.54953 93129 81645	-0.8CAE 9BC1
$\sqrt{2}$	1.41421 35623 73095	1.6A09 E668
$\ln 2$	0.69314 71805 59945	0.B172 17F8
$\log_{10} 2$	0.30102 99956 63981	0.4D10 4D42
$\sqrt{10}$	3.16227 76601 68379	3.298B 075C
$\ln 10$	2.30258 40929 94046	2.4D76 3777

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	.999 999 999 068 677 425 384 521 484 375
2 147 483 647		31	.999 999 999 534 338 712 692 260 742 187 5
4 294 967 295		32	.999 999 999 767 169 356 346 130 371 093 75
8 589 934 591		33	.999 999 999 883 584 678 173 065 185 546 875
17 179 869 183	10	34	.999 999 999 941 792 339 086 532 592 773 437 5
34 359 738 367		35	.999 999 999 970 896 169 543 266 296 386 718 75
68 719 476 735		36	.999 999 999 985 448 034 771 633 148 193 359 375
137 438 953 471	11	37	.999 999 999 992 724 042 385 816 574 096 679 687 5
274 877 906 943		38	.999 999 999 996 362 021 192 908 287 048 339 843 75
549 755 813 887		39	.999 999 999 998 181 010 596 454 143 524 169 921 875
1 099 511 627 775	12	40	.999 999 999 999 090 505 298 227 071 762 084 960 937 5
2 199 023 255 551		41	.999 999 999 999 545 252 649 113 535 881 042 480 468 75
4 398 046 511 103		42	.999 999 999 999 772 626 324 556 767 940 521 240 234 375
8 796 093 022 207		43	.999 999 999 999 886 313 162 278 383 970 260 620 117 187 5
17 592 186 044 415	13	44	.999 999 999 999 943 156 581 139 191 985 130 310 058 593 75
35 184 372 088 831		45	.999 999 999 999 971 578 290 569 595 992 565 155 029 296 875
70 368 744 177 663		46	.999 999 999 999 985 789 145 284 797 996 282 577 514 648 437 5
140 737 488 355 327	14	47	.999 999 999 999 992 894 572 642 398 998 141 288 757 324 218 75
281 474 976 710 655		48	.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	.999 999 999 999 999 555 910 790 149 937 383 830 547 332 763 671 875
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.

INDEX

NOTE

Parenthesized entries refer to figures or tables.

A

Addend Bus 2-10, 2-14, 2-15
Addresses 2-5 to 2-6, 7-6
Addressing Modes 1-5, 2-17 to 2-18, (1-10)
 Absolute 2-20, (2-19)
 Base relative 1-9, 2-21, 3-4, C-2, (2-11, 2-19)
 Direct 2-17, 2-23, 7-15 to 7-16, (2-19)
 Indexed 2-25, 2-26, (2-11, 2-19)
 Indirect 2-17, 2-24, (2-19)
 Program relative 1-9, 2-22, 3-3, (2-19)
 Literal 2-23
Addressing Parameters 3-2
 m (addressing mode) 3-2
 * (direct/indirect) 3-2
 i (indexing) 3-2
 Address expression 3-2
Assembler (CAP-16 Assembly language) 3-1, 3-2, C-8 to C-9
 Macro assembler C-9
Arithmetic and Logical Unit 2-10, (2-9)
Arithmetic Processing Unit 7-17
Augend Bus 2-10, 2-16, 2-17
Auto Restart Feature 1-8, 5-13, 7-3, 7-4, (2-32)
ASCII codes B-1 to B-4

B

BASIC C-8
"Bit" Instructions 1-5, 2-15, 2-26, 3-23, 3-24, 3-25
Bootstrapping 1-5, 6-7 to 6-8
 Teletype bootstrap 6-8
"Byte" Instructions 1-5, 2-27, 3-19, 3-20, 3-52 to 3-53

C

Cable Interface Driver 5-6, (5-55)
Cabling 8-18 to 8-21
Central Processing Unit (CPU) 2-1, 8-3, (2-9, 8-2, 8-4, 8-5)
Chaining 5-32, 5-34, 5-42, (5-45)
Channel Address Register (CAR) 2-35, 4-5, 5-27, 5-28, 5-34, (2-32, 4-6)
 See Direct Memory Access
Character Codes B-1 to B-4
Cold Start 1-5, 5-13
Configurations 1-1 to 1-2, 8-1, 8-3, 8-11, (1-2, 1-3, 2-29)
Console Section 6
Console Switches 3-55, 6-1
Control Instruction 2-30, 2-34, (5-17)
Controller, Peripheral 2-28, Section 5
Core Load Builder C-7
Cycle Speed (1-2, 3-4)
Cycle Stealing 1-7, 5-4, 5-46

D

Data Bus 2-10, 2-12, 2-14, 2-15, 2-16, 2-17
Data Channel Assignments (4-6)
Data Channel Bus 5-4, 5-34, 5-51, (5-5)
 Signals on 5-32 to 5-37
Data Channel Module 5-4, 5-35, 5-54, (5-5)
 Example of controller 5-53
 Input block transfers 5-36
 Output block transfers 5-42
Data Channels for Peripheral Equipment (4-6)
Data Format 2-2, 2-6 to 2-7, 7-17

