Publication Number 29-588

MODEL 5/16
USER'S MANUAL

INTERDATA®
A UNIT OF
PERKIN-ELMER DATA SYSTEMS
2 Crescent Place, Oceanport, N.J. 07757 • (201) 229-4040

February 1977

PAGE REVISION STATUS SHEET

PUBLICATION NUMBER 29-588

TITLE Model 5/16 User's Manual

REVISION R00 DATE 4/28/77

PAGE	REV.	DATE	PAGE	REV.	DATE	PAGE	REV.	DATE
i/ii	R00	4/77	A2-1 thru	R00	4/77			
iii thru vi	R00	4/77	A2-3/ A2-4					
1-1 thru 1-3/ 1-4	R00	4/77	A3-1 thru A3-3/ A3-4	R00	4/77			
2-1 thru 2-9/ 2-10	R00	4/77	A4-1, A4-2	R00	4/77			
3-1 thru 3-31/ 3-32	R00	4/77	A5-1 thru A5-6	R00	4/77			
4-1 thru 4-23/ 4-24	R00	4/77	A6-1, A6-2	R00	4/77			
5-1 thru 5-16	R00	4/77	A7-1 thru A7-4	R00	4/77			
6-1 thru 6-8	R00	4/77	A8-1, A8-2	R00	4/77			
7-1 thru 7-21/ 7-22	R00	4/77	A9-1 thru A9-3/ A9-4	R00	4/77			
8-1 thru 8-38	R00	4/77	A10-1 thru A10-6	R00	4/77			
9-1 thru A1-1/ A1-2	R00	4/77	A11-1 thru A11-3/ A11-4	R00	4/77			
			I-1 thru I-3/ I-4	R00	4/77			

FOREWORD

The Model 5/16 User's Manual provides information for the programmer, designer, and operator in the use of the Model 5/16 computer system. The programmer is given a detailed description of each instruction in the Model 5/16 instruction set. Such information includes valuable system related information presented in the form of programming notes, examples, and Condition Codes. To assist the designer in choosing his own controller interface to the Model 5/16 system, a wealth of design information is presented on the Input/Output portion of the Model 5/16, corresponding to the Micro I/O and the Multiplexor Busses. ASCII Programmer's Console information is given in the manual to facilitate program preparation and execution for the programmer and operator of the system. Finally, several Appendices are included as valuable reference information for Model 5/16 users.

Table of Contents

CHAPTER 1 INTRODUCTION 1-1

SYSTEM ARCHITECTURE 1-1
MEMORY 1-1
INSTRUCTION SET 1-1
INPUT/OUTPUT SYSTEM 1-2
SOFTWARE 1-2
PROCESSOR OPTIONS AND PERIPHERALS 1-2

CHAPTER 2 SYSTEM DESCRIPTION 2-1

PROCESSOR 2-1
Program Status Word 2-2
Processor Interrupts 2-3
General Registers 2-3
Reserved Memory Locations 2-3
MEMORY 2-4
Memory Expansion 2-4
Memory Addressable ROM 2-5
Implementation 2-5
PROCESSOR OPERATIONS 2-5
DATA FORMATS 2-6
Fixed Point Data 2-6
Logical Data 2-6
INSTRUCTION FORMATS 2-6
Branch Instruction Formats 2-7
Programming Examples 2-7
Register to Register (RR) Format 2-7
Short Form (SF) Format 2-7
Register and Indexed Storage (RX) Format 2-7
Register and Immediate Storage (RI) Format 2-8

CHAPTER 3 LOGICAL OPERATIONS 3-1

INTRODUCTION 3-1
DATA FORMATS 3-2
Boolean Operations 3-2
List Processing 3-3
LOGICAL INSTRUCTION FORMATS 3-4
LOGICAL INSTRUCTIONS 3-4

CHAPTER 4 BRANCHING 4-1

OPERATIONS 4-1
Decision Making 4-1
Subroutine Linkage 4-1
BRANCH INSTRUCTION FORMATS 4-1
BRANCH INSTRUCTIONS 4-1
EXTENDED BRANCH MNEMONICS 4-7

CHAPTER 5 FIXED POINT ARITHMETIC 5-1

DATA FORMATS 5-1
FIXED POINT NUMBER RANGE 5-1
FIXED POINT INSTRUCTION FORMATS 5-2
FIXED POINT INSTRUCTIONS 5-2

CHAPTER 6 STATUS SWITCHING AND INTERRUPTS 6-1

INTRODUCTION 6-1

PROGRAM STATUS WORD 6-1
Wait State 6-2

INTERRUPT SYSTEM 6-2
External Interrupt 6-3
Fixed Point Fault Interrupt 6-3
Immediate Interrupt 6-3
System Queue Interrupt 6-4
Illegal Instruction Interrupt 6-4
Supervisor Call Interrupt 6-4
Simulated Interrupt 6-4
STATUS SWITCHING INSTRUCTION FORMATS 6-4
STATUS SWITCHING INSTRUCTIONS 6-4

CHAPTER 7 INPUT/OUTPUT OPERATIONS 7-1

DEVICE CONTROLLERS 7-1
Device Addressing 7-1
Processor/Controller Communication 7-2
Device Priorities 7-2
INTERRUPT SERVICE POINTER TABLE 7-2
I/O INSTRUCTION FORMATS 7-3
I/O INSTRUCTIONS 7-3
CONTROL OF I/O OPERATIONS 7-17
INTERRUPT DRIVEN I/O 7-18
Automatic Vectoring 7-18
Software Vectoring 7-19
DIRECT MEMORY ACCESS 7-19
DMA Devices 7-20
DMA Operation 7-20

CHAPTER 8 INPUT/OUTPUT SYSTEMS 8-1

INTRODUCTION 8-1
MICRO I/O BUS 8-1
Address Lines 8-1
Data Lines 8-1
Control Lines 8-3
Initialize Line 8-4
Micro I/O Bus Operation 8-4
MICRO I/O INTERFACE DESIGN 8-4
Micro I/O Bus 8-4
Micro I/O Bus Device Addressing 8-8
Micro I/O Bus Data Line Buffering 8-9
Interrupt Logic 8-10
Data and Status Output 8-11
Micro I/O Bus Timing 8-13
Read Data or Sense Status 8-15
Write Data or Output Command 8-15
Acknowledge Interrupt 8-16
Block I/O 8-16
Micro-DMA Bus 8-17
MULTIPLEXOR BUS 8-19
Data Lines (D00:15) 8-20
Control Lines 8-20
Initialize Line 8-20
MULTIPLEXOR I/O DEVICE CONTROLLER LOGIC DESIGN 8-21
Multiplexor Bus 8-21
Multiplexor Bus Loading Rules 8-21
Multiplexor Bus Length Restrictions 8-23
Multiplexor Bus Terminators 8-23
Device Controller Addressing 8-23
Interrupt Control 8-23
Multiplexor Bus Wiring 8-25
Multiplexor Bus Timing 8-25
General Multiplexor Bus Interface 8-27
DC-DC Converter Specifications 8-28
MULTIPLEXOR I/O INTERFACE DESIGN (PROGRAMMING CHARACTERISTIC) 8-30
Data and Status Input 8-31
Data and Command Output 8-31
Byte-Oriented Device Controller Design 8-31
I/O Bus Sequence and Timing 8-33
Data Transfer 8-33

MULTIPLEXOR I/O INTERFACE PHYSICAL PACKAGING, CABLING, AND CONNECTIONS 8-35
7 Inch Half-boards 8-35
15 Inch Boards 8-35
Functions Common to the 7 Inch and 15 Inch I/O Interface Boards 8-37/8-38

CHAPTER 9 ASCII PROGRAMMER'S CONSOLE 9-1

CONFIGURATION 9-1
PROGRAMMING INSTRUCTIONS 9-2
Sense Status (SS or SSR) 9-2
Output Command (OC or OCR) 9-2
Key Operated Security Lock 9-2
Control Switches 9-3
OPERATION PROCEDURES 9-3
Power Up 9-3
ASCII Console 9-3
Open Cell and Examine 9-3
Open and Examine Next Cell 9-4
Open and Examine Location Shown 9-4
Modify Open Cell 9-4
Program Execution 9-4
Program Termination 9-4
General Register Examination and Modification 9-4
Program Status Word Examination and Modification 9-4
Memory Initialization 9-5
DATA FORMAT 9-6
Status and Command Bytes 9-6
Status 9-6
Commands 9-6
PROGRAMMING SEQUENCES 9-7
Programming Notes 9-7
Status Monitoring I/O 9-7
Interrupt I/O Control 9-7
INTERRUPTS 9-7

TABLES

TABLE 2-1. INTERPRETATION OF PSW BITS 2-2
TABLE 5-1. FIXED POINT FORMAT RELATIONS 5-1
TABLE 8-1. MICRO I/O BUS LINES 8-2
TABLE 8-2. FUNCTIONS OF THE C/D DATA LINE 8-3
TABLE 8-3. SIGNALS CARRIED BY MICRO I/O BUS CABLES 8-6
TABLE 8-4. TRANSMITTER CHARACTERISTICS 8-6
TABLE 8-5. RECEIVER CHARACTERISTICS 8-6
TABLE 8-6. RELATIONSHIP BETWEEN MICRO I/O BUS AND MOTOROLA AND INTEL CHIPS 8-7
TABLE 8-7. MULTIPLEXOR BUS LINES 8-19
TABLE 9-1. ASCII CONSOLE COMMAND SUMMARY 9-2
TABLE 9-2. SERIAL INPUT/OUTPUT PORT STATUS AND COMMAND DATA 9-6

ILLUSTRATIONS

Figure 2-1. System Block Diagram 2-1
Figure 2-2. Program Status Word Format 2-2
Figure 2-3. M51-104 Memory Expansion Option 2-4
Figure 2-4. M51-105 Memory Expansion Option 2-5
Figure 2-5. Instruction Formats 2-6
Figure 2-6. 16-Bit Instruction Format Examples 2-9/2-10
Figure 3-1. Logical Data 3-2
Figure 3-2. Circular List Definition 3-3
Figure 3-3. Circular List 3-3
Figure 5-1. Fixed Point Data Words Format 5-1
Figure 6-1. Program Status Word Format 6-1
Figure 8-1. System Interface Block Diagram 8-2
Figure 8-2. Micro I/O Bus Communication Circuits 8-5
Figure 8-3. Typical Motorola Peripheral Chip Control 8-7

Figure 8-4. Typical INTEL Peripheral Chip Control 8-8
Figure 8-5. Micro I/O Bus Device Addressing 8-8
Figure 8-6. Micro I/O Bus Data Line Transceivers 8-9
Figure 8-7. Micro I/O Bus Control Line Receivers and Interrupt Logic 8-10
Figure 8-8. Data Output Multiplexor 8-11
Figure 8-9. Teletype Controller 8-12
Figure 8-10. Tape Recorder Interface 8-13
Figure 8-11. Centronics Line Printer Interface 8-14
Figure 8-12. ENABLE Timing 8-14
Figure 8-13. Micro I/O Bus Input Timing 8-15
Figure 8-14. Micro I/O Bus Output Timing 8-16
Figure 8-15. Micro I/O Bus Acknowledge Timing 8-17
Figure 8-16. Micro I/O Bus Timing for Read Block 8-18
Figure 8-17. Micro I/O Bus Timing for Write Block 8-18
Figure 8-18. Logic and Timing for Model 5/16 Micro-DMA 8-19
Figure 8-19. I/O Interface Transmit and Receive Characteristics 8-22
Figure 8-20. General Interface to Multiplexor Bus (Byte or Halfword Oriented) 8-24
Figure 8-21. Multiplexor Bus Output Timing 8-26
Figure 8-22. Multiplexor Bus Input Timing 8-26
Figure 8-23. DC/DC Converter 8-28
Figure 8-24. Status Byte 8-30
Figure 8-25. Command Byte 8-30
Figure 8-26. Bus Sequence for Byte or Halfword Device 8-32
Figure 8-27. Read/Write Data Transfer Bus Sequence 8-33
Figure 8-28. Address and Data Transfer Timing Between Processor and I/O Device Interface (Command and Data Available) 8-34
Figure 8-29. Address and Data Transfer Timing Between I/O Interface and Processor (Data Request and Sense Status) 8-34
Figure 8-30. Interrupt Timing 8-35
Figure 8-31. 16-398 Half Board Adapter Kit 8-36
Figure 8-32. 15" x 15" Printed Circuit Board 8-36
Figure 8-33. I/O Back Panel Connections 8-37/8-38
Figure 9-1. Model 5/16 Turnkey Console 9-1
Figure 9-2. Keyboard Layout 9-1
Figure 9-3. ASCII Character U, Eleven Bit Code 9-2
Figure A8-1. Bootstrap Loader Turnkey Console A8-1
Figure A8-2. Bootstrap Loader A8-2

INDEX I-1

APPENDICES

APPENDIX 1 MODEL 5/16 OP-CODE MAP A1-1/A1-2
APPENDIX 2 INSTRUCTION SUMMARY – ALPHABETICAL WITH ATTRIBUTES A2-1
APPENDIX 3 INSTRUCTION SUMMARY – NUMERICAL A3-1
APPENDIX 4 EXTENDED BRANCH MNEMONICS A4-1
APPENDIX 5 ARITHMETIC REFERENCES A5-1
APPENDIX 6 INSTRUCTION TIMING A6-1
APPENDIX 7 I/O REFERENCES A7-1
APPENDIX 8 BOOTSTRAP LOADER OPTION A8-1
APPENDIX 9 CONSOLE ROUTINE FLOWCHART A9-1
APPENDIX 10 SAMPLE PROGRAM A10-1
APPENDIX 11 PHYSICAL AND ELECTRICAL CHARACTERISTICS A11-1

CHAPTER 1
INTRODUCTION

The INTERDATA Model 5/16 Processor combines powerful minicomputer performance and architecture with microprocessor interfacing economy for the lowest possible system cost. The Model 5/16 is manufactured on a single fifteen-inch (38.1cm X 38.1cm) printed circuit board to increase reliability and lower packaging costs. The basic system configuration includes the Model 5/16 Processor with 100 separate instructions, 8KB of MOS memory, 16 general purpose registers, hardware vectoring for up to 255 external devices, INTERDATA's standard Multiplexor Channel I/O Bus and a microprocessor compatible Micro I/O Bus and a power supply. A variety of standard off-the-shelf options permits the user to tailor his system to meet present and future needs. The overall efficiency of the 5/16 makes it well suited for a variety of applications that concentrate on small, dedicated systems, but do not preclude larger multi-user systems.

Original equipment manufactured products are provided with a low-cost production processor, combined with field-proven hardware and software of the INTERDATA computer family. In addition, the 5/16 takes advantage of all the new, low-cost peripheral interface chips designed for use with the Intel 8080 and Motorola 6800 microprocessors.

SYSTEM ARCHITECTURE

INTERDATA Model 5/16 Processor design is based on the powerful third-generation architecture. The advantages inherent in this type of architecture greatly simplify system design, programming, and debugging. A large, task-oriented instruction set allows the programmer to concentrate on system-level programming instead of re-creating basic functions, such as Exclusive-OR, multiple shifts, or byte processing.

The multi-accumulator architecture pioneered by INTERDATA provides 16 general purpose registers to increase programming flexibility and to eliminate awkward accumulator house-keeping, characteristic of machines with fewer general registers. All 16 registers are available for use at the programmer's discretion. Since none of these registers are dedicated to any specific function, such as index registers, stack pointers, subroutine return pointers, or location counter, all 16 are available for the programmer's use. He is free to use these registers for storage of partial results, frequently used constants, loop management control, or whatever else is needed.

Of the 16 general registers, 15 are available as index registers. Register 0 cannot be used as an index register; because a zero placed in the index register select field imples no indexing is to be performed.

Additionally, the architectural design provides 100% directly addressable memory, thus eliminating time-consuming design problems and wasted memory associated with paging and indirect addressing. Programmers can write straight-forward, simple, in-line code without concern for depleting base registers or exceeding page boundaries.

The 5/16 Processor features such enhancements as:

ASCII Programmer Console
Real Time Clock Input
Boot Strap Loader
Processor Self Test Feature
Turnkey Console Support

MEMORY

Main memory is built around read/write MOS modules or read-only memory (ROM) modules available in 8KB increments. The first 8KB or 16KB must be read/write. The remaining memory, up to the 64KB limit, can be read/write or read-only in several mixes. The MOS and ROM modules have a 600 nanosecond cycle time.

INSTRUCTION SET

The Model 5/16 instruction set consists of 100 individual instructions designed to provide the programmer with the means to write programs with a miminal effort. These are 100 separate and truly individual instructions. For example, "Load A" and "Load B" are not considered separate instructions. INTERDATA's tally of instructions is made exclusive of possible destination register specifications, or variation in the branch condition or index register select field. To add extra meaning to the variations in the branch condition field, the Assembler recognizes 14 additional extended branch mnemonics, thus bringing the total number of available discrete mnemonics to 114.

The Model 5/16 instruction set uses both 16-bit and 32-bit instruction formats. It permits operations between any two general registers, between a general register and memory, or between a general register and an "immediate" data constant contained in the instruction word. The instruction set includes a complete set of arithmetic and logical instructions and a complete set of conditional branch instructions that allow transfer of control to any instruction in memory. The byte processing instructions simplify the handling of byte strings and allow for more effective use of available memory. The input/output instructions permit operations between peripheral devices and general registers, or between peripheral devices and memory.

INPUT/OUTPUT SYSTEM

The 5/16 includes as standard the Micro I/O Bus and the INTERDATA Multiplexor Bus. This unique dual-channel I/O system allows for operation of standard INTERDATA supplied peripheral controllers over the Multiplexor Channel. The Micro I/O Bus allows the user to design peripheral controllers around new, low-cost interface chips available for use with most popular microprocessors. INTERDATA's own flexible Floppy Disc system is designed to interface directly to the Micro I/O Bus. Direct to memory transfers, conducted on an instruction cycle-stealing basis over the Micro I/O Bus, can operate up to 952,000 bytes per second, whereas, the standard Multiplexor Channel in the block mode operates up to 277,000 bytes per second. Both channels operate on a request-response basis, allowing simple and reliable peripheral controller design.

Under program control, automatic hardware interrupt vectoring of up to 256 devices is provided by means of the Interrupt Service Pointer (ISP) Table located in memory. This capability provides a separate driver in memory for each device in the system. INTERDATA offers a broad line of peripherals that are both program and interface compatible with all members of the INTERDATA family. INTERDATA also offers standard, low-cost interface modules to aid the user in interface design.

SOFTWARE

Standard software available for INTERDATA 16-bit Processors includes a symbolic Assembler, an interactive Text Editor, an interactive Debug package, an Extended FORTRAN IV Compiler, a FORTRAN V Level 1 Compiler, interactive Extended BASIC Level II, utility programs, and the OS/16 Multi-Task Operating System (OS/16 MT2).

In addition, the INTERDATA User's group called Interchange, has a large software library available to the 16-Bit Processor user.

PROCESSOR OPTIONS AND PERIPHERALS

The 5/16 Processor provides both a flexible and expandable hardware system to efficiently meet user needs.

Processor Options:

1. ASCII Programmer's Console. The basic display, modification, and control elements of a conventional operator's panel are based in the microcode supported ASCII Programmer's Console. An ASCII terminal such as a Carousel, CRT, or teletypewriter can be used for processor control. Human engineered keyboard commands permit quick and accurate entry and retrieval of data. From the ASCII Programmer's Console, the operator can examine and modify main memory locations, examine and modify the General Registers, examine and modify the Program Status Word, and initiate program execution.

2. Serial Input/Output Port. This option provides inexpensive 20 milliampere current loop interface to the ASCII Programmer Console device. The same device can also be used as the operating system command console or as a general input/output terminal.

3. Turnkey Console. This option provides switch control for power, initalization, and execution for the processor in dedicated systems.

4. Real Time Clock. The Real Time Clock input provides for connection of an external frequency source or event key. If enabled by PSW bits, an immediate interrupt is simulated from device number '07'.

5. Bootstrap Loader. The Bootstrap Loader provides a simple, single switch bootstrap load capability. It accomodates a 1KB PROM or a 2KB ROM for program loading or unattended restarts.

6. Read-Only replacement for up to 48KB of main memory provides program security and immediate system utility following power failure.

Peripheral Products include:

- Teletype (ASR 33, 35)
- Carousel (Model 15, 30, 35, 300)
- CRT (Non-editing, Editing, and Graphic)
- Flexible Floppy Disc System
- Digital Multiplexor System
- Mini Input/Output System (D/A and A/D)
- Real Time Analog System
- Clock Modules (Line Frequency and Precision Interval)
- Universal Logic Interfaces
- I/O Bus Switch
- Line Printers (60, 200, 300, and 600 LPM)
- Card Readers (400, 1000 CPM)
- Paper Tape Reader/Punch
- Cassette System
- Industry Compatible Magnetic Tapes
  - 9 Track, 45 IPS, 800 BPI
  - 9 Track, 45 IPS, 1600 BPI
  - 7 Track, 45 IPS, 200 BPI
  - 7 Track 45 IPS, 556 BPI
  - 9 Track, 45 IPS 800 BPI
- Quad Synchronous Adaptor (BISYNC or ZBID)

CHAPTER 2
SYSTEM DESCRIPTION

The Model 5/16, contained on a single printed circuit board, has the processor and up to 16KB of memory. Memory expansion beyond this is accomplished with a plug-on module. The optional serial current loop interface for use with an ASCII console device also connects to the processor board. An optional chassis with integral power supply provides space for up to six seven-inch interface boards or up to three fifteen-inch interface boards. The Model 5/16 packaging is consistent with INTERDATA standards for ruggedness, durability, and reliability.

The unique design characteristics of the INTERDATA 16 Bit Processor line allow for a fully integrated system with module expansion for additional memory or peripherals. Figure 2-1 shows how various elements of the Model 5/16 system are combined.

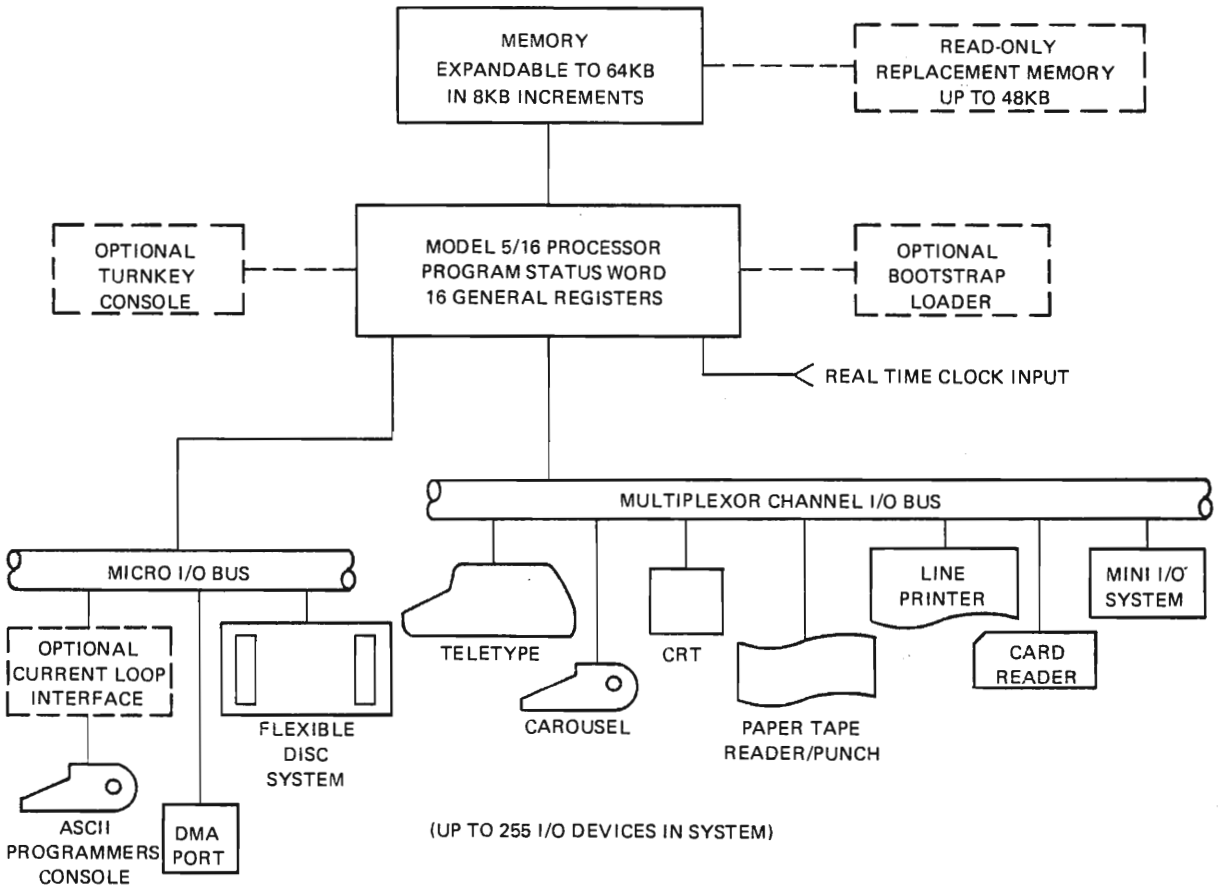


Figure 2-1. System Block Diagram

PROCESSOR

The processor controls activities in the system. It executes instructions in a specific sequence and performs arithmetic and logical functions. Included in the processor are:

Program Status Word Register
16 General Registers
Signed Multiply/Divide Hardware

Program Status Word

The Program Status Word (PSW), shown in Figure 2-2, defines the state of the processor at any given time.

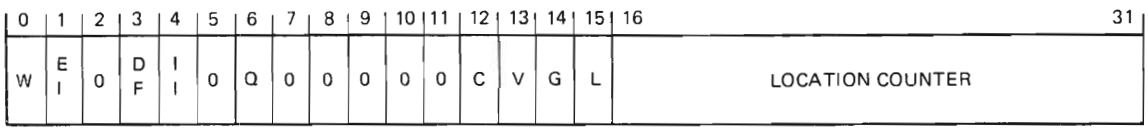


Figure 2-2. Program Status Word Format

Bits 0:15 are reserved for status information and for interrupt masks. Bits 16:31 contain the Location Counter. Unassigned PSW bits should not be used and should be zero. Status information and interrupt mask bits are defined as in Table 2-1.

TABLE 2-1. INTERPRETATION OF PSW BITS

MASK BIT	DESIGNATION	INTERPRETATION
0	WAIT STATE (W)	IF SET, THE PROCESSOR HALTS NORMAL PROGRAM EXECUTION AND ASSUMES AN IDLE STATE. IN THIS STATE, THE PROCESSOR STILL RESPONDS TO EXTERNAL AND IMMEDIATE INTERRUPTS IF ENABLED BY OTHER PSW BITS.
1	EXTERNAL INTERRUPT MASK (EI)	THIS BIT CONTROLS REQUESTS FOR SERVICE FROM DEVICES ON THE MICRO I/O BUS OR THE MULTIPLEXER CHANNEL I/O BUS. IF SET, THE PROCESSOR IS RESPONSIVE TO INTERRUPT REQUESTS. IF RESET, THESE REQUESTS REMAIN QUEUED AT THE INTERFACE BUT ARE IGNORED BY THE PROCESSOR.
2	NOT USED	
3	FIXED POINT DIVIDE FAULT INTERRUPT MASK (DF)	THIS BIT CONTROLS INTERRUPTS GENERATED IF A FIXED POINT DIVIDE OPERATION RESULTS IN QUOTIENT OVERFLOW OR IF DIVISION BY ZERO IS ATTEMPTED. IF SET, THE INTERRUPT IS TAKEN. IF RESET, THE INTERRUPT CONDITION IS IGNORED AND NOT QUEUED.
4	IMMEDIATE INTERRUPT MASK (II)	THIS BIT CONTROLS THE AUTOMATIC VECTORED IMMEDIATE INTERRUPT. IF THIS BIT AND BIT 1 ARE SET, IMMEDIATE INTERRUPTS ARE ENABLED. IF RESET, THE INTERRUPT DOES NOT OCCUR, BUT THE EXTERNAL INTERRUPT CAN STILL OCCUR IF ENABLED BY BIT 1. REFER TO CHAPTER 6.
5	NOT USED	
6	SYSTEM QUEUE SERVICE INTERRUPT MASK (Q)	THIS BIT CONTROLS OPERATION OF THE SYSTEM QUEUE INTERRUPT. IF SET AND THE QUEUE REQUIRES SERVICE, THE INTERRUPT IS TAKEN.
7-11	NOT USED	
12-15	CONDITION CODE (C,V,G,L)	THESE BITS, SET BY THE PROCESSOR, INDICATE THE RESULT OF THE CONCLUSION OF THOSE INSTRUCTIONS THAT UPDATE THE CONDITION CODE. THE GENERAL INTERPRETATION OF THESE BITS IS: BIT 12 : C – CARRY OR BORROW BIT 13 : V – OVERFLOW BIT 14 : G – GREATER THAN ZERO BIT 15 : L – LESS THAN ZERO

Processor Interrupts

Interrupt conditions cause the entire PSW to be replaced by a new PSW, thus breaking the usual flow of instruction execution. When an interrupt condition arises, the processor saves its current PSW in a memory location unique to the type of interrupt. That location is called the 'Old PSW'. The processor then loads a new PSW from a corresponding memory location and assumes the state specified by the new PSW.

General Registers

There are sixteen 16-bit general purpose registers numbered 0 through 15. None of these registers has a preset function and thus can be used by the programmer's discretion as accumulators or temporary storage of data. Registers 1 through 15 may be used as index registers. If register zero is specified as an index register, no indexing occurs.

Reserved Memory Locations

The following memory locations are reserved for interrupt pointers, PSWs, and system constants:

Location	Use
X'0000' - X'001F'	General Register Save Area for ASCII Console
X'0020' - X'0023'	Reserved (not used)
X'0024' X'0027'	Current PSW Save Area for ASCII Console
X'0028' - X'002F'	Reserved (not used)
X'0030' - X'0033'	Illegal Instruction Interrupt Old PSW
X'0034' - X'0037'	Illegal Instruction Interrupt New PSW
X'0038' - X'003F'	Reserved (not used)
X'0040' - X'0043'	External Interrupt Old PSW
X'0044' - X'0047'	External Interrupt New PSW
X'0048' - X'004B'	Fixed Point Divide Fault Old PSW
X'004C' - X'004F'	Fixed Point Divide Fault New PSW
X'0050' - X'007F'	Bootstrap Loader and Device Definition Table
X'0080' - X'0081'	System Queue Pointer
X'0082' - X'0085'	System Queue Service Interrupt Old PSW
X'0086' - X'0089'	System Queue Service Interrupt New PSW
X'008A' - X'0093'	Reserved (not used)
X'0094' - X'0095'	Supervisor Call Argument Pointer
X'0096' - X'0099'	Supervisor Call Interrupt Old PSW
X'009A' - X'009B'	Supervisor Call Interrupt New PSW Status
X'009C' - X'00BB'	Supervisor Call Interrupt New Location Counter Values
X'00BC' - X'00CF'	Reserved (not used)
X'00D0' - X'02CF'	Interrupt Service Pointer Table

These reserved locations play an important role in console support, interrupt processing and input/output processing. For a detailed description, refer to Chapters 6 and 7.

MEMORY

The Model 5/16 Processor may have 8KB or 16KB of read/write MOS memory on the processor card. Additional memory can be installed on one of two optional piggy-back cards that connect with the processor card. With either option, the total system memory space can be brought up to a 65,536 byte maximum.

Memory Expansion

The M51-104 memory expansion option allows installation of additional read/write MOS memory in 8KB increments. As much as 48KB of read/write MOS can be added to the M51-104 memory option. See Figure 2-3. Using the same M51-104 memory option, the uppermost 8KB of address space (addresses X'E000' through X'FFFF') may be replaced with Read-Only-Memory. This allows 8KB of non-volatile user program for those applications that require immediate system integrity following power fail situations.

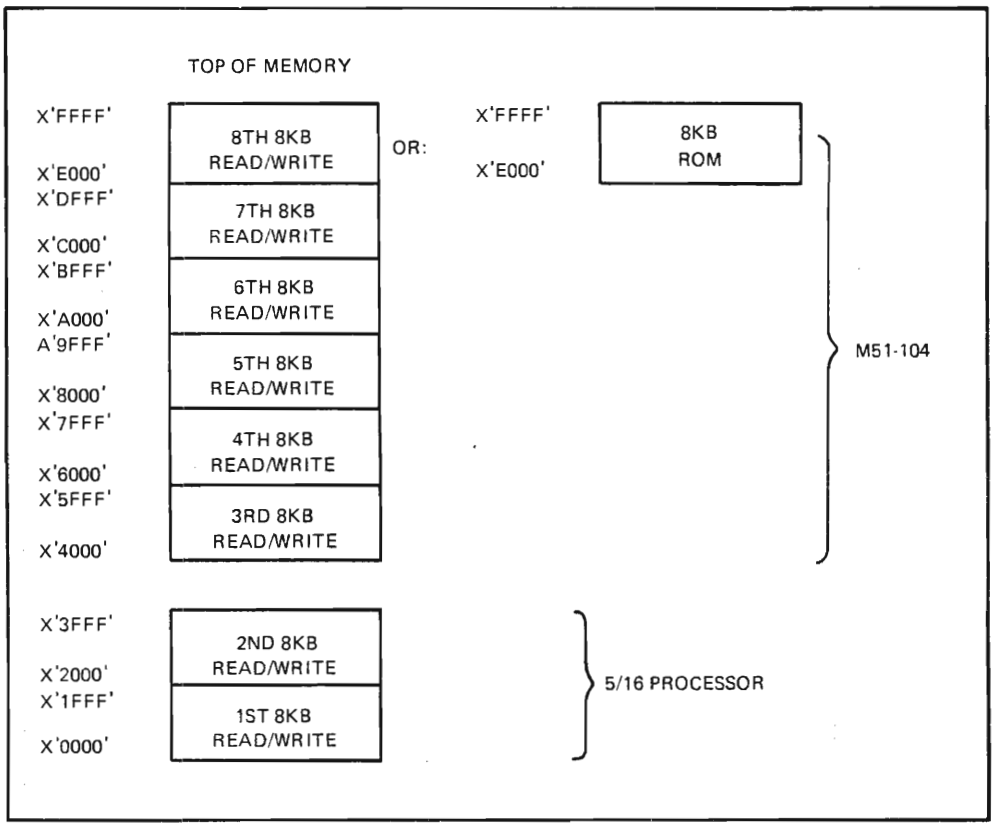


Figure 2-3. M51-104 Memory Expansion Option

NOTE

The 8KB of Read-Only-Memory comprises four 2K x 8 bit ROM devices (INTEL 2716 or equivalent). If only 4KB of non-volatile program is needed, 1K x 8 bit ROM devices can be substituted (INTEL 2708 or equivalent). The Read-Only segment will then represent addresses X'F000' through X'FFFF'. Addresses X'E000' through X'EFFF' become unavailable.

The M51-105 Memory expansion option allows adding up to 48KB of Read-Only Memory. No read/write memory can be added. The 8KB or 16KB of read/write MOS is still available on the basic processor card. See Figure 2-4. With this option, the first 8KB of Read-Only Memory must be installed in address space X'E000' through X'FFFF'.

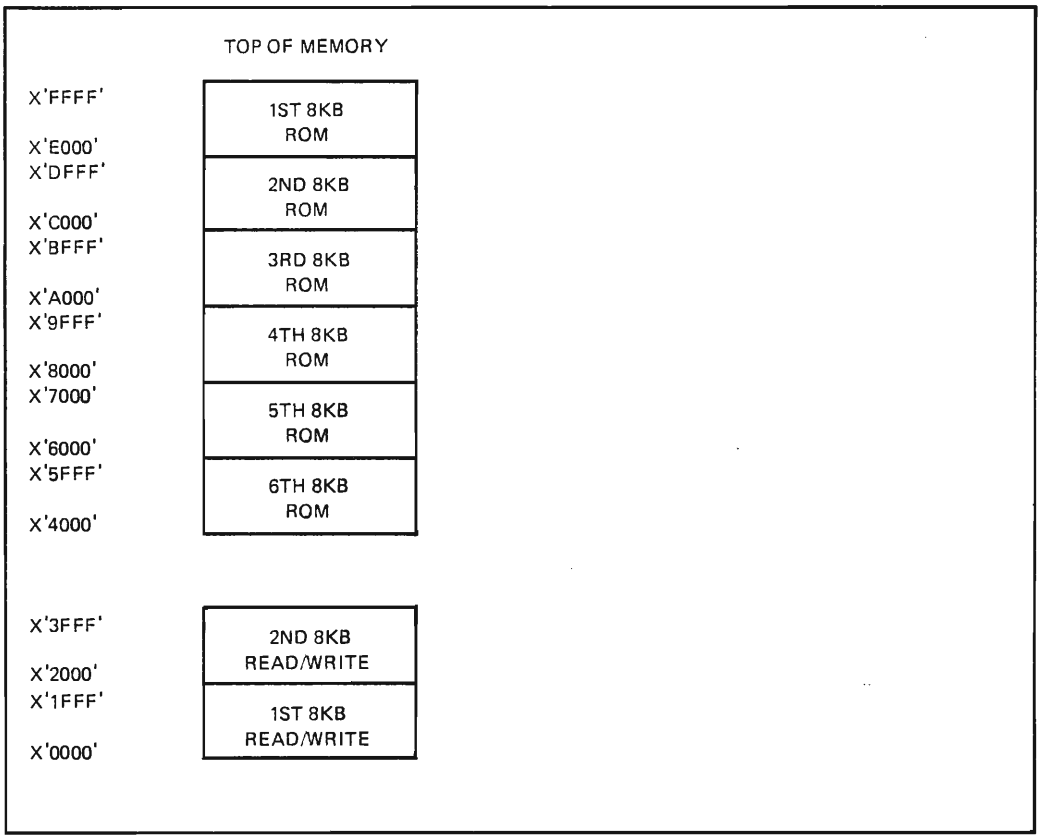


Figure 2-4. M51-105 Memory Expansion Option

Memory Addressable ROM

The presence of Memory Addressable ROM is indicated to the microprogram by a testable strap (ARST). On power-up or after an initialize sequence, if any amount of Memory Addressable ROM is equipped, the PSW is set to X'0000' and the instruction Location Counter is set to X'FFFC'. This is the address of the last fullword location in the Read-Only segment of memory. The instruction at that location must be a branch to the actual starting address of the user's fixed program in the ROM. Refer to the *Model 5/16 Maintenance Manual,* Publication Number 29-603, for details.

On power-up or after initialization, the microcoded diagnostic routine is executed and then user program execution begins at address X'FFFC'.

Implementation

The fixed program for implementation can be developed in 5/16 Assembly Language using standard programming techniques. After this, the customer is on his own for getting ROM devices built.

PROCESSOR OPERATIONS

The processor performs logical and fixed point arithmetic operations between the contents of two registers or between the contents of a register and a halfword from main memory. When the second operand is contained in memory, it may be part of the instruction itself (immediate operand), or it may be located in indexed storage.

DATA FORMATS

The processor performs logical and arithmetic operations on 8-bit bytes, 16-bit halfwords, or 32-bit fullwords. This data may represent a fixed point number or logical information.

Fixed Point Data

Fixed point arithmetic operands are 16-bit halfwords or 32-bit fullwords. In both of these formats, the most significant bit is the sign bit, and the remaining bits represent the magnitude. Positive quantities are expressed in true binary form with a sign bit of zero. Negative quantities are expressed in two's complement form with a sign bit of one. The numerical value of zero is represented with all bits zero.

Logical Data

Logical operations manipulate 8-bit bytes, 16-bit halfwords, or 32-bit fullwords. All bits participate in logical operations and the sign bit has no particular significance.

INSTRUCTION FORMATS

The INTERDATA instruction formats provide a concise method of representing required operations for easy interpretation by the processor. There are four basic formats, shown in Figure 2-5. The abbreviations used in the figure have the following meanings:

OP	Operation code
R1	First operand register
R2	Second operand register
N	A four bit immediate value
X2	Second operand single index register
A	Second operand 16-bit address
I	Second operand 16-bit immediate value

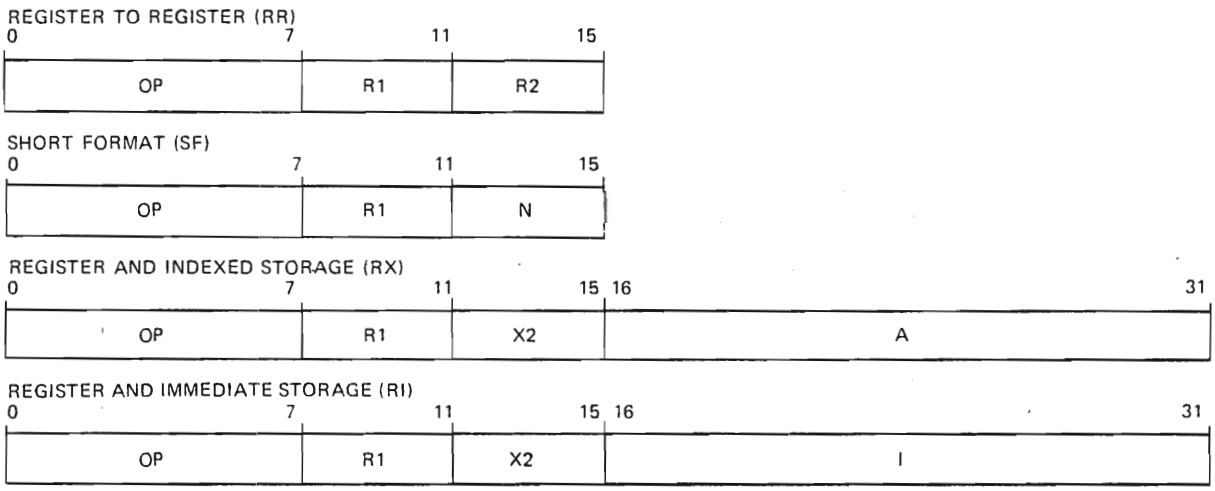


Figure 2-5. Instruction Formats

Many instructions may be expressed in two or more formats. This feature provides flexibility in data organization and instruction sequencing.