INDEX (Continued)

D

Data Input/Output Instructions 2-30, 5-18, (5-19, 5-20)
Dedicated Memory Locations 2-31, 2-35, 3-73, 7-1, (2-32)
Device Select Codes 2-30, 3-79, 5-1, (5-2)
Direct Memory Access (DMA) 1-6, 2-34 to 2-35, 4-5 to 4-6, 5-4, 5-23, 5-10, 5-11
  Examples 5-48, 5-51
  Versus programmed I/O 5-4
Double Precision 7-17
Dynamic Storage Allocation 3-4, 3-73, C-2

E

Effective Address 2-18, 2-23, 2-24, 2-25, 3-5, (2-19, 3-8, 3-15)
E Register 1-6, 2-11, 2-15, 2-33, 3-11, 3-72, 7-11
Execution Time (1-11)
External I/O Cable 5-7, (5-55)
External I/O Package 1-2, 5-6

F

Foreground/Background 1-6, 2-11, 2-14, 3-68, 3-69, 3-70
FORTRAN C-3, C-4, C-5, C-7

G

General Purpose Registers 1-6 to 1-7, 2-10 to 2-11, (2-11, 3-5)

I

IDLE Mode 6-1, 6-4, 6-5, 7-3
Inbus 5-7, 5-24, 5-39
Instruction Formats 2-7 to 2-9, Section 3, (A-1)
Instruction Set 1-5 to 1-6, Section 3, A-1, (1-11, 3-96 to 3-98)
Integral I/O Package 1-2, 5-4, 5-57, (1-2, 5-59)
Interrupt Capability 5-1
Interrupt--Driven I/O
 See Programmed I/O
Interrupt Masking 2-31, 2-33, 2-34, 4-7, 5-23, 7-1, 7-4, 7-11, (4-8)
Interrupt Priority Levels
 See Priority Interrupt System
Interrupts 1-6, 1-7, 1-8, 2-31, 2-33, 3-11, 3-71, 4-3, 5-24, (1-12, 1-13, 5-25)
  External 2-33, 3-69
  Internal 2-33, 3-69, 4-7, (4-8)
Interrupt Service Routines 2-31, 3-72, 4-3
Interrupt System Enable (ISE) 2-15, 2-31, 2-33, 3-11, 3-57, 3-58, 3-67, 3-69, (2-14)
Interrupt Vectors 2-31, 4-3, 5-3, (2-32, 5-2)
I/O 1-6, 2-15, 2-28, 2-30, 5-15, (1-12, 5-5, 5-16, 7-13)
I/O Bus 1-3, 2-28, 2-34, 5-4, 5-6, 5-7, 5-55
  Signals on 5-7 to 5-13, (5-56, 5-59)
  Terminations 5-14
  Timing (5-30)
I/O Enclosure, External 1-3, 5-6, 5-51, (5-52, 8-7, 8-8)
I/O Transfer Rates (1-11)

K

K Register 2-17, 5-12, 6-1

L

LARS (Load All Registers and Status) 1-6, 1-9, 2-14, 2-33, 3-20 to 3-21
Link Indicator 2-10, 2-12, 2-13, 3-33, 3-34, 3-53, 3-70
Literal 2-23
Loaders 6-7, 6-8
Logical Operations 2-3 to 2-4
L Register 2-15

M

Machine Language 3-1
Memory Buffer Register 2-15

INDEX (Continued)

M

Memory Configurations 1-3, 1-5, 7-6, (1-3, 1-4, 1-10)
Extended memory 7-5, (6-7)
Memory, Dedicated (2-32)
Memory Protection 7-19, (2-32)
M Register 2-15
Multiply/Divide Option 3-83

O

Operating Systems 1-1, C-2
Disk/drum based (DBOS-16) C-4 to C-6
Free standing (FSOS-16) C-3 to C-4
Real-time Executive (RTX-16) C-6
Real-time operating system (RTOS-16) C-6 to C-8
Teletype only (TTY only) C-2 to C-3
Operation Codes 2-7, (2-8)
Operations Monitor Alarm (OMA) 1-8, 3-70, 7-3
Outbus 2-15, 5-7, 5-18, 5-39
Overflow Indicator 2-10, 2-12, 2-13, 3-53

P

Parameters, Addressing
See Addressing Parameters
Peripheral Equipment 1-1, 2-28, 3-79, Section 4, 5-1, (4-6, 5-2)
Plus Indicator 2-12, 2-14, 3-21, 3-23
Power Fail Detect Feature 1-7, 1-8, 2-15, 2-31, 7-3, (2-32)
Power Requirements 8-23
P Register 2-14 to 2-15, 2-31, 2-33, 3-11, 6-3
Priority Interrupt System 2-31, 2-33, 2-34, 5-3, 5-24
Process Control Enclosure 5-6
Program Counter
See P Register
Programmed I/O 2-30, 3-78, 4-1, 5-1, 5-14
Data phase 5-14
Device select phase 5-14, 5-18
Interrupt driven 2-31, 4-3
Versus DMA 5-4
Test loop-driven 2-30, 4-1
Programs, Execution of 6-7
Pulse Operations Monitor Alarm (PMA) 1-7, 3-70, 7-3