When working with the INTERDATA Common Assembler Language (CAL) assembler, it is not necessary to specify the instruction format explicitly. The assembler chooses the most economical format and supplies the required bits in the machine code.

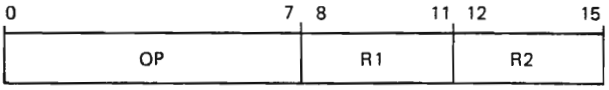
Branch Instruction Formats

The Branch instructions use the RR, SF, and the RX formats. However, in the Conditional Branch instructions, the R1 field does not specify a register. Instead, it contains a mask value (labeled M1 in the instruction descriptions), which is tested against the Condition Code. The INTERDATA CAL assembler provides a series of Extended Branch Mnemonics which make it possible to specify a Conditional Branch without specifying the mask value explicitly. For a summary of the Extended Branch Mnemonics, see Appendix 4.

Programming Examples

Each of the following programming examples refers to the sample assembly language program shown in Figure 2-6. Note the use of symbolic equates for general registers. Machine code generated and the result of each instruction are dependent upon the physical and logical placement of the instructions, respectively.

Register to Register (RR) Format

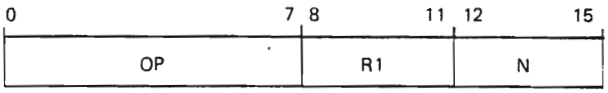


In this 16-bit format, Bits 0:7 contain the operation code. Bits 8:11 contain the R1 field, and Bits 12:15 contain the R2 field. In most RR instructions, the register specified by R1 contains the first operand, and the register specified by R2 contains the second operand. For example:

Machine Code	Label	Assembler Notation	
0865	RR	LHR	R6,R5

- 5 — Second Operand
- 6 — First Operand
- 08 — 'LHR' Instruction Op-Code (Load Halfword Register)

Short Form (SF) Format

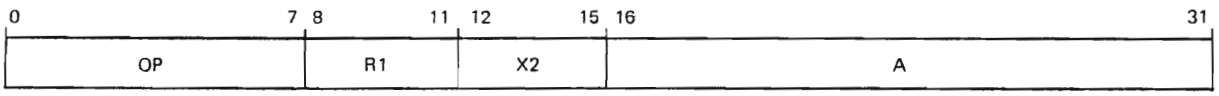


This 16-bit format provides space economy when working with small values. Bits 0:7 contain the operation code. Bits 8:11 contain the R1 field. Bits 12:15 contain the N field. In arithmetic and logical operations, the register specified by R1 contains the first operand. The N field contains a four bit immediate value (12:15) used as the second operand. For example:

Machine Code	Label	Assembler Notation	
245E	SF	LIS	R5,14

- E — Second Operand
- 5 — First Operand
- 24 — 'LIS' Instruction Op-Code (Load Immediate Short)

Register and Indexed Storage (RX) Format

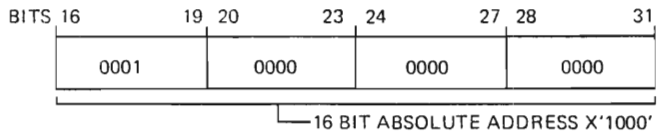


This is a 32-bit format in which Bits 0:7 contain the operation code, Bits 8:11 contain the R1 field, Bits 12:15 contain the X2 field, and Bits 16:31 contain the A field. In general, the register specified by R1 contains the first operand. The second operand is located in memory at the address obtained by adding the contents of the second operand index register, specified by X2, and the 16-bit absolute address contained in the A field. For example:

Machine Code	Label	Assembler Notation	
4050 1000	RX.EX1	STH	R5,X'1000'

- 1000 — Defines Second Operand Address
- 0 — No Index Register Specified
- 5 — First Operand
- 40 — 'STH' Instruction Op-Code (Store Halfword)

The Second Operand address is calculated as follows:

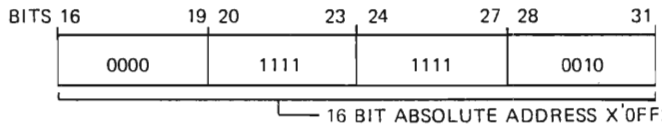


No indexing is specified. Therefore, the second operand address is X'1000'.

Machine Code	Label	Assembler Notation
4056 0FF2	RX.EX2	STH R5,X'0FF2'(R6)

- Defines Second Operand Address
- Register 6 to be used for Indexing
- First Operand
- 'STH' Instruction Op-Code (Store Halfword)

The Second Operand address is calculated as follows:



Second Operand Address

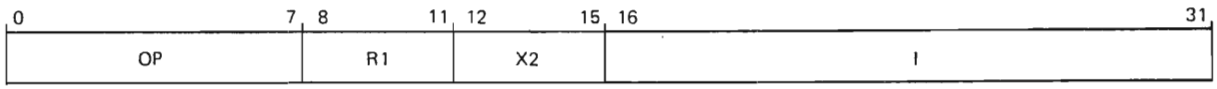
= contents of A field + contents of the Index Register 6 (see Figure 2-4)

= X'0FF2' + X'000E'

= X'1000'

The formation of the second operand address has absolutely no effect on setting the Condition Code.

Register and Immediate Storage (RI) Format



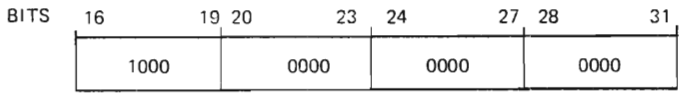
This format represents a 32-bit instruction word. Bits 0:7 contain the operation code. Bits 8:11 contain the R1 specification. Bits 12:15 contain the X2 specification. Bits 16:31 contain the 16 bit immediate value, I.

In this format, the register specified by R1 contains the first operand. The 16-bit effective second operand is obtained by adding together the 16-bit value contained in the I field, and the contents of the register specified by X2. For example:

Machine Code	Label	Assembler Notation
C890 8000	RI.EX1	LHI R9,X'8000'

- 16-Bit Immediate Value
- No Index Register Specified
- First Operand
- 'LHI' Instruction Op-Code (Load Halfword Immediate)

The Second Operand is calculated as follows:



Second Operand

= X'8000'

Machine Code	Label	Assembler Notation
C895 8000	RI.EX2	LHI R9,X'8000'(R5)

- 16-Bit Immediate Value
- Index Register 5 Specified
- First Operand
- 'LHI' Instruction Op-Code (Load Halfword Immediate)

The Second Operand is calculated as follows:

BITS	16 — 19	20 — 23	24 — 27	28 — 31
	1000	0000	0000	0000

Second Operand

= X'8000' + the contents of Index Register 5 (See Figure 2-6).

= X'8000' + X'000E'

= X'800E'

```
      16-BIT INSTRUCTION FORMAT EXAMPLES                                PAGE    1

PROG= EXAMPL       ASSEMBLED BY CAL 05-066R04(16-BIT)

                                    1          SCRAT
                                    2          TARGT  16
                                    3          WIDTH  120
                                    5  *
                0000 0005           6  R5      EQU    5                 GENERAL REGISTER 5
                0000 0006           7  R6      EQU    6                 GENERAL REGISTER 6
                0000 0009           8  R9      EQU    9                 GENERAL REGISTER 9
                                    9  *
                                   10  *
    0000R  245E                     11  SF      LIS    R5,14             (R5) = X'000E'
                                   12  *
    0002R  0865                     13  RR      LHR    R6,R5             (R6) = X'000E'
                                   14  *
    0004R  4050 1000                15  RX,EX1  STH    R5,X'1000'        (X'1000') = X'000E'
                                   16  *
    0008R  4056 0FF2                17  RX,EX2  STH    R5,X'0FF2'(R6)    (X'1000') = X'000E'
                                   18  *
    000CR  2302                     19          BS     RI,EX1
                                   20  *
    000ER  0000                     21  LOC1    DC     H'0'
                                   22  *
    0010R  C890 8000                23  RI,EX1  LHI    R9,X'8000'        (R9) = X'8000'
                                   24  *
    0014R  C895 8000                25  RI,EX2  LHI    R9,X'8000'(R5)    (R9) = X'800E'
                                   26  *
    0018R  4300 0000R               27          B      SF
                                   28  *
    001CR                          29          END
```

Memory Address — Machine Codes — Statement Number Assigned by the Assembler — Program Labels — Instruction Mnemonics — Operands — Comments

Figure 2-6. 16-Bit Instruction Format Examples

CHAPTER 3
LOGICAL OPERATIONS

The set of logical instructions provides a means for the manipulation of binary data. Many of the instructions grouped with the logical set may also be used in arithmetic and other operations. These instructions include loads, stores, compares, shifts, and list processing.

INTRODUCTION

The instruction repertoire has been grouped by function. The use and operation of each instruction is presented in the following format:

1. An instruction word chart for each instruction including: Mnemonic operation code, and first and second operand designations in the correct assembler format. The format type is designated by SF, RR, RI, or RX.

2. A description of instruction operation.

   An example of a diagrammatic representation of instruction operation is shown below.

   SIS: (R1) ← (R1) - N
   SHR: (R1) ← (R1) -- (R2)
   SH: (R1) ← (R1) -- [A + (X2)]
   SHI: (R1) ← (R1) -- I - (X2)

3. A chart illustrating the possible variations of the Condition Code in the Current Program Status Word as a result of performing the instructions: a one indicates set, a zero indicates reset. It is important to note that any instruction which changes the Condition Code can change all four bits. The conditions listed on the chart are only those conditions which are meaningful after a particular instruction. Other bits may be changed, but their condition is not meaningful, for example:

   Resulting Condition Code:

12	13	14	15	
C	V	G	L	
0	0	0	0	DIFFERENCE IS ZERO
X	X	0	1	DIFFERENCE IS LESS THAN ZERO
X	X	1	0	DIFFERENCE IS GREATER THAN ZERO
X	1	X	X	ARITHMETIC OVERFLOW
1	X	X	X	BORROW

4. A programming note to provide additional pertinent or clarifying information. All privileged instructions and those instructions which may cause a memory protect violation are so noted.

5. Examples.

The symbols and abbreviations used in the instruction diagrams are defined as follows:

( )
[ ] Parentheses or Brackets. Read as "the content of . . .."

⇆ Arrow. Read as "is replaced by . . ." or "replaces . . .."

A The 16-bit halfword address which is a part of the RX instruction.

I The 16-bit halfword immediate field of the RI instruction.

R1 The address of a General Register containing the first operand.

M1 Mask of four bits specifying Branch on Condition testing.

R2 The address of a General Register containing the second operand of an RR instruction.

X2 The address of a General Register containing an index value.

N The four-bit second operand used with Short Format Immediate and Short Format Branch instructions.

(0:7) A bit grouping within a byte, a halfword, or a fullword. Read as "0 through 7 inclusive."
(8:15) "Bits 8 through 15 inclusive", etc.
(16:31)

PSW Program Status Word of 32 bits containing the Status, Condition Code, and current instruction address.

CC Condition Code of four-bits contained in the PSW.

C Carry Bit contained in the Condition Code (Bit 12 of PSW).

V Overflow Bit contained in the Condition Code (Bit 13 of PSW).

G Greater Than Bit contained in the Condition Code (Bit 14 of PSW).

L Less Than Bit contained in the Condition Code (Bit 15 of PSW).

\+ Arithmetic operations – Add,
\- Subtract,
* Multiply,
/ and Divide respectively.

: Logical comparison, (e.g., R1:R2).

DATA FORMATS

Logical data may be organized as bytes, halfwords, or fullwords as shown in Figure 3-1.

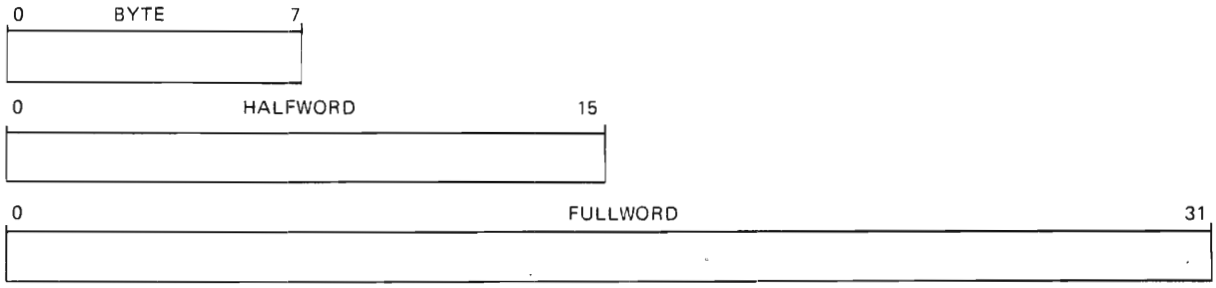


Figure 3-1. Logical Data

Boolean Operations

The Boolean operators AND, OR, and Exclusive OR (XOR) operate on halfword quantities. All bits in both operands participate individually. The Boolean functions are defined as follows:

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1 (logical product)

0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1 (logical sum)

0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1
1 XOR 1 = 0 (logical difference)

List Processing

The list processing instructions manipulate a circular list as defined in Figure 3-2.

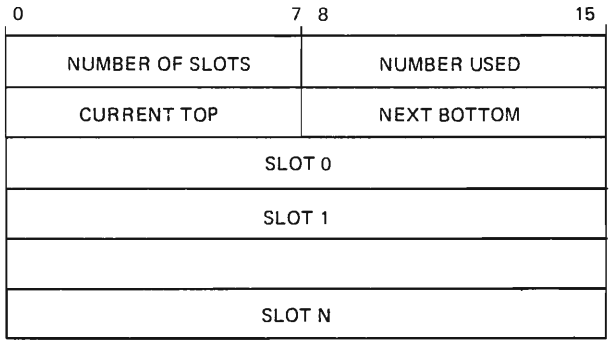


Figure 3-2. Circular List Definition

The first two halfwords contain the list parameters. Immediately following the parameter block is the list itself. The first halfword in the list is designated Slot 0. The remaining slots are designated 1, 2, 3, etc., up to a maximum slot number which is equal to the number in the list minus one. An absolute maximum of 255 halfword slots may be specified. (Slots are designated 0 through X'FE'.)

The first parameter byte indicates the number of slots (halfwords) in the entire list. The second parameter byte indicates the current number of slots being used. When this byte equals zero, the list is empty. When this byte equals the number of slots in the list, the list is full. Once initialized, this byte is maintained automatically. It is incremented when elements are added to the list and decremented when elements are removed.

The third and fourth bytes of the list parameter block specify the current top of the list and the next bottom of the list respectively. These pointers are also updated automatically. See Figure 3-3.

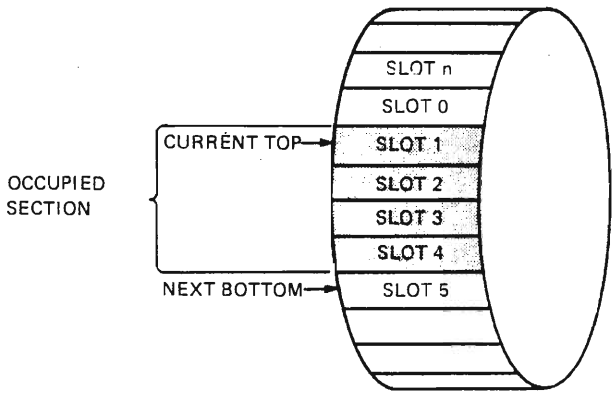


Figure 3-3. Circular List

LOGICAL INSTRUCTION FORMATS

The logical instructions use the Register to Register (RR), the Short Form (SF), the Register and Indexed Storage (RX), and the Register and Immediate Storage (RI) instruction formats.

LOGICAL INSTRUCTIONS

The instructions described in this section are:

LIS	Load Immediate Short
LCS	Load Complement Short
LH	Load Halfword
LHI	Load Halfword Immediate
LHR	Load Halfword Register
LM	Load Multiple
LB	Load Byte
LBR	Load Byte Register
EXBR	Exchange Byte Register
STH	Store Halfword
STM	Store Multiple
STB	Store Byte
STBR	Store Byte Register
CLHR	Compare Logical Halfword Register
CLH	Compare Logical Halfword
CLHI	Compare Logical Halfword Immediate
CLB	Compare Logical Byte
NHR	AND Halfword Register
NH	AND Halfword
NHI	AND Halfword Immediate
OHR	OR Halfword Register
OH	OR Halfword
OHI	OR Halfword Immediate
XHR	Exclusive OR Halfword Register
XH	Exclusive OR Halfword
XHI	Exclusive OR Halfword Immediate
THI	Test Halfword Immediate
SLL	Shift Left Logical
SLLS	Shift Left Logical Short
SRL	Shift Right Logical
SRLS	Shift Right Logical Short
SLHL	Shift Left Halfword Logical
SRHL	Shift Right Halfword Logical
RLL	Rotate Left Logical
RRL	Rotate Right Logical
ATL	Add to Top of List
ABL	Add to Bottom of List
RTL	Remove from Top of List
RBL	Remove from Bottom of List

INSTRUCTIONS

Load Halfword Register (LHR)
Load Immediate Short (LIS)
Load Complement Short (LCS)

Assembler Notation		Op-Code	Format
LHR	R1,R2	08	RR
LIS	R1,N	24	SF
LCS	R1,N	25	SF

Operation

The second operand replaces the contents of the register specified in R1.

LHR: (R1) ← (R2)
LIS: (R1) ← N
LCS: (R1) ← -N

Condition Code

C	V	G	L	
0	0	0	0	Value is ZERO
0	0	0	1	Value is not ZERO
0	0	1	0	Value is not ZERO

Programming Note

These instructions may be used to preset a register with an index value, load a register with the first operand for a subsequent arithmetic operation (e.g., add, multiply), or set the Condition Code for supplemental testing by a Branch on Condition instruction.

The Load Immediate Short instruction causes the four bit second operand to be expanded to a 16-bit halfword with high order bits forced to ZERO. This halfword replaces the contents of the register specified by R1.

The load complement short instruction causes the two's complement value of the four bit second operand to be expanded to a 16-bit halfword with high order bits forced to one. This value replaces the contents of the register specified by R1.

When the Load instructions operate on fixed point data, the Condition Code indicates ZERO (no flags), negative (L flag), or positive (G flag) value.

In the RR format, if R1 equals R2, the Load instruction functions as a test on the contents of the register.

Example: LCS

Assembler Notation		Machine Code	Comments
LCS	REG8,7	2587	LOAD -7 INTO REG8

Result of LCS Instruction

(REG8) = FFF9
Condition Code = 0001 (L = 1)

INSTRUCTIONS

Load Halfword (LH)
Load Halfword Immediate (LHI)

Assembler Notation		Op-Code	Format
LH	R1,A (X2)	48	RX
LHI	R1,I (X2)	C8	RI

Operation

The halfword second operand replaces the contents of the register specified by R1.

LH: (R1) ← [A + (X2)]
LHI: (R1) ← I + (X2)

Condition Code

C	V	G	L	
0	0	0	0	Value is ZERO
0	0	0	1	Value is not ZERO
0	0	1	0	Value is not ZERO

Programming Note

These instructions may be used to preset a register with an index value, load a register with the first operand for a subsequent arithmetic operation (e.g., add, multiply), or set the Condition Code for supplemental testing by a Branch on Condition instruction.

When the Load Halfword instructions operate on fixed point data, the Condition Code indicates zero (no flags), negative (L flag), or positive (G flag) value.

In the RX format, the second operand must be located on a halfword boundary.

INSTRUCTION

Load Multiple (LM)

Assembler Notation		Op-Code	Format
LM	R1,A (X2)	D1	RX

Operation

Successive registers, starting with the register specified by R1, are loaded from successive memory locations, starting with the location specified as the effective address of the second operand. Each register is loaded with a halfword from memory. The process stops when Register 15 has been loaded.

1. (R1) ← [A + (X2)]
2. R1: X'F'
   if R1 = X'F', then the instruction is finished.
   if R1 ≠ X'F', then:
3. R1 ← R1 + 1
4. A ← A+2, return to Step 1.

Condition Code

Unchanged

Programming Note

The second operand must be located on a halfword boundary.

Load Byte (LB)
Load Byte Register (LBR)

Assembler Notation		Op-Code	Format
LB	R1,A (X2)	D3	RX
LBR	R1,R2	93	RR

Operation

The eight-bit second operand replaces the least significant bits (Bits 8:15) of the register specified by R1. Bits 0:7 of the register are forced to ZERO.

LB: R1(8:15)←[A + (X2)]
R1(0:7) ←ZERO
LBR: R1(8:15)←R2 (8:15)
R1(0:7) ←ZERO

Condition Code

Unchanged

Programming Note

In the Load Byte Register instruction, the second operand is taken from the least significant eight bits (Bits 8:15) of the register specified by R2.

INSTRUCTION

Exchange Byte Register (EXBR)

Assembler Notation		Op-Code	Format
EXBR	R1,R2	94	RR

Operation

The two eight-bit bytes contained in the register specified by R2 are exchanged and loaded into the register specified by R1. The register specified by R2 is unchanged.

EXBR: R1 (0:7) ← R2 (8:15)
R1 (8:15) ← R2 (0:7)

Condition Code

Unchanged

Programming Note

R1 and R2 may specify the same register. In this case, the two bytes in the register specified by R2 are exchanged.

Example: EXBR

Assembler Notation		Machine Code	Comments
LHI	REG7,X'3C4D'	C870 3C4D	(REG7) = 3C4D
LHI	REG3,X'1234'	C830 1234	(REG3) = 1234
EXBR	REG7,REG3	9473	

Result of EXBR Instruction

(REG7) = 3412
(REG3) = 1234
Condition Code = Unchanged

INSTRUCTION

Store Halfword (STH)

Assembler Notation		Op-Code	Format
STH	R1,A (X2)	40	RX

Operation

The 16-Bit contents of the register specified by R1 replace the contents of the halfword memory location specified by the effective address of the second operand.

STH: (R1)⟶[A + (X2)]

Condition Code

Unchanged

Programming Note

The second operand location must be on a halfword boundary.

INSTRUCTION

Store Multiple (STM)

Assembler Notation		Op-Code	Format
STM	R1,A (X2)	D0	RX

Operation

The halfword contents of registers, starting with the register specified by R1, replace the contents of successive memory locations, starting with the location specified by the effective address of the second operand. The process stops when Register 15 has been stored.

STM:
1. (R1) ⟶ [A + (X2)]
2. R1: X'F'
   if R1 = X'F', then the instruction is finished.
   if R1 ≠ X'F', then:
3. R1 ⟵ R1 + 1
4. A ⟵ A + 2, return to Step 1.

Condition Code

Unchanged

Programming Note

The second operand location must be on a halfword boundary.

The Store Multiple (STM) instruction, in conjunction with the Load Multiple (LM) instruction, is an aid to subroutine execution. They permit the easy saving and restoring of the registers required by the subroutine. The Store Multiple instruction can be used upon entering the subroutine, and the Load Multiple can be the last instruction executed before returning from the subroutine.

INSTRUCTIONS

Store Byte (STB)
Store Byte Register (STBR)

Assembler Notation		Op-Code	Format
STB	R1,A (X2)	D2	RX
STBR	R1,R2	92	RR

Operation

The least significant eight bits (Bits 8:15) of the register specified by R1 are stored in the second operand location.

STB: [R1(8:15)] ⟶ [A + (X2)]
STBR: [R1(8:15)] ⟶ R2(8:15)

Condition Code

Unchanged

Programming Note

In the Store Byte Register instruction, the eight bit quantity is stored in Bits 8:15 of the register specified by R2. Bits 0:7 of the register are unchanged.

Example: STBR

Assembler Notation		Machine Code	Comments
LHI	REG4, X'7531'	C840 7531	(REG4) = 7531
LHI	REG3, X'8642'	C830 8642	(REG3) = 8642
.			
.			
.			
STBR	REG4, REG3	9243	

Result of STBR Instruction

(REG4) = 7531
(REG3) = 8631
Condition Code = Unchanged

INSTRUCTIONS

Compare Logical Halfword (CLH)
Compare Logical Halfword Register (CLHR)
Compare Logical Halfword Immediate (CLHI)

Assembler Notation		Op-Code	Format
CLH	R1,A(X2)	45	RX
CLHR	R1,R2	05	RR
CLHI	R1,I(X2)	C5	RI

Operation

The first operand, the contents of the register specified by R1, is compared logically to the second operand. The result is indicated by the Condition Code setting. Neither operand is changed.

CLH: (R1): [A + (X2)]
CLHR: (R1): (R2)
CLHI: (R1): I + (X2)

Condition Code

C	V	G	L	
0	X	0	0	First operand equal to second
1	X	0	1	First operand less than second
1	X	1	0	First operand less than second
0	X	0	1	First operand greater than second
0	X	1	0	First operand greater than second

Programming Note

In the RX format, the second operand must be located on a halfword boundary.

The state of the V flag is undefined.

It is meaningful to check the following condition code mask (M1) after a logical comparison:

Mask	True/False*	Inference
3	False	First operand equal to second
3	True	First operand not equal to second
8	False	First operand not less than second
8	True	First operand less than second

*Refer to Chapter 4, Branching, for True/False concept in branch instructions.

INSTRUCTION

Compare Logical Byte (CLB)

Assembler Notation		Op-Code	Format
CLB	R1,A(X2)	D4	RX

Operation

The byte quantity, contained in Bits 8:15 of the register specified by R1, is compared with the 8-bit second operand. The result is indicated by the Condition Code setting. Neither operand is changed.

CLB: R1(8:15) : [A + (X2)]

Condition Code

C	V	G	L	
0	0	0	0	First operand equal to second
1	0	0	1	First operand less than second
0	0	1	0	First operand greater than second

Programming Note

It is meaningful to check the following condition code mask (M1) after a logical comparison:

Mask	True/False*	Inference
3	False	First operand equal to second
3	True	First operand not equal to second
8	False	First operand not less than second
8	True	First operand less than second

*Refer to Chapter 4, Branching, for True/False concept in branch instructions.

AND Halfword (NH)
AND Halfword Register (NHR)
AND Halfword Immediate (NHI)

Assembler Notation		Op-Code	Format
NH	R1,A(X2)	44	RX
NHR	R1,R2	04	RR
NHI	R1,I(X2)	C4	RI

Operation

The logical product of the 16-bit second operand and the contents of the register specified by R1 replace the contents of the register specified by R1. The 16 bit logical product is formed on a bit-by-bit basis.

NHR: (R1) ← (R1) AND (R2)
NH: (R1) ← (R1) AND [A + (X2)]
NHI: (R1) ← (R1) AND I + (X2)

Condition Code

C	V	G	L	
0	0	0	0	Result is ZERO
0	0	0	1	Result is not ZERO
0	0	1	0	Result is not ZERO

Programming Note

In the RX formats, the second operand must be located on a halfword boundary.

When operating on fixed-point data, the Condition Code indicates ZERO (no flags), negative (L flag), or positive (G flag) result.

OR Halfword (OH)
OR Halfword Register (OHR)
OR Halfword Immediate (OHI)

Assembler Notation		Op-Code	Format
OH	R1,A(X2)	46	RX
OHR	R1,R2	06	RR
OHI	R1,I(X2)	C6	RI

Operation

The logical sum of the 16-bit second operand and the contents of the register specified by R1 replace the contents of the register specified by R1. The 16 bit sum is formed on a bit-by-bit basis.

OH: (R1) ← (R1) OR [A + (X2)]
OHR: (R1) ← (R1) OR (R2)
OHI: (R1) ← (R1) OR I + (X2)

Condition Code

C	V	G	L	
0	0	0	0	Result is ZERO
0	0	0	1	Result is not ZERO
0	0	1	0	Result is not ZERO

Programming Note

In the RX format, the second operand must be located on a halfword boundary.

When operating on fixed-point data, the Condition Code indicates ZERO (no flags), negative (L flag), or positive (G flag) result.

Exclusive OR Halfword (XH)
Exclusive OR Halfword Register (XHR)
Exclusive OR Halfword Immediate (XHI)

Assembler Notation		Op-Code	Format
XH	R1,A(X2)	47	RX
XHR	R1,R2	07	RR
XHI	R1,I(X2)	C7	RI

Operation

The logical difference of the 16-bit second operand and the contents of the register specified by R1 replace the contents of the register specified by R1. The 16-bit difference is formed on a bit-by-bit basis.

XH: (R1) ← (R1) XOR [A + (X2)]
XHR: (R1) ← (R1) XOR (R2)
XHI: (R1) ← (R1) XOR I + (X2)

Condition Code

C	V	G	L	
0	0	0	0	Result is ZERO
0	0	0	1	Result is not ZERO
0	0	1	0	Result is not ZERO

Programming Note

In the RX formats, the second operand must be located on a halfword boundary.

When operating on fixed-point data, the Condition Code indicates ZERO (no flags), negative (L flag), or positive (G flag) result.

INSTRUCTION

Test Halfword Immediate (THI)

Assembler Notation		Op-Code	Format
THI	R1,I(X2)	C3	RI

Operation

Each bit in the 16-bit second operand is logically ANDed with the corresponding bit contained in the register specified by R1. Neither operand is changed.

THI: (R1) AND I + (X2)

Condition Code

C	V	G	L	
0	0	0	0	Result is ZERO
0	0	0	1	Result is not ZERO
0	0	1	0	Result is not ZERO

Programming Note

When operating on fixed-point data, the Condition Code indicates ZERO (no flags), negative (L flag), or positive (G flag) result.

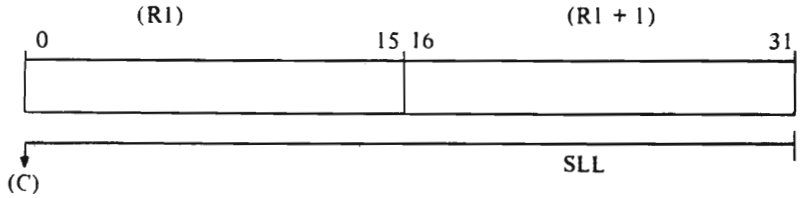
INSTRUCTION

Shift Left Logical (SLL)

Assembler Notation		Op-Code	Format
SLL	R1,I(X2)	ED	RI

Operation

In this instruction, the register specified by R1 and the register implied by the value of R1+1 are linked together to form a fullword operand. This operand is shifted left the number of binary places specified by the second operand. Bits shifted out of Position 0 in the register specified by R1 are shifted through the carry flag of the Condition Code, and then lost. The last bit shifted remains in the carry flag. Bits shifted from Position 0 of the second register move into Position 15 of the first. Zeros are moved into Position 15 of the second register.



Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is not ZERO
X	0	1	0	Result is not ZERO
1	0	X	X	Carry

Programming Note

The shift count is specified by the least significant five bits of the second operand.

The state of the C flag indicates the state of the last bit shifted out of Position 0 of register R1.

If the second operand specifies a shift of zero places, the Condition Code is set in accordance with the value contained in the registers. The state of the C flag is undefined.

The register specified by R1 must be an even numbered register.

When the registers R1 and R1+1 contain fixed point data, the L flag set indicates a negative result, the G flag set indicates a positive result.

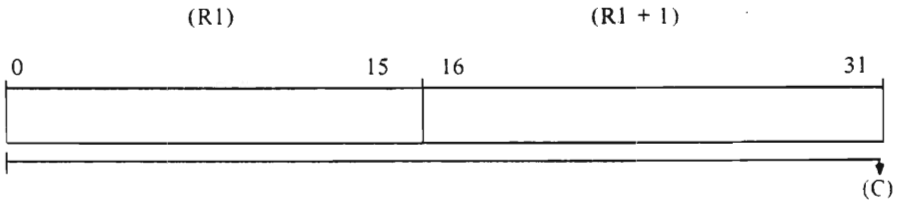
INSTRUCTION

Shift Right Logical (SRL)

Assembler Notation	Op-Code	Format
SRL R1,I(X2)	EC	RI

Operation

In this instruction, the register specified by R1 and the register implied by the value of R1+1 are linked together to form a fullword operand. This operand is shifted right the number of binary places specified by the second operand. Bits shifted out of Position 15 of the second register are shifted through the carry flag of the Condition Code, and then lost. The last bit shifted remains in the carry flag. Bits shifted from Position 15 of the first register move into Position 0 of the second. Zeros are moved into Position 0 of the first register.



Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is not ZERO
X	0	1	0	Result is not ZERO
1	0	X	X	Carry

Programming Note

The shift count is specified by the least significant five bits of the second operand.

The state of the C flag indicates the state of the last bit shifted out of Position 15 of register R1+1.

When the first operand contains fixed point data, the L flag set indicates a negative result, the G flag set indicates a positive result.

If the second operand specifies a shift of zero places, the Condition Code is set in accordance with the value contained in the registers. The state of the C flag is undefined.

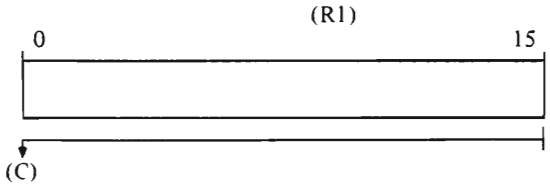
The register specified by R1 must be an even numbered register.

Shift Left Halfword Logical (SLHL)
Shift Left Logical Short (SLLS)

Assembler Notation		Op-Code	Format
SLHL	R1,I(X2)	CD	RI
SLLS	R1,N	91	SF

Operation

Bits 0:16 of the register specified by R1 are shifted left the number of places specified by the second operand. Bits shifted out of Position 0 are shifted through the carry flag and lost. The last bit shifted remains in the carry flag. Zeros are shifted into Position 15.



Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is not ZERO
X	0	1	0	Result is not ZERO
1	0	X	X	Carry

Programming Note

In the RI format, the shift count is specified by the least significant four bits of the second operand.

The state of the C flag indicates the state of the last bit shifted out of Position 0.

When the register specified by R1 contains fixed point data, the L flag set indicates a negative result, the G flag set indicates a positive result.

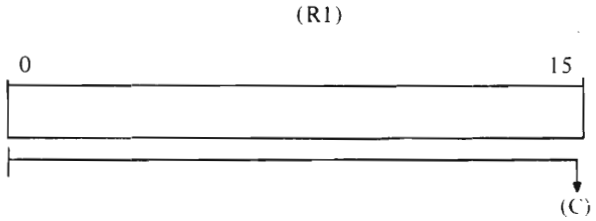
If the second operand specified a shift of zero places, the Condition Code is set in accordance with the value contained in the register. The state of the C flag is undefined.

Shift Right Halfword Logical (SRHL)
Shift Right Logical Short (SRLS)

Assembler Notation		Op-Code	Format
SRHL	R1,I(X2)	CC	RI
SRLS	R1,N	90	SF

Operation

Bits 0:15 of the register specified by R1 are shifted right the number of places specified by the second operand. Bits shifted out of Position 15 are shifted through the carry flag and lost. The last bit shifted remains in the carry flag. Zeros are shifted into Position 0.



Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is not ZERO
X	0	1	0	Result is not ZERO
1	0	X	X	Carry

Programming Note

In the RI format, the shift count is specified by the least significant four bits of the second operand.

The state of the C flag indicates the state of the last bit shifted out of Position 15.

When the register specified by R1 contains fixed point data, the L flag set indicates a negative result, the G flag set indicates a positive result.

If the second operand specifies a shift of zero places, the Condition Code is set in accordance with the value contained in the register. The state of the C flag is undefined.

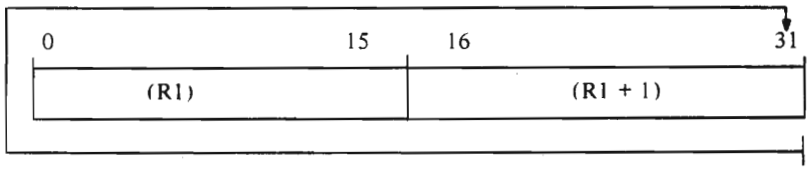
INSTRUCTION

Rotate Left Logical (RLL)

Assembler Notation		Op-Code	Format
RLL	R1,I(X2)	EB	RI

Operation

In this instruction, the register specified by R1 and the register implied by the value of R1+1 are linked together to form a fullword operand. This operand is rotated left the number of binary places specified by the second operand. Bits moved from Position 0 of the first register move into Position 15 of the second register.



Condition Code

C	V	G	L	
0	0	0	0	Result is ZERO
0	0	0	1	Result is not ZERO
0	0	1	0	Result is not ZERO

Programming Note

The register specified by R1 must be an even numbered register.

The shift count is specified by the least significant five bits of the second operand.

If the second operand specifies a shift of zero places, the Condition Code is set in accordance with the value contained in the registers.

When the register specified by R1 contains fixed point data, the L flag set indicates a negative result, the G flag set indicates a positive result.

Example: RLL

1.

Assembler Notation		Machine Code	Comments
LHI	REG8,X'5678'	C880 5678	(REG8) = 5678
LHI	REG9,X'9ABC'	C890 9ABC	(REG9) = 9ABC
RLL	REG8,X'0004'	EB80 0004	

Result of RLL Instruction

(REG8) = 6789 (REG9) = ABC5
Condition Code = 0010 (G = 1)

2.

Assembler Notation		Machine Code	Comments
LHI	REG8, X'8888'	C880 8888	(REG8) = 8888
LHI	REG9, X'8888'	C890 8888	(REG9) = 8888
RLL	REG8, X'0003'	EB80 0003	

Result of RLL Instruction

(REG9) = 4444 (REG8) = 4444
Condition Code = 0010 (G = 1)

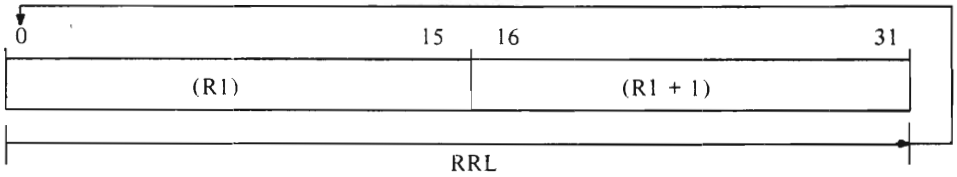
INSTRUCTION

Rotate Right Logical (RRL)

Assembler Notation		Op-Code	Format
RRL	R1,I(X2)	EA	RI

Operation

In this instruction, the register specified by R1 and the register implied by the value of R1+1 are linked together to form a fullword operand. This operand is rotated right the number of binary places specified by the second operand. Bits moved from Position 15 of the second register move into Position 0 of the first register.



Condition Code

C	V	G	L	
0	0	0	0	Result is ZERO
0	0	0	1	Result is not ZERO
0	0	1	0	Result is not ZERO

Programming Note

The register specified by R1 must be an even numbered register.

The shift count is specified by the least significant five bits of the second operand.

If the second operand specifies a shift of zero places, the Condition Code is set in accordance with the value contained in the registers.

When the register specified by R1 contains fixed point data, the L flag set indicates a negative result, the G flag set indicates a positive result.

Example: RRL

1.

Assembler Notation		Machine Code	Comments
LHI	REG4, X'1234'	C840 1234	(REG4) = 1234
LHI	REG5, X'5678'	C850 5678	(REG5) = 5678
RRL	REG4, X'0004'	EA40 0004	

Result of RRL Instruction

(REG4) = 8123 (REG5) = 4567
Condition Code = 0001 (L = 1)

2.

Assembler Notation		Machine Code	Comments
LHI	REG4, X'1111'	C840 1111	(REG4) = 1111
LHI	REG5, X'1111'	C850 1111	(REG5) = 1111
RRL	REG4, X'0001'	EA40 0001	

Result of RRL Operation

(REG4) = 8888 (REG5) = 8888
Condition Code = 0001 (L = 1)

INSTRUCTION

Add to Top of List (ATL)
Add to Bottom of List (ABL)

Assembler Notation		Op-Code	Format
ATL	R1,A(X2)	64	RX
ABL	R1,A(X2)	65	RX

Operation

The register specified by R1 contains the halfword element to be added to the list. The list is located in memory at the address specified by the second operand. The number of slots used tally is compared with the number of slots in the list. If the number of slots used equals the number of slots in the list, an overflow condition exists. The element is not added to the list and the overflow flag in the Condition Code is set. If the number of slots used tally is less than the number of slots in the list, it is incremented by one, the appropriate pointer (current top or next bottom) is changed, and the element is added to the list.

Condition Code

C	V	G	L	
0	0	0	0	Element added successfully
0	1	0	0	List overflow

Programming Note

These instructions manipulate circular lists as described in the introduction to this chapter.

The second operand location must be on a halfword boundary.

The Add to Top of List instruction manipulates the current top pointer in the list. If no overflow occurs, the current top pointer, which points to the last element added to the top of the list, is decremented by one and the element is inserted in the slot pointed to by the new current top pointer. If the current top pointer was zero on entering this instruction, the current top pointer is set to the maximum slot number in the list. This condition is referred to as list wrap.

The Add to Bottom of List instruction manipulates the next bottom pointer. If no overflow occurs, the element is inserted in the slot pointed to by the next bottom pointer, and the next bottom pointer is incremented by one. If the incremented next bottom pointer is greater than the maximum slot number in the list, the next bottom pointer is set to zero. This condition is referred to as list wrap.

Examples:

See examples for next instruction.

Remove from Top of List (RTL)
Remove from Bottom of List (RBL)

Assembler Notation		Op-Code	Format
RTL	R1,A(X2)	66	RX
RBL	R1,A(X2)	67	RX

Operation

The list is located at the address specified by the second operand. The halfword element removed from the list replaces the contents of the register specified by R1. If, at the start of the instruction execution, the number of slots used tally is ZERO, the list is already empty and the instruction terminates with the overflow flag set in the Condition Code. This condition is referred to as list underflow; in this case, (R1) is undefined. If underflow does not occur, the number of slots used tally is decremented by one, the appropriate pointer is changed, and the element is extracted and placed in the register specified by R1.

Condition Code

C	V	G	L	
0	0	0	0	List is now empty
0	0	1	0	List is not yet empty
0	1	0	0	List was already empty

Programming Note

These instructions manipulate circular lists as described in the introduction to this chapter.

The second operand location must be on a halfword boundary.

In the case of list underflow, the contents of the register specified by R1 are undefined.

The Remove from Top of List instruction manipulates the current top pointer. If no underflow occurs, the current top pointer points to the element to be extracted. The element is extracted, and placed in the register specified by R1. The current top pointer is incremented by one and compared to the maximum slot number. If the current top pointer is greater than the maximum slot number, the current top pointer is set to ZERO. This condition is referred to as list wrap.

The Remove from Bottom of List instruction manipulates the next bottom pointer. If no underflow occurs, and the next bottom pointer is ZERO, it is set to the maximum slot number (list wrap); otherwise, it is decremented by one, and the element now pointed to is extracted and placed in the register specified by R1.

Examples: List Instructions (ATL, ABL, RTL, RBL)

The following are examples of the use of the four list processing instructions.

The original list is normally set up as shown in Figure 3-4.

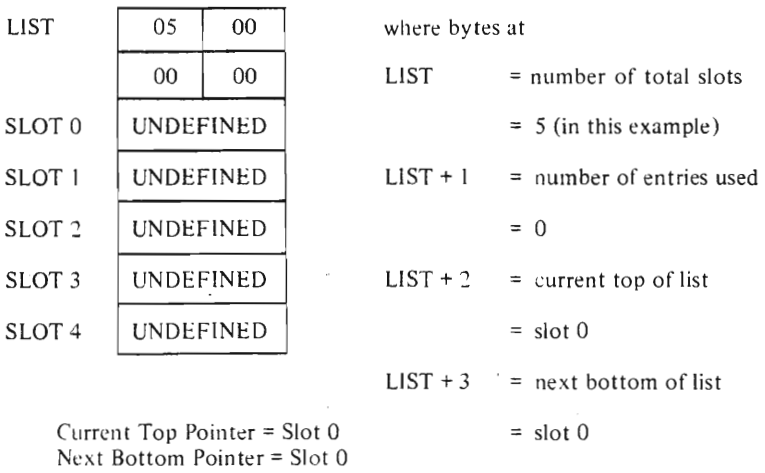


Figure 3-4. List Processing Instructions

Labels	Assembler Notation		Results and Comments
	LIS	REG0,0	
	STB	REG0,LIST+1	INITIALIZE NO. OF ENTRIES USED TO 0
	STH	REG0,LIST+2	INITIALIZE POINTERS TO 0
	LIS	REG1,1	REGISTERS 1 THRU 6 CONTAIN
	LIS	REG2,2	1 THRU 6 RESPECTIVELY
	LIS	REG3,3	
	LIS	REG4,4	
	LIS	REG5,5	
	LIS	REG6,6	
	STB	REG5,LIST	TOTAL NO. OF ENTRIES = 5

```
REF1    ATL     REG1  LIST
```

LIST	05	01
	04	00
SLOT 0	UNDEFINED	
SLOT 1	UNDEFINED	
SLOT 2	UNDEFINED	
SLOT 3	UNDEFINED	
SLOT 4	0001	

Condition Code = 0000
Current Top Pointer = Slot 4
Next Bottom Pointer = Slot 0

```
REF2    ATL     REG2, LIST
```

LIST	05	02
	03	00
SLOT 0	UNDEFINED	
SLOT 1	UNDEFINED	
SLOT 2	UNDEFINED	
SLOT 3	0002	
SLOT 4	0001	

Condition Code = 0000
Current Top Pointer = Slot 3
Next Bottom Pointer = Slot 0

```
REF3    ATL    REG3, LIST
```

LIST	05	03
	02	00
SLOT 0	UNDEFINED	
SLOT 1	UNDEFINED	
SLOT 2	0003	
SLOT 3	0002	
SLOT 4	0001	

Condition Code = 0000
Current Top Pointer = Slot 2
Next Bottom Pointer = Slot 0

```
REF4    ABL    REG4, LIST
```

LIST	05	04
	02	01
SLOT 0	0004	
SLOT 1	UNDEFINED	
SLOT 2	0003	
SLOT 3	0002	
SLOT 4	0001	

Condition Code = 0000
Current Top Pointer = Slot 2
Next Bottom Pointer = Slot 1

```
REF5    ABL    REG5, LIST
```

LIST	05	05
	02	02
SLOT 0	0004	
SLOT 1	0005	
SLOT 2	0003	
SLOT 3	0002	
SLOT 4	0001	

Condition Code = 0000
Current Top Pointer = Slot 2
Next Bottom Pointer = Slot 2

```
REF6    ABL    REG6, LIST
```

LIST	05	05	
	02	02	
SLOT 0	0004		
SLOT 1	0005		
SLOT 2	0003		
SLOT 3	0002		
SLOT 4	0001		

Condition Code = 0100
Condition Code = 0100 (List Overflow)
Next Bottom Pointer = Slot 2

```
REF7    RTL    REG7, LIST
```

LIST	05	04	
	03	02	
SLOT 0	0004		
SLOT 1	0005		
SLOT 2 X	0003		
SLOT 3	0002		
SLOT 4	0001		

(REG 7) = 0003
Condition Code = 0010
Current Top Pointer = Slot 3
Next Bottom Pointer = Slot 2

```
REF8    RBL    REG8, LIST
```

LIST	05	03	
	03	01	
SLOT 0	0004		
SLOT 1 X	0005		
SLOT 2 X	0003		
SLOT 3	0002		
SLOT 4	0001		

(REG 8) = 0005
Condition Code = 0010
Current Top Pointer = Slot 3
Next Bottom Pointer = Slot 1

NOTE

X = Entry removed from list, and is not accessible through further manipulation of list instructions.

REF9 RTL REG9, LIST

LIST		05	02
		04	01
SLOT 0		0004	
SLOT 1	X	0005	
SLOT 2	X	0003	
SLOT 3	X	0002	
SLOT 4		0001	

(REG 9) = 0002
Condition Code = 0010
Current Top Pointer = Slot 4
Next Bottom Pointer = Slot 1

REF10 RBL REG10, LIST

LIST		05	01
		04	00
SLOT 0	X	0004	
SLOT 1	X	0005	
SLOT 2	X	0003	
SLOT 3	X	0002	
SLOT 4		0001	

(REG 10) = 0004
Condition Code = 0010
Current Top Pointer = Slot 4
Next Bottom Pointer = Slot 0

REF11 RTL REG11, LIST

LIST		05	00
		00	00
SLOT 0	X	0004	
SLOT 1	X	0005	
SLOT 2	X	0003	
SLOT 3	X	0002	
SLOT 4	X	0001	

(REG 11) = 0001
Condition Code = 0000 (List is now empty)
Current Top Pointer = Slot 0
Next Bottom Pointer = Slot 0

NOTE

X =Entry removed from list, and is not accessible through further manipulation of list instructions.

REF12 RTL REG12, LIST

LIST		05	00
		00	00
SLOT 0	X	0004	
SLOT 1	X	0005	
SLOT 2	X	0003	
SLOT 3	X	0002	
SLOT 4	X	0001	

(REG 12) = UNDEFINED
Condition Code = 0100 (List Was Already Empty)
Current Top Pointer = Slot 0
Next Bottom Pointer = Slot 0

NOTE

X = Entry removed from list, and is not accessible through further manipulation of list instructions.

CHAPTER 4
BRANCHING

In normal operations, the Processor executes instructions in sequential order. The Branch instructions allow this sequential mode of operation to be varied, so that programs can loop, transfer control to subroutines, or make decisions based on the results of previous operations.

OPERATIONS

The second operand in Branch instructions is the address of the memory location to which control is transferred. The address may be contained in a register, or it may be specified in the instruction as the second operand address.

Decision Making

The Conditional Branch instructions permit the program to make decisions based on previous results. In these instructions, the R1 field contains a four bit mask, M1, which is tested against the Condition Code. The result of the test determines whether the branch is taken, or the next sequential instruction is executed.

The following examples show current Condition Code, mask specified in a branch instruction, and the result of the test on which a branch or no branch decision is made.

Current Condition Code	Mask (M1)	Result of Test	(True/False)
0000	0010	0000	(False)
0001	1010	0000	(False)
1001	1000	1000	(True)
0100	0100	0100	(True)
1010	0010	0010	(True)
0010	0011	0010	(True)
0010	0000	0000	(False)

Subroutine Linkage

The Branch and Link instructions allow branching to subroutines in such a way that a return address is passed to the subroutine. In these instructions, the address of the instruction immediately following the Branch instruction is saved in the register specified by R1.

BRANCH INSTRUCTION FORMATS

The Branch instructions use the Register to Register (RR), the Short Form (SF), and the Register and Indexed Storage (RX) format.

BRANCH INSTRUCTIONS

The instructions described in this section are:

BFC	Branch on False Condition
BFCR	Branch on False Condition Register
BFBS	Branch on False Condition Backward Short
BFFS	Branch on False Condition Forward Short
BTC	Branch on True Condition
BTCR	Branch on True Condition Register
BTBS	Branch on True Condition Backward Short
BTFS	Branch on True Condition Forward Short
BAL	Branch and Link
BALR	Branch and Link Register
BXLE	Branch on Index Low or Equal
BXH	Branch on Index High

INSTRUCTIONS

Branch on True Condition (BTC)
Branch on True Condition Register (BTCR)
Branch on True Condition Backward Short (BTBS)
Branch on True Condition Forward Short (BTFS)

Assembler Notation		Op-Code	Format
BTC	M1,A(X2)	42	RX
BTCR	M1,R2	02	RR
BTBS	M1,N	20	SF
BTFS	M1,N	21	SF

Operation

The Condition Code of the Program Status Word is tested for the conditions specified by the mask field, M1. If any of the conditions tested are found to be true, a branch is executed to the second operand location. If none of the conditions tested is found to be true, the next sequential instruction is executed.

Tested Condition True:

BTC: [PSW (16:31)] ← A + (X2)
BTCR: [PSW (16:31)] ← (R2)
BTBS: [PSW (16:31)] ← [PSW (16:31)] -2N
BTFS: [PSW (16:31)] ← [PSW (16:31)] +2N

Tested Condition False:

BTC: [PSW (16:31)] ← [PSW (16:31)] +4

BTBS:
BTFS: } [PSW (16:31)] ← [PSW (16:31)] +2
BTCR:

Condition Code

Unchanged

Programming Note

In the RR format, the branch address is contained in the register specified by R2.

In the SF format, the N field contains the number of *halfwords* to be added or subtracted from the current Location Counter to obtain the branch address.

In the RR and RX formats, the branch address must be located on a halfword boundary.

Example: BTC

Assembler Notation		Machine Code	Comments
LH	R1,X 100	4810 0100	Load halfword (X'1234') located at X'100'. Condition Code is set to CVGL = 0010. Mask is 3, i.e., M1 = 0011. Perform logical AND between CVGL and M1, i.e., 0010 AND 0011. The result is 0010, i.e., true; therefore, a branch is taken to LOC.
BTC	3, LOC	4230 ABC0	

Branch on False Condition (BFC)
Branch on False Condition Register (BFCR)
Branch on False Condition Backward Short (BFBS)
Branch on False Condition Forward Short (BFFS)

Assembler Notation		Op-Code	Format
BFC	M1,A(X2)	43	RX
BFCR	M1,R2	03	RR
BFBS	M1,N	22	SF
BFFS	M1,N	23	SF

Operation

The Condition Code of the Program Status Word is tested for the conditions specified in the mask field, M1. If all conditions tested are found to be false, a branch is executed to the second operand location. If any of the conditions tested is found to be true, the next sequential instruction is executed.

Tested Condition False

BFC: [PSW (16:31)] ← A+(X2)
BFCR: [PSW (16:31)] ← (R2)
BFBS: [PSW (16:31)] ← [PSW (16:31)] -2N
BFFS: [PSW (16:31)] ← [PSW (16:31)] +2N

Tested Condition True

BFC: [PSW (16:31)] ← [PSW (16:31)] +4

BFCR:
BFBS: } [PSW (16:31)] ← [PSW (16:31)] +2
BFFS:

Condition Code

Unchanged

Programming Note

In the RR format, the branch address is contained in the register specified by R2.

In the SF format, the N field contains the number of *halfwords* to be added to or subtracted from the current Location Counter to obtain the branch address.

In the RR and RX formats, the branch address must be located on a halfword boundary.

Branch on False Condition with a mask of 0 is an Unconditional Branch.

Example: BFC

Assembler Notation		Machine Code	Comments
LCS	R1,2	2512	(R1) = X'FFFE'. Condition Code, CVGL = 0001 Mask is 1001. Perform logical AND between mask and CVGL, i.e., 1001 AND 0001. The result is 0001, i.e., true, therefore, a branch is not taken to LOC.
BFC	9, LOC	4390 ABC0	

Branch and Link (BAL)
Branch and Link Register (BALR)

Assembler Notation		Op-Code	Format
BAL	R1,A(X2)	41	RX
BALR	R1,R2	01	RR

Operation

The address of the next sequential instruction is saved in the register specified by R1, and a branch is taken to the second operand address.

BAL: R1 ← [PSW (16:31)] + 4
[PSW (16:31)] ← A + (X2)

BALR: R1 ← [PSW (16:31)] + 2
[PSW (16:31)] ← (R2)

Condition Code

Unchanged

Programming Note

The second operand location must be on a halfword boundary.

The branch address is calculated before the register specified by R1 is changed. R1 may specify the same register as X2 or R2.

Example: BAL

The following example illustrates the use of the BAL instruction. The instruction causes control to be transferred to a subroutine called SUBROUT. After completion of the subroutine, the linking register is used to branch back to the next sequential instruction after the BAL, i.e., the instruction labeled RETURN.

	Labels	Assembler Notation		Comments
	BEGIN	BAL	REG4,SUBROUT	TRANSFER TO SUBROUT
MAIN	RETURN	XHR	R6,R6	
PROG		STH	R6,LAB+4	
		:		
	SUBROUT	LH	R8,LOC	THE RETURN ADDRESS OF THE SUBROUTINE IS IN REG4
SUBROUTINE		AHI	R8,10	
		:		
	RTNEND	BR	REG4	RETURN TO XHR INST

NOTE

Within the subroutine, the linking register (REG4 in the example) should not be used unless stored and reloaded within the subroutine.

Result of BAL Instruction

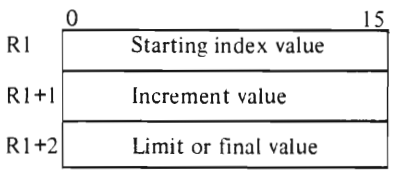
Condition Code = Unchanged

INSTRUCTION

Branch on Index Low or Equal (BXLE)

Assembler Notation		Op-Code	Format
BXLE	R1,A(X2)	C1	RX

Set Up



Prior to execution of this instruction, the register specified by R1 must contain a starting index value. The register specified by R1+1 must contain an increment value. The register specified by R1+2 must contain a comparand (limit or final value). All values may be signed.

Operation

Execution of this instruction causes the increment value to be added to the index value. The result is logically compared to the limit or final value. If the index value is less than or equal to the limit value, a branch is executed to the second operand location. If the index value is greater than the limit value, the next sequential instruction is executed.

BXLE: (R1) ← (R1) + (R1+1)
(R1): (R1+2)
If (R1) ≤ (R1+2), then [PSW (16:31)] ← A + (X2)
If (R1) > (R1+2), then [PSW (16:31)] ← [PSW (16:31)] + 4

Condition Code

Unchanged

Programming Note

The incremented index value replaces the contents of the register specified by R1.

The register specified by R1 must not be greater than 13.

The second operand location must be on a halfword boundary.

The branch address is calculated before incrementing the starting index value contained in the register specified by R1.

The register specified by R1 may be the same as X2.

Example: BXLE

Transfer 10 bytes in memory starting at Memory Location Labelled BUF0 to memory location labelled BUF1.

Labels	Assembler Notation		Comments
	LIS	REG3,0	(REG 3) = STARTING INDEX VALUE = 0
	LIS	REG4,1	(REG 4) = INCREMENT VALUE = 1
	LIS	R5,9	(REG 5) = FINAL VALUE = 9
AGAIN	LB	REG0, BUF0(R3)	(REG 0) = 1 BYTE FROM BUF0
	STB	REG0, BUF1(R3)	COPY 1 BYTE TO BUF1
	BXLE	R3,AGAIN	IF (REG 3) = (REG 5), DONE
	.		
	.		
	.		
BUF0	DS	10	
BUF1	DS	10	

Result of BXLE Instruction

Condition Code = Unchanged by BXLE Instruction
(REG1) = 000A
(REG2) = 0001
(REG3) = 0009

INSTRUCTION

Branch on Index High (BXH)

Assembler Notation		Op-Code	Format
BXH	R1,A(X2)	C0	RX

Set Up

	0	15
R1	Starting index value	
R1+1	Increment value	
R1+2	Limit or final value	

Prior to execution of this instruction, the register specified by R1 must contain a starting index value. The register specified by R1+1 must contain an increment value. The register specified by R1+2 must contain a comparand (limit or final value). All values may be signed.

Operation

Execution of this instruction causes the increment value to be added to the index value. The result is logically compared to the limit or final value. If the index value is greater than the limit value, a branch is executed to the second operand location. If the index value is equal to or less than the limit value, the next sequential instruction is executed.

BXH: (R1) ← (R1) + (R1 + 1)
(R1): (R1 + 2)
if (R1) ≤ (R1 + 2), then [PSW (16:31)] ← [PSW(16:31)] + 4
if (R1) > (R1 + 2), then [PSW (16:31)] ← A + (X2)

Condition Code

Unchanged

Programming Note

The incremented index value replaces the contents of the register specified by R1.

The second operand location must be on a halfword boundary.

The branch address is calculated before incrementing the starting index value contained in the register specified by R1.

The register specified by R1 may be the same as X2.

The register specified by R1 must not be greater than 13.

Example: BXH

The following example shows how to set up a counter (1 - 9) using the BXH instruction.

Label	Assembler Notation		Comment
	LIS	REG1,1	(REG 1) = 0001 (INDEX)
	LIS	REG2,1	(REG 2) = 0001 (INCREMENT)
	LIS	REG3,9	(REG 3) = 0009 (COMPARAND)
BEGIN	BXH	REG1, LABEL	COMPARE INDEX WITH COMPARAND
	LH	R6,COUNT	
	.		
	.		
	.		
	B	BEGIN	BRANCH TO BXH INSTRUCTION
LABEL	LH	R8,RTN	EXIT FROM BXH
	ST	R8,MEM	

Result of BXH Instruction

The code between the instructions labelled BEGIN and LABEL is executed 9 times.

Condition Code = Unchanged by BXH instruction
(REG1) = 000A
(REG2) = 0001
(REG3) = 0009

EXTENDED BRANCH MNEMONICS

The CAL Assembler supports 14 extended branch mnemonics that generate the branch op-code (true or false conditional) and the condition code mask required. The programmer must supply the second operand address (symbolic or absolute). In the case of short format (SF) branch instructions, the second operand branch address must be within ± 15 halfwords of the current location counter. The CAL Assembler determines the backward or forward relationship of the second operand address and generates the appropriate operation code.

Examples of extended branch mnemonic:

```
                .
                .
                .
                LH      R5,LOOP1
                BNZ     LOERR
LAP             SRLS    R6,1
                BNCS    LAP
                BS      CONTIN
LOERR           LIS     R6,0
ERROR1          AIS     R6,1
                SIS     R5,4
                BPS     ERROR1
                SIS     R8,1
                BO      ERROR2
CONTIN          LH      R1,TIME
                .
                .
                .
```

Appendix 4 lists the extended branch mnemonics and the proper operand form to be used with each mnemonic. The actual machine code generated is also listed.

The instructions described in this section are:

BC	Branch on Carry	BP	Branch on Plus
BCR	Branch on Carry Register	BPR	Branch on Plus Register
BCS	Branch on Carry Short	BPS	Branch on Plus Short
BNC	Branch on No Carry	BNP	Branch on Not Plus
BNCR	Branch on No Carry Register	BNPR	Branch on Not Plus Register
BNCS	Branch on No Carry Short	BNPS	Branch on Not Plus Short
BE	Branch on Equal	BO	Branch on Overflow
BER	Branch on Equal Register	BOR	Branch on Overflow Register
BES	Branch on Equal Short	BOS	Branch on Overflow Short
BNE	Branch on Not Equal	BNO	Branch on No Overflow
BNER	Branch on Not Equal Register	BNOR	Branch on No Overflow Register
BNES	Branch on Not Equal Short	BNOS	Branch on No Overflow Short
BL	Branch on Low	BZ	Branch on Zero
BLR	Branch on Low Register	BZR	Branch on Zero Register
BLS	Branch on Low Short	BZS	Branch on Zero Short
BNL	Branch on Not Low	BNZ	Branch on Not Zero
BNLR	Branch on Not Low Register	BNZR	Branch on Not Zero Register
BNLS	Branch on Not Low Short	BNZS	Branch on Not Zero Short
BM	Branch on Minus	B	Branch (Unconditional)
BMR	Branch on Minus Register	BR	Branch Register (Unconditional)
BMS	Branch on Minus Short	BS	Branch Short (Unconditional)
BNM	Branch on Not Minus		
BNMR	Branch on Not Minus Register	NOP	No Operation
BNMS	Branch on Not Minus Short	NOPR	No Operation Register

INSTRUCTION

Branch on Carry (BC)
Branch on Carry Register (BCR)
Branch on Carry Short (BCS)

Assembler Notation		Op-Code + M1	Format
BC	A(X2)	428	RX
BCR	R2	028	RR
BCS	A	208 (Backward) 218 (Forward)	SF

Operation

If the Carry (C) flag in the Condition Code is set, a branch is executed to the second operand location. If the Carry flag is not set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

Example: BCS

Assembler Notation			Machine Code	Comments
SHIFT	SLLS	R9,1	9191	Register 9 is shifted left until the first zero bit is shifted out (left).
	BCS	SHIFT	2081	

INSTRUCTION

Branch on No Carry (BNC)
Branch on No Carry Register (BNCR)
Branch on No Carry Short (BNCS)

Assembler Notation		Op-Code + M1	Format
BNC	A(X2)	438	RX
BNCR	R2	038	RR
BNCS	A	228 (Backward) 238 (Forward)	SF

Operation

If the Carry (C) flag in the Condition Code is not set, a branch is executed to the second operand location. If the Carry flag is set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Equal (BE)
Branch on Equal Register (BER)
Branch on Equal Short (BES)

Assembler Notation		Op-Code + M1	Format
BE	A(X2)	433	RX
BER	R2	033	RR
BES	A	223 (Backward) 233 (Forward)	SF

Operation

If the G flag and the L flag are both reset in the Condition Code, a branch is executed to the second operand location. If either flag is set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

Example: BE

Assembler Notation		Machine Code	Comments
CLHI	R4,X'23'	C540 0023	If R4 contains X'23', a branch is executed to location X'A00'. Otherwise the next sequential instruction is executed.
BE	OPTIN	4330 0A00	

INSTRUCTION

Branch on Not Equal (BNE)
Branch on Not Equal Register (BNER)
Branch on Not Equal Short (BNES)

Assembler Notation		Op-Code + M1	Format
BNE	A(X2)	423	RX
BNER	R2	023	RR
BNES	A	203 (Backward) 213 (Forward)	SF

Operation

If the G flag or the L flag is set in the Condition Code, a branch is executed to the second operand location. If neither flag is set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Low (BL)
Branch on Low Register (BLR)
Branch on Low Short (BLS)

Assembler Notation		Op-Code + M1	Format
BL	A(X2)	428	RX
BLR	R2	028	RR
BLS	A	208 (Backward) 218 (Forward)	SF

Operation

When two operands are compared, the C flag is set in the Condition Code of the PSW if the first operand is less than the second operand. If the Carry (C) flag in the Condition Code is set, a Branch on Low is executed to the second operand address. If the Carry flag is not set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

Example: BL

Assembler Notation		Machine Code	Comments
CLHI	R1,X'FF'	C510 00FF	R1 is compared to X'00FF'.
BL	RESTART	4280 0A00	If R1 is less than X'FF', a branch is taken to memory location X'0A00'.

INSTRUCTION

Branch on Not Low (BNL)
Branch on Not Low Register (BNLR)
Branch on Not Low Short (BNLS)

Assembler Notation		Op-Code + M1	Format
BNL	A(X2)	438	RX
BNLR	R2	038	RR
BNLS	A	228 (Backward) 238 (Forward)	SF

Operation

When two operands are compared, the C flag is not set in the Condition Code of the PSW if the first operand is not less than the second operand. If the Carry (C) flag in the Condition Code is not set, a Branch is executed to the second operand address. If the Carry flag is set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Minus (BM)
Branch on Minus Register (BMR)
Branch on Minus Short (BMS)

Assembler Notation		Op-Code + M1	Format
BM	A(X2)	421	RX
BMR	R2	021	RR
BMS	A	201 (Backward) 211 (Forward)	SF

Operation

If the Less-Than (L) flag in the Condition Code is set, a branch is executed to the second operand location. If the L flag is not set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

Example: BM

Assembler Notation		Machine Code	Comments
SIS	R3,1	2631	If R3 is less than 0 after
BM	CONTINUE	4210 10A0	the subtraction, a branch is taken to X'10A0'.

INSTRUCTION

Branch on Not Minus (BNM)
Branch on Not Minus Register (BNMR)
Branch on Not Minus Short (BNMS)

Assembler Notation		Op-Code + M1	Format
BNM	A(X2)	431	RX
BNMR	R2	031	RR
BNMS	A	221 (Backward) 231 (Forward)	SF

Operation

If the Less-Than (L) flag in the Condition Code is not set, a branch is executed to the second operand location. If the L flag is set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Plus (BP)
Branch on Plus Register (BPR)
Branch on Plus Short (BPS)

Assembler Notation		Op-Code + M1	Format
BP	A(X2)	422	RX
BPR	R2	022	RR
BPS	A	202 (Backward) 212 (Forward)	SF

Operation

If the Greater-Than (G) flag in the Condition Code is set, a branch is executed to the second operand location. If the G flag is not set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Not Plus (BNP)
Branch on Not Plus Register (BNPR)
Branch on Not Plus Short (BNPS)

Assembler Notation		Op-Code + M1	Format
BNP	A(X2)	432	RX
BNPR	R2	032	RR
BNPS	A	222 (Backward) 232 (Forward)	SF

Operation

If the Greater-Than (G) flag in the Condition Code is reset, a branch is executed to the second operand location. If the G flag is set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Overflow (BO)
Branch on Overflow Register (BOR)
Branch on Overflow Short (BOS)

Assembler Notation		Op-Code + M1	Format
BO	A(X2)	424	RX
BOR	R2	024	RR
BOS	A	204 (Backward) 214 (Forward)	SF

Operation

If the Overflow (V) flag in the Condition Code is set, a branch is executed to the second operand location. If the V flag is not set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on No Overflow (BNO)
Branch on No Overflow Register (BNOR)
Branch on No Overflow Short (BNOS)

Assembler Notation		Op-Code + M1	Format
BNO	A(X2)	434	RX
BNOR	R2	034	RR
BNOS	A	224 (Backward) 234 (Forward)	SF

Operation

If the Overflow (V) flag in the Condition Code is not set, a branch is executed to the second operand location. If the V flag is set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Zero (BZ)
Branch on Zero Register (BZR)
Branch on Zero Short (BZS)

Assembler Notation		Op-Code + M1	Format
BZ	A(X2)	433	RX
BZR	R2	033	RR
BZS	A	223 (Backward) 233 (Forward)	SF

Operation

If the G and L flag are both reset in the Condition Code, a branch is executed to the second operand location. If the G or L flag is set, the next sequential instruction is executed.

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Not Zero (BNZ)
Branch on Not Zero Register (BNZR)
Branch on Not Zero Short (BNZS)

Assembler Notation		Op-Code + M1	Format
BNZ	A(X2)	423	RX
BNZR	R2	023	RR
BNZS	A	203 (Backward) 213 (Forward)	SF

Operation

If the G or L flag is set in the Condition Code, a branch is executed to the second operand address. If the G and L flags are both reset, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch (Unconditional) (B)
Branch Register (Unconditional) (BR)
Branch Short (Unconditional) (BS)

Assembler Notation		Op-Code + M1	Format
B	A(X2)	430	RX
BR	R2	030	RR
BS	A	220 (Backward) 230 (Forward)	SF

Operation

A branch is unconditionally executed to the second operand address.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

Example: B

Assembler Notation		Machine Code	Comments
B	OPTIN	4300 0A00	An unconditional branch is executed.

INSTRUCTION

No Operation (NOP)
No Operation Register (NOPR)

Assembler Notation		Op-Code + M1	Format
NOP	A(X2)	420	RX
NOPR	R2	020	RR

Operation

After a short delay (instruction decode time), the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

A(X2) and R2 are meaningless and usually equal ZERO (0).

Example: NOP, NOPR

Assembler Notation		Machine Code	Comments
NOP	0	4200 0000	No Operation
NOPR	0	0200	No Operation

CHAPTER 5
FIXED POINT ARITHMETIC

Fixed Point Arithmetic instructions provide a complete set of operations for calculating addresses and indices, for counting, and for general purpose fixed point arithmetic.

DATA FORMATS

Figure 5-1 shows the two formats for fixed point data: halfword and fullword. In each of these formats, the most significant bit (Bit 0) is the Sign bit. The remaining bits, either 15 or 31, represent the magnitude.

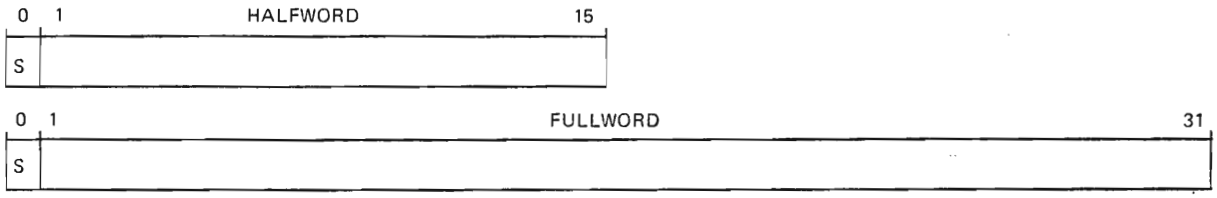


Figure 5-1. Fixed Point Data Words Format

Positive values are represented in true binary form with a Sign bit of ZERO. Negative values are represented in two's complement form with a Sign bit of ONE. To change the sign of a number, the two's complement of the number is produced as follows:

1. Change all zeros to ones, and all ones to zeros.
2. Add one.

FIXED POINT NUMBER RANGE

Fixed point numbers represent integers. Table 5-1 shows the relation between different formats along with decimal values.

TABLE 5-1. FIXED POINT FORMAT RELATIONS

FULLWORD	HALFWORD	DECIMAL
80000000 (MOST NEGATIVE)		-21474 83648
	8000 (MOST NEGATIVE)	-32768
FFFFFFFF	FFFF (LEAST NEGATIVE)	-1
00000000	0000	0
00000001	0001	1
	7FFF (MOST POSITIVE)	32767
7FFFFFFF (MOST POSITIVE)		21474 83647

CONDITION CODE

Most Fixed Point Arithmetic Instructions affect the Condition Code. (The exceptions are Multiply and Divide.) The Condition Code indicates the effect of the operation on the 16-bit result.

In fixed point Add and Subtract operations, because the arguments are represented in two's complement form, all bits, sign included, participate in forming the result. Consequently, the occurrence of a carry or borrow has no real arithmetic significance.

For example, an Add operation between a minus one (FFFF) and a plus two (0002) produces the correct result of plus one (0001) and a carry. The Condition Code is set to 1010 (C = 1 and G = 1). "Carry only" means that the complete result, which in this case would have been 10001, would not fit in 16 bits.

An overflow occurs when the result does not fit in 15 bits. Note that bit "zero" must be reserved for the sign of the result, e.g., adding one to the largest positive fixed point value produces an overflow:

```
 7FFF
+0001
=8000
```

The condition code is 0101 (V = 1 and L = 1)

The result, 8000, is logically correct, but because the sign bit is negative when the result should be positive, the overflow condition exists.

The columns of the Condition Code table show the state of the C, V, G, and L flags for the specific result.

The 'X' in the Condition Code column means that particular flag is not defined, i.e., the flag can be 0 or 1. Hence, no inference should be drawn by testing that particular flag.

FIXED POINT INSTRUCTION FORMATS

The fixed point instructions use the Register to Register (RR), the Short Form (SF), the Register and Indexed Storage (RX), and the Register and Immediate (RI) instruction formats.

FIXED POINT INSTRUCTIONS

The fixed point instructions described in this section are:

AHR	Add Halfword Register
AIS	Add Immediate Short
AH	Add Halfword
AHI	Add Halfword Immediate
AHM	Add Halfword to Memory
SHR	Subtract Halfword Register
SIS	Subtract Immediate Short
SH	Subtract Halfword
SHI	Subtract Halfword Immediate
CHR	Compare Halfword Register
CH	Compare Halfword
CHI	Compare Halfword Immediate
MH	Multiply Halfword
MHR	Multiply Halfword Register
DH	Divide Halfword
DHR	Divide Halfword Register
SLA	Shift Left Arithmetic
SLHA	Shift Left Halfword Arithmetic
SRA	Shift Right Arithmetic
SRHA	Shift Right Halfword Arithmetic
ACH	Add With Carry Halfword
ACHR	Add With Carry Halfword Register
SCH	Subtract with Carry Halfword
SCHR	Subtract with Carry Halfword Register
MHU	Multiply Halfword Unsigned
MHUR	Multiply Halfword Unsigned Register

Add Halfword (AH)
Add Halfword Register (AHR)
Add Halfword Immediate (AHI)
Add Immediate Short (AIS)

Assembler Notation		Op-Code	Format
AH	R1,A(X2)	4A	RX
AHR	R1,R2	0A	RR
AHI	R1,I(X2)	CA	RI
AIS	R1,N	26	SF

Operation

The second operand is added algebraically to the contents of the register specified by R1. The result replaces the contents of the register specified by R1.

AH	(R1) ← (R1) + [A + (X2)]
AHR	(R1) ← (R1) + (R2)
AHI	(R1) ← (R1) + I + (X2)
AIS	(R1) ← (R1) + N

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO
X	1	X	X	Arithmetic overflow
1	X	X	X	Carry

Programming Note

In the RX format, the second operand must be located on a halfword boundary.

The second operand for the Add Immediate Short instruction is obtained by expanding the four bit data field, N, to a 16 bit halfword by forcing the high order bits to zero.

Example: AH

This example adds the halfword at memory location labeled LAB to the contents of Register 4.

1. Register 4 contains X'0002'
   Halfword at memory location LAB contains X'FFFF'

Assembler Notation	Comments
AH REG4,LAB	ADD (LAB) TO (REG4)

Result of AH Instruction

(REG4) = 0001
(LAB) = unchanged by this instruction
Condition Code = 1010 (C=1, G=1)

2. Register 5 contains X'FFF5'
   LAB contains X'FFF2'

Assembler Notation	Comments
AH REG5, LAB	ADD (LAB) TO (REG5)

Result of AH Instruction

(REG5) = FFE7
(LAB) = unchanged by this instruction
Condition Code = 1001 (C=1, L=1)

INSTRUCTION

Add Halfword to Memory (AHM)

Assembler Notation	Op-Code	Format
AHM R1,A(X2)	61	RX

Operation

The content of the register specified by R1 is added algebraically to the contents of the memory location specified by the effective address of the second operand. The 16-bit result replaces the contents of the memory location specified by the effective address of the second operand.

AHM [A + (X2)] ←——— (R1) + [A + (X2)]

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	1	X	X	Arithmetic overflow
1	X	X	X	Carry

Programming Note

The second operand must be located on a halfword boundary.

Example: AHM

This example adds the contents of Register 5 to the contents of memory location LAB.

1. Register 5 contains X'0002'
   Halfword in memory at LAB contains X'FFFF'.

Assembler Notation	Comments
AHM REG5,LAB	ADD (REG5) TO (LAB)

Result of AHM Instruction

(REG5) = unchanged by this instruction
(LAB) = 0001
Condition Code = 1010 (C=1,G=1)

2. Register 6 contains X'FFF5'
   LAB contains X'FFF2'

Assembler Notation	Comments
AHM REG6,LAB	ADD (REG6) TO (LAB)

Result of AHM Instruction

(REG6) = unchanged by this instruction
(LAB) = FFE7
Condition Code = 1001 (C=1,L=1)

INSTRUCTIONS

Subtract Halfword (SH)
Subtract Halfword Register (SHR)
Subtract Halfword Immediate (SHI)
Subtract Immediate Short (SIS)

Assembler Notation		Op-Code	Format
SH	R1,A(X2)	4B	RX
SHR	R1,R2	0B	RR
SHI	R1,I(X2)	CB	RI
SIS	R1,N	27	SF

Operation

The halfword second operand is subtracted from the contents of the register specified by R1. The result replaces the contents of the register specified by R1. The second operand is unchanged.

SH (R1) ← (R1)−[A+(X2)]
SHR (R1) ← (R1)−(R2)
SHI (R1) ← (R1)−I−(X2)
SIS (R1) ← (R1)−N

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO
X	1	X	X	Arithmetic overflow
1	X	X	X	Borrow

Programming Note

The second operand for the Subtract Immediate Short instruction is obtained by expanding the four bit data field, N, to a 16-bit halfword by forcing the high order bits to zero.

In the RX format, the second operand must be located on a halfword boundary.

Example: SH

This example subtracts the halfword at memory location LOC from the contents of register 9.

1. Register 9 contains X'3456'
   LOC contains X'FFF4'

Assembler Notation	Comments
SH REG9,LOC	Subtract contents of LOC from (REG9)

Result of SH Instruction

(REG9) = 3462
(LOC) = FFF4
Condition Code = 1010

2. Register 9 contains X'4567'
   LOC contains X'2345'

Assembler Notation	Comments
SH REG9,LOC	Subtract contents of LOC from (REG9)

Result of SH Instruction

(REG9) = 2222
(LOC) = 2345
Condition Code = 0010

Add with Carry Halfword (ACH)
Add with Carry Halfword Register (ACHR)

Assembler Notation		Op-Code	Format
ACH	R1,A(X2)	4E	RX
ACHR	R1,R2	0E	RR

Operation

The 16-bit second operand and the carry of the previous operation are added algebraically to the contents of the register specified by R1. The result replaces the contents of the register specified by R1 and is reflected by the setting of the Condition Code. The second operand is unchanged.

ACH (R1) ← (R1) + [A2 + (X2)] + C
ACHR (R1) ← (R1) + (R2) + C

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO
X	1	X	X	Arithmetic overflow
1	X	X	X	Carry

Programming Note

Multiple precision addition operations require a carry forward from the least significant operands to the most significant. To accomplish this, the locations containing the least significant portions of the two operands are summed, using the Add Halfword instruction. A carry forward, if it occurs, is retained in the carry bit of the Condition Code. The locations containing the next least significant portions of the two operands are then summed, using the Add with Carry instruction. The carry bit contained in the Condition Code, set from the previous operation, participates in this sum. The carry bit is then set to reflect the new result. The Add with Carry instruction is used on succeeding pairs of operands until the most significant operands of the multiple precision words have been summed. The resulting Condition Code is valid for testing the multiple precision word.

Subtract with Carry Halfword (SCH)
Subtract with Carry Halfword Register (SCHR)

Assembler Notation		Op-Code	Format
SCH	R1,A(X2)	4F	RX
SCHR	R1,R2	0F	RR

Operation

The 16-bit second operand and the borrow from the previous operation are subtracted from the contents of the register specified by R1. The result replaces the contents of the register specified by R1 and is reflected by the setting of the Condition Code. The second operand is unchanged.

SCH (R1) ← (R1) − [A + (X2)] - C
SCHR (R1) ← (R1) − (R2) - C

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO
X	1	X	X	Arithmetic overflow
1	0	X	X	Carry (Borrow)

Programming Note

Multiple precision subtraction operations require a carry forward from the least significant operands to the most significant. To accomplish this, the locations containing the least significant portions of the two operands are subtracted, using the Subtract Halfword instruction. A carry forward, if it occurs, is retained in the carry bit of the Condition Code. The locations containing the next least significant portions of the two operands are then subtracted, using the Subtract with Carry instruction. The carry bit contained in the Condition Code, set from the previous operation, participates in this operation. The carry bit is then set to reflect the new result. The Subtract with Carry instruction is used on succeeding pairs of operands until the most significant operands of the multiple precision words have been subtracted. The resulting Condition Code is valid for testing the multiple precision word.

Compare Halfword (CH)
Compare Halfword Register (CHR)
Compare Halfword Immediate (CHI)

Assembler Notation		Op-Code	Format
CH	R1,A(X2)	49	RX
CHR	R1,R2	05	RR
CHI	R1,I(X2)	C9	RI

Operation

The Halfword second operand is compared algebraically with the first operand, the contents of the register specified by R1. The result is indicated by the Condition Code setting. Neither operand is changed.

Condition Code

C	V	G	L	
0	X	0	0	First operand is equal to second operand
1	X	0	1	First operand is less than second operand
0	X	1	0	First operand is greater than second operand

Programming Note

In the RX format, the second operand must be located on a halfword boundary.

The state of the V flag is undefined.

Example: CH

This example compares the contents of REG8 to the halfword at LAB.

Register 8 contains X'7891'
Halfword at LAB contains X'3123'

Assembler Notation	Comments
CH REG8,LAB	Compare (REG8) to (LAB)

Result of CH Instruction

(REG8) = unchanged by this instruction
(LAB) = unchanged by this instruction
Condition Code = 0010 (G=1)

Multiply Halfword (MH)
Multiply Halfword Register (MHR)

Assembler Notation		Op-Code	Format
MH	R1,A(X2)	4C	RX
MHR	R1,R2	0C	RR

Operation

The signed (halfword) first operand, contained in the register specified by R1+1, is multiplied by the signed (halfword) second operand. The 32-bit result replaces the contents of the registers specified by R1 and R1+1.