Q

Q Register 2-12, 2-16
Q-R SHFT 2-16

R

Read-Only Memory (ROM) 1-2, 1-5, 3-4, 3-35, 3-68
Recursive Code 1-9, 3-74, C-2
Re-entrant Code 1-6, 1-9, 3-74, C-2
Register Identifier Codes 2-11, (3-5)
Register Display Indicator 2-17, 6-1, (6-2)
Relative Time Clock (RTC) 1-7, 7-1, (2-32)
R Register 2-12, 2-16
R SWAP 2-16
RUN Mode 6-1, 6-4, 6-6, 7-3

S

SARS (Store All Registers and Status) 1-6, 1-8, 2-14, 2-33, 3-21
Scan Control Register (SCR) 2-35, 4-5, 4-6, 5-27, 5-32, (2-32)
See Direct Memory Access
Shift Counter 2-12, 2-13, 3-53, 6-3
Software 6-7, C-1
Status Register
See S Register
S Register 2-12, 2-14, 6-3
STEP Switch 6-4
Subroutine Library C-7 to C-8
Subroutine Linkage 1-6
Subroutine Nesting 2-33
Systems Reset (SYRT) 1-8, 3-70, 5-12, 5-26, 6-4, 7-3, (6-2)
System Safe Signal (SFEC) 1-8, 5-12, 7-3
SYNC Pulse 3-71, 3-72, 5-12

INDEX (Continued)

T

TEST Instruction 2-30, 2-34, (5-17)
Text Editor C-8
Timing 5-14
 Control instruction (5-17)
 Data channel initialization (5-38)
 Data input (5-21)
 Data output (5-20)
 DMA chaining (5-45)
 DMA data input (5-30, 5-31, 5-40, 5-41)
 DMA data output (5-28, 5-29, 5-40, 5-43, 5-44)
 Interrupt sequence (5-25)
 I/O bus (5-33)
 Test instruction (5-17)
Two's Complement 2-2, 2-3
 WAIT Mode 3-71, 6-6

W

W-R SHFT 2-17
Word Format 2-2
Word Length 1-5
W Register 2-16 to 2-17, 6-3

X

XIO Instructions 1-7, 2-30, 2-33
 For extended memory 7-6

Z

Zero Indicator 2-12, 2-14, 3-32

LIST OF INSTRUCTIONS BY MNEMONIC

ADD	3-36	EXIT	3-58	PMA	3-70	STA	3-10
ADDC	3-39					STBY	3-20
ADDS	3-53	FMS	3-69	RBIT	3-24	STR	3-18
ADDV	3-43			RCSM	3-82	SUB	3-37
ADDVC	3-45	INCM	3-22	RCSR	3-82	SUBC	3-39
AND	3-37	INCR	3-50	RCSW	3-55	SUBV	3-43
ANDC	3-40	INE	3-69	RISE	3-58	SUBVC	3-45
ADNV	3-44	INH	3-69	RLK	3-53	SYNC	3-71
ANDVC	3-45			RTR	3-36		
		JMP	3-10	RTRN	3-57	TBIT	3-25
BMS	3-70	JSR	3-11			TEST	3-81
				SARS	3-21	TRS	3-55
CMPL	3-51	LARS	3-20	SBIT	3-23	TSR	3-54
CMR	3-18	LDA	3-9	SKIP	3-29		
CTRL	3-81	LDBY	3-19	SKM	3-33	WAIT	3-72
		LDR	3-15	SKN	3-32		
DECM	3-23	LDV	3-43	SKOF	3-33	XEC	3-57
DECR	3-50	LKR	3-71	SKOT	3-33	XOR	3-38
DIV	3-89	LKS	3-71	SKP	3-32	XORC	3-40
DSPL	3-56			SKR	3-34	XORV	3-44
DTIM	3-81	MPY	3-84	SKS	3-33	XORVC	3-46
DTIR	3-80			SKZ	3-32		
DTOM	3-80	OR	3-38	SRA	3-62	ZERO	3-51
DTOR	3-79	ORC	3-40	SRC	3-63	ZLBY	3-52
		ORV	3-44	SRCL	3-64	ZRBY	3-52
EXBY	3-52	ORVC	3-46	SRLC	3-63		

COMMENT SHEET

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, CALIF.

BUSINESS REPLY MAIL
No Postage Necessary if Mailed in the United States

Postage Will Be Paid By . . .

GENERAL AUTOMATION, INC.
1055 East Street
Anaheim, California 92803

Attention: Technical Publications

FOLD

CUT ALONG LINE

General Automation, Inc.
1055 East Street
Anaheim, California 92803

ADDITIONAL COMMENTS:



GENERAL AUTOMATION, INC.

1055 South East Street, Anaheim, California 92805 (714) 778-4800