MH (R1,R1 + 1) ⟵ (R1 + 1) * [A + (X2)]
MHR (R1,R1 + 1) ⟵ (R1 + 1) * (R2)

Condition Code

Unchanged

Programming Note

After multiplication, the most significant 15 bits with sign bit are contained in R1. The least significant 16 bits are contained in R1+1. The sign of the result is determined by the rules of algebra.

In the RX format, the second operand must be located on a halfword boundary.

The R1 field of these instructions must specify an even numbered register.

Example: MH

This example multiplies the halfword contents of Register 9 by the halfword in memory location LAB.

Register 9 contains X'0045'
Halfword at memory location LAB contains X'8674'

Assembler Notation	Comments
MH REG8,LAB	Multiply (REG9) by (LAB)

Result of MH Instruction

(REG8) = FFDF (REG9) = 3D44
(LAB) = unchanged by this instruction
Condition Code = unchanged by this instruction

INSTRUCTIONS

Multiply Halfword Unsigned (MHU)
Multiply Halfword Unsigned Register (MHUR)

Assembler Notation		Op-Code	Format
MHU	R1,A(X2)	DC	RX
MHUR	R1,R2	9C	RR

Operation

The 16-bit second operand is multiplied by the contents of the register specified by R1+1. All 16 bits of both operands are considered magnitude. The resulting 32 bit product is contained in the registers specified by R1 and R1+1.

MHU (R1,R1 + 1) ⟵ (R1 + 1)*[A + (X2)]
MHUR (R1,R1 + 1) ⟵ (R1 + 1)*(R2)

Condition Code

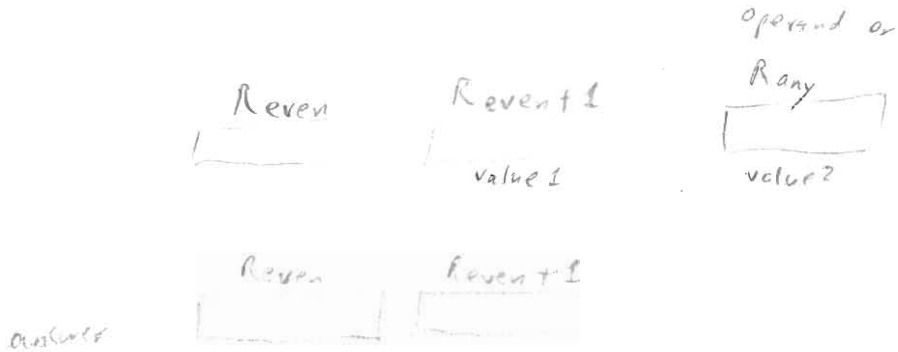
Unchanged

Programming Note

The R1 field must specify an even numbered register.

In the RR format, R2 may specify any register.

This instruction is most useful in applications requiring multiple precision multiply capability.



Divide Halfword (DH)
Divide Halfword Register (DHR)

Assembler Notation		Op-Code	Format
DH	R1,A(X2)	4D	RX
DHR	R1,R2	0D	RR

Operation

The 32-bit signed dividend contained in the register specified by R1 and R1+1 is divided by the 16-bit signed second operand (divisor). The 16-bit signed remainder is stored in the register specified by R1. The 16-bit signed quotient is stored in the register specified by R1+1.

DH	(R1 + 1) ←	(R1,R1 + 1)/[A + (X2)]
	(R1) ←	REMAINDER
DHR	(R1 + 1) ←	(R1,R1 + 1)/(R2)
	(R1) ←	REMAINDER

Condition Code

Unchanged

Programming Note

The R1 field of these instructions must specify an even-numbered register. Otherwise, the results are undefined.

In the RX formats, the second operand must be located on a halfword boundary.

If the divisor is equal to zero, the instruction is not executed, the operand registers are unchanged, and the Divide Fault Interrupt is taken, if enabled by bit 3 of the current program status word. If the interrupt is not enabled, the next sequential instruction is executed.

If the value of the quotient is greater than X'7FFF' or less than (more negative than) X'8000', quotient overflow is said to occur.

If quotient overflow occurs, the operand registers are not changed, and the Divide Fault Interrupt is taken, if enabled by bit 3 of the current program status word. If the interrupt is not enabled, the next sequential instruction is executed.

The sign of the quotient is determined by rules of algebra.

The sign of the remainder is the same as the sign of the dividend.

Example: DH

In this example, the contents of Registers 8 and 9 are divided by the halfword contents of memory location LOC.

1. Register 8 contains X'0000' = Dividend
   Register 9 contains X'0054' = Divisor
   LOC contains X'0008'

Assembler Notation	Comments
DH REG8,LOC	Divide (REG8, REG9) by (LOC)

Result of DH Instruction

(REG8) = 0004 = Remainder
(REG9) = 000A = Quotient
(LOC) = 0008
Condition Code = unchanged by this instruction

2. Register 8 contains X'0000' = Dividend
   Register 9 contains X'1234' = Divisor
   LOC contains X'0000'

Assembler Notation	Comments
DH REG8,LOC	Divide (REG8, REG9) by (LOC)

Result of DH Instruction

Division by zero causes arithmetic fault to be taken if bit 3 of PSW is enabled.

Operands and Condition Code remain unchanged by this instruction.

3. Register 8 contains X'FFFF'
Register 9 contains X'8002' = Dividend
LOC contains X'0001'

Assembler Notation	Comments
DH REG8,LOC	Divide (REG8. REG9) by (LOC)

Result of DH Instruction

Quotient overflow causes arithmetic fault to be taken if bit 3 of PSW is enabled.

Operands and Condition Code remain unchanged by this instruction.

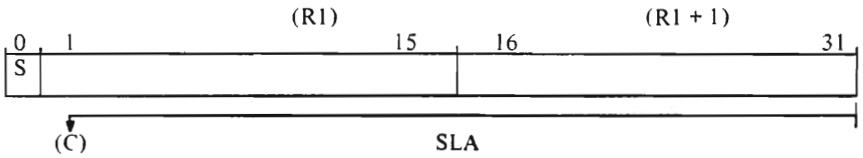
INSTRUCTION

Shift Left Arithmetic (SLA)

Assembler Notation		Op-Code	Format
SLA	R1,I(X2)	EF	RI

Operation

In this instruction, the register specified by R1 and the register implied by the value R1+1 are linked together to form a fullword operand. Bit 0 of the register specified by R1 is the Sign bit. Bits 1:15 of the register specified by R1 and Bits 0:15 of the register specified by R1+1 are shifted left the number of binary places specified by the second operand. The Sign bit is not shifted. Bits shifted out of Position 1 of the first register are shifted into the carry flag of the PSW and then lost. Zeros are moved into Position 15 of the second register.



Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO

Programming Note

R1 must specify an even-numbered register.

The shift count is specified by the least significant five bits of the second operand.

A shift of zero places causes the Condition Code to be set in accordance with the value contained in the register specified by R1. The state of the C flag is undefined in this case.

The state of the C flag indicates the state of the last bit shifted.

Example: SLA

This example shifts the bits in Registers 4 and 5 left by the number specified by the second operand.

Register 4 contains X'8047'
Register 5 contains X'ABCD'

Assembler Notation		Comments
SLA	REG4,4	Shift Left 4 Places

Result of SLA Instruction

(REG4) = X'847A', (REG5) = X'BCD0'
Condition Code = 0001 (L=1)

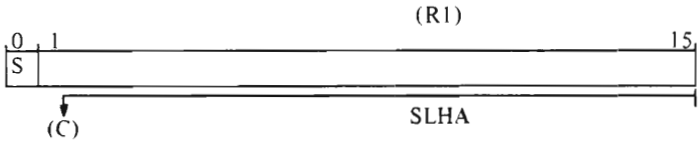
INSTRUCTION

Shift Left Halfword Arithmetic (SLHA)

Assembler Notation		Op-Code	Format
SLHA	R1,I(X2)	CF	RI

Operation

Bits 1:15 of the register specified by R1 are shifted left the number of places specified by the second operand. Bit 0 of the register, the Sign bit, remains unchanged. Bits shifted out of Position 1 are shifted through the carry flag and then lost. The last bit shifted remains in the carry flag. Zeros are shifted into Position 15.



Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO

Programming Note

The state of the C flag indicates the state of the last bit shifted.

The shift count is specified by the least significant four bits of the second operand.

A shift of zero places causes the state of the Condition Code to be set in accordance with the value contained in the register. The C flag is undefined.

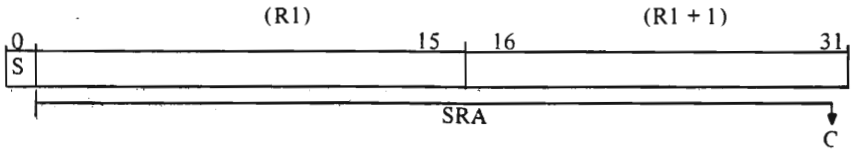
INSTRUCTION

Shift Right Arithmetic (SRA)

Assembler Notation		Op-Code	Format
SRA	R1,I(X2)	EE	RI

Operation

In this instruction, the register specified by R1 and the register implied by the value R1+1 are linked together forming a fullword operand. Bit 0 of the register specified by R1 is the Sign bit. Bits 1:15 of the register specified by R1 and Bits 0:15 of the register specified by R1+1 are shifted right the number of binary places specified by the second operand. The Sign bit remains unchanged and is propagated right as many positions as specified by the second operand. Bits shifted out of Position 15 of the second register are shifted into the carry flag of the PSW, and then lost.



Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO

Programming Note

R1 must specify an even-numbered register.

The state of the C flag indicates the state of the last bit shifted.

The shift count is specified by the least significant five bits of the second operand.

A shift of zero places causes the Condition Code to be set in accordance with the value contained in the registers. The C flag is undefined.

Example: SRA

This example shifts the contents of Registers 8 and 9 right the number of places specified by the second operand.

Register 8 contains X'8ABC'
Register 9 contains X'4256'

Assembler Notation	Comments
SRA REG8,8	Shift (REG9) right 8 bits

Result of SRA Instruction

(REG8) = FF8A (REG9) = BC42
Condition Code = 0001 (L=1)

INSTRUCTION

Shift Right Halfword Arithmetic (SRHA)

Assembler Notation		Op-Code	Format
SRHA	R1,I(X2)	CE	RI

Operation

Bits 1:15 of the register specified by R1 are shifted right the number of places specified by the second operand. Bit-0 of the register, the halfword Sign bit, remains unchanged and is propagated right the number of positions specified by the second operand. Bits shifted out of Position 15 are shifted through the carry flag and lost. The last bit shifted remains in the carry flag.

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO

Programming Notes

The shift count is specified by the low order four bits of the second operand.

The state of the C flag indicates the state of the last bit shifted.

If the second operand specified a shift of zero places, the Condition Code is set in accordance with the value contained in the register. The state of the C flag is undefined.

CHAPTER 6
STATUS SWITCHING AND INTERRUPTS

INTRODUCTION

At any given time, the processor may be in either the Stop or the Run mode. In the Stop mode, the normal execution of instructions is suspended. The processor is under control of the operator who can, through the ASCII Programmer's Console:

Examine the contents of any memory location

Change the contents of any memory location

Examine the contents of any general register

Examine the contents of the Program Status Word

Put the processor in the Run mode

Once the processor has been put in the Run mode, the current Program Status Word controls the operation of the processor. By changing the contents of the current PSW, a running program can:

Put the processor in the Wait state

Enable or disable various interrupts

Vary the normal sequential execution of instructions

PROGRAM STATUS WORD

The Program Status Word is a 32 bit fullword as shown in Figure 6-1.

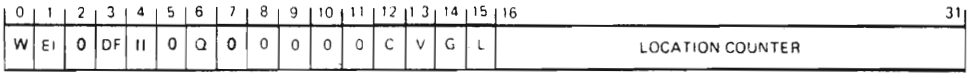


Figure 6-1. Program Status Word Format

Bits 0:15 of the PSW are reserved for status definitions. Note that Bits 8:11 are not currently assigned specific functions. These bits must always be zero. Bits 12:15 are reserved for the Condition Code. Bits 16:31 are reserved for the Location Counter. The status definition bits are interpreted as follows:

Bit 0	(W) Wait state
Bit 1	(EI) External interrupt mask
Bit 2	Reserved, must be ZERO
Bit 3	(DF) Fixed point fault interrupt mask
Bit 4	(II) Automatic I/O and immediate interrupt mask
Bit 5	Reserved, must be ZERO
Bit 6	(Q) Queue service interrupt mask
Bit 7	Reserved, must be ZERO

The current PSW is contained in a hardware register within the Processor. Status switching results when the current PSW, or at least the first half (Bits 0:15) of the current PSW is replaced. The occurrence of an interrupt or the execution of a Status Switching instruction can cause the replacement of the current PSW.

Wait State

Replacing the current PSW with one in which Bit 0 is set puts the processor in the Wait state. When the processor is in the Wait state, program execution is halted. However, the processor is still responsive to external and immediate interrupts, if they are enabled. If the processor is put in the Wait state with all interrupts disabled, only operator intervention from the Turnkey Console can force the processor out of the Wait state.

INTERRUPT SYSTEM

The interrupt system of the processor provides rapid response to external and internal events that require service by special software routines. In the interrupt response procedure, the processor preserves its current state, and transfers control to the required interrupt handler. This software routine may optionally restore the previous state of the processor upon completion of the service.

Some interrupts are controlled by bits in the current Program Status Word. That is, they can be enabled or disabled by setting or resetting a bit in the PSW. Other interrupts are not controlled by PSW bits, and are always enabled. The following is a list of processor interrupts and their controlling PSW bits, if any:

Interrupt	PSW Bit
External (I/O)	1
Fixed Point Fault	3
Immediate I/O	4
System Queue Service	6
Supervisor Call	none
Simulated	none
Illegal Instruction	none

Interrupts occur at various times during processing. The external, immediate and machine malfunction interrupts occur between the execution of instructions. The system queue service, arithmetic fault, supervisor call, and simulated interrupts occur during the execution of instructions. The illegal instruction interrupt occurs before the execution of the improper instruction.

The interrupt procedure is based on the concept of old, current, and new Program Status Words. The current PSW, contained in the hardware register, defines the operating state of the Processor. When this state must be changed, the current PSW becomes the old PSW. The new PSW becomes the current PSW. The current PSW now contains the operating status and the Location Counter for the interrupt service routine.

External Interrupt

This I/O interrupt provides compatibility with previous INTERDATA Processors. Bit 1 of the current PSW controls this interrupt. If this bit is set and Bit 4 reset (see immediate interrupt), and an external device requests processor service, the following action takes place:

The current Program Status Word replaces the contents of memory locations X'0040' - X'0043'.

The new Program Status Word from locations X'0044' - X'0047' becomes the current Program Status Word.

From this point it is up to the software to identify the interrupting device, and take appropriate action.

Fixed Point Fault Interrupt

Bit 3 of the current PSW controls this interrupt. If this bit is set, the interrupt is enabled. A fixed point fault interrupt occurs for either of two reasons:

The divisor in a Fixed Point Divide instruction is zero.

The signed quotient resulting from a fixed point divide operation cannot be expressed in 16 bits.

This interrupt is always taken before any operand is changed. The current PSW is saved in memory locations X'48' - X'4B'. The new PSW, contained in memory locations X'4C' - X'4F', becomes the current PSW. The Location Counter of the old PSW contains the address of the instruction following the one that caused the interrupt.

If Bit 3 of the current PSW is reset, quotient overflow or attempted division by zero do not cause an interrupt. The operands are unchanged, and the next sequential instruction is executed.

Immediate Interrupt

If both Bit 1 and Bit 4 of the current Program Status Word are set, an interrupt request from a peripheral device results in an immediate interrupt.

When the processor receives the interrupt request, it automatically acknowledges the request. The device, in turn, responds with its unique device number. The processor doubles this number, and uses the result as an index into the interrupt service pointer table, which must contain a half-word entry for each of the possible 256 device numbers. The table starts at memory location X'00D0', and extends through location X'02CF'. Chapter 7, Input/Output Operations, contains detailed descriptions of the make-up of this table and its use in interrupt driven I/O.

When an immediate interrupt is taken, the following events occur:

1. The current Program Status Word is saved in the location specified by the entry in the table.

2. The status portion (Bits 0:15) of the Program Status Word is loaded with the value contained in the memory location obtained by adding four to the value contained in the table.

3. The Location Counter of the current Program Status Word is loaded with a value obtained by adding six to the address contained in the table.

The immediate interrupt provides hardware vectoring of external interrupt requests. Each device on the system may have a unique location for the interrupt service routine. If several devices of the same type are included in the system, one service routine may be used for all, if the interrupting device is first identified and then a branch is taken to the common service routine.

System Queue Interrupt

Whenever the processor executes a load Program Status Word or an Exchange Program Status Register instruction, it checks Bit 6 of the current Program Status Word. If this bit is set, and there is an item in the system queue, the processor takes the system queue interrupt. Taking this interrupt causes the current Program Status Word to be saved in memory locations X'0082' - X'0085'. The new Program Status Word contained in memory locations X'0086' - X'0089' becomes the current Program Status Word.

Illegal Instruction Interrupt

The illegal instruction interrupt cannot be disabled. The interrupt occurs whenever the processor reads an instruction word containing an operation code that is not one of those permitted by the system. The processor saves the current Program Status Word in memory locations X'0030' – X'0033'. The new Program Status Word contained in memory locations X'0034' – X'0037' becomes the current Program Status Word.

When the processor encounters an illegal instruction, it makes no attempt to execute it. The Location Counter of the old Program Status Word contains the address of the illegal instruction.

Supervisor Call Interrupt

This interrupt occurs as the result of the execution of a Supervisor Call instruction. This instruction provides a means for user level programs to communicate with system programs. The supervisor call interrupt is always enabled. When the processor executes a Supervisor Call instruction, it:

Saves the current PSW in memory locations X'0096' - X'0099'.

Places the address of the supervisor call parameter block (address of the second operand) in memory locations X'0094' – X'0095'.

Loads the current PSW status with the value contained in locations X'009A' – X'009B'.

Loads the current PSW Location Counter from one of the supervisor call new PSW Location Counters.

Simulated Interrupt

The Simultate Interrupt instruction simulates a request for service from an external device. When this instruction is executed, the processor goes through the automatic I/O procedure, using the device address presented in the instruction word. An immediate interrupt is taken.

STATUS SWITCHING INSTRUCTION FORMATS

The Status Switching instructions use the Register to Register (RR), and the Register and Indexed Storage (RX) instruction formats. In two cases, Load Program Status Word and Simulate Interrupt, the R1 field of the instruction has no significance, and must be zero.

STATUS SWITCHING INSTRUCTIONS

The Status Switching instructions provide for software control of the interrupt structure of the system. They also allow user level programs to communicate with control software.

The instructions described in this section are:

LPSW	Load Program Status Word
EPSR	Exchange Program Status Register
SINT	Simulate Interrupt
SVC	Supervisor Call

INSTRUCTION

Load Program Status Word (LPSW)

Assembler Notation		Op-Code	Format
LPSW	A(X2)	C2	RX

Operation

The 32 bit second operand becomes the current Program Status Word. The second operand is unchanged.

LPSW: $PSW(0:15) \leftarrow [A+(X2)]$

$PSW(16:31) \leftarrow [A+(X2)+2]$

Condition Code

Determined by the new PSW

Programming Note

The R1 field of this instruction is not used by the processor. The Assembler sets the R1 field to zero.

The quantity to be loaded into the current Program Status Word must be located on a full-word boundary.

INSTRUCTION

Exchange Program Status Register (EPSR)

Assembler Notation		Op-Code	Format
EPSR	R1, R2	95	RR

Operation

Bits 0:15 of the current Program Status Word replace the contents of the register specified by R1. The contents of the register specified by R2 replace Bits 0:15 of the current Program Status Word.

EPSR: PSW (0:15) ⟶ R1

PSW (0:15) ⟵ R2

Condition Code

Determined by the new status

Programming Note

If R1=R2, Bits 0:15 of the current PSW are copied into the register specified by R1, but otherwise remain unchanged.

INSTRUCTION

Simulate Interrupt (SINT)

Assembler Notation	Op-Code	Format
SINT 0,I(X2)	E2	RI

Operation

The least significant eight bits of the second operand are presented to the interrupt handler as a device number. The device number is used to index into the interrupt service pointer table, simulating an interrupt request from an external device. This results in an immediate interrupt.

Condition Code

Determined by the new status.

Programming Note

The R1 field of this instruction must contain zero.

INSTRUCTION

Supervisor Call (SVC)

Assembler Notation		Op-Code	Format
SVC	R1, A(X2)	E1	RX

Operation

The address of the second operand replaces the contents of memory locations X'0094' - X'0095'. The current Program Status Word replaces the contents of memory locations X'0096' - X'0099'. The halfword quantity in memory locations X'009A' - X'009B' becomes the new status. The R1 field of the instruction is doubled and added to X'009C'. The halfword value found at the resulting address becomes the new Location Counter.

SVC:

[X'0094'] ← A + (X2)

[X'0096'] ← PSW (0:31)

[X'009A'] → PSW (0:15)

[X'009C'+2* R1] → PSW (16:31)

Condition Code

Determined by the new status

Programming Note

The second operand must be located on a halfword boundary.

CHAPTER 7
INPUT/OUTPUT OPERATIONS

Input/output operations provide a versatile means for the exchange of information between the processor, memory, and external devices. Communication between the processor and external devices is accomplished over the Micro I/O Bus, or Multiplexor Bus. Data transfers to or from external devices may be performed in the byte mode, the halfword mode, or the burst mode. Byte and halfword transfers require processor intervention, either programmed or automatic, for each item transferred. Burst mode transfers, which require a DMA device, proceed independently of the processor.

DEVICE CONTROLLERS

The basic functions of all device controllers are:

To provide synchronization with the processor and to provide device address recognition.

To transmit operational commands from the processor to the device.

To translate device status into meaningful information for the processor.

To request processor attention when required.

In addition, controllers may generate parity, convert serial data to parallel, buffer incoming or outgoing data, or perform other device dependent functions.

Device Addressing

The system design allows as many as 255 external devices. Each device must have its own unique device number, or address. Device numbers may range from X'01' through X'FF'. (Device number X'00' is not used.)

Processor/Controller Communication

Device controllers are attached directly to the Micro I/O Bus or to the Multiplexor Bus. Communication between the processor and controllers is a bidirectional, request-response type of operation.

If the processor initiates the communication, it sends the device address out on the I/O Bus. When a controller recognizes its address, it returns a synchronization signal to the processor, and remains ready to accept commands from the processor. The processor waits up to 15 microseconds for the synchronization signal. If no signal is received in this period of time, the processor aborts the operation, and notifies the controlling program. Controller malfunction and software failure (incorrect device address) are the most common causes of this type of time-out.

In the other direction, a controller can initiate communication with the processor. It does this by generating an attention signal. If the processor is in the interruptable state (Bit 1 of the current PSW set) it temporarily suspends the normal "fetch instruction, execute, fetch next instruction" operation at the end of the execute phase, and transmits an acknowledge signal over the I/O Bus. The controller requesting attention responds with a synchronization signal, and transmits its device number to the processor. (The acknowledge signal may be automatic or programmed, depending on the current state of the processor.)

Device Priorities

Requests for attention are asynchronous. Therefore, more than one request may be pending at any time. The system resolves these conflicts according to device priority. The placement of the controllers on the I/O Bus determines their priority. When two or more controllers request attention at the same time, the one closest to the processor receives the acknowledge signal first, and responds first. Those further down the line must wait until the processor has acknowledged and acted upon requests from higher priority controllers. Requests for attention remain queued until all have been serviced. Devices on the Micro I/O Bus have a higher priority than devices on the Multiplexor Bus.

INTERRUPT SERVICE POINTER TABLE

When automatic I/O is enabled (Bits 1 and 4 of the current PSW set), device requests for service result in an immediate interrupt operation.

The interrupt service pointer table is an ordered list containing one entry for each possible device number in the system. The table starts at memory location X'00D0' and extends through X'02CF'. The software controlling I/O operations must set up the table.

When, having acknowledged a request for service, the processor receives the device address, it adds two times the device address to X'00D0'. The result is the address, within the table, of the entry reserved for the device requesting attention. The processor takes an immediate interrupt, and transfers control to the appropriate software routine.

At the time the processor transfers control to the software routine, the old PSW (current at the time of the device request) has been saved at the location specified in the table; the current status has been loaded from the halfword immediately following the old PSW save location; the current Location Counter has been forced to a value equal to the address of the next halfword following the new status.

I/O INSTRUCTION FORMATS

The I/O instructions use the Register to Register (RR), and the Register and Indexed Storage (RX) instruction formats.

I/O INSTRUCTIONS

Following most I/O instructions, the V flag in the Condition Code indicates an instruction time-out. This means that the operation was not completed, either because the device did not respond, or because it responded incorrectly.

In the sense status and block I/O instructions, the V flag can also mean examine status. To distinguish between these two conditions, the program should test Bits 0:3 of the status byte. If all of these bits are zero, instruction time-out has occurred.

The instructions described in this section are:

ACK (AI)	Acknowledge Interrupt
ACKR (AIR)	Acknowledge Interrupt Register
SS	Sense Status
SSR	Sense Status Register
OC	Output Command
OCR	Output Command Register
RD	Read Data
RDR	Read Data Register
RH	Read Halfword
RHR	Read Halfword Register
RB	Read Block
RBR	Read Block Register
WD	Write Data
WDR	Write Data Register
WH	Write Halfword
WHR	Write Halfword Register
WB	Write Block
WBR	Write Block Register
AL	Autoload
BRK	**Break Point**

INSTRUCTION

Acknowledge Interrupt (ACK)
Acknowledge Interrupt Register (ACKR)

Assembler Notation		Op-Code	Format
ACK (AI)	R1, A2(X2)	DF	RX
ACKR (AIR)	R1, R2	9F	RR

Operation

The address of the interrupting device replaces the contents of the register specified by R1. The eight bit device status replaces the contents of the second operand. The Condition Code is set equal to the right-most four-bits of the device status byte. The device interrupt condition is then cleared.

ACK:
- [R1 (8:15)] ← Device number
- [R1 (0:7)] ← Zero
- [A+(X2)] ← Status byte
- [PSW (12:15)] ← Status byte (4:7)

ACKR:
- [R1 (8:15)] ← Device address
- [R1 (0:7)] ← Zero
- [R2 (8:15)] ← Status byte
- [R2 (0:7)] ← Zero
- [PSW (12:15)] ← Status byte (4:7)

Condition Code

C	V	G	L	
X	X	X	1	Device unavailable
X	X	1	X	End of medium
X	1	X	X	Examine status or time-out
1	X	X	X	Device busy

Programming Note

The Condition Code settings described above assume standard INTERDATA device controllers.

The mnemonics for these instructions under the CAL Assembler (03-066) and the CAL 16 Assembler (03-101) have been changed to avoid confusion with the Add Immediate (AI) instruction in the INTERDATA 32-Bit Processor line. The O.S. Assembler (03-025) continues to accept AI and AIR for these instructions, but does not accept ACK or ACKR.

INSTRUCTION

Sense Status (SS)
Sense Status Register (SSR)

Assembler Notation		Op-Code	Format
SS	R1, A(X2)	DD	RX
SSR	R1, R2	9D	RR

Operation

Bits 8:15 of the register specified by R1 contain the 8-bit device address. The device is addressed, and the 8-bit device status is placed in the second operand location. The Condition Code is set equal to the least significant four bits of the device status byte. The first operand is unchanged.

SSR:
[R2 (8:15)] ← Status byte
[R2 (0:7)] ← Zero
[PSW (12:15)] ← Status byte (4:7)

SS:
[A +(X2)] ← Status byte
[PSW (12:15)] ← Status byte (4:7)

Condition Code

C	V	G	L	
0	0	0	0	Acceptable status
X	X	X	1	Device unavailable
X	X	1	X	End of medium
X	1	X	X	Examine status or time-out
1	X	X	X	Device busy

Programming Note

The Condition Code interpretations of status assume standard INTERDATA device controllers.

In the RR format, the device status byte replaces Bits 8:15 of the register specified by R2. Bits 0:7 are forced to zero.

INSTRUCTION

Output Command (OC)
Output Command Register (OCR)

Assembler Notation		Op-Code	Format
OC	R1, A(X2)	DE	RX
OCR	R1, R2	9E	RR

Operation

Bits 8:15 of the register specified by R1 contain the 8-bit device address. The processor addresses the device and transmits an 8-bit command byte from the second operand location to the device. Neither operand is changed.

OCR: Device ← R2 (8:15)

CC: Device ← [A + (X2)]

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
0	1	0	0	Instruction time-out

Programming Note

In the RR format, Bits 8:15 of the register specified by R2 contain the device command.

INSTRUCTION

Read Data (RD)
Read Data Register (RDR)

Assembler Notation		Op-Code	Format
RD	R1, A(X2)	DB	RX
RDR	R1, R2	9B	RR

Operation

Bits 8:15 of the register specified by R1 contain the 8-bit device address. The processor addresses the device. The device responds by transmitting an 8-bit data byte. This byte is placed in the second operand location.

RD: [A + (X2)] ← Data byte

RDR: [R2 (8:15)] ← Data byte
[R2 (0:7)] ← ZERO

Condition Code

C	V	G	L	
0	0	0	0	Operation Successful
0	1	0	0	Instruction time-out

Programming Note

In the RR format, the 8-bit data byte replaces Bits 8:15 of the register specified by R2. Bits 0:7 of the register are forced to zero.

INSTRUCTION

Read Halfword (RH)
Read Halfword Register (RHR)

Assembler Notation		Op-Code	Format
RH	R1, A(X2)	D9	RX
RHR	R1, R2	99	RR

Operation

Bits 8:15 of the register specified by R1 contain the 8-bit device address. The processor addresses the device. If the device is halfword oriented, the processor transmits 16 bits of data from the device to the second operand location. If the device is byte oriented, the processor transmits two 8-bit bytes in successive operations.

RH:
[A + (X2)] ← First data byte (8-bit oriented device controller)
[A + (X2)+1] ← Second data byte (8-bit oriented device controller)
or
[A + (X2)] ← Halfword of data (16-bit oriented device controller)

RHR:
[R2 (0:7)] ← First data byte (8-bit oriented device controller)
[R2 (8:15)] ← Second data byte (8-bit oriented device controller)
or
[R2 (0:15)] ← Halfword of data (16-bit oriented device controller)

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
0	1	0	0	Instruction time-out

Programming Notes

In the RR format, the data received from a halfword device replaces the contents of the register specified by R2. The first byte of data from a byte device replaces Bits 0:7 of the register specified by R2 and the second byte replaces Bits 8:15.

In the RX format, the second operand must be located on a halfword boundary.

INSTRUCTION

Read Block (RB)

Assembler Notation		Op-Code	Format
RB	R1, A(X2)	D7	RX

Operation

Bits 8:15 of the register specified by R1 contain the 8-bit device address. Bits 0:15 of the halfword located at the second operand address contain the starting address of the data buffer. Bits 0:15 of the halfword located at the second operand address plus two contain the ending address of the data buffer.

The processor transmits 8-bit data bytes from the device to consecutive locations in the specified buffer.

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
X	X	X	1	Device unavailable
X	X	1	X	End of medium
X	1	X	X	Examine status or time-out
1	X	X	X	Device busy

The Condition Code interpretations of status assume standard INTERDATA device controllers.

Programming Note

The starting address must be less than, or equal to, the ending address. If the starting address is greater than the ending address, no transfer takes place, and the processor forces the Condition Code to zero. If the addresses are equal, one data byte is transmitted.

The processor is in a non-interruptable state during the transfer.

This instruction should not be used with 16-bit oriented device controllers.

INSTRUCTION

Read Block Register (RBR)

Assembler Notation		Op-Code	Format
RBR	R1,R2	97	RR

Operation

Bits 8:15 of the register specified by R1 contain the 8-bit device address. The register specified by R2 contains the starting address of the data buffer. The register specified by R2+1 contains the ending address of the data buffer.

The processor transmits 8-bit data bytes from the device to consecutive locations in the specified buffer.

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
X	X	X	1	Device unavailable
X	X	1	X	End of medium
X	1	X	X	Examine status or time-out
1	X	X	X	Device busy

The Condition Code interpretations of status assume standard INTERDATA controllers.

Programming Note

The maximum value for R2 is 14.

The starting address must be less than, or equal to, the ending address. If the starting address is greater than the ending address, no transfer takes place, and the processor forces the Condition Code to zero. If the addresses are equal, one byte is transmitted.

The processor is in a non-interruptable state during the transfer.

This instruction should not be used with 16-bit oriented device controllers.

INSTRUCTION

Write Data (WD)
Write Data Register (WDR)

Assembler Notation		Op-Code	Format
WD	R1, A(X2)	DA	RX
WDR	R1, R2	9A	RR

Operation

Bits 8:15 of the register specified by R1 contain the 8-bit device address. The processor addresses the device, and transmits an 8-bit data byte from the second operand location to the device. Neither operand is changed.

WD: [A + (X2)] ⟶ Device

WDR: [R2 (8:15)] ⟶ Device

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
0	1	0	0	Instruction time-out

Programming Note

In the RR format, the data byte is taken from Bits 8:15 of the register specified by R2.

INSTRUCTION

Write Halfword (WH)
Write Halfword Register (WHR)

Assembler Notation		Op-Code	Format
WH	R1, A(X2)	D8	RX
WHR	R1, R2	98	RR

Operation

Bits 8:15 of the register specified by R1 contain the 8-bit device address. The processor addresses the device. If the device is halfword oriented, the processor transmits 16 bits of data from the second operand location to the device. If the device is byte oriented, the processor transmits two 8-bit data bytes in successive operations.

WH:	[A + (X2)] ⟶ Device	8-bit oriented device controller
	[A + (X2)+1] ⟶ Device	8-bit oriented device controller
	or	
	[A + (X2)] ⟶ Device	16-bit oriented device controller
WHR:	[R2 (0:7)] ⟶ Device	8-bit oriented device controller
	[R2 (8:15)] ⟶ Device	8-bit oriented device controller
	or	
	[R2 (0:15)] ⟶ Device	16-bit oriented device controller

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
0	1	0	0	Instruction time-out

Programming Notes

In the RR format, the data transmitted to a halfword device comes from Bits 0:15 of the register specified by R2. The first byte transmitted to a byte device comes from Bits 0:7 of the register specified by R2 and the second byte comes from Bits 8:15.

INSTRUCTION

Write Block (WB)

Assembler Notation		Op-Code	Format
WB	R1, A(X2)	D6	RX

Operation

Bits 8:15 of the register specified by R1 contain the 8-bit device address. Bits 0:15 of the halfword located at the second operand address contain the starting address of the data buffer. Bits 0:15 of the halfword located at the second operand address plus two contain the ending address of the data buffer.

The processor transmits 8-bit data bytes from consecutive locations in the specified buffer to the device.

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
X	X	X	1	Device unavailable
X	X	1	X	End of medium
X	1	X	X	Examine status or time-out
1	X	X	X	Device busy

Programming Note

The Condition Code interpretations of status assume standard INTERDATA controllers.

The starting address must be less than, or equal to, the ending address. If the starting address is greater than the ending address, no transfer takes place, and the processor forces the Condition Code to zero. If the addresses are equal, one byte is transmitted.

The processor is in a non-interruptable state during the transfer.

This instruction should not be used with 16-bit oriented device controllers.

INSTRUCTION

Write Block Register (WBR)

Assembler Notation		Op-Code	Format
WBR	R1,R2	96	RR

Operation

Bits 8:15 of the register specified by R1 contain the 8-bit device address. The register specified by R2 contains the starting address of the data buffer. The register specified by R2+1 contains the ending address of the data buffer.

The processor transmits 8-bit data bytes from consecutive locations in the specified buffer to the device.

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
X	X	X	1	Device unavailable
X	X	1	X	End of medium
X	1	X	X	Examine status or time-out
1	X	X	X	Device busy

Programming Note

The maximum value for R2 is 14.

The Condition Code interpretations of status assume standard INTERDATA controllers.

The starting address must be less than, or equal to, the ending address. If the starting address is greater than the ending address, no transfer takes place, and the processor forces the Condition Code to zero. If the addresses are equal, one byte is transmitted.

The processor is in a non-interruptable state during the transfer.

This instruction should not be used with 16-bit oriented device controllers.

INSTRUCTION

Autoload (AL)

Assembler Notation		Op-Code	Format
AL	A(X2)	D5	RX

Operation

The Autoload instruction loads memory with a block of data from a byte oriented input device. The data is read a byte at a time, and stored in successive memory locations starting with location X'0080'. The last byte is loaded into the memory location specified by the address of the second operand. Any blank or zero bytes that are input prior to the first non-zero byte are considered to be leader, and are ignored. All other zero bytes are stored as data. The input device is specified by memory location X'0078'. The device command code is specified by memory location X'0079'.

1. If byte = 0, fetch next byte, otherwise step 2.
2. n ← zero
3. [X'80'+n] ← byte
4. n ← n+1
5. If A+(X2) < X'80' +n, the instruction is finished, otherwise return to step 3.

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
X	X	X	1	Device unavailable
X	X	1	X	End of medium
X	1	X	X	Examine status or time-out
1	X	X	X	Device busy

Programming Note

The R1 field of this instruction must be zero.

The Condition Code interpretations of status assume standard INTERDATA device controllers.

INSTRUCTION

Breakpoint (BRK)

Assembler Notation	Op-Code	Format
BRK	88	RR

Operation

Normal Program execution is suspended. The action taken is the same as though the execute switch on the Turnkey Panel had been depressed. The current contents of all 16 General Registers are saved in memory locations X'000' through X'001F'. The Current PSW with bit zero forced reset is saved in memory location X'0024'. The current value of the Instruction Location Counter plus two is saved in memory location X'0026'.

Control is then given to the operator at the ASCII Programmers Console. Refer to Chapter 9 for details.

Condition Code

Unchanged

Programming Note

The R1 and R2 fields of this instruction are not used.

This instruction is optional and is unique to the Model 5/16 Processor. If the option is not equipped, the occurance of this instruction evokes on Illegal Instruction interrupt. If the option is equipped (Refer to the 5/16 Maintenance Manual, Publication Number 29-603), then the actions described under operation will occur.

CONTROL OF I/O OPERATIONS

The design of the I/O structure allows data transfers in any of several ways. The choice of which I/O method to use depends on the particular application, and on the characteristics of the external devices. The primary methods of data transfer between the processor and external devices are:

One byte or one halfword to or from any one of the general registers.

One byte or one halfword to or from memory.

A block of data to or from memory under direct Processor control.

A block of data to or from memory under control of a DMA device.

INTERDATA standard device controllers expect a predetermined sequence of commands to effect data transfers. These commands address the device, put it in the correct mode, and cause data to be transferred. I/O control programs should exercise care in enabling external interrupts.

STATUS MONITORING I/O

The simplest form of I/O programming is status monitoring I/O. In this mode of operation, only one device is handled at a time, and the Processor cannot overlap other operations with the data transfer. The sequence of operations in this type of programming is:

1. Address the device and set the proper mode (Output Command instruction).
2. Test the device status (Sense Status instruction).
3. Loop back to the Sense Status instruction until the status byte indicates that the device is ready (Conditional Branch instruction).
4. When the device is ready, transfer the data (Read or Write instruction).
5. If the transfer is not complete, branch back to the Sense Status instruction (step 2). If it is complete, terminate.

A variation on this type of programming makes use of the block I/O instructions. In this kind of programming, the program prepares the device, and waits for it to become ready. It then executes a block I/O instruction. The processor takes over control and completes the transfer, one byte at a time, to or from memory. The processor monitors device status during the transfer. Block I/O instructions may be used only with byte oriented devices whose ready status is zero.

INTERRUPT DRIVEN I/O

Interrupt driven I/O allows the processor to take advantage of the disparity in speed between itself and the external devices being controlled. With status monitoring, the processor spends much of its time waiting for the device. With interrupt driven programming, the Processor can use much of this time to perform other functions. This kind of programming establishes two levels of operation. On one level are the interrupt service programs. They usually run with external interrupts disabled. On the other level are the interruptable programs. They run with interrupts enabled.

Automatic Vectoring

The use of the automatic I/O features of the 16-Bit Processor allows hardware vectoring of external interrupts. In this type of programming, the software is relieved of the burden of identifying explicitly the interrupt source. This is done by the hardware through the interrupt service pointer table and the immediate interrupt. Automatic I/O is controlled by Bits 1 and 4 of the current PSW.

Before starting operations of this type, the interrupt service pointer table must be set up. This table starts at memory location X'00D0'. It must contain a halfword address entry for every possible device. The value placed in the location reserved for a device is the address of the interrupt service routine for that device. The interrupt service routine must start with a 32 bit old PSW save area. This is followed by a halfword constant that defines the New PSW status. The first instruction of the routine must follow immediately after this constant.

Although there may be gaps in the device address assignments, the interrupt service pointer table should be completely filled. Entries for non-existent devices can point to an error recovery routine. (This precaution prevents system failure in the event of spurious interrupts caused by improper use of the simulate interrupt instruction.)

The next step is to prepare the device for data transfer. This is best done with the external interrupt disabled. Once the table pointer has been set up, and the device prepared, the Processor can move on to an interruptable program.

When the device signals that it requires service, the processor saves its current state, and transfers control to the interrupt service routine. At this time, the old PSW has been saved in the first two halfword locations of the routine, the new status has been loaded, and the current Location Counter contains the address of the first instruction of the routine. The software routine can now:

1. Save any registers used in the routine.
2. Check the device status, and if satisfactory,
3. Make the data transfer, and
4. Restore the registers, then
5. Return to the interrupted program by reloading the old PSW from its save location.

The interrupt service routine for a device may enable immediate interrupts, provided it first disables interrupts from the particular device being serviced.* Because INTERDATA hardware allows interrupts to be disabled at either the device level or the Processor level, nesting of interrupts is both possible and practical.

*Certain devices do not support this; refer to the specific device manual.

Software Vectoring

Software vectoring of interrupts is provided for compatibility with previous INTERDATA processors. The processor reverts to this mode when Bit 4 of the current PSW is reset and Bit 1 of the current PSW is set.

The software must first set up the new external interrupt Program Status Word in memory locations X'0044' - X'0047'. This new PSW should disable external interrupts by resetting Bit 1. The Location Counter of this new PSW contains the address of the interrupt service routine. Upon receipt of the interrupt signal, the processor saves the current PSW in memory locations X'0040' - X'0043', and loads the new external interrupt PSW. This transfers control to the interrupt service routine which must:

1. Save any registers to be used.
2. Acknowledge the interrupt request to get the interrupting device address.
3. Transfer to an appropriate subroutine, based on the device address.

The subroutine then:

4. Checks the device status, and if satisfactory,
5. Makes the transfer.
6. Restores the registers.
7. Returns to the interrupted program by loading the PSW from location X'0040'.

This method for I/O transfers is not as efficient as is the use of the immediate interrupt. In addition, it is not practical with this method to nest interrupts.

DIRECT MEMORY ACCESS

The Direct Memory Access (DMA) feature operates on an instruction cycle stealing basis. That is, between execution of user instructions, a single DMA transfer may occur or a block DMA transfer may occur. Instruction execution is suspended while the DMA transfer takes place.

The DMA device requests processor service when it is ready to transfer data. This request cannot be disabled at the processor end. At the end of the current user level instruction, the processor automatically acknowledges the DMA request. The target main memory address is collected and then a halfword of data is transferred to or from the device. The processor will continue to transfer halfwords until the DMA request is dropped. Because a halfword at a time is transferred, the transfer must always involve an integral number of halfwords. The starting address must always be on an even byte (halfword) boundary. The ending address must always be on an odd byte boundary.

DMA Devices

The DMA mechanism allows a medium to high speed device to gain access to main memory. The memory access is actually performed by the microprogram resident in the Model 5/16 processor. The data transfer occurs over the Micro I/O Bus. Between the Micro I/O Bus and the device requiring service, the customer must build a DMA port. This port must contain a 16-bit Memory Address register and either a 16-bit final transfer address register or a counter. The port contains a Micro I/O Bus interface on the processor side with DMA attention and request acknowledge circuitry. On the device side of the port another I/O Bus interface may be implemented. Depending upon implementation, as many as 16 devices may be attached to the DMA port.

When the DMA port is idle, its private bus is connected directly to the processor's Micro I/O Bus. While this condition exists, the processor can address, command, transfer data and accept interrupt requests from the devices on the DMA private bus. When the DMA port is busy, this connection is broken. All communication between the processor and devices on the DMA private bus is cut off.

The DMA capability of the 5/16 processor, coupled with the user's DMA port allows an external device to transfer data with main memory simultaneous with other program activity.

DMA Operation

The DMA port is a device on the Micro I/O Bus. It has its own unique device number which it recognizes when addressed by the processor. Two registers in the DMA port hold the current memory address and the final memory address or the current memory address and a count of the number of halfwords to transfer. Before starting a DMA operation, the controlling software, using Write Data instructions, places the address of the first halfword to transfer in the current address register. Again, using Write Data instructions, the final address register or count register is set up.

The controlling software then sets up the intended device on the DMA private bus. When the device is ready, the DMA port is issued a start command. The direct connection between the Micro I/O Bus and the DMA private bus is severed and the DMA transfer begins.

The DMA port monitors the device status. Each time the device controller is ready, the DMA port makes a single byte transfer. As soon as the port has accumulated two bytes of data, it makes a DMA request of the processor. The processor acknowledges the request, then collects the halfword of DMA data. The data halfword is stored in the main memory location specified. The processor then increments its copy of the DMA current address register, anticipating another halfword of data from the DMA port. Meanwhile, the DMA port increments its current address register and tests for completion of transfer. If the transfer is not complete and the device is ready with two more data bytes, the DMA request is held active. The processor immediately reads the new halfword from the DMA port and stores it in main memory. Holding the DMA line active forces the "burst" mode.

If the DMA transfer is complete or if the device is not ready with more data, the DMA port drops the request line. Another request cannot be acknowledged until after the next user level instruction.

The above discussion was for a DMA transfer from a device to memory (Read). The DMA write operation is essentially the same. When the device is ready, the DMA port requests a halfword from main memory then sends it, one byte at a time, to the device. This is repeated any time the device indicates that it is ready until the specified number of halfwords has been transferred.

CHAPTER 8
INPUT/OUTPUT SYSTEM

INTRODUCTION

The term interface is used with digital systems to define the junction between two different devices, elements, or pieces of equipment. Interface circuits may perform translation in voltage level, timing, or both.

Digital logic systems operate with a source of input data and an output medium. Inputs may consist of digital or analog signals (i.e., Keyboard, card reader, data set, etc.). Outputs may be a visual display (CRT), or a hard copy terminal (i.e., line printer or Teletypewriter) or control signals. Each signal processed by interfacing hardware must be adequately specified and defined for successful interfacing.

When planning an I/O system to which specific devices must interface, each line of the interface has a dedicated function such as:

Transfer data to or from the processor.
Convey control and timing signals to the peripheral devices.
Transfer status from the peripheral devices to the processor.

Input/output systems provide communication between the processor and its peripheral devices or other system elements. Methods of communication vary in speed, sophistication, and the amount of attention required by the processor. The standard Multiplexor Channel I/O Bus and the Micro I/O Bus are both available in the Model 5/16 Processor. Figure 8-1 is a block diagram of the Model 5/16 System interface.

This chapter defines both the electrical and mechanical specifications of Interdata's Input/Output System. A functional description of each I/O subsystem follows with a description of the layout and interconnection for a typical system interface. Input/output instruction sequences with considerations and specifications for designing device controllers are discussed.

MICRO I/O BUS

The Micro I/O Bus is a byte-oriented I/O system which can directly communicate with 64 different peripheral devices. In the 5/16 Processor, any I/O instruction directed to a device whose address is in the range X'C0' through X'FF', is executed through the Micro I/O Bus rather than the normal Multiplexor Channel. The Micro I/O Bus consists of 28 lines - 8 device address lines, 8 bi-directional data lines, 7 control lines, 4 test lines, and 1 initialize line as shown in Table 8-1. When doing a DMA (Direct to Memory Access) operation, the 8 device address lines double as 8 additional bi-directional data lines. This allows halfword transfers to occur.

Address Lines

The 8 address lines are used to present a device address from the processor to a peripheral device controller. The device address remains latched on the address lines until a different device is addressed or an initialize occurs.

Data Lines

The 8 bi-directional data lines are used to transfer an 8-bit data byte, command byte, or status byte between the processor and the device. A transfer occurs when the device address is valid on the address lines as indicated by the VPA (Valid Peripheral Address) control line. One byte of data is transferred from the processor to the device when the Write control line is active and the C/$\overline{D}$ (Command/$\overline{Data}$) control line is active. One byte of data is transferred from the device to the processor when the Read control line is active and the C/$\overline{D}$ control line is active. When the C/$\overline{D}$ control line is inactive, the Write line means that a command byte is being transferred from the processor to the device, and the Read line means that a status byte is being transferred from the device to the processor.

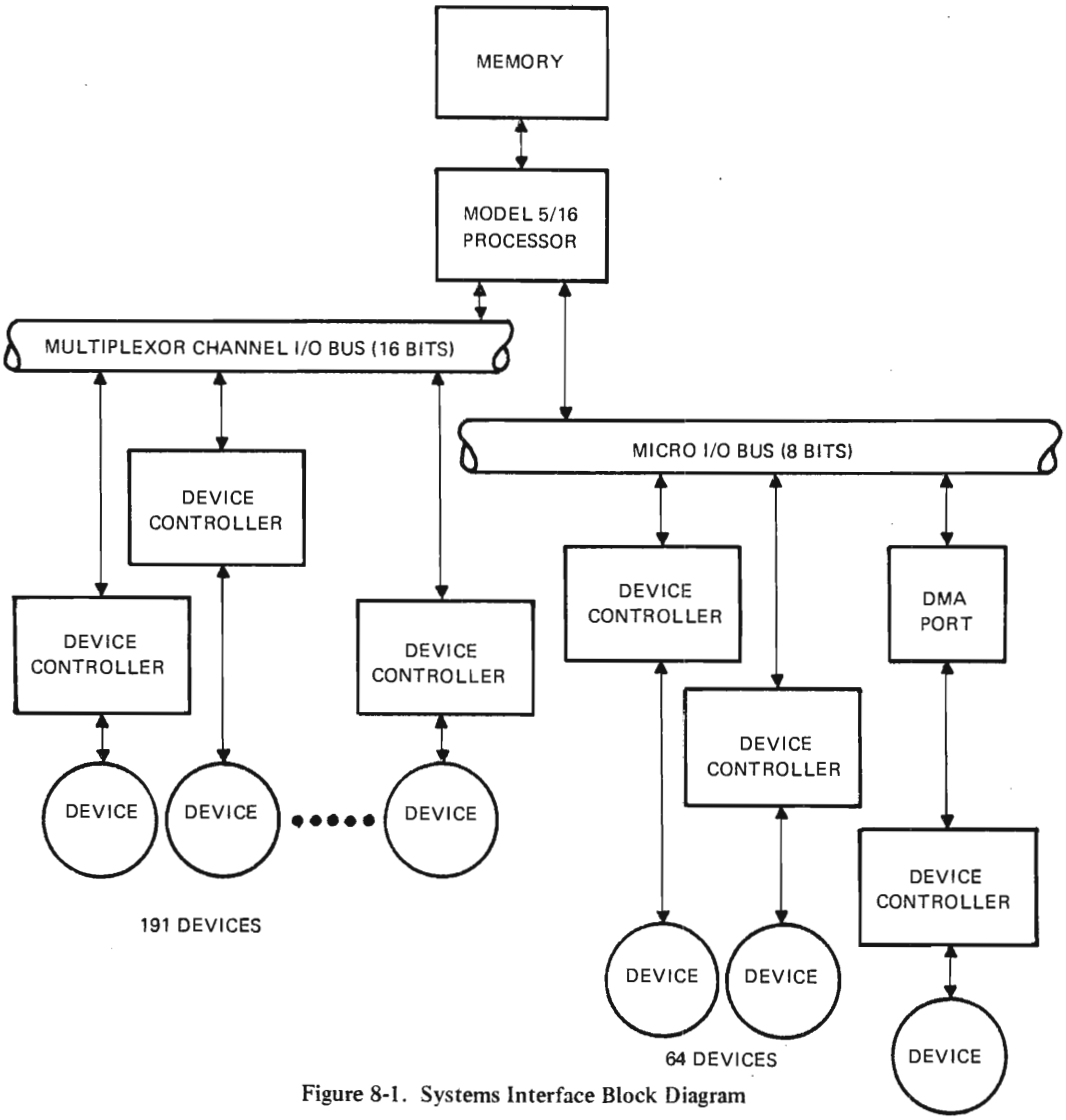


Figure 8-1. Systems Interface Block Diagram

TABLE 8-1. MICRO I/O BUS LINES

FUNCTION	DESIGNATION	DIRECTION
ADDRESS LINES	AD08:15 OR D00:07	PROCESSOR → DEVICE PROCESSOR ↔ DEVICE
DATA LINES	D08:15	PROCESSOR ↔ DEVICE
CONTROL LINES	C/$\overline{D}$ VPA ENABLE READ WRITE ACK DACK	PROCESSOR → DEVICE
TEST LINES	READY ATN DATN DRDY	PROCESSOR ← DEVICE
INITIALIZE LINE	RESET	PROCESSOR → DEVICE

Control Lines

C/$\overline{D}$ (Command/$\overline{Data}$) Table 8-2 shows the function of this line in relationship to the Read and Write control lines. A '1' indicates high level, '0' indicates low level.

TABLE 8-2. FUNCTIONS OF THE C/$\overline{D}$ DATA LINE

C/$\overline{D}$	READ	WRITE	FUNCTION
0	0	1	DATA REQUEST. THE DEVICE CONTROLLER RETURNS DATA ON D08:15.
0	1	0	DATA AVAILABLE. THE PROCESSOR PRESENTS DATA ON D08:15 FOR TRANSFER TO THE DEVICE.
1	0	1	STATUS REQUEST. THE DEVICE CONTROLLER RETURNS STATUS INFORMATION ON D08:15.
1	1	0	OUTPUT COMMAND. THE PROCESSOR PRESENTS A COMMAND BYTE OR D08:15 FOR TRANSFER TO THE DEVICE.

VPA The Valid Peripheral Address line goes active when Address lines DA08:15 are valid and one of the four functions specified in Table 8-2 is being performed.

ENABLE This is a timing signal generated by the processor for I/O transfer synchronization.

READ This low active line tells the device controller that the processor expects to receive data or status information. (See C/$\overline{D}$).

WRITE This low active line tells the device controller that the processor has placed a data or command byte on D08:15. (See C/$\overline{D}$).

ACK In response to the Acknowledge line, an interrupting device controller places its address on Data lines D08:15.

DACK In response to the DMA Acknowledge line, an interrupting DMA device places itself on line and gets ready to present a 16-bit word on the concatenated address lines and data lines. The most significant 8 bits are on the address lines and the least significant 8 bits are on the data lines. See Section on DMA.

Test Lines

READY This signal is generated by the addressed device controller to indicate that it is accepting the data presented by the processor or that it is presenting data to the processor.

ATN Any device desiring to interrupt the processor may do so by activating the Attention line. This line must be held active until an acknowledge is received from the processor.

DATN The DMA Attention line is a separate interrupt line activated by a DMA device when it is ready to transfer data with the processor. This line must be held active until a DMA Acknowledge (DACK) is received from the processor.

DRDY The DMA Ready line goes active after the processor has acknowledged a DMA request and remains active for the duration of the multiple halfword transfer. This line goes inactive to signal the completion of the transfer.

Initialize Line

The Initialize line, known as SCLR within the processor and as Reset on the Micro I/O Bus, is a low active signal that occurs during a power fail or an initialize sequence.

NOTE

All lines except ACK and DACK are connected in parallel to all Micro I/O Bus devices. The ACK line is connected in series with all devices. The DACK line is connected in series with all DMA devices. The ACK or DACK signal, generated by the processor is sent to the first device on the bus. If that device did not generate the interrupt, the ACK or DACK signal is passed on to the next device on the bus. Any device that is the origin of the interrupt signal will capture the ACK or DACK signal and pass it no further. That device then responds as appropriate for ACK or DACK definition.

Micro I/O Bus Operations

Communication over the Micro I/O Bus is on a request/response basis and is controlled by the microprogram resident in the processor's Control Store ROM. A typical sequence of operations over the Micro I/O Bus is:

1. The processor addresses the device controller by an 8-bit address on the address lines. At this time, the device controller becomes selected and will remain selected as long as its address is on the address lines. Meanwhile, the addressed device will respond to subsequent activity on the Micro I/O Bus.

2. If the I/O instruction involves transferring data from the processor to the device, the processor places data on the data lines, adjusts the state of the C/$\overline{D}$ line, and activates the VPA line and the Write line. The timing of the transfer is then controlled by the Enable line.

3. If the I/O instruction involves transferring data from the device to the processor, the processor adjusts the C/$\overline{D}$ line and activates the VPA line and the Read line. The device controller responds by placing data on the data lines. The timing of the transfer is controlled by the Enable line.

In all cases, I/O transfers are accompanied by the VPA control line and are synchronized to the Enable line. The sequence described here is simplified for clarity. The exact sequence for each I/O instruction is discussed later. The programming characteristics of the I/O instructions are found in Chapter 7.

MICRO I/O INTERFACE DESIGN

This section describes those procedures for designing Micro I/O Bus controllers. Since it is impossible to cover all possible controllers, representative circuits are described in sufficient detail to facilitate the design of most controllers.

Micro I/O Bus

The Micro I/O Bus consists of 28 lines which may be divided into four groups:

1. Device Address lines form a group of eight lines from the processor. These lines (AD08:15) hold the Micro I/O Bus interface device address. On DMA transfers, these same eight lines serve as the most significant eight bits of a halfword transfer.

2. Data lines form a group of eight bi-directional lines which transfer command, status, or data bytes between the processor and the addressed device.

3. Control lines form a group of seven unidirectional lines from the processor. The control lines provide the synchronization of data transfers and define the use or intent of the data lines. One of these lines carries the interrupt acknowledge signal (ACK). This is not a shared line but breaks up into a series of short lines to form the daisy-chain priority system. The device controller closest to the processor has the highest priority since the ACK signal must first pass through it.

4. The four test lines and the System Initialize line (RESET0) form the last group. The initialize signal is generated in the processor. The test lines are generated by Micro I/O Bus devices to indicate their need for processor attention or to show response to the signals on the control lines.

Figure 8-2 shows the Micro I/O Bus communication circuits. The physical bus consists of two flat cables carrying the signals shown in Table 8-3.

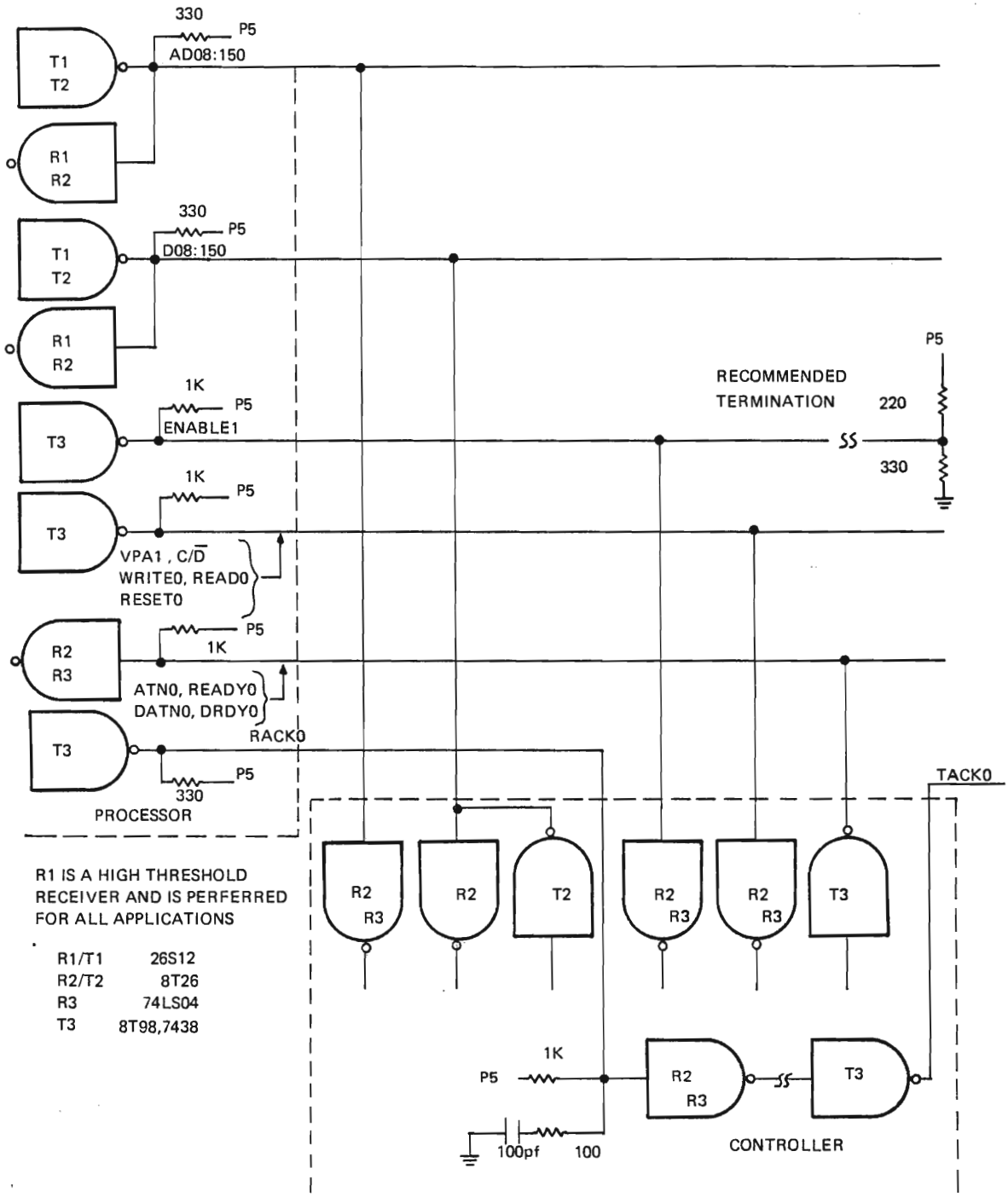


Figure 8-2. Micro I/O Bus Communication Circuits

Each device controller is permitted one TTL load, 2 milliamperes maximum, on any of the device address lines, data lines, control lines, or Reset line. Each device controller is permitted one high power open collector TTL OR-tied onto each of the data lines and each of the test lines. The open collector bus driver must be capable of sinking 48 milliamperes at 0.5VDC maximum V_{CE}. See Figure 8-2.

The cables are 34 conductor 3-M Ribbon Cable, INTERDATA Part Number 17-126F02. The lengths of these cables and the method of termination are best determined by the user and his application. The charactistics of the drivers and receivers on Figure 8-2 are summarized in Tables 8-4 and 8-5:

TABLE 8-3. SIGNALS CARRIED BY MICRO I/O BUS CABLES

	CABLE 1		CABLE 2	
	1	2	1	2
00	AD150	GND	ENABLE1	GND
01	AD140	GND	VPA1	GND
02	AD130	GND	RESET0	GND
03	AD120	GND	READ0	GND
04	AD110	GND	WRITE0	GND
05	AD100	GND	C/$\overline{D}$	GND
06	AD090	GND	RACK0	GND
07	AD080	GND	ATN0	GND
08	D150	GND	READY0	GND
09	D140	GND	DATN0	GND
10	D130	GND	DACK0	GND
11	D120	GND	DRDY0	GND
12	D110	GND	TACK0	GND
13	D100	GND	N.C.	N.C.
14	D090	GND	N.C.	N.C.
15	D080	GND	N.C.	N.C.
16	N.C.	GND	N.C.	N.C.

TABLE 8-4. TRANSMITTER CHARACTERISTICS

PARAMETER	T1	T2	T3
VOL LOW LEVEL OUTPUT	.5V @ 48MA	0.5V @ 48MA	0.5V @ 48MA
VOH HIGH LEVEL OUTPUT	5.5 MAX.	2.4 MIN.	5.5V MAX.
IOH HIGH LEVEL LEAKAGE, VOH = 5.5V	100UA	.200UA	.25UA
+PLH DELAY, LOW TO HIGH	24NS MAX.	20NS MAX.	22NS MAX.
+PHL DELAY, HIGH TO LOW	24NS MAX.	20 NS MAX.	18NS MAX.

TABLE 8-5. RECEIVER CHARACTERISTICS

PARAMETER	R1	R2	R3
VIH INPUT THRESHOLD HIGH	1.8V MIN.	2V. MAX.	2V. MIN.
VIL INPUT THRESHOLD LOW	1.6 MAX.	0.85V. MIN.	0.8V. MAX.
IIH INPUT LEAKAGE, HIGH @5.5V.	200MA MAX.	25MA MAX.	20UA MAX.
IIL INPUT LEAKAGE, LOW @0.4V.	-50UA MAX.	-200UA MAX.	360UA MAX.
TPLH DELAY, LOW TO HIGH	26NS MAX.	17NS MAX.	20NS MAX.
TPHL DELAY, HIGH TO LOW	26NS MAX.	17NS MAX.	20NS MAX.

The following sections show how to design a Micro I/O Bus interface around the Motorola 6800 line Peripheral Interface Adapter (PIA) and the Motorola Asynchronous Communication Interface Adapter (ACIA). Similar parts are available in the INTEL 8080 line. The relationship between the Micro I/O Bus signals and the signals required by the Motorola or INTEL type interface chips is summarized in Table 8-6.

TABLE 8-6. RELATIONSHIP BETWEEN MICRO I/O BUS AND MOTOROLA AND INTEL CHIPS

MICRO I/O BUS SIGNAL	ORIENTATION	MOTOROLA DESIGNATION	INTEL DESIGNATION
AD080:AD150	ACTIVE LOW	A0:A7	A0:A7
*D080:D150	ACTIVE LOW	D0:D7	D0:D7
C/$\overline{D}$0	C1 ACTIVE HIGH D0 ACTIVE LOW	–	C/$\overline{D}$
*VPA1	ACTIVE HIGH	VMA	–
*ENABLE1	ACTIVE HIGH	ENABLE	–
READ0	ACTIVE LOW	–	RD
*WRITE0	ACTIVE LOW	R/W	WR
ACK0	ACTIVE LOW	–	INTA
*ATN0	ACTIVE LOW	IRQ	INT
READY0	ACTIVE LOW	TSC & DBE	READY
*RESET0	ACTIVE LOW	RESET	RESET
DATN0	ACTIVE LOW	–	–
DACK0	ACTIVE LOW	–	–
DRDY0	ACTIVE LOW	–	–
–	–	–	CLK

*THESE SIGNALS CONFORM TO THE MOTOROLA EXORCISOR BUS REGARDING SIGNAL ORIENTATION. THE CLK (CLOCK) SIGNAL REQUIRED BY INTEL PARTS MUST BE DEVELOPED LOCALLY ON THE INTERFACE.

Figure 8-3 shows how the Micro I/O Bus signals relate to the requirements of a Motorola peripheral chip. Figure 8-4 shows the same information for an INTEL peripheral chip.

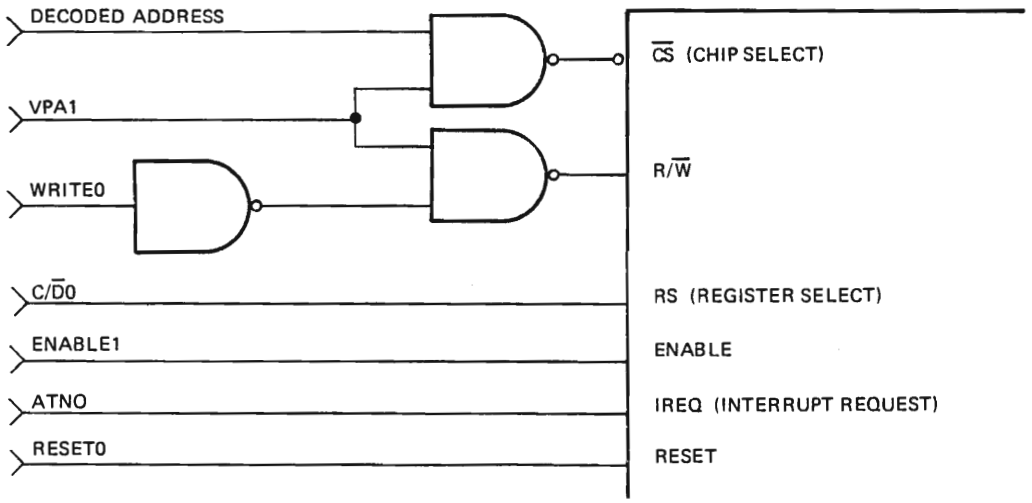


Figure 8-3. Typical Motorola Peripheral Chip Control (For Example Only)

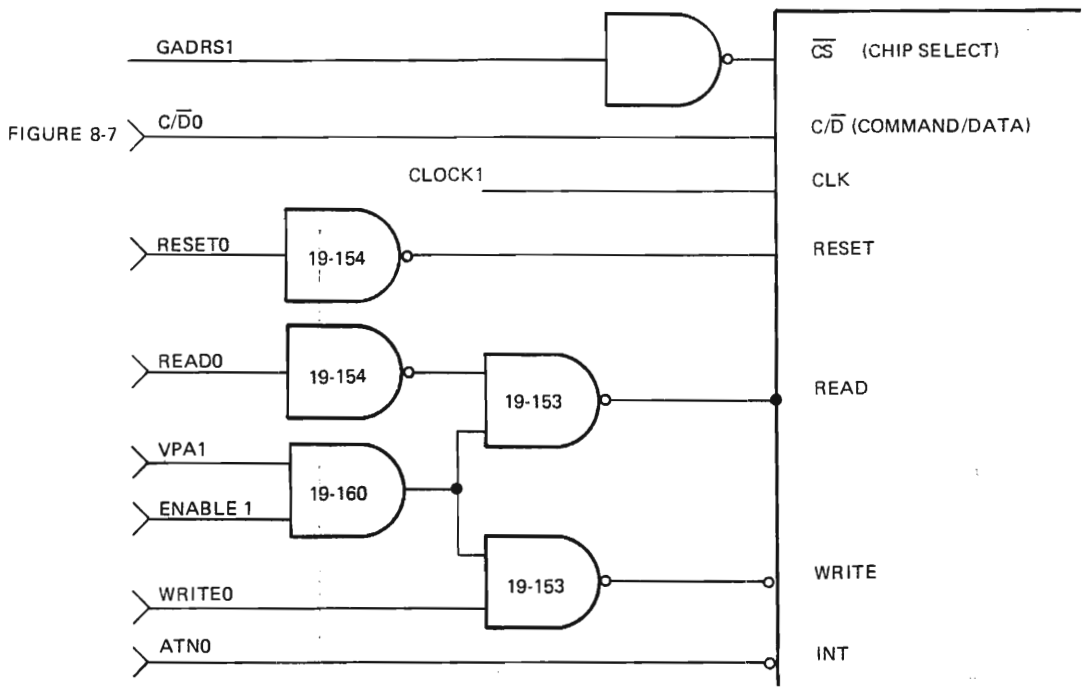


Figure 8-4. Typical INTEL Peripheral Chip Control (For Example Only)

Micro I/O Bus Device Addressing

Figure 8-5 shows how a Micro I/O Bus interface may be addressed. When communicating with a Micro I/O Bus device, the eight-bit address is latched on the device address lines (AD081 through AD151). For any device, the top four address bits are constant (1100_2). The bottom four address bits select one of sixteen possible addresses. The address of this particular interface is switch selectable with INTERDATA Part Number 33-032. The eight input gate will produce a low active signal (ADRS0) only when the address of this controller is on the device address lines. The signal ADRS0 can be used as one of the chip select inputs.

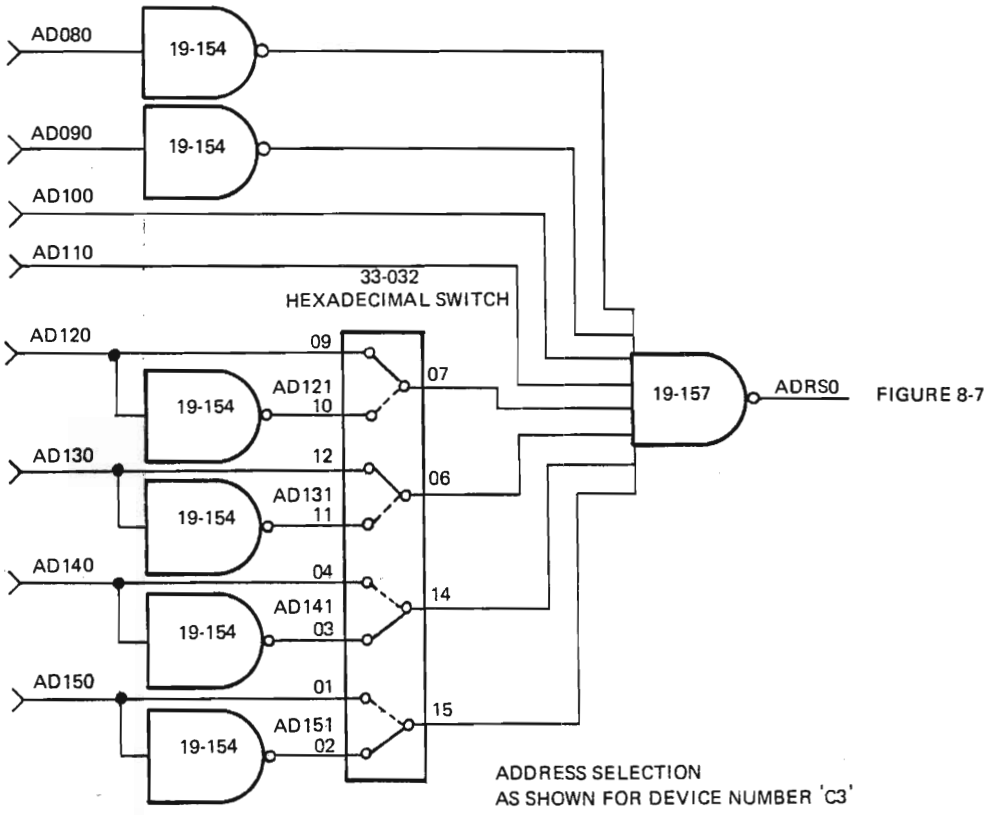


Figure 8-5. Micro I/O Bus Device Addressing

Micro I/O Bus Data Line Buffering

Figure 8-6 shows the Micro I/O Bus data line transceivers. Data from the bi-directional data lines is gated into the interface when the signal WRTEN0 is low active. Data is gated from the interface onto the data lines when the signal RDEN1 is high active. Figure 8-7 shows how the data line transceiver gating signals may be developed. The signal ADRS0 is combined with the Valid Peripheral Address (VPA1) control line to produce GADRS1. This signal is the prime gating signal for developing WRTEN0, RDEN1, CMD0, and STAT0. WRTEN0 is active on a data output or command output from the processor. RDEN1 is active on a data request or a status request from the processor. The signal CMD0 is only active on an output command and the signal STAT0 is only active on a status request.

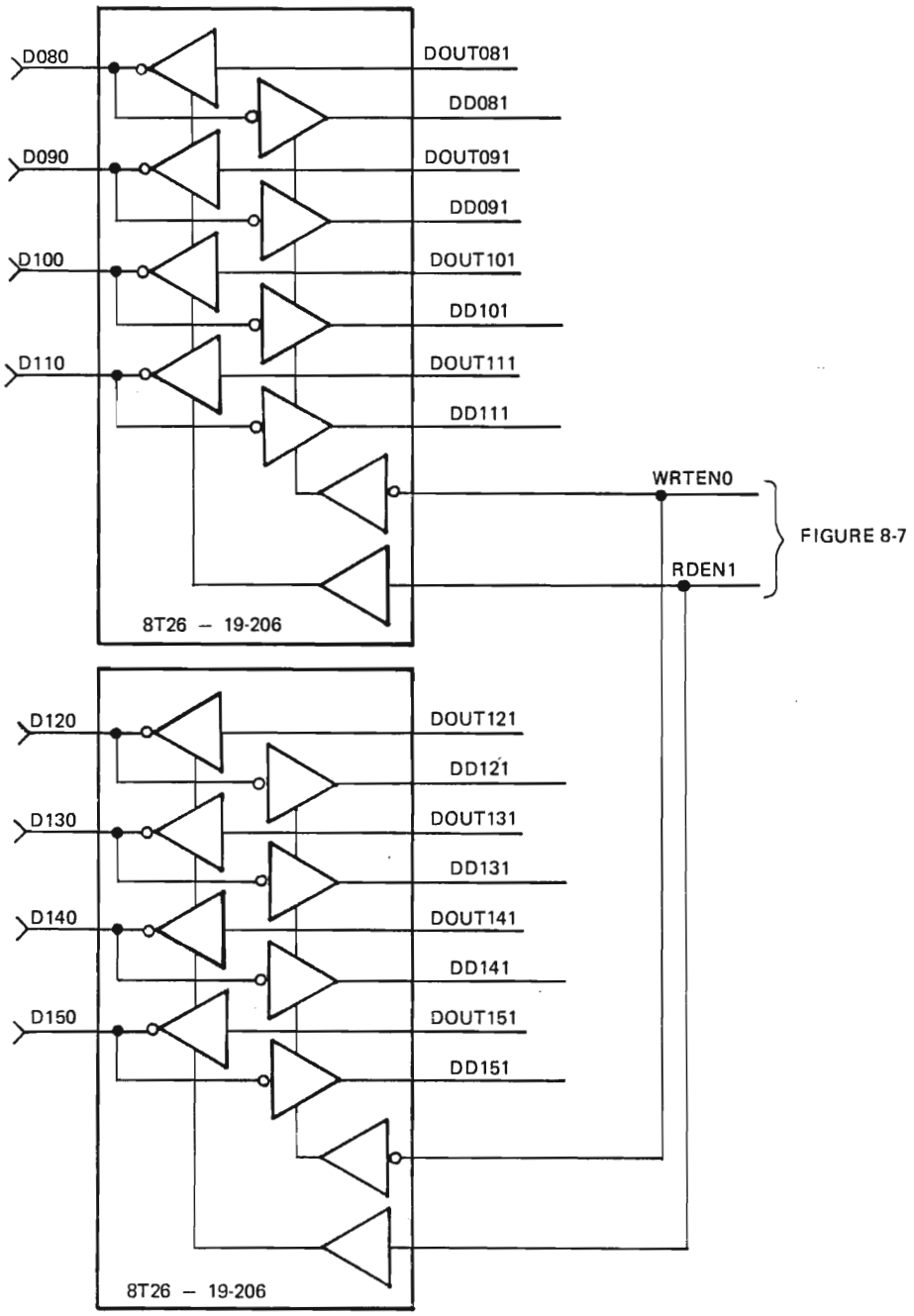


Figure 8-6. Micro I/O Bus Data Line Transceivers

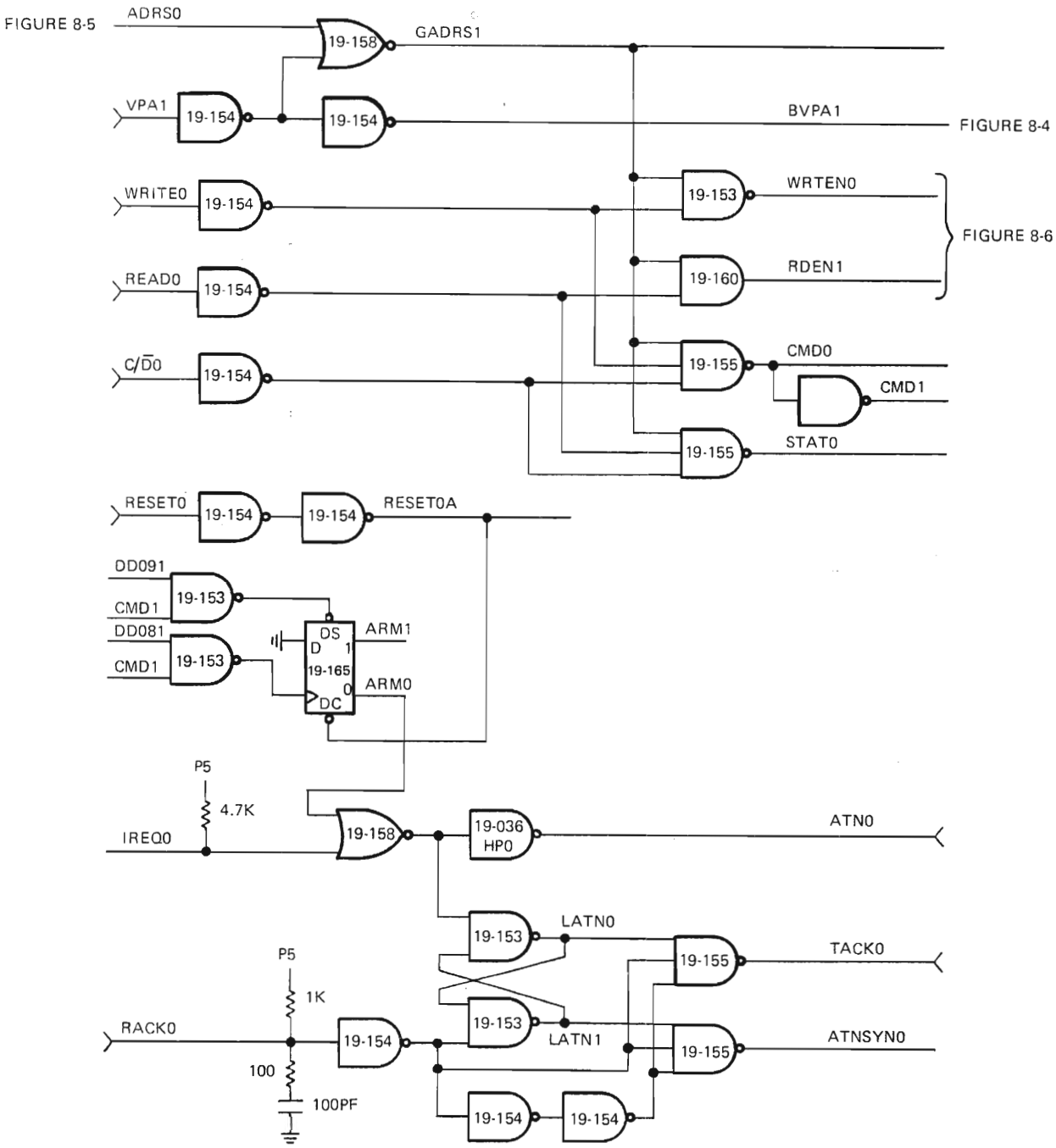


Figure 8-7. Micro I/O Bus Control Line Receivers and Interrupt Logic

Interrupt Logic

The interrupt logic is shown on Figure 8-7. The interface chip generates the signal IREQ0 for whatever reason. This signal is double buffered to generate the ATN0 signal to the processor. The signal LATN0 goes low inhibiting a Transmitted Acknowledge (TACK0). When the processor acknowledges the interrupt, the RACK0 signal is captured to produce the Attention Sync signal ATSYN0. This signal is used to transmit the device address back on the data lines. It may also be used by the peripheral chip to clear IREQ0.

If there is no interrupt request from this controller, a RACK0 signal received from the processor will be transmitted on to the next device controller as TACK0.

Data And Status Output

Figure 8-8 shows the data output multiplexor. Based upon the state of the signals ATSYN0 and STAT0, the following information will be gated onto the DOUT08:15 lines.

ATSYN0	STAT0	DOUT08:15
0	0	Zero Illegal Combination
0	1	Device Number 'C3'
1	0	Device Status (SR08:15)*
1	1	Device Data (DD08:15)

* Optional signals. Device Status Bus normally same as Device Data Bus.

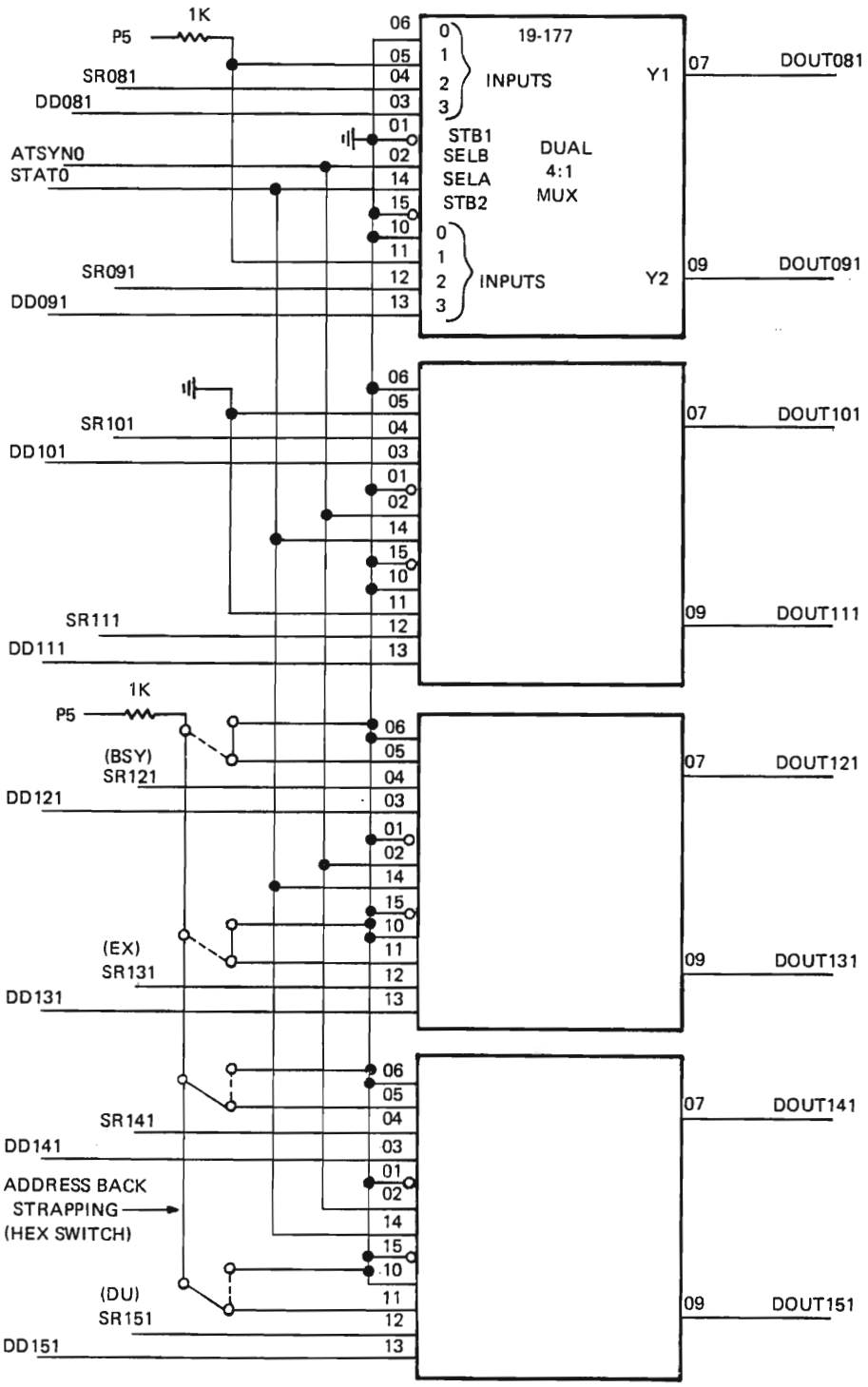


Figure 8-8. Data Output Multiplexor

Figure 8-9, combined with Figures 8-5 through 8-8, forms a complete Teletype current loop interface. Figure 8-10, combined with Figures 8-5 through 8-8, forms a complete paper tape reader interface. Figure 8-11, combined with Figures 8-5 through 8-8, forms a complete Centronics line printer interface.

In each of the three sample interfaces, the address strapping shown on Figure 8-5 and on Figure 8-8, has the interface responding to device address X'C3'. The strapping in both figures must be changed to change the device address.

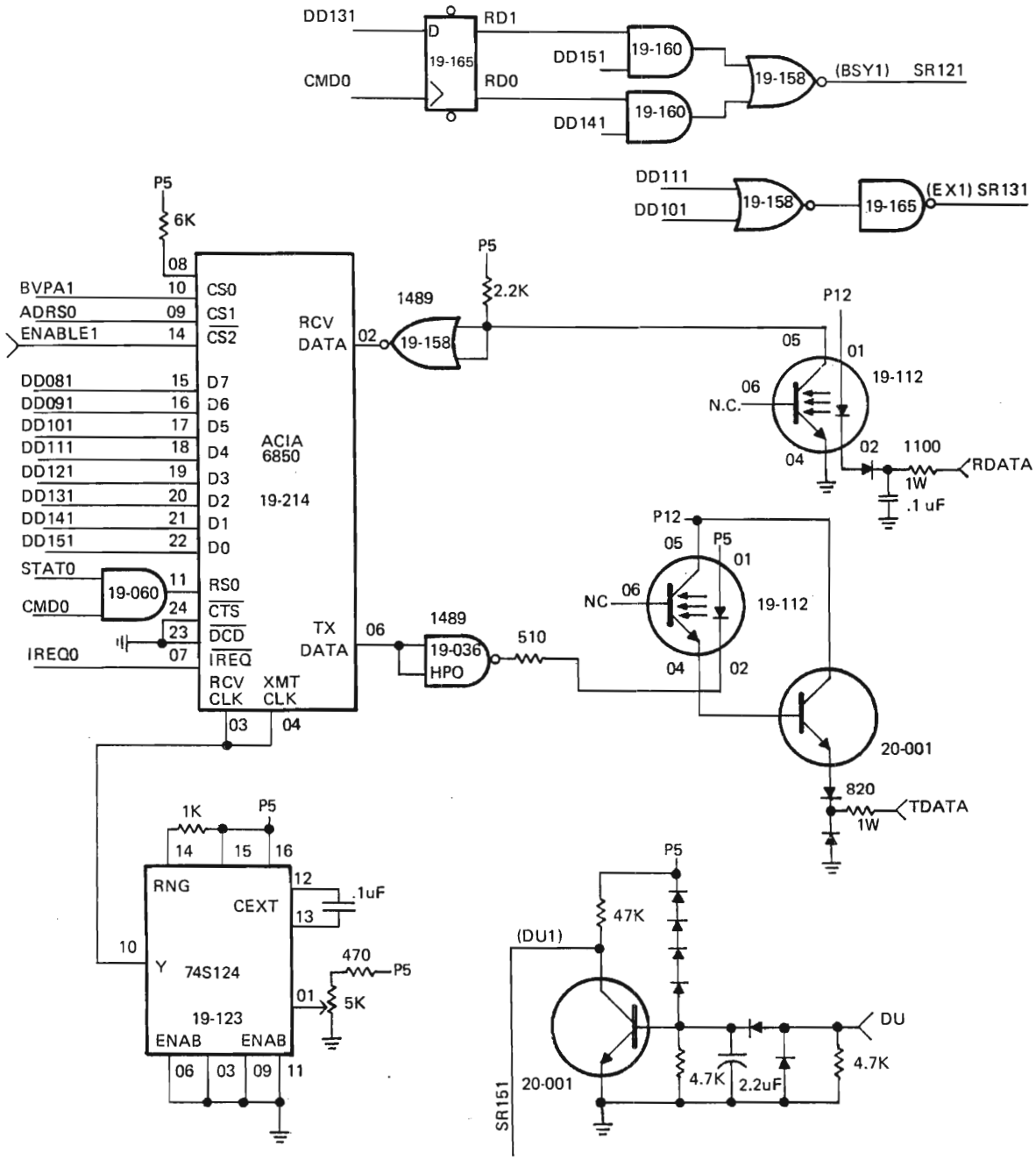


Figure 8-9. Teletype Controller

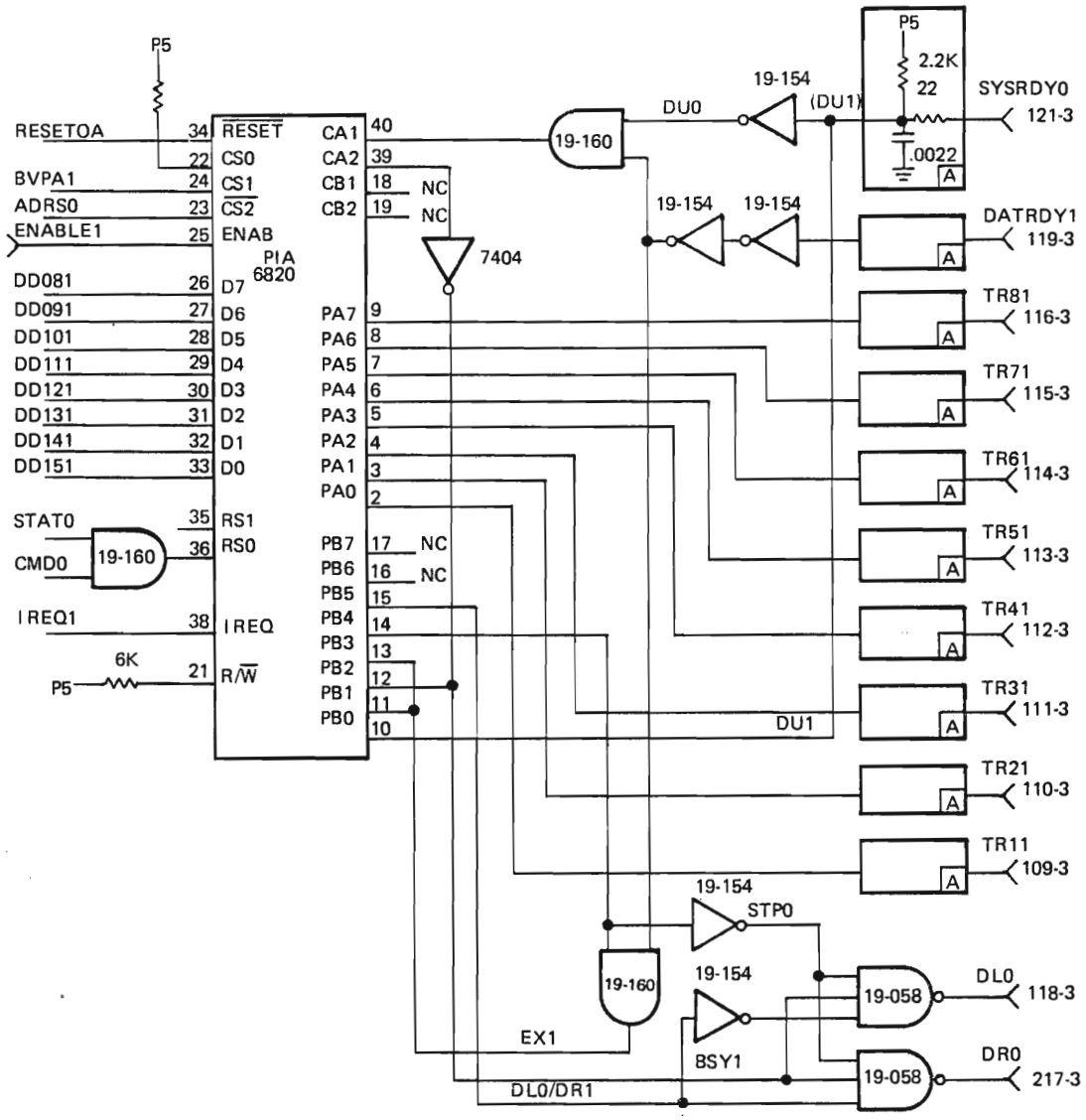


Figure 8-10. Tape Reader Interface

Micro I/O Bus Timing

This section contains timing for various signals on the Micro I/O Bus, including relationships to each other and their sequence of occurence for various I/O instructions. The timing values are adjusted for worst case delays at the controller level. All timing is based on microprocessor manufacturer's data sheets for various devices. Both input and output operations on the Micro I/O Bus make use of "request-response" signaling. This allows the system to run at its maximum speed whenever possible, but permits a graceful slowdown if the characteristics of a particular device controller require signals of longer duration. All transfers are synchronized at the controller level to the Enable signal. See Figure 8-12. The Enable signal is derived from the basic processor clock and is a continuous square wave of 1200 nanosecond period. Because it is continuously running, it is not synchronized with user level I/O instructions. An instruction may not catch Enable at the appropriate point in its cycle. Consequently, it is not possible to accurately predict the Micro I/O Bus timing values.

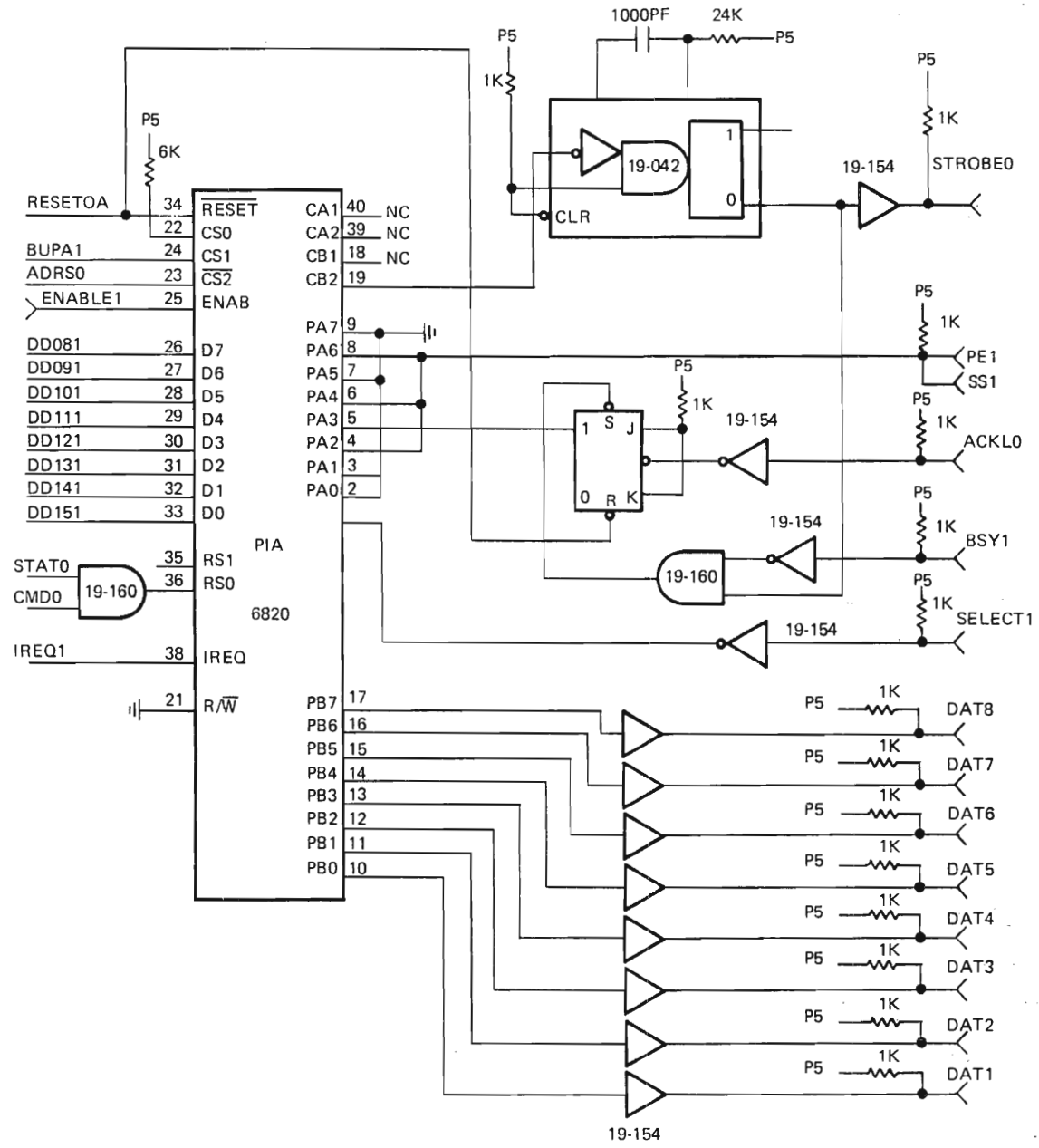


Figure 8-11. Centronics Line Printer Interface

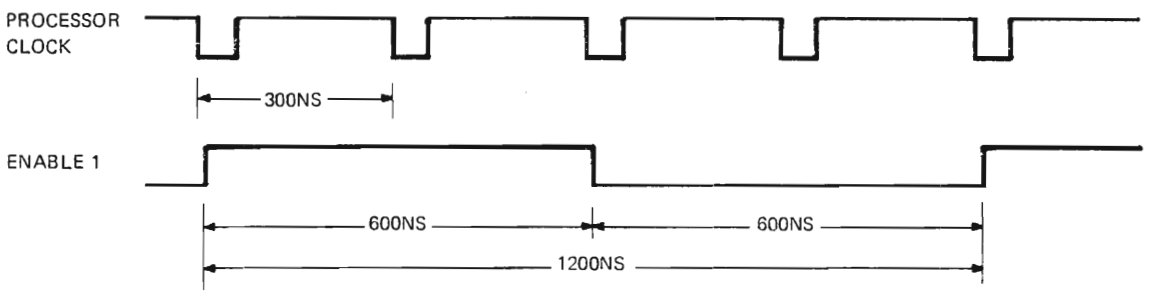


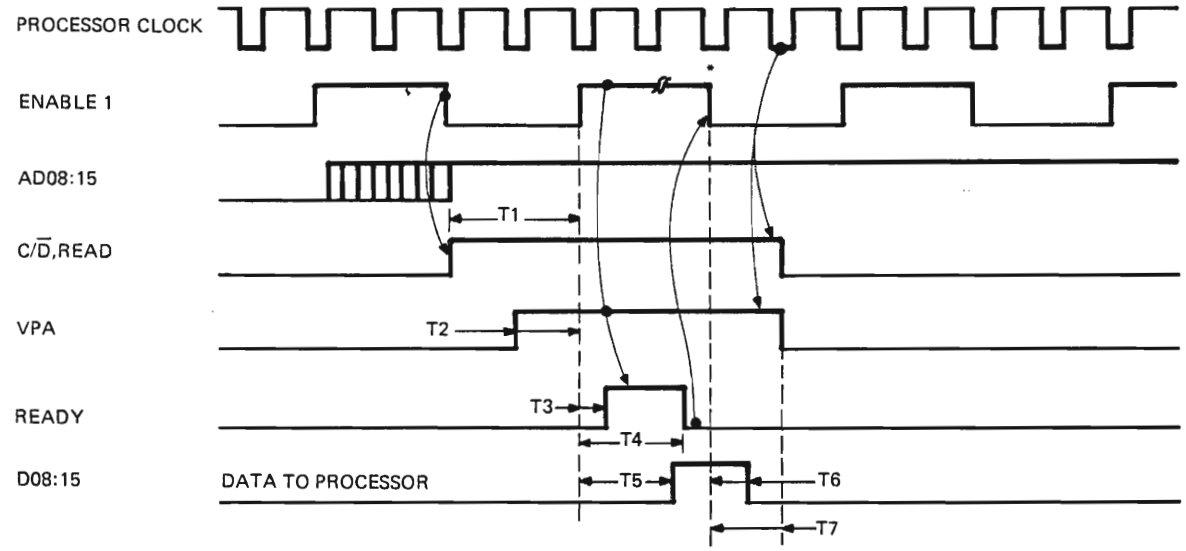
Figure 8-12. ENABLE Timing

Read Data or Sense Status

Execution of a Read Data instruction (RD or RDR) causes an 8-bit data byte to be transferred from the specified device to the processor. The General Register specified by the R1 field of the Read Data instruction contains the device address. This address is latched on the AD lines then a data byte is requested. The returned data byte is placed in the second operand location.

Execution of a Sense Status instruction (SS or SSR) causes the 8-bit device status byte to be transferred to the processor. The General Register specified by the R1 field of the instruction contains the device address. This address is latched on the AD lines then the status byte is requested. The returned status byte is placed in the second operand location.

Figure 8-13 shows the Micro I/O Bus timing for a Read Data or a Sense Status instruction. The processor establishes the device address on the AD lines some 600 nanoseconds in advance of the enable that actually synchronizes the transfer. The Read line is activated and the C/D line is adjusted, then the VPA line is activated. When the Enable line goes active, the device controller responds with a Ready signal and gates the data byte or status byte onto the Data lines. The processor holds the Enable line active until the device drops Ready. The Enable line can be held active for as many additional 300 nanosecond intervals as it takes for the Ready line to become de-activated. To prevent the processor locking up an I/O that cannot be completed, a 15 microsecond time-out aborts the operation (false SYNC) and drops Enable. When Enable goes inactive, the processor accepts the data on the data lines and de-activates the Read and VPA lines.



	MIN	TYP	MAX
T1	210NS	600NS	
T2	210NS	300NS	
T3	0	100NS	240NS
T4	470NS	500NS	25us
T5	400NS	400NS	
T6	10NS	10NS	
T7	60NS	300NS	

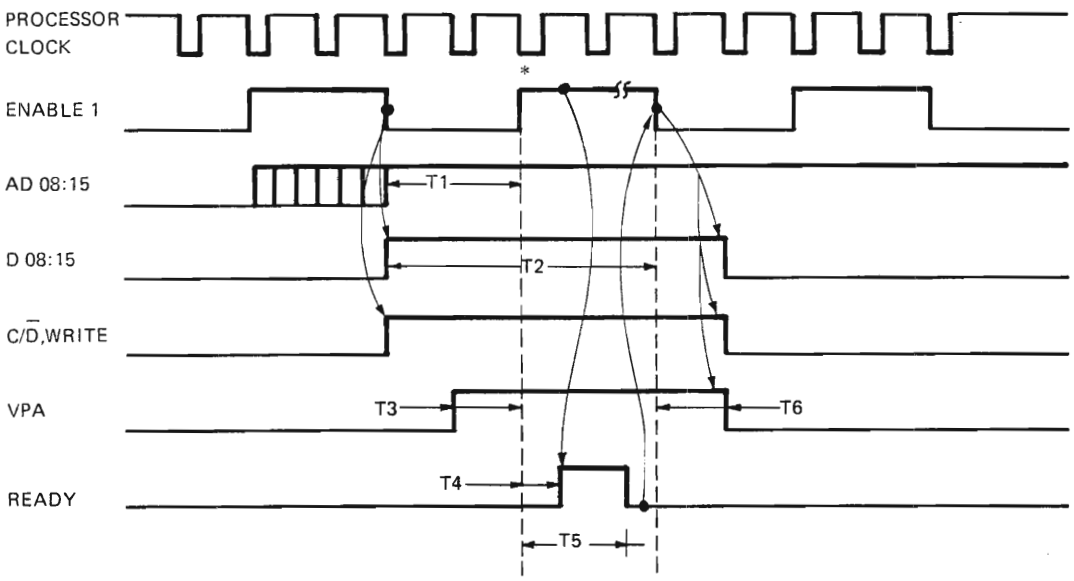
Figure 8-13. Micro I/O Bus Input Timing

Write Data or Output Command

Execution of a Write Data instruction (WD or WDR) causes an 8-bit byte from the second operand location to be transferred to the specified device. The General Register specified by the R1 field of the instruction contains the device address. This address is latched on the AD lines then the second operand data byte is transferred to the selected device.

Execution of an Output Command instruction (OC or OCR) causes an 8-bit command byte from the second operand location to be transferred to the specified device. The General Register specified by the R1 field of the instruction contains the device address. This address is latched on the AD lines then the second operand command byte is transferred to the selected device.

Figure 8-14 shows the Micro I/O Bus timing for a Write Data or an Output Command instruction. The processor establishes the device address on the AD lines at least 600 nanoseconds in advance of the enable that is to actually synchronize the transfer. The data to be output is statically gated onto the data lines. The C/D line is adjusted and the Write line is activated, then the VPA line is activated. When the Enable line goes active, the device controller responds with Ready and accepts the data byte or command byte. The processor holds Enable active until the device drops ready. Enable can be held active for as many additional 300 nanosecond intervals as it takes for the Ready Line to become de-activated. To prevent the processor locking up an I/O that cannot be completed, a 15 microsecond time-out aborts the operation (False SYNC) and drops Enable. After Enable goes inactive, the processor removes the data from the data lines and de-activates the Write and VPA lines.



	MIN	TYP	MAX
T1	210NS	600NS	—
T2	245NS	1500NS	—
T3	210NS	300NS	—
T4	0	100NS	240NS
T5	470NS	500NS	25us
T6	60NS		

*DEVICE CONTROLLER MAY USE THIS EDGE TO LATCH DATA IF DATA IS AVAILABLE MINIMUM OF 50NS BEFORE.

Figure 8-14. Micro I/O Bus Output Timing

Acknowledge Interrupt

Interrupts are provided to detect events both within and external to the processor. The occurrance of an interrupt may cause the processor to leave the normal program sequence and enter an interrupt service subroutine which performs some operation appropriate to the interrupt. For example, an external device may interrupt to indicate that it is ready to transfer another byte of data. The interrupt subroutine would then initiate the data transfer. After servicing the interrupt, the main program is resumed from the point at which it was interrupted.

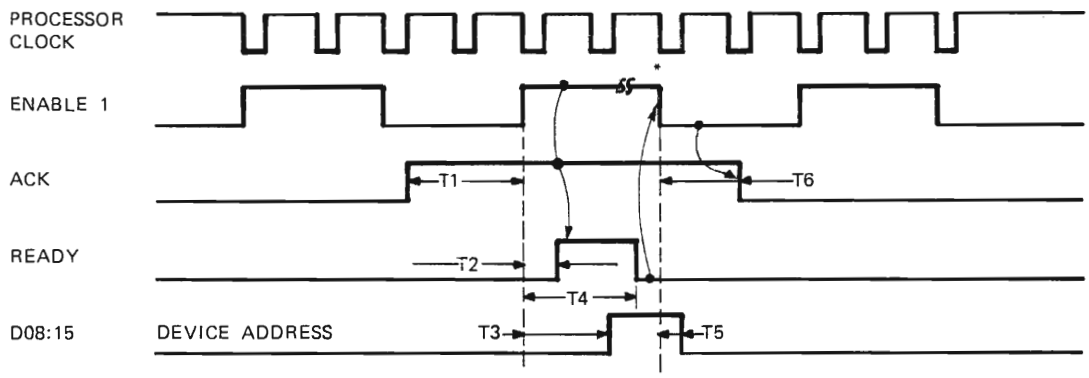
Figure 8-15 shows the Micro I/O Bus timing for an Acknowledge Interrupt instruction. If enabled by the PSW, the microcode automatically acknowledges device interrupts. The timing on this is the same. The ACK line is connected in series to all Micro I/O Bus devices and Multiplexor Channel devices. Devices on the Micro I/O Bus have a higher priority than devices on the Multiplexor Bus. The first device controller in the ACK chain that is responsible for an I/O Attention responds. Other devices further down the chain must wait for service. If it is a Micro I/O Bus device, the interrupting device responds by activating the Ready line. It then places its address on the data lines D08:15 and drops Ready. After Ready drops, the processor accepts the returned device number and drops the ACK line.

The acknowledge sequence may be programmed or automatic depending on the state of the processor when the interrupt occurred. Refer to Chapter 6. If doing an Acknowledge Interrupt instruction (ACK or ACKR), the returned device number is copied into the General Register specified by the R1 field of the instruction. The microcode then performs a Sense Status sequence and places the returned status byte in the second operand location. In the automatic interrupt service mode, no Sense Status is performed. Instead, a software routine unique to the interrupting device number is entered.

Block I/O

The Block I/O instructions allow transferring multiple byte strings. No other processor activity occurs while the Transfer takes place.

Execution of a Read Block instruction (RB or RBR) causes multiple data bytes to be read from the device and stored in the area of memory specified by the second operand. The General Register specified by the R1 field of the instruction contains the device address. The device is addressed then the block transfer sequence begins. Each byte transfer is preceeded by a status request. If the status indicates that the device is busy, the status request is repeated until the device is not busy. When the status is good, a data request is made. The returned data byte is stored in memory at the location specified by the starting memory address. The starting address is incremented and compared to the final memory address. If the start address is less than the final address, the sequence is repeated. The instruction terminates when the last byte has been transferred. The instruction also terminates if a bad status indication is received from the device.



	MIN	TYP	MAX
T1	480NS	600NS	–
T2	0	100NS	240NS
T3		400NS	400NS
T4	470NS	500NS	10us
T5	60NS	60NS	–
T6	60NS	300NS	–

* PROCESSOR LATCHES DATA ON THIS CLOCK.

Figure 8-15. Micro I/O Bus Acknowledge Timing

The Block I/O instructions only use the least significant four bits of the device status. If these bits are 1000, the device must be busy. If these bits are 0000, the device is not busy and a data transfer will occur. Any other indication in the 4 status bits will abort the instruction.

A Write Block instruction (WB or WBR) is identical to a Read Block instruction except for the direction of the data transfer.

Figure 8-16 shows the Micro I/O Bus timing for a Read Block instruction. Figure 8-17 shows the Micro I/O Bus timing for a Write Block instruction.

Micro-DMA Bus

The DMA over the Micro I/O Bus is the only DMA port available on the Model 5/16. On the 5/16 this DMA is serviced on an instruction steal basis. The "Micro-DMA" is designed to pass data to or from a device controller or Selector Channel in a burst mode and is efficient only in this mode. This 'burst' can be from one to 'n' halfwords. The DMA system is not presently supported by INTERDATA software or device controllers.

The Micro-DMA bus has nineteen lines consisting of 16 bi-directional lines used for data and address. These are the same lines used by the Micro I/O Bus for data and address. In addition to these data lines a unique Attention line (DATN0), a unique Acknowledge line (DACK0), and a unique Ready line (DRDY0) are used.

A description of the operational sequence of the Micro-DMA on the Model 5/16 follows:

1. The device when ready to transfer data must activate the DMA Attention line (DATN0). The micro-program tests this line after each user instruction and when active goes to a micro-subroutine to handle the data transfer.

2. The micro-program will issue two DMA Acknowledge (DACK0) signals to the Micro I/O Bus. The highest priority DMA device requesting service will respond with a 16-bit word. Bits 0-14 represent the starting address of the block transfer and Bit 15 indicates whether the operation to be performed is either a Read or a Write. The micro-program tests bit 15, logical '0' indicates a memory read and a logical '1' indicates a memory write, and proceeds to the appropriate subroutine.

3. Data is transferred via data requests or data availables until the DMA Ready line goes inactive at which time the next user instruction is executed. The micro-program uses the starting memory address presented by the device controller and increments it by two for each data transfer.

NOTE

No status requests are made between data transfers; therefore the device must be able to respond to the next control line within 300 nanoseconds or deactivate its DMA READY.

The programming of a Selector Channel on this bus could be identical to the present Selector Channel.

Figure 8-18 indicates the protocol and timing necessary on the Model 5/16 Micro-DMA.

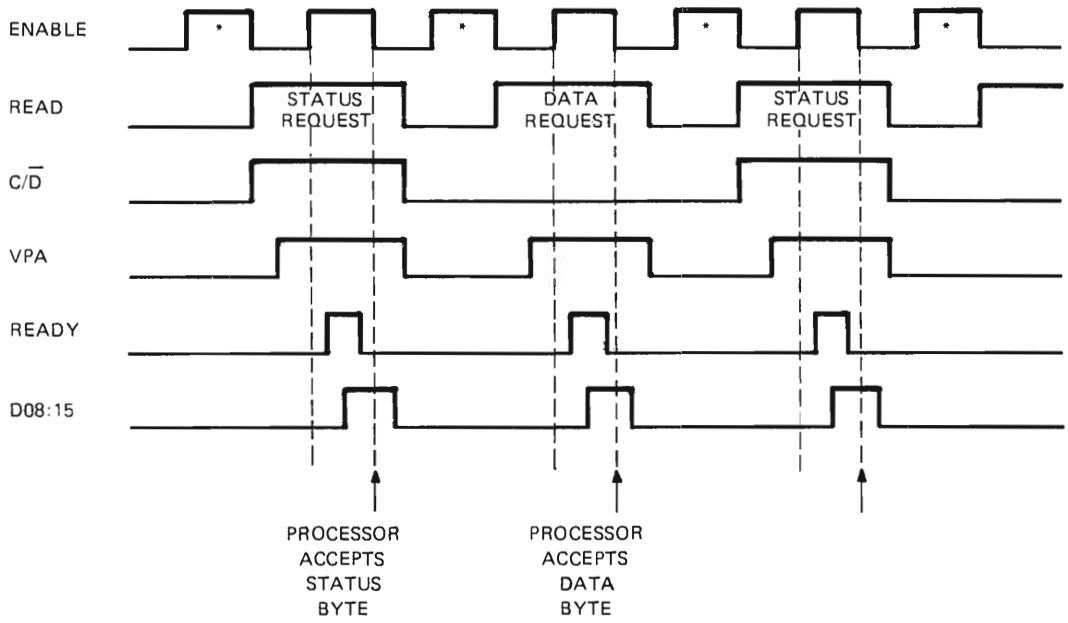


Figure 8-16. Micro I/O Bus Timing for Read Block

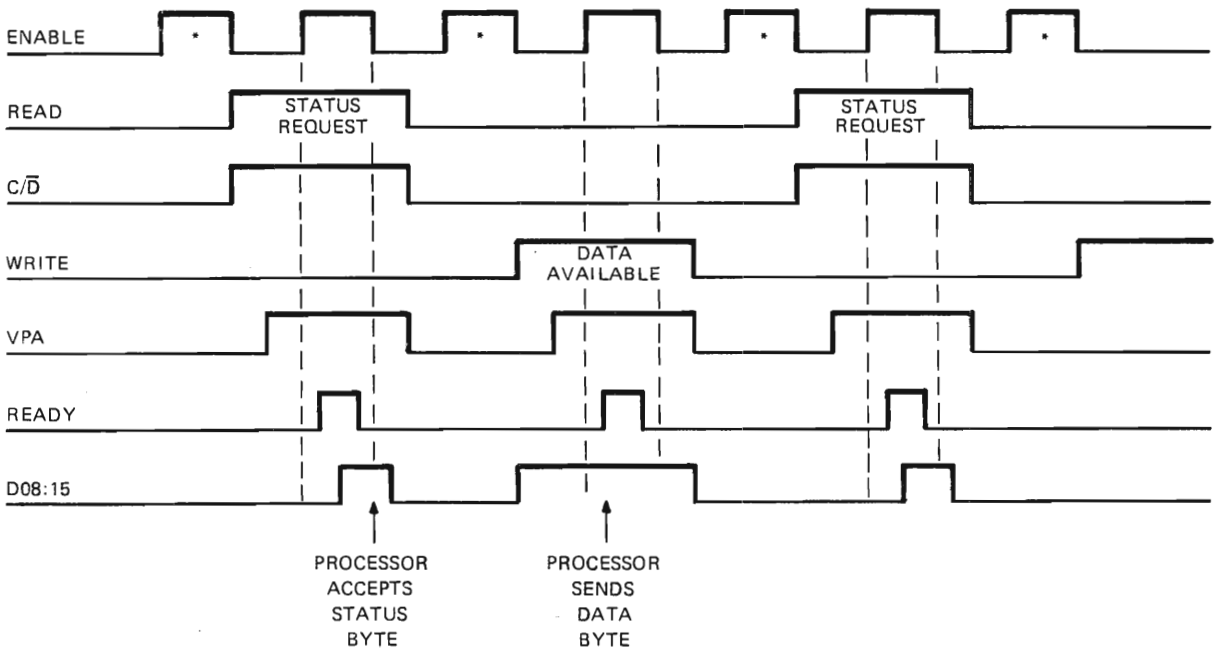


Figure 8-17. Micro I/O Bus Timing For Write Block

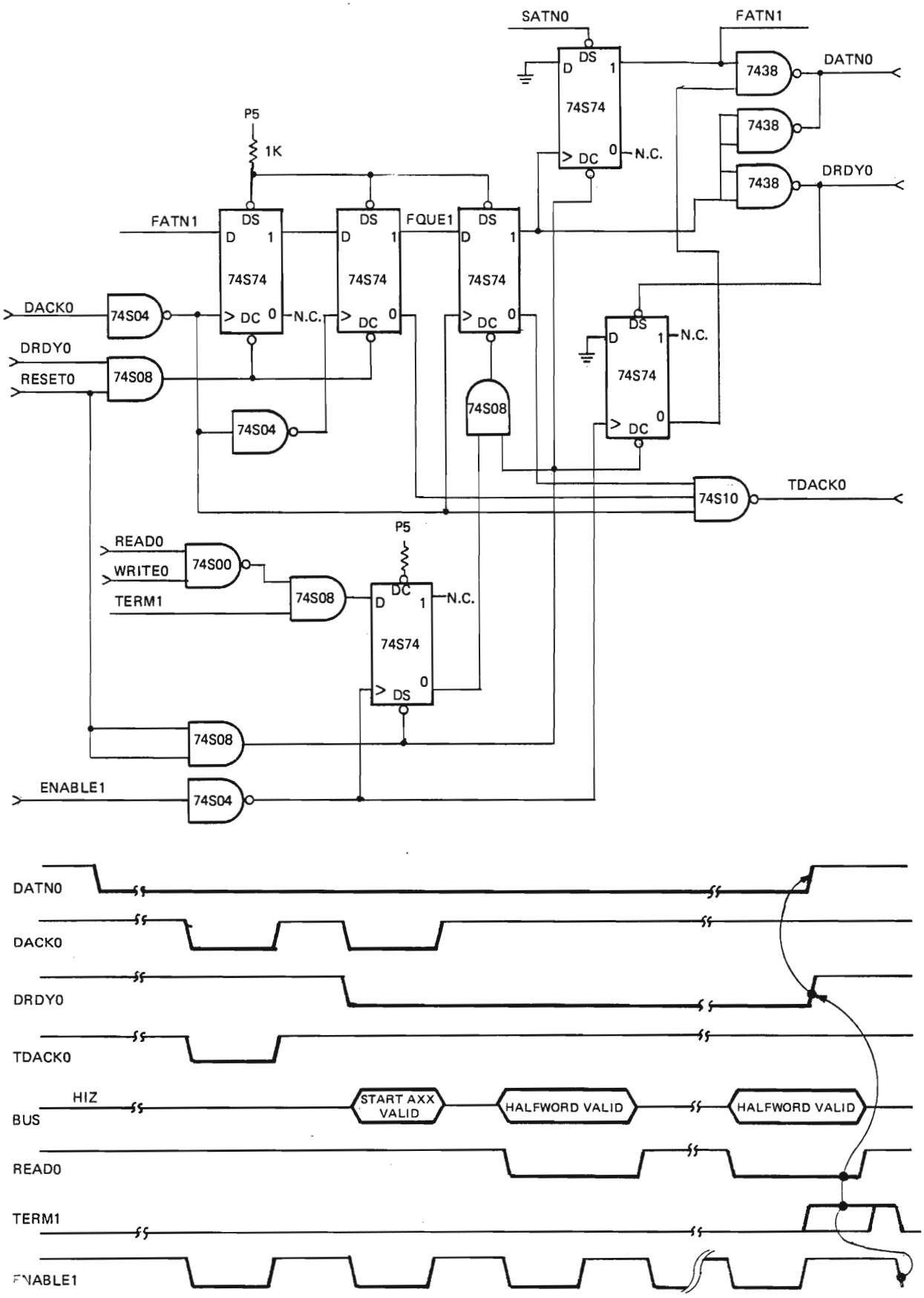


Figure 8-18. Logic And Timing For Model 5/16 Micro-DMA

MULTIPLEXOR BUS

The Multiplexor Bus is a byte or halfword oriented I/O system which can communicate with up to 191 peripheral devices. Sixty-four of the possible 255 device address (Address X'00' is not used) are represented by the Micro I/O Bus. INTERDATA's complete line of peripheral equipment which does not require a Selector Channel can be interfaced to the Multiplexor Bus. The Multiplexor Bus comprises 28 lines – 16 bi-directional data lines, 7 control lines, 4 test lines, and 1 initialize line shown in Table 8-7.

TABLE 8-7. MULTIPLEXOR BUS LINES

FUNCTION	DESIGNATION	DIRECTION
DATA LINES	D00:15	PROCESSOR ⟷ DEVICE
CONTROL LINES	SR DR CMD DA ADRS ACK CL07	PROCESSOR ⟶ DEVICE
TEST LINES	ATN SPATN SYN HW	PROCESSOR ⟵ DEVICE
INITIALIZE	SCLR	PROCESSOR ⟶ DEVICE

The following general definitions apply to the Multiplexor Bus lines.

Data Lines (D00:15)

The data lines are used to transfer an 8-bit byte or a 16-bit halfword of data between the processor and the device. An 8-bit device address is transferred from the processor to the device over data lines 08:15 when accompanied by the Address (ADRS) control line. An 8-bit command byte is transferred over D08:15 accompanied by the Command (CMD) control line. One byte or one halfword is transferred from the processor to the device accompanied by the Data Available (DA) control line. The device, in response to an Acknowledge (ACK) control line, sends an 8-bit address to the processor over D08:15, or 8-bits of status information over D08:15 in response to the Status Request (SR) control line. In response to the Data Request (DR) control line, the device sends either an 8-bit byte or a 16-bit halfword of data to the processor.

NOTE

The device always sends a Synchronize (SYN) signal to the processor after it has accepted an operation from the processor. The SYN signal is then removed immediately after the processor removes the control line.

Control Lines

SR	Status Request. The device controller returns device status on D08:15.
DR	Data Request. The device controller returns data to D08:15 or D00:15. If a halfword of data is presented, the Controller also activates the Halfword (HW) test line and returns data on D00:15.
ACK	Acknowledge. The interrupting device controller returns its address on D08:15 if ATN is active.
DA	Data Available. The processor presents data on D00:15 for transfer to the device. The device controller accepts the low byte or the entire halfword and responds with a SYN.
CMD	Command. The processor presents control information to the device on D08:15.
ADRS	Address. The processor presents an 8-bit address on D08:15.
CL070	Early Power Fail Warning. This control line is activated by the processor when a Power Fail condition is detected by the processor. This line is held active until the SCLR0 signal occurs.

Test Lines

ATN	Attention. Any device desiring to interrupt the processor activates the ATN line and holds this line active until an ACK is received from the processor The device controller must not deactivate ATN until the processor deactivates ACK.
SPATN	Special Attention. This interrupt line is provided for attachment of special event keys or clock devices. The microcode does not acknowledge this interrupt, but rather assumes the device address to be X'07'.
HW	Halfword. The HW line is activated by a halfword oriented device controller whenever it is communicating normally with the processor.
SYN	Synchronize. This signal is generated by the device to inform the processor that it has properly responded to a control line.

Initialize Line

SCLR	System Clear. This is a metallic contact to ground that occurs during Power Fail, Power Up, or Initialize.

NOTE

All control lines, except ACK, are connected in parallel to all devices. The ACK line is activated by the processor in response to an external interrupt and is connected in series with all devices. If no interrupt is pending in the first controller when the ACK signal arrives, the signal is passed in daisy-chain fashion to the next controller, and so on until it is captured by the interrupting controller. See definition of ACK.

Communication over the Multiplexor Bus is performed on a request/response basis where each sequence is controlled by the micro-program in the processor's Read-Only-Memory (ROM). A typical sequence to perform an I/O instruction with a device controller is:

1. The processor addresses the device controller by placing an 8-bit address on the data lines and activates the ADRS control line. The device controller whose address corresponds to the 8-bit address on the data lines responds by setting its Address flip-flop and returning SYN to the processor. (All other device controllers reset their Address flip-flops.) Once a device controller is addressed, it remains so until another device is addressed or until the system is initialized. The addressed device controller responds to subsequent activity on the Multiplexor Bus until another controller is addressed.

2. If the I/O instruction involves transferring data from the processor to the device controller, the Processor places the data on the data lines and activates the DA control line. The addressed device controller responds with a SYN after it has received the data, and the processor removes DA.

3. If the I/O instruction involves transferring data to the processor from the device controller, the processor activates the DR control line, and waits for the device controller to respond by placing the data on the data lines then activating SYN. When the processor receives SYN, it accepts the data and removes the DR.

4. In all cases, the device controller removes the SYN whenever the processor removes the control line.

The sequence described here is somewhat simplified for the sake of clarity. The exact sequence for each I/O instruction is listed later.

Whenever a device controller detects an extraordinary condition, it may interrupt the processor by activating the ATN test line. This may be done by any device controller at any time, provided that device interrupts are enabled, regardless of whether it is addressed or not. If interrupts are enabled in the Current Program Status Word, the processor responds to ATN by interrupting the currently running program and directing the processor to a new program (or a new micro-program) which identifies and services the interrupt as required.

MULTIPLEXOR I/O DEVICE CONTROLLER LOGIC DESIGN

This section describes the procedures to follow in designing device controllers which connect to the Multiplexor Bus. While it is impossible to describe all possible controllers, this section explains representative circuits in sufficient detail to facilitate design of most controllers.

Multiplexor Bus

The Multiplexor Channel is a byte or halfword oriented I/O system which communicates with up to 191 peripheral devices. The Multiplexor Bus consists of 27 lines: 16 bi-directional Data Lines, 7 Control Lines, 3 Test Lines, and an Initialize Line as described previously.

All busses are false type, i.e., low level is active, high level is inactive. The device controller circuits used to communicate with the Multiplexor Bus are shown in Figure 8-19.

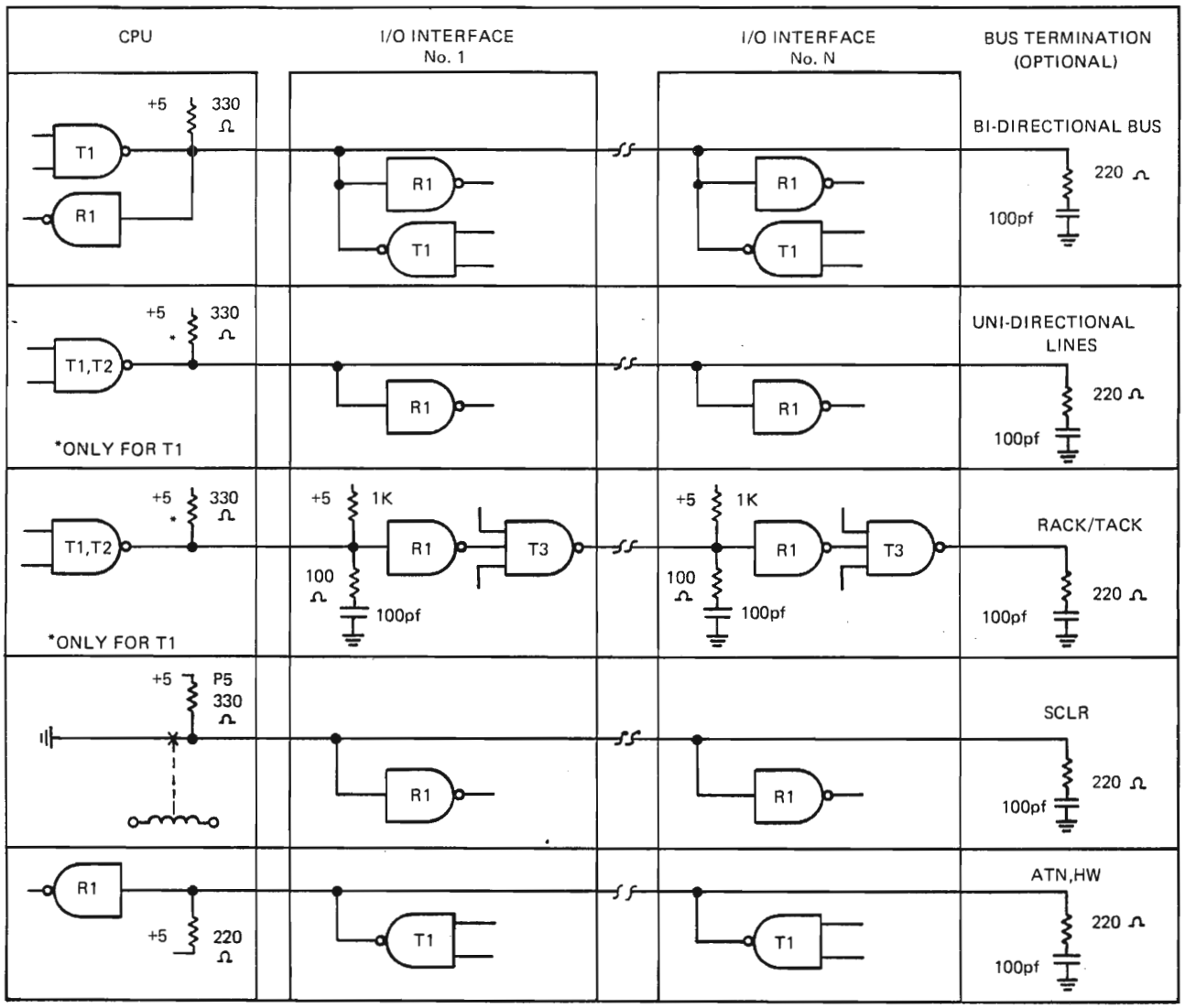
In a typical case, a device controller receives an 8-bit address, an 8-bit Command Byte, and either an 8-bit data byte or a 16-bit data halfword from the processor over the 16 bi-directional Data Lines (D00:15). When only a byte of data is transferred, that byte is passed over the lower eight Data Lines (D08:15). The load resistors for all lines in the Multiplexor Bus are located in the Processor.

Each device controller is permitted one TTL load, 2 milliamperes maximum, on any of the 16 bi-directional Data Lines, the 7 Control Lines, or the single Initialize Line. Each device controller is permitted one high power open collector TTL OR-tied onto each of the 16 bi-directional Data Lines and each of the 3 Test Lines. (The open collector bus driver must be capable of sinking 48 milliamperes at 0.5 VDC maximum V_{CE}. (See Figure 8-19.)

Multiplexor Bus Loading Rules

The Multiplexor Bus, generated at the processor, is capable of driving a total of 16 I/O controllers.

The Multiplexor Bus Buffer or I/O Bus Switch extend the bus drive by regenerating the bus. These devices all represent one load to the bus they are driven by. Each of these devices is capable of driving up to 16 loads.



TRANSMITTER CHARACTERISTICS

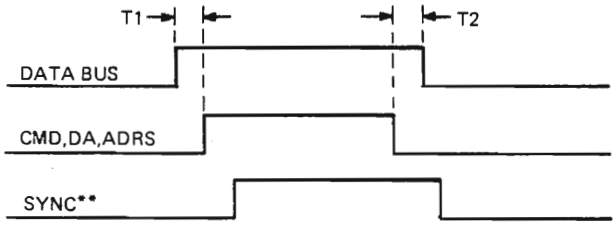
PARAMETER	T1	T2	T3
VOL, LOW LEVEL OUTPUT	0.4V@48 ma.	0.4V@48 ma.	0.4V@20 ma.
VOH, HIGH LEVEL OUTPUT	5.5V max.	2.4V min.	2.4V min.
IOH, HIGH LEVEL LEAKAGE, VOH=5.5V	250 ua.	N.A.	N.A.
tPLH, DELAY, LOW TO HIGH	22ns max.	22ns max.	10ns max.
tPHL, DELAY, HIGH TO LOW	18ns max.	18ns max.	10ns max.

RECEIVER CHARACTERISTICS

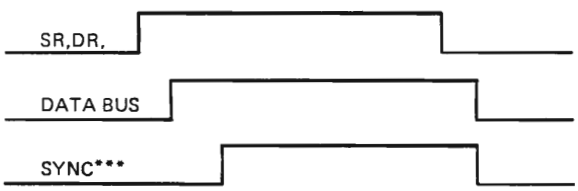
PARAMETER	R1
V_{IH}, INPUT THRESHOLD,HIGH	2.0V MIN
V_{IL}, INPUT THRESHOLD,LOW	0.8V MAX
I_{IH}, INPUT LEAKAGE,HIGH@5.5V	1mA MAX
I_{IL}, INPUT LOW LEVEL, V_{IN}=0.4V	2mA MAX
t_{PLH}, DELAY,LOW TO HIGH	15NS MAX
t_{PHL}, DELAY,HIGH TO LOW	12NS MAX

MAXIMUM BUS LOAD (DATA & CONTROL) : 29 mA
MAXIMUM BUS LOAD (TACK, SCLR) : 2 mA

T1 (TYPICAL) 7438 T3 (TYPICAL) 74H10
T2 (TYPICAL) 7437 R1 (TYPICAL) 74H04



CPU OUTPUT TIMING



INTERFACE OUTPUT TIMING

T1 = T2 ≥ 75 ns

** RETURN SYNC AFTER DATA HAS BEEN ACCEPTED

*** RETURN SYNC AFTER DATA HAS BEEN PRESENTED

Figure 8-19. I/O Interface Transmit and Receive Characteristics

Multiplexor Bus Length Restrictions

The processor's Multiplexor Bus must be complete within the processor chassis and two adjacent 7" expansion chassis. The Multiplexor Bus normally may not be extended to any expansion chassis by use of a cable longer than 4". For this configuration, a bus buffer must be used to extend the bus.

Private I/O Busses, which are generated by a Bus Buffer or I/O Bus Switch, must be complete within the chassis the bus is generated in, and a maximum of two 7" expansion chassis. Any private bus may be extended by no more than one 36 inch cable plus an additional cable with a maximum length of 4 inches.

Multiplexor Bus Terminators

I/O Terminators (INTERDATA Part Number 35-433 or 35-434) must be installed at the end of the Multiplexor Bus. If the Multiplexor bus is present on both connector 0 and connector 1 of a chassis, terminators must be installed on both sides. If a given Bus extends no more than 4 adjacent slots in a single chassis, the I/O terminator is optional; however, a terminator should be installed if reflection problems are noted.

Device Controller Addressing

Refer to Figure 8-20 during the following description. When a device controller is addressed, the 10-bit address code is placed on the Data Lines (D060:150). The least significant 8-bits are switch selectable with INTERDATA Part Number 33-032 switches. Bits D06 and D07 generate CHNAD1. If these bits are both ZERO, CHNAD1 is high. This is ANDed with the switch selected 8-bits. The resultant logic function is strobed with ADRS to set or reset the AD flip flop.

The Synchronize (SYN) signal is returned to the processor, during the presence of ADRS1. The ONE output from the Address flip-flop, AD1, is used to gate all other I/O control lines to the device controller. When a different device is addressed, the ADRS1 strobe line resets the AD flip-flop thus inhibiting further communications between the Processor and controller. Thus, only one device controller may be addressed at any time. During the address cycle, only the device which was addressed returns a SYN.

NOTE

The device controller must be designed such that when some other device is addressed, the previously addressed controller clears its Address flip-flop in no more than 350 nanoseconds after the receipt of ADRS. Otherwise the system could have two devices addressed simultaneously.

The device controller logic must delay SYN until it has reacted to the Multiplexor Bus control line, however, unnecessarily long delays reduce the system input/output operation.

NOTE

If the device controller is a 16-bit halfword-oriented controller, the Halfword Enable line (HW0) is activated while its Address flip-flop is set, if the interface is strapped for Halfword mode. (See Figure 6.) The HW0 is used by the processor to determine if the device is capable of sending or receiving 16-bit halfword data in parallel.

Interrupt Control

Figure 8-20 shows a complete general purpose interrupt and interrupt acknowledge logic system. When an interrupt is generated, the ATN flip-flop is set. The output from the ATN flip-flop generates an Attention signal (ATN0) to the processor. The program responds with an Acknowledge Interrupt signal, which is received by the controller as Receive Acknowledge (RACK). Since the ATN flip-flop was set prior to receiving RACK, ATSYN0 goes low, and the Transmit Acknowledge (TACK0) output is held high. Thus, the Acknowledge Interrupt signal is captured by the interrupting controller, and is prevented from propagating to the next device as Receive Acknowledge (RACK0). The ATSYN0 signal active causes the device address to be returned on the Multiplexor Bus Data Lines D080-150 when input number three is selected on the 4:1 multiplexor, and then causes a SYN to be returned to the Processor. On receiving the SYN0, the processor deactivates RACK0, causing the ATN flip-flop to reset, deactivating ATN0. The device controller must not deactivate ATN0 until the processor deactivates RACK0.

NOTE

If the ATN flip-flop is reset, the RACK1 signal passes through the device to TACK0, and on to the next device.

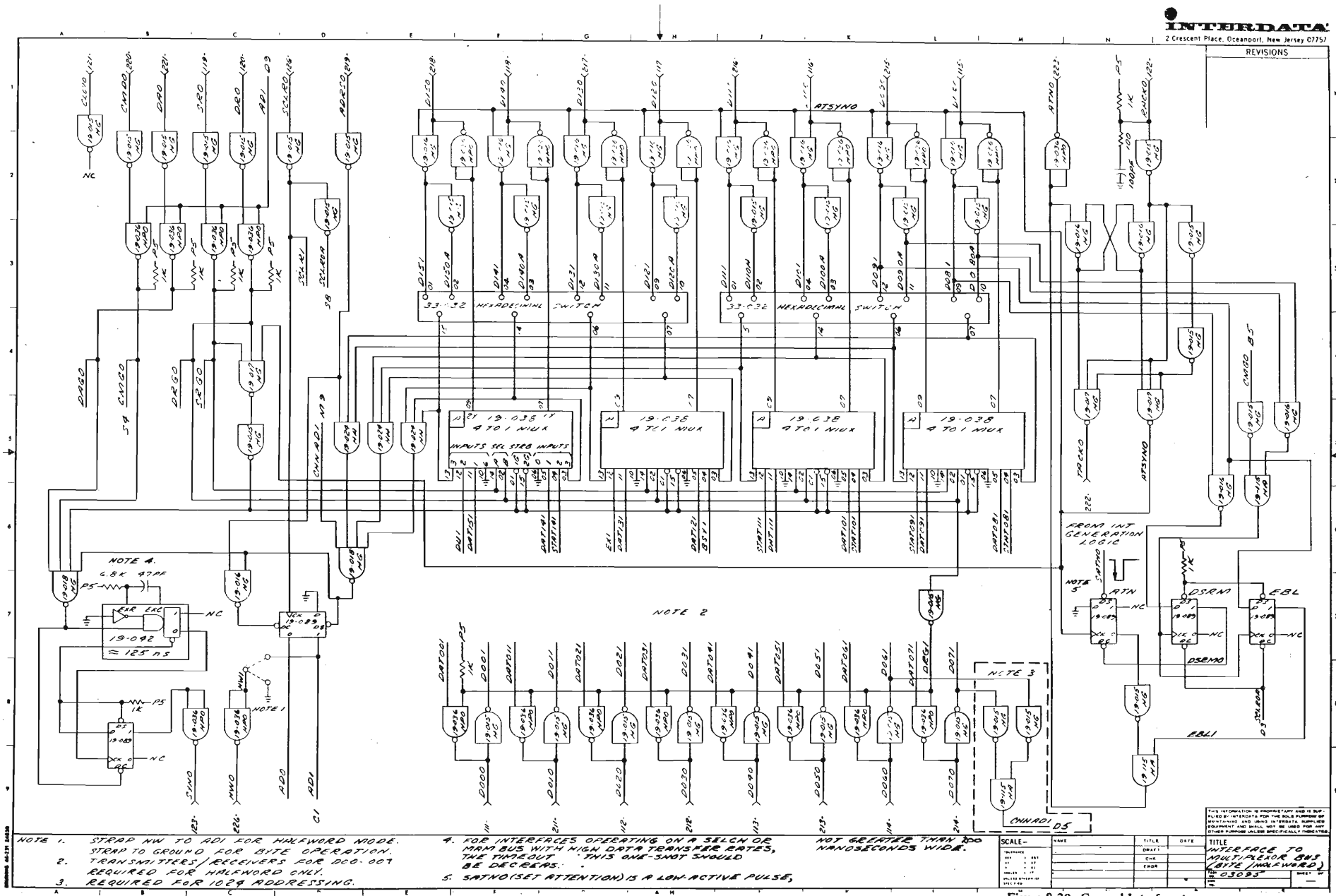


Figure 8-20. General Interface to Multiplexor Bus (Byte or Halfword Oriented)

The EBL and DSRM flip-flops provide control over the interrupt ATN flip-flop and the ATN control line to the processor. Two bits of the Command Byte (Bits 8 and 9) are decoded as one of three possible functions: Enable, Disable, or Disarm.

A command with Bit 8 reset (ZERO) and Bit 9 set (ONE) causes interrupts to be Enabled. The output from the DSRM flip-flop (DSRM0) is high, allowing the ATN flip-flop to be set by a SATN0 pulse, and the output from the EBL flip-flop (EBL1) is high, enabling the output from the ATN flip-flop to activate ATN to the Processor.

A command with Bit 8 set (ONE) and Bit 9 reset (ZERO) causes interrupts to be Disabled. The output from the DSRM flip-flop (DSRM0) is high, allowing the ATN flip-flop to be set by a SATN0 pulse, however, the output from the EBL flip-flop (EBL1) is low, preventing the ATN flip-flop from activating ATN. Thus, interrupts are allowed to queue, but are not passed to the Processor.

A command with both Bits 8 and 9 set (ONEs) causes interrupts to be Disarmed. The output from the DSRM flip-flop (DSRM0) is low, forcing the clear input to the ATN flip-flop low, thus disallowing an interrupt to queue. Note that the output from the EBL flip-flop is a don't-care condition when DSRM0 is low.

A command with both Bits 8 and 9 reset (ZERO) causes no change in the interrupt controls. This condition prevents the command pulse from loading the DSRM and EBL flip-flops.

As described previously, RACK from the processor is the Interrupt Acknowledge (ACK) signal. This line breaks up into a series of short lines to form the daisy-chain priority system. The RACK signal must pass through every device controller that is equipped with Interrupt Control circuits. This implies that the device controller's interrupt priority in the INTERDATA system is determined by the physical location within the system. That is, the controller nearest to the Processor (first in line in the ACK daisy-chain) has the highest priority.

At any I/O slot, the Received ACK (RACK0) appears at Pin 122-1 and the Transmitted ACK (TACK0) at Pin 222-1. The daisy-chain bus is formed by a series of isolated lines which connect Terminal 222-1 of a given position to Terminal 122-1 of the next position (lower priority). On unequipped positions, a back panel jumper shorts 122-1 and 222-1 on the same connector to complete the bus. Back panels are wired with jumpers on all I/O positions. Whenever a card chassis position is equipped with a device controller that has interrupt circuits, the jumper from 122-1 to 222-1 must be removed from the back panel at that position.

For controllers that occupy several positions, the jumper is removed only at the position where the controller board has interrupt circuits.

Multiplexor Bus Wiring

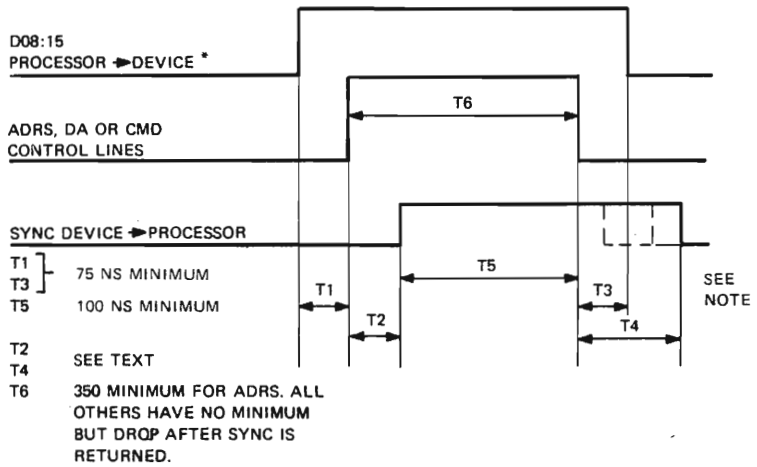
Wiring for the Multiplexor Bus is identical in the processor and Expansion chassis. Each card position contains two connectors with the Multiplexor Bus wired to each at pin positions indicated in Figure 8-33.

Multiplexor Bus Timing

Both the Input and Output operations on the Multiplexor Bus make use of request/response signaling. This allows the system to run at its maximum speed whenever possible, but permits a graceful slowdown if the characteristics of a particular device controller require signals of longer duration. Device controller designs should keep Multiplexor Bus usage as fast as possible, consistent with practical circuit margins. Doing this assures the fastest computer input/output operation when a system is configured with many peripheral devices.

Timing for typical Output operations is shown on Figure 8-21. On the Output operation, the processor places a signal on the data lines followed by an appropriate control line signal. This stagger (T1) varies but it is guaranteed to be at least 75 nanoseconds. When the device controller has received the Output Byte, the SYN signal is returned,to the processor, which terminates the control line signal. Realizing that T5 is 100 nanoseconds minimum, the SYN delay T2 should be only long enough to guarantee proper reception of the Output Byte. The control line/data line removal time (T3) is important where single-rail to double-rail operation is used, e.g., the ADRS flip-flop of Figure 8-20. A minimum of 75 nanoseconds is guaranteed for T3. For SYN generation as per Figure 8-21, the control line signal is DC coupled through the gates to form the SYN signal. The SYN removal time (T4) should be minimized. This delay should not be extended unnecessarily since the processor does not begin another Input/Output operation until SYN is removed.

It should be emphasized that the times shown on Figure 8-21 are defined for signals on the Multiplexor Bus. Within a given device controller, one signal may flow through more gates than another signal and these delays must be considered.



* THE PROCESSOR MUST DEACTIVATE THE DATA LINES MIN. 100 NS BEFORE ACTIVATING ANY CONTROL LINE FOR THE NEXT I/O OPERATION.

Figure 8-21. Multiplexor Bus Output Timing

NOTE

The time between the completion of one I/O operation and the start of the next I/O operation is undefined. In certain cases, there is no delay between consecutive I/O operations. The device controller must be ready to respond immediately.

Timing for typical Input operations is shown on Figure 8-22. For the Input operation, the processor activates a control line. The currently addressed device controller should gate signals to the data lines as soon as possible to keep T1 at a minimum. The SYN delay (T2) must guarantee that the Input Byte is on the data lines, considering the slowest data gates and the fastest SYN gates. The processor removes the control line signal when SYN is received with a minimum delay (T4) of 100 nanoseconds. With SYN and the byte gate DC coupled to the control line, the removal delay (T3) is the sum of the corresponding gate delays. The processor considers the operation complete when SYN deactivates.

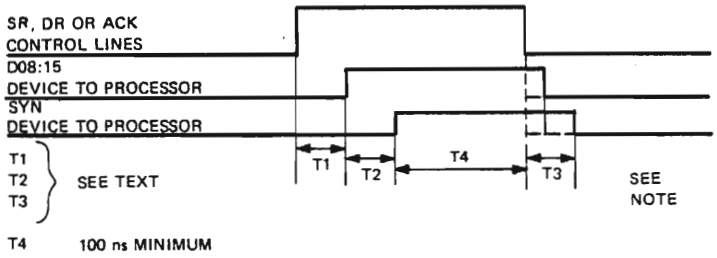


Figure 8-22. Multiplexor Bus Input Timing

NOTE

The time between the completion of one I/O operation and the start of the next I/O operation is undefined. In certain cases, there is no delay between consecutive I/O operations. The device controller must *never* hold the data lines any longer than necessary.

When the control signal is ACK, the delay (T1) includes the cumulative gate delays (see Figure 8-20) for all the device controllers between the responding device controller and the processor. This is less than the processor time-out even with the maximum limit of 255 device controllers.

NOTE

With a SYN delay of 50 nanoseconds, device controllers must be designed to accept a minimum width of 170 nanoseconds on all control line pulses except ADRS which is guaranteed to be 350 nanoseconds minimum. The SYN delay in the device controller may be increased to effectively lengthen the control line pulses if it is absolutely necessary. It is essential to realize that, after the processor initiates an I/O operation, the processor does nothing until the SYN signal is returned by the device controller; one or more processor clock cycles may be skipped if necessary and the data throughput decreased proportionally. While this may not affect a particular device controller, the overall system performance is degraded. Furthermore, if a device controller fails to respond with a SYN within the time out period of approximately 15 to 35 microseconds, the processor aborts the I/O operation and removes the control line signal.

General Multiplexor Bus Interface

Figure 8-20 illustrates a general interface to the Multiplexor Bus which may be used when designing custom device controllers, either 8-bit byte or 16-bit halfword oriented. (If an 8-bit byte oriented interface is being designed, gates connecting to D001:071 and to DAT001:071 can be eliminated.) The figure illustrates an interface with the 74H logic family; however, other logic families may be used provided they conform to the characteristics in Figure 8-19. The address straps can be hardwired by the user for any device number from X'002' to X'3FF' with the exception of X'007', which is reserved for the manually selected clock. The user can use the Gated Status Request (SRG0) or the Gated Data Request (DRG0) control lines to gate status or data from appropriate points in his logic. Data from the processor is available to the user's circuits, double rail, at the points labeled D001:151 and D000A:150A. The user can use the Gated Data Available (DAG0) and the Gated Command (CMG0) control lines to gate the data from the processor to appropriate points in his logic. The delay of the SYN0 signal should be arranged such that it is the minimum delay necessary for the custom controllers to function properly, per the Multiplexor Bus Timing Section.

Additional Requirements for I/O Interface

1. +5 volts ± 5% is assumed for all interface designs. Other voltages from the system power supply should not be used in the design of the interface. If voltages other than +5 volts are required in the design of the interface, an on-board DC to DC converter may be used. (See DC-DC Converter Specifications.)

2. When appropriate, use 16-bit I/O for devices.

3. In general, all unused inputs of the logic packages, flip-flop (F/F), counter, etc., should be terminated to +5 volts through a pull-up resistor (1K ohm). A maximum of 25 74H gate loads can typically be connected to one pull-up resistor.

4. Avoid the use of capacitors for delaying the edge of a logic signal. Use a one-shot 19-042, 19-031, etc., or delay lines for delays.

5. The interface should be designed with the assumption that all data transferred to and from the processor is valid when the BSY status bit (BSY1) changes from a logic 1 to a logic 0.

6. Avoid the use of RC networks for differentiation of logic signal edges. Use a one-shot 19-042, 19-031, etc., for differentiation. The timing components used on these one-shots must be immediately adjacent to the IC.

7. High-frequency decoupling capacitors should be located adjacent to ICs as required. The number of decouplers required is a function of the logic family being used – 74S, 74LS, etc. As a minimum, there should be one decoupler for each two ICs. Provide extra decoupling and/or isolated P5 and GND connections for high-current driver ICs.

8. It is good design practice, in cases where address, command, or data is loaded from the Multiplexor Bus into an edge-triggered flip-flop, to accomplish the loading operation on the leading edge of the appropriate control signal (i.e., coincident with the high-to-low transition of ADRS0, CMD0, or DA0).

9. Device controllers which have more than one device address, should have contiguous addresses (i.e., X'A0', 'A1', 'A2', 'A3').

10. Device controller addresses must not be restricted to a single fixed address or fixed range of addresses and must decode ten address bits.

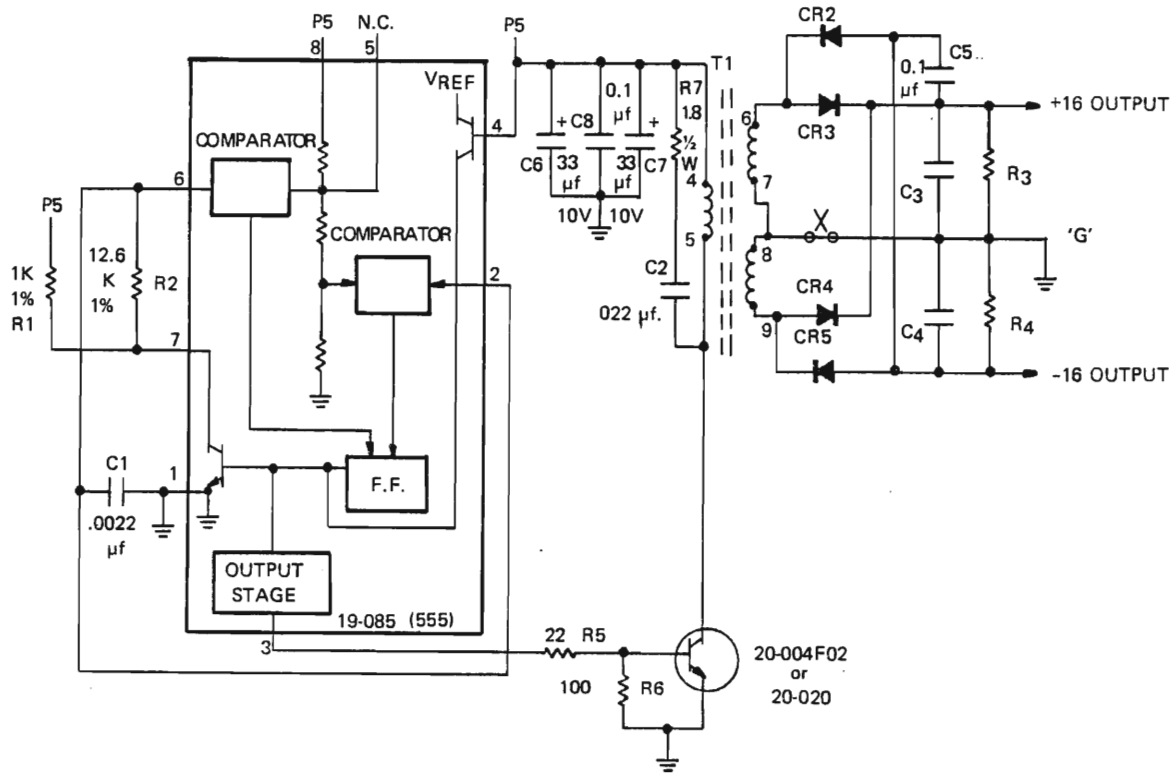
11. The 15" x 15" controllers are limited to an absolute maximum of 13 amperes of P5 power, and the 7" x 15" controllers are limited to an absolute maximum of 7 amperes of P5 power. These limitations are imposed by the standard INTERDATA back panel connector.

12. In cases where command information is loaded into a level-triggered or R-S flip-flop by the presence of CMG0, the I/O interface must allow for wide variation of the width of the CMG0 pulse.

DC-DC Converter Specifications

Functional Use

The circuit shown in Figure 8-23 may be used to provide a means of developing low current voltage levels to operate Teletypewriter (TTY) I/O interfaces, operational amplifiers, etc., from on-board +5 volts logic power.



NOTES

1. ALL DIODES 23-001
2. ALL RESISTORS ¼W. + 5% TOLERANCE UNLESS OTHERWISE SPECIFIED
3. $C_3 = C_4 = 15\ \mu f\ 20V$
4. $R_3 = R_4 = 3.9K$
5. T1 = 30-020

Figure 8-23. DC/DC Converter

General Description

The circuit is comprised of a 19-085 IC timer connected as a 24K Hz oscillator having a square wave output, which drives a switching transistor feeding a step-up transformer. The output from the transformer is rectified and filtered to provide +16 and -16 volts at 50 milliamperes each, or optional levels as listed in the following specifications.

Specifications

Input +5 VDC ±5% @ 200 to 500 milliamperes depending on output load.

Output: Standard circuit supplies + and - 18 volts at no load (with 5 volt input) decreasing to + and -16 volts at full load (100 milliamperes total sum of either or both polarity).

Output Power Options

1. Option 1 supplies +18V only at no load, decreasing to +16 volts at full load (100 milliamperes).

2. Option 2 supplies -18V only at no load, decreasing to -16 volts at full load (100 milliamperes).

3. Option 3 supplies +36V only at no load, decreasing to +32 volts at full load (50 milliamperes).

4. Option 4 supplies -36V only at no load, decreasing to -32 volts at full load (50 milliamperes).

Output Ripple: Standard circuit and Options 1 and 2 = 150 millivolts peak-to-peak (p/p) maximum – Options 3 and 4 = 300 millivolts p/p maximum.

Line Regulation: Output varies directly as input.

Operating Frequency: Approximately 24K Hz.

Start-up Time: Output voltage is up to nominal level within 200 milliseconds after power is applied.

Output Configuration Options. Refer to Figure 8-23

1. For single polarity +16 volts output (full load functional variations), delete CR2, CR5, C4, and R4.

2. For single polarity -16 volts output (full load), delete CR3, CR4, C3, and R3.

3. For single polarity +32 volts output (full load), open ground connection to transformer at point X, delete ground point G, and ground -16 volts output.

4. For single polarity -32 volts output (full load), open ground connection to transformer at point X, delete ground point G, and ground +16 volts output.

NOTE

If voltage levels other than those supplied by the converter are needed, or close regulation is required, an IC regulator such as a 19-094 can be attached to the output of the DC to DC Converter.

The recommended format for the Status Byte of an I/O device controller is shown in Figure 8-24. There may be some exceptions to the status format, depending upon the function of the I/O device controller.

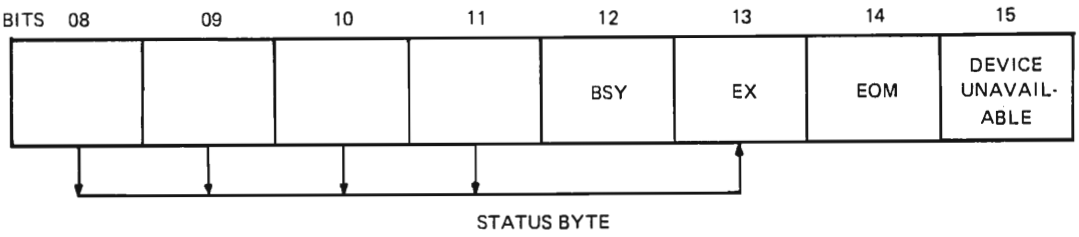
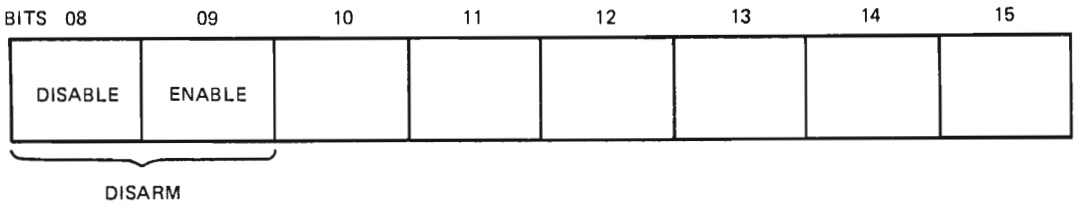


Figure 8-24. Status Byte

At least one of Bits 08:11 shall be set when the Examine bit (EX) is set. In many cases, all of the upper 4-bits (Bits 08:11) of the Status Byte may be used to set the Examine bit depending upon the function of the device controller.

The standard Output Command Byte is shown in Figure 8-25. Bits 10:15 are used for control functions of the device controller. Bits 08 and 09 must be used to control the interrupt circuit of the device controller. The meaning of Bits 08 and 09 are as shown on Figure 8-25.



BITS		MEANING
08	09	
1	0	DISABLE INTERRUPT FUNCTION (QUEUE INTERRUPTS)
0	1	ENABLE INTERRUPT FUNCTION (PASS INTERRUPTS TO PROCESSOR)
1	1	DISARM INTERRUPT FUNCTION (DO NOT QUEUE INTERRUPTS)
0	0	NO CHANGE IN THE INTERRUPT CONTROLS

Figure 8-25. Command Byte

Data and Status Input

Data

Figure 8-20 shows how a byte or halfword of data may be read into the processor. When the byte-oriented device controller is addressed, AD1 is high, enabling the Data Request (DR) control line from the Processor. (The HW1 is strapped to ground for byte operation.) The DR enables the data byte onto the eight bottom Data Lines D080:150. If a halfword-oriented device controller is addressed, 16 bits are gated onto Data Lines D000:150 since the halfword input is strapped to a high AD1 output from the Address flip-flop, enabling the Halfword mode. A system requirement is that the addressed controller must respond to all control lines (i.e., Data Request) with a SYN.

Status

Figure 8-20 shows how a byte of status may be read into the processor. When the byte or halfword oriented device controller is addressed, AD1 is high, enabling the Status Request (SR) control line from the processor. Open collector gates are used for OR tying the 4:1 multiplexor onto the eight Data Lines (D080:150).

The device controller logic should place a high on BSY1 until the data is ready. The processor may now be synchronized to the device data rate by testing the device status until the Busy bit is low. When the Busy bit is low, the program may transfer data. Device synchronization can also be achieved by generating an interrupt when the data is ready.

The End of Medium (EOM) bit is normally placed active at the termination of the device medium, such as End of Tape. The Device Unavailable (DU) bit, when active, typically signifies that device power is not turned on.

The Examine Status (EX) bit is used to signify other appropriate device conditions. In this case, the user assigns D08 through D11 to appropriate conditions, such as Parity Error, etc.

It is appropriate to note here that the Busy Status is unconditionally defined such that data cannot be transferred unless Busy is inactive. The remaining status bits are defined as required by the device controller. Not all device controllers require all eight status bits.

Data and Command Output

Data

Figure 8-20 shows how a data byte may be output from the processor. The buffered true and false Data Lines (D081:151 and D080:150) connect to the set and reset inputs of the Data Register.

When the device is addressed, AD1 is high, enabling the control line DAG0 to return the SYN signal to the processor.

Command

The command lines are shown on Figure 8-20 as being used to enable/disable interrupts, etc.

Again, note that definition of the command bits is a function of the device controller only. Not all device controllers require eight separate bits.

Byte-Oriented Device Controller Design

A byte-oriented device controller may be designed to accommodate Halfword Data Transfer instructions (RH/RHR and WH/WHR). This allows slightly more efficient I/O programming (one user-level instruction instead of two) for each two bytes of data transfer. To accommodate these instructions, the device controller must be designed to transfer data in accordance with the I/O sequence shown on Figure 8-26.

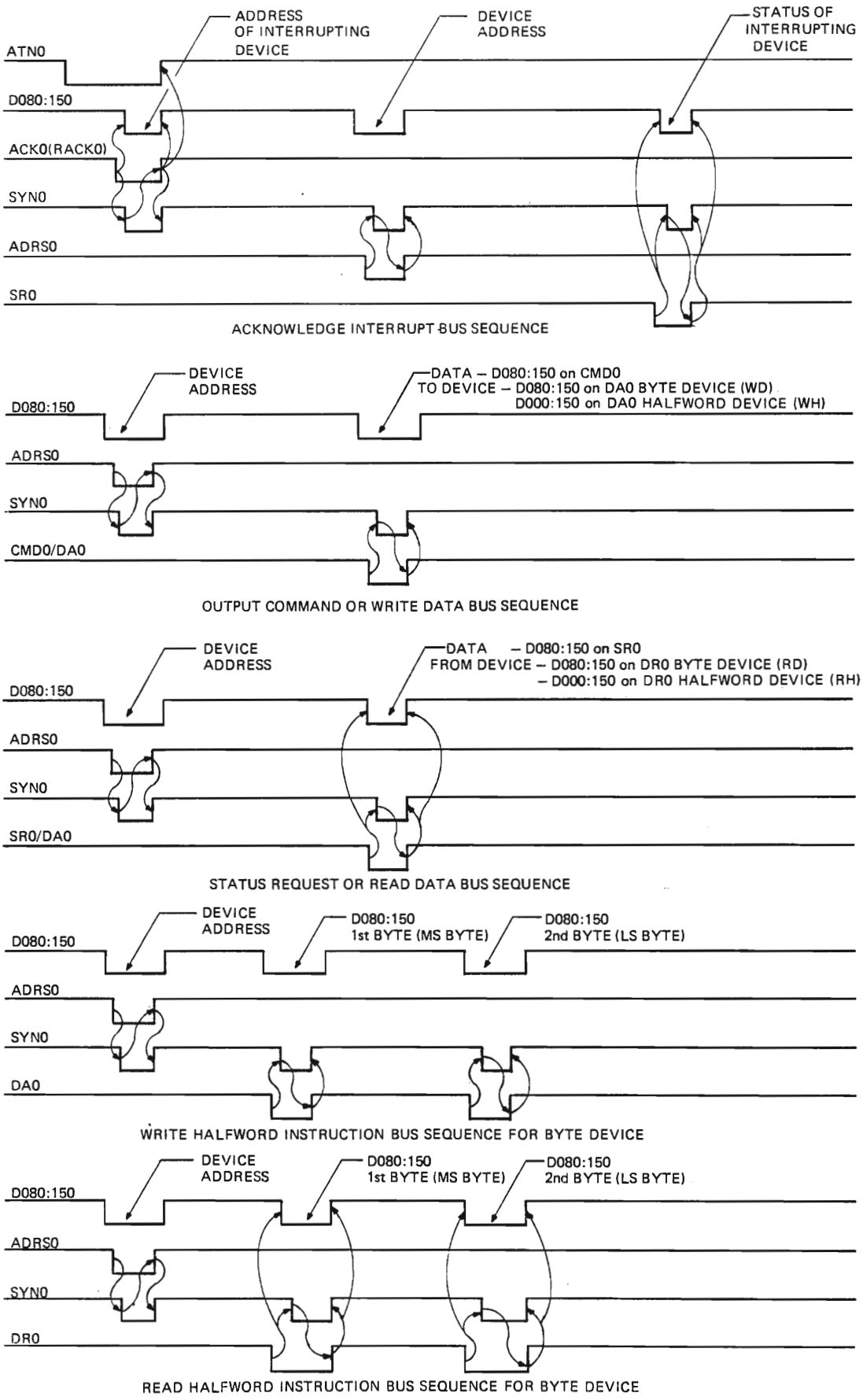


Figure 8-26. Bus Sequence For Byte Or Halfword Device

I/O Bus Sequence and Timing

The standard I/O Bus sequence and timing for the I/O Instruction Set is shown on Figures 8-26 through 8-30 for reference. reference.

NOTE

The Busy Status bit must always be set only on the trailing edge of DA0 or DR0.

Data Transfer

The data transfer sequence between the processor and the device controller in the Block Transfer mode (Read Block/Write Block) is shown on Figure 8-27 for reference.

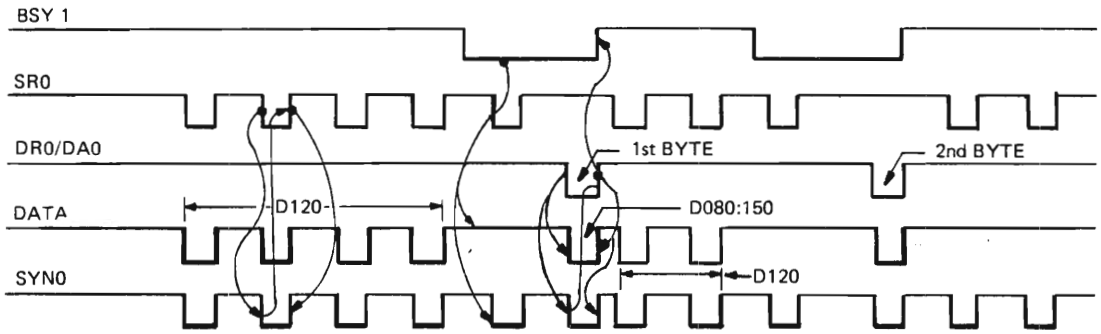


Figure 8-27. Read/Write Data Transfer Bus Sequence

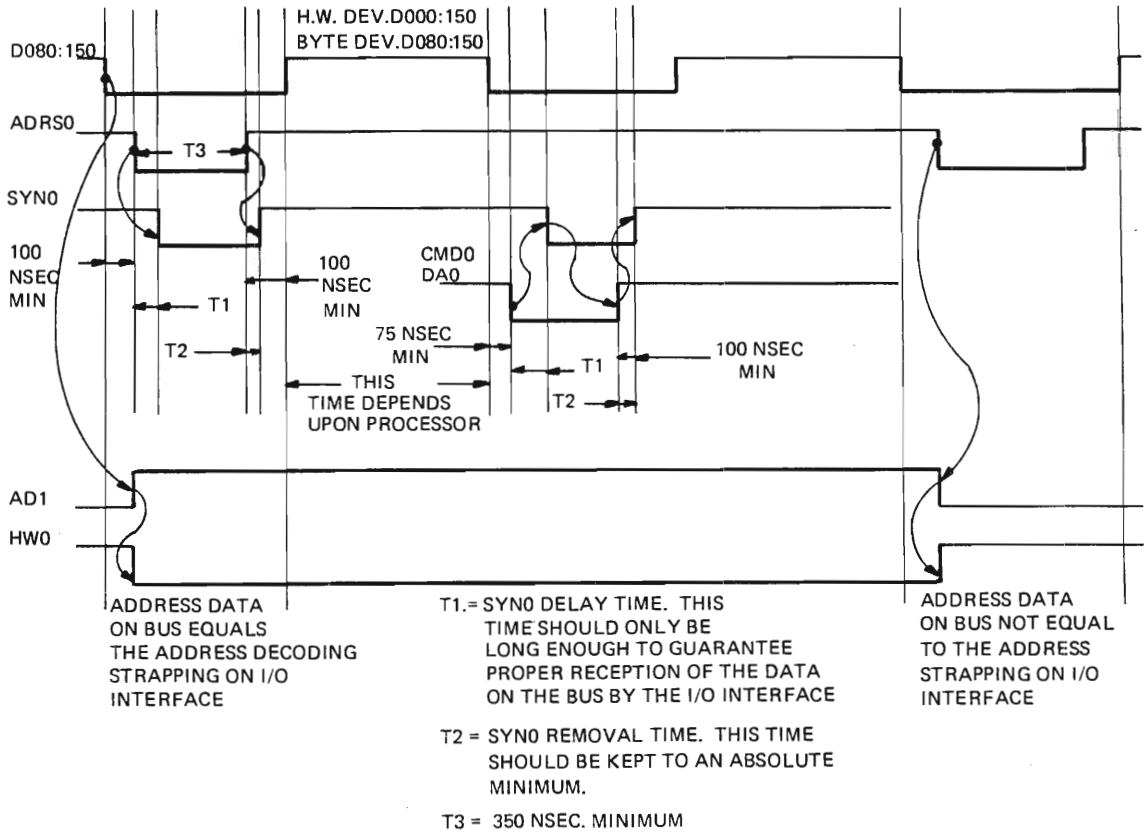


Figure 8-28. Address and Data Transfer Timing Between Processor and I/O Device Interface (Command and Data Available)

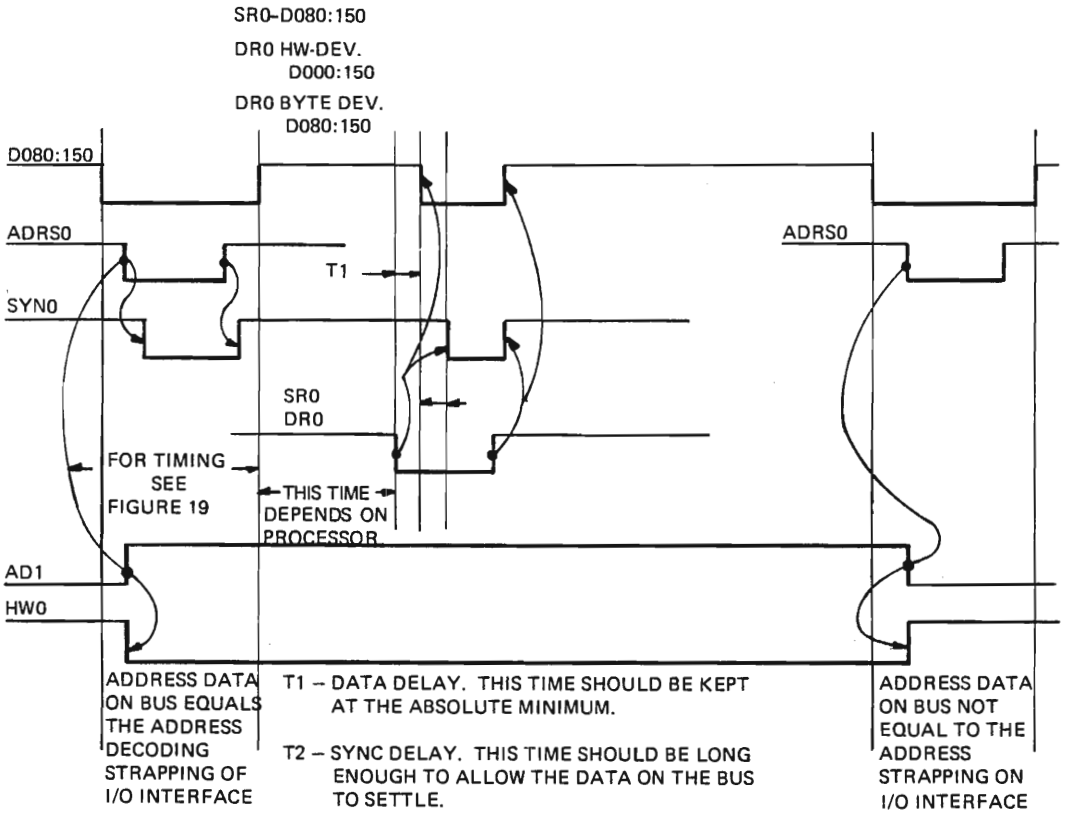


Figure 8-29. Address and Data Transfer Timing Between I/O Interface and Processor (Data Request and Sense Status)

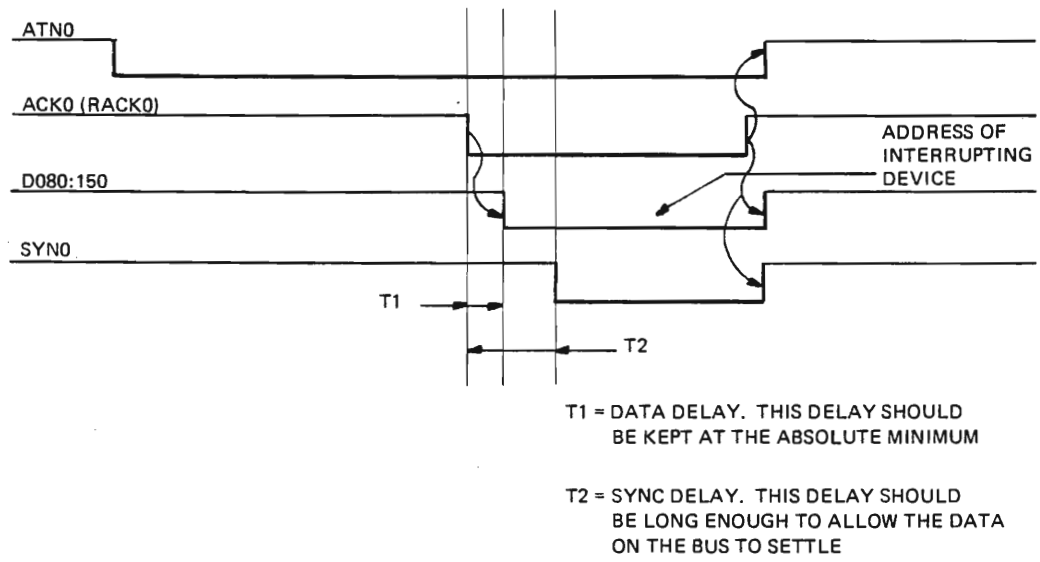


Figure 8-30. Interrupt Timing

MULTIPLEXOR I/O INTERFACE PHYSICAL PACKAGING, CABLING, AND CONNECTIONS

The I/O interface between any peripheral device and the processor Multiplexor I/O Bus uses 15" x 15" printed circuit boards or 7" x 15" printed circuit half-boards, as required. Hereafter, these boards are referred to as 15" boards or 7" half-boards respectively. The size of the printed circuit board used in an I/O interface depends upon the amount of logic required in its design.

7 Inch Half-boards

Two 7" half-boards can be inserted into a 15" chassis via the 16-398 Half-Board Adapter Kit (see Figure 8-31). The 16-398 Half-Board Adapter Kit may contain two active 7" half-boards or one active and one blank 7" half-board, depending on the system requirement. No wiring takes place between the boards and the adapters. The adapters are designed such that the 84 pin connector on the board plugs directly into the back panel connector in the chassis.

The 7" half-board contains one 84 pin back panel connector to pick up the I/O Bus signals. For peripheral device connection, one front connector is provided. The number of pins used (up to 50) is dictated by the application. All the connections are mechanically mounted per drawing SK653. The 84 pin back panel connector and the front connector may be Connectors 1 (CONN1) and 3 (CONN3), or Connectors 0 (CONN0) and 2 (CONN2) respectively, depending upon which side of the 7" half-board is to be connected to the I/O Bus. Refer to Figure 8-31. Generally, if the I/O interface contains less than 60 ICs the I/O interface fits on one 7" half-board.

15 Inch Boards

Each 15" board may contain two 84 pin back panel connectors, labeled Connector 1 (CONN1) and Connector 0 (CONN0), to pick up the I/O Bus, and two 50 pin front connectors, labeled Connector 3 (CONN3) and Connector 2 (CONN2), for peripheral device connection. All the connectors are mechanically mounted per drawing SK653. Refer to Figure 8-32.

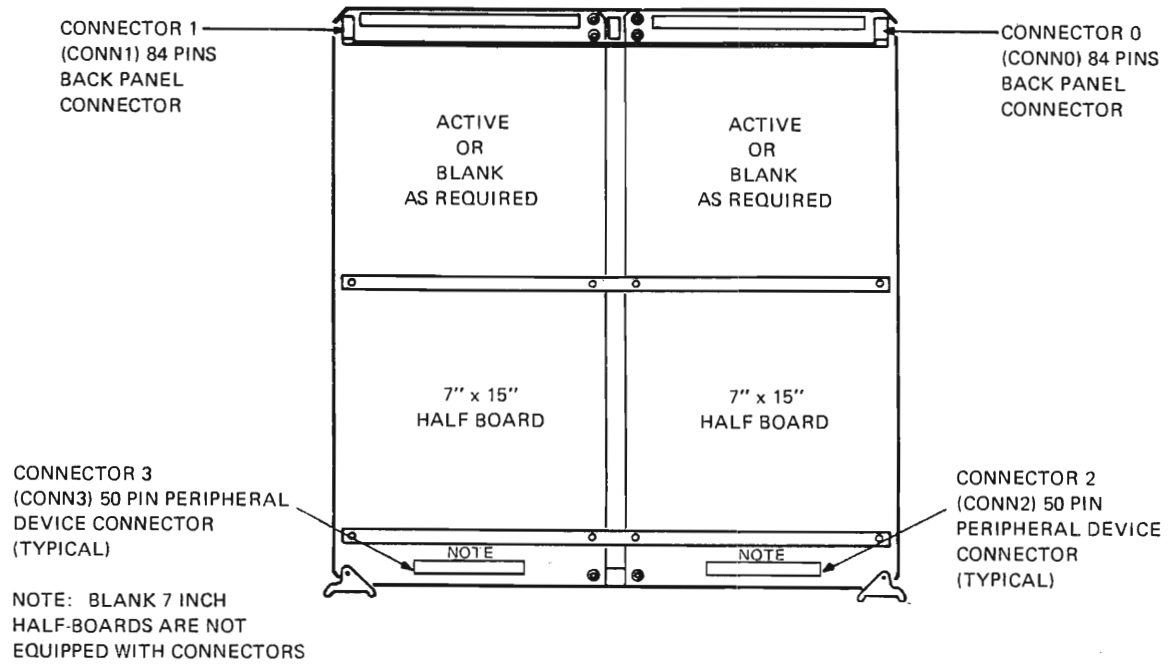


Figure 8-31. 16-398 Half Board Adapter Kit

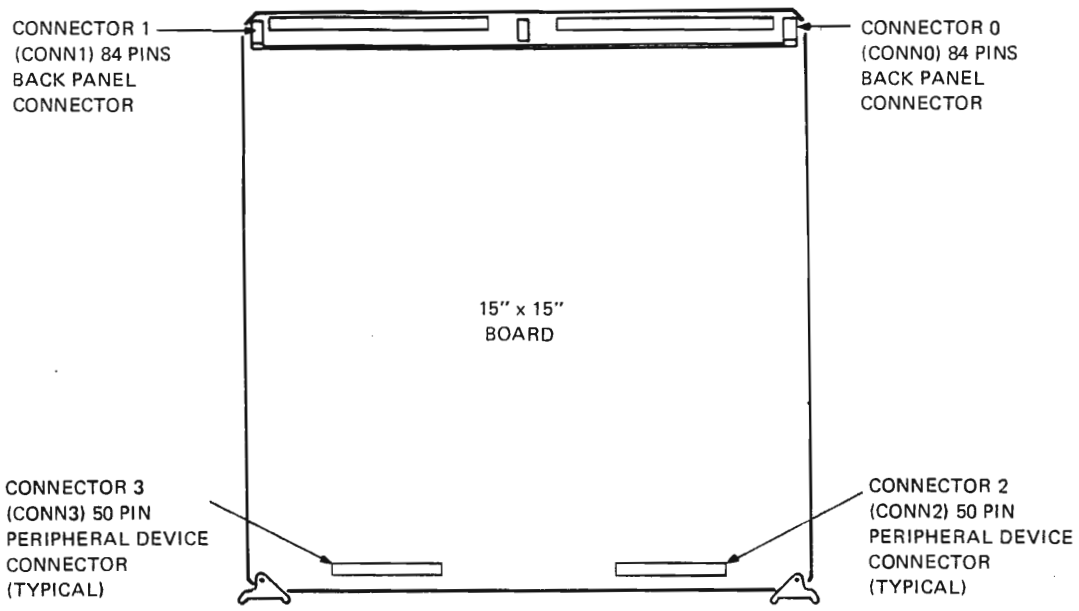


Figure 8-32. 15" x 15" Printed Circuit Board

If the proposed I/O interface design requires more than a 7" half-board, one or more 15" boards may be used. No back panel stitch pattern exists to interconnect multi-board designs. Thus, cables must be used for board-to-board interconnects.

When designing a 15" board, use either Connector 0 (CONN0) or Connector 1 (CONN1) to pick up the I/O signals from the back panel.

NOTE

Do not pick up some I/O signals from one connector and some from another for convenience of layout. Always use one connector.

Functions Common to the 7 Inch and 15 Inch I/O Interface Boards

1. No pins on the back panel may be used for I/O purposes other than those listed in Figure 8-33. All other pins are reserved for Memory connections or other purposes.
2. The I/O signals are duplicated in the Connector 0 and Connector 1 84 pin connectors in the same pin positions. That is, 119-0 (CONN0) has the same logic function as 119-1 (CONN1).
3. Only the standard board cable connector is used for low and medium speed signals. The maximum number of connections is 50 per connector.
4. The 3M-type Ribbon cable may be used for high speed signals. The maximum number of connections is 50 per connector.
5. All cables used in the interior of the cabinet shall be covered with a U.L. approved material.

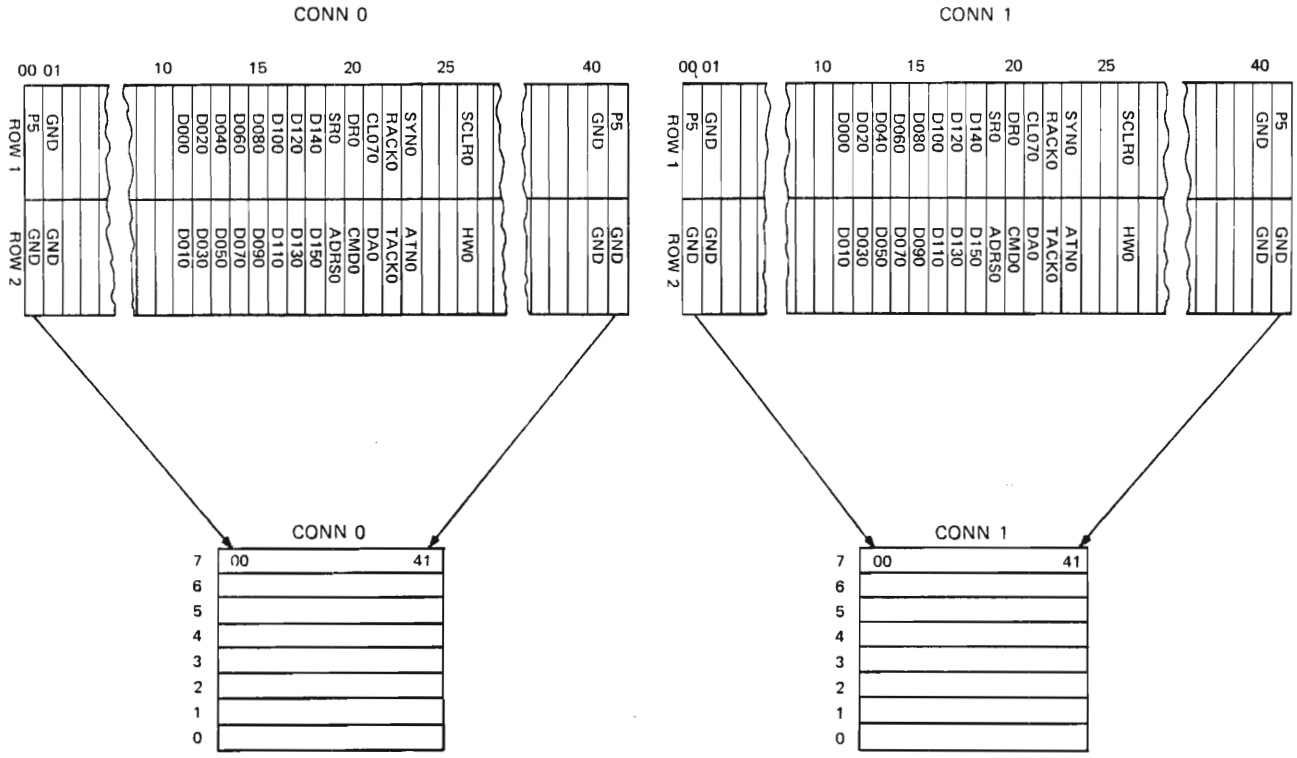


Figure 8-33. I/O Back Panel Connections

CHAPTER 9
ASCII PROGRAMMER'S CONSOLE

Operator control is provided by the M51-103 Turnkey Console and a microcode driven ASCII Console device on the M51-100 Serial Input/Output Port. The ASCII Console device may be an ASR/KSR 33 or 35 Teletype, CRT Terminal, Carousel 30 Terminal, or a Carousel 35 Terminal. The M51-100 Serial Input/Output Port is a current-loop interface addressed on the Micro I/O Bus as device number X'C0'.

CONFIGURATION

The M51-103 Turnkey Console, shown in Figure 9-1, is a RETMA standard 5-1/4" x 19" panel which is plug removable from the processor. It controls primary power to the system and provides for system initialization and interruption of running programs.

Keyboard commands through the ASCII Console device allow the operator to examine and modify processor registers and main memory locations and start program execution. Refer to Figure 9-2. The hexadecimal characters and a number of special characters are recognized by the ASCII Console microcode support. The characters accepted along with their meaning are shown in Table 9-1. All other characters are rejected for command input.

When not being used for operator control, the ASCII Console device is available to any running program as an I/O device.

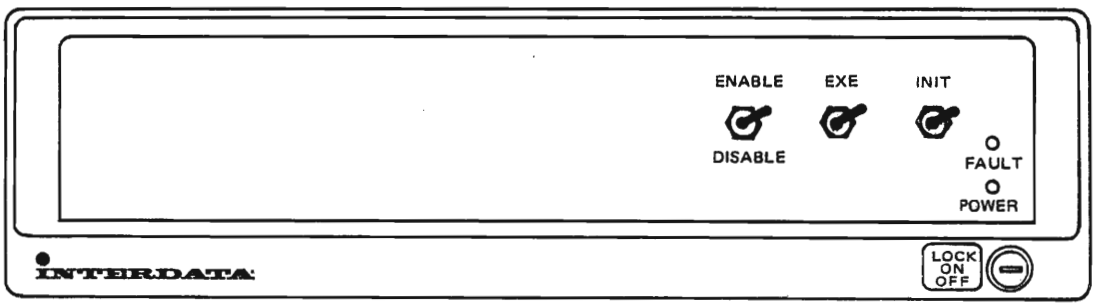


Figure 9-1. Model 5/16 Turnkey Console

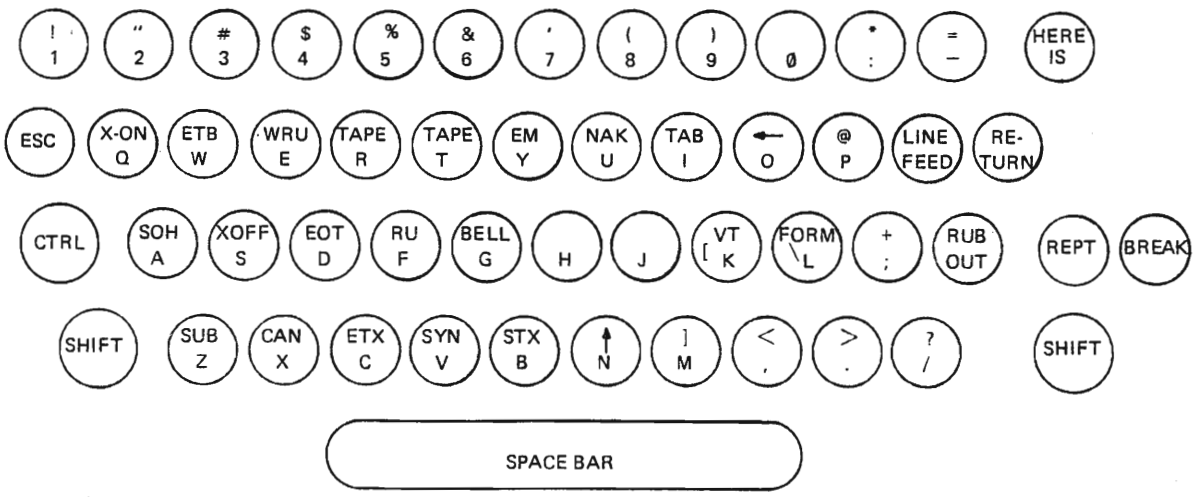


Figure 9-2. Keyboard Layout

TABLE 9-1. ASCII CONSOLE COMMAND SUMMARY

CHARACTERS	MEANING
0 1 2 3 4 5 6 7 8 9 A B C D E F	HEXADECIMAL DIGITS
@	OPEN CELL AND EXAMINE
+	OPEN AND EXAMINE NEXT CELL
?	OPEN AND EXAMINE LOCATION SHOWN
=	MODIFY OPEN CELL
*	BEGIN EXECUTION AT OPEN CELL ADDRESS
RETURN	CARRIAGE RETURN USED TO TERMINATE COMMAND INPUT

The following is a list of salient Serial Input/Output Port specifications:

Baud rate is selectable to yield 110, 300 or 1200 baud
Character format is 8 level, 11 unit code (one start bit, 8 data bits - no parity- and two stop bits).
Data is double buffered to permit a full character "grab time."

Figure 9-3 illustrates the character format.

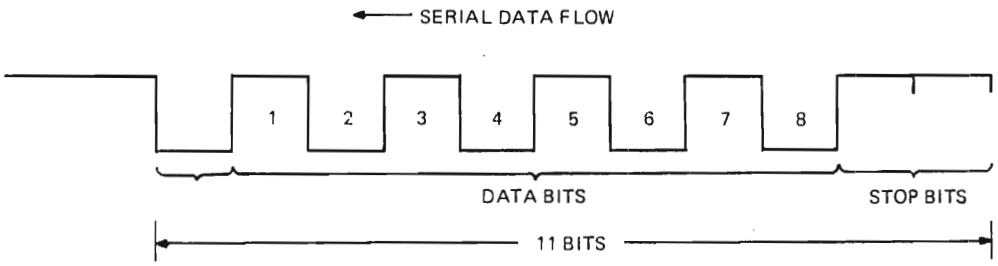


Figure 9-3. ASCII Character U, Eleven Bit Code

PROGRAMMING INSTRUCTIONS

When not being used for operator control, the ASCII Console device on the Serial Input/Output Port is available to any running program as an I/O device. The following processor I/O instructions may be used to control and communicate with the ASCII Console device.

Sense Status (SS or SSR)

The Sense Status instruction is used to determine if the ASCII Console device is ready to transfer data.

Output Command (OC or OCR)

The Output Command instruction is used to initialize the interface and to establish the Read mode or the Write mode.

Write Data (WD or WDR)

The Write Data instruction is used to output a character to the ASCII Console device.

Key Operated Security Lock

This is a three position, OFF-ON-LOCK, key operated switch. It controls primary power to the system. It can also disable (LOCK) the initialize and console switches, thereby preventing any accidental manual input to the system. The power indicator lamp (POWER) is on when the key lock is in the ON or LOCK position.

Control Switches

The momentary contact control switches are only active when the key-operated locking switch is in the ON position.

INITIALIZE (INIT) — The Initialize Switch (INIT) causes the system to be initialized. After the initialize sequence, all device controllers on the system I/O Bus are cleared and certain functions in the processor are reset. The Fault lamp (FAULT) comes on with initialize and is extinguished when the micro diagnostic routine is completed.

NOTE

A function of the micro diagnostic routine is to write known data into the first 8KB of main memory. Since this overwrites any data that may have been there, operation of the Initialize Switch should be carefully considered.

EXECUTE — The Execute Switch (EXE) causes the running program to be interrupted. Control is given to the microcode driven ASCII console device. If the processor was already in the ASCII Console routine, the run mode is entered.

ENABLE/DISABLE — The Enable/Disable Switch provides 12 VAC, 50/60 HRz, to the Non-Maskable Interrupt logic in the processor. In the Enable (UP) position the Real Time Clock feature is enabled.

OPERATING PROCEDURES

Power Up

To power up the system, turn the key-operated security lock clockwise from the OFF position to the ON position. This action provides electrical power to the system and leaves all device controllers on the Micro I/O Bus and Multiplexor Bus in an initialized state. The Fault lamp (FAULT) on the Turnkey Console is lit and the micro-diagnostic routine is entered. Operation of the Initialize switch (INIT) also places the processor in this initial state.

The micro-diagnostic routine comprises a number of small microcode sequences that exercise internal data paths and registers and test the first 8KB of main memory. This diagnostic is purposely limited in scope, only serving to indicate a go/no-go condition. If an error is detected, the microcode loops indefinitely on the failing subtest. The Fault (FAULT) lamp on the Turnkey Console remains on. If no errors are detected, the diagnostic takes about 25 milliseconds so that the Fault lamp does not appear to have been on.

The exercise of the first 8KB of memory involves writing data to each halfword location from X'0000' through X'1FFE'. For locations X'0000' through X'07FE' and X'1000' through X'17FE', the data used equals the address of the location. For locations X'0800' through X'0FFE' and X'1800' through X'1FFE', the data used equals the ones complement of the address of the location. Each location is then read back and tested against address true or address false, as appropriate. Because the first 8KB of memory are effectively wiped out by the diagnostic, operation of the Initialize Switch should be carefully considered.

After the micro-diagnostic, if there are no machine options such as Memory Addressable ROM or the Bootstrap Loader option, the ASCII Console routine is entered. Refer to Appendix 9.

ASCII Console

When the ASCII Console routine is entered from power up or initialize, a Carriage Return and Line Feed sequence is output followed by the current Location Counter value which will be X'FFFC'. Another Carriage Return, Line Feed sequence is output followed by an asterisk (*) operator prompt character. The prompt character indicates that the system is ready to receive operator commands.

Open Cell and Examine

For the remainder of this discussion, the term "cell" refers to a halfword location in main memory. The cell whose address is the contents of the Location Counter is the currently open cell. The character (@) places the microcode in the 'address' mode. This character may be followed by up to four hexadecimal digits. Leading zeros are not requred. If more than four hex characters are entered, the last four are taken. A Carriage Return is used to terminate the address input. At that time, the instruction Location Counter and its image at memory location X'0026' are set equal to the new open cell address. A Carriage Return and Line Feed are output followed by the hexadecimal data contained in the open cell. Another Carriage Return and Line Feed sequence is output followed by an asterisk (*) operator prompt character.

If, while entering the address, a character other than a Carriage Return or a valid hexadecimal character is struck, the 'address' mode is cancelled. A Carriage Return, Line Feed sequence is output followed by the asterisk (*) operator prompt character.

Open and Examine Next Cell

The plus sign character (+) is used to advance the instruction Location Counter to the next sequential open cell address. No Carriage Return is required. The Location Counter and its image at location X'0026' are incremented by two. A Carriage Return and Line Feed sequence is output followed by the hexadecimal data contained in the new open cell. Another Carriage Return and Line Feed sequence is output followed by the asterisk (*) operator prompt. Repeat to examine sequential locations.

Open and Examine Location Shown

A question mark (?) followed by a Carriage Return causes the halfword data last displayed to be taken as the new open cell address. The data is copied into the Location Counter and its image at location X'0026'. A Carriage Return, Line Feed sequence is output followed by the hexadecimal data contained in the new open cell. Another Carriage Return and Line Feed sequence is then output followed by the asterisk (*) operator prompt.

This command may also be issued after a memory write operation. The new open cell address in this case is the last written data halfword.

Modify Open Cell

The equals sign (=) places the microcode in the 'memory write' mode. This character may be followed by up to four hexadecimal digits. Leading zeros are not required. If more than four hexadecimal characters are entered, the last four are taken. A Carriage Return is used to terminate the data input. At that time, the halfword of data is written into the currently open cell. The instruction Location Counter is advanced to the next sequential open cell address. A Carriage Return, Line Feed sequence is output followed by the asterisk (*) operator prompt. Repeat to modify sequential locations.

If, while entering the data, a character other than a Carriage Return or a valid hexidecimal character is struck, the 'memory write' mode is cancelled. A Carriage Return and Line Feed sequence is output followed by the asterisk (*) operator prompt.

Program Execution

To begin execution of a user program, select the program start address as in Open Cell and Examine. The contents of the open cell are output followed by an operator prompt (*). Type an asterisk character (*) to begin execution. The 16 General Registers are loaded from memory locations X'0000' through X'001F'. The PSW is loaded from halfword location X'0024' and program execution begins at the open cell address.

Program Termination

To halt a running program and regain control at the ASCII Console, or regain control when a program has entered the wait state, operate the Console Switch on the Turnkey Console. The 16 General Registers are saved in memory locations X'0000' through X'001F'. The PSW with bit zero forced reset is saved in halfword location X'0024'. The current Location Counter value is saved in location X'0026'. An Output Command Reset is issued to the ASCII Console device and a Carriage Return, Line Feed sequence is output. The contents of the current Location Counter are output followed by another Carriage Return, Line Feed sequence and the operator prompt (*).

These events also occur when the Console routine is entered from the power up sequence, but the contents of the General Registers are unpredictable. The PSW, however, will be zero and the Location Counter will be equal to X'FFFC'.

General Register Examination and Modification

After entering the Console routine, the General Registers can be individually displayed. Remembering that the 16 General Registers were saved in memory locations X'0000' through X'001F', each register can be examined by opening the memory location whose address is twice the register number. Open location X'0000' to see what was in Register 0. Open location X'0002' to see what was in Register 1. Location X'0006' reflects the content of Register 3, etc.

The General Registers can be individually modified by doing a memory write to the appropriate location. When program execution is begun as in Program Execution, the General Registers are updated from their save area at locations X'0000' through X'001F'.

Program Status Word Examination and Modification

After entering the Console routine, the PSW with bit 0 forced to the reset state is saved in memory location X'0024' and the current Location Counter value is written to the ASCII Console. LOC is also saved at location X'0026', but any attempt to examine this location results in the number X'0026' being displayed.

To examine the PSW that was active when the console routine was entered, open location X'0024'. To modify the PSW, do a memory write to location X'0024'. When program execution is begun as in Program Execution, the PSW is updated from halfword location X'0024'. If the modified PSW has bit 0 set, the processor enters the wait state when execution is started.

Memory Initialization

The following sequence shows how to set up low memory and load the Relocating Loader.

The system was initialized

FFFC	Shows LOC = 'FFFC'			
*@34 (CR)	Select Address '0034'			
0034	Shows that '0034' = '0034'			
* 0 (CR)	Set new PSW for illegal instruction			
* = 50 (CR)	Interrupt = '0000', '0050'			
*? (CR)	Go to location '0050'			
0050	Shows that '0050' = '0050'			
* = D500 (CR)	Write "50 Sequence"	'D500'	AL	X'CF'
* = CF (CR)		'00CF'		
* = 4300 (CR)		'4300'	B	X'80'
* = 80 (CR)		'0080'		
*@78 (CR)	Select address X'0078'			
0078				
* = 1399 (CR)	Set Binary Input device for HSPTRP			
*@24 (CR)				
0024				
* = 0 (CR)	Clear Program Status Word			
*@50 (CR)	Select Address X'0050'			
**	Go			

After loading, the Relocating (REL) Loader places the processor in the wait state. Depress CONSOLE to regain control.

3C00	Current LOC = X'3C00' (Starting ddress of REL Loader)
**	Start REL Loader by typing *

DATA FORMAT

The M51-100 Serial Input/Output Port is a current loop interface addressed on the Micro I/O Bus as device number X'C0'. It will adapt a single ASR/KSR 33 or 35 Teletype, CRT Terminal, Carousel 30 Terminal, or a Carousel 35 Terminal to the Micro I/O Bus. For the operating procedures pertaining to these devices, refer to the following manuals:

Teletype Interface Programming Manual, Publication Number 29-442
CRT Programming Manual, Publication Number 29-451
Current Loop Interface Programming Manual, Publication Number 29-461

Read Data (RD or RDR)

The Read Data instruction is used to read a character from the ASCII Console device.

Acknowledge Interrupt (ACK or ACKR)

The Acknowledge Interrupt instruction is used to service interrupt request. Execution of this instruction returns the device number in the first operand register and the device status in the second operand register or memory location. The interrupt condition is not reset. The interrupt is cleared by a Read operation in the Read mode, a Write operation in the Write mode, or by a RESET command.

Status and Command Bytes

Table 9-2 contains the Serial Input/Output Port status and command bytes.

TABLE 9-2. SERIAL INPUT/OUTPUT PORT STATUS AND COMMAND DATA

	BIT NUMBER	0	1	2	3	4	5	6	7
	STATUS	OV	1	BRK	0	BSY	EX	0	DU
COMMAND	RESET COMMAND	0	0	0	0	0	0	1	1
	WRITE-DISABLE	0	0	0	1	0	0	1	0
	WRITE ENABLE	0	0	1	1	0	0	1	0
	READ	1	0	0	1	0	0	1	0

Status

OV — The Overflow status bit indicates that one or more characters in the data stream were lost. That is, one or more new characters were received before the processor had a chance to read the present character. The Overflow indication does not set until the processor actually reads the present character. The BSY bit then remains reset until the OV bit resets which occurs the next time the processor does a Read operation. Overflow is also reset by the Reset command.

BRK — The Line Break bit is set when the Serial input data line is a zero (space) for longer than one character period. Specifically, BRK sets when a character is received that appeared to have a Start bit but when it came time for the first Stop bit, the line was ZERO. See Figure 9-3. Break status remains active until the input data line goes to a ONE (Mark) and the character is read by the program.

BSY — The Busy status bit is set when the Serial Input/Output port cannot yet transfer a character. In the Read mode, BSY is normally set. It resets when a complete character has been received and is ready to be read by the processor. In the Write mode, BSY is normally reset. It sets after the processor writes a character to the interface and remains set until the interface transmits the character. Thus, when BSY is reset, the processor can transfer data with the Serial Input/Output port. In the transition from BSY set to BSY reset, an interrupt is generated.

EX — The Examine status bit is set when the Overflow or the Line Break status bits are set.

DU — The Device Unavailable status bit is set when the ASCII Console device is powered down or off-line.

Commands

WRITE DISABLE	An Output Command instruction X'02' places the interface in the Write mode with Write interrupts disabled. In this mode when BSY is reset, the processor can write data to the interface.
WRITE ENABLE	An Output Command instruction X'32' places the interface in the Write mode. In this mode, when BSY is reset, the processor can write data to the interface.
READ	An Output Command instruction X'92' places the interface in the Read mode. In this mode, when BSY is reset, a character has been received from the ASCII console device and is available for the processor to read.
RESET	This Output Command, X'03', initializes the interface. The OV, BRK, BSY and EX status bits are reset and any pending interrupts are cleared. The Reset command must be issued after a power failure.

PROGRAMMING SEQUENCES

Programming Notes

Characters read from the keyboard are not echoed back for display. The program must perform the character echo. This is accomplished by doing a Write Data instruction immediately after reading the character. It is not necessary to switch from Read mode to Write mode for the character echo.

Status Monitoring I/O

This form of I/O programming uses loops to continually interrogate the status of the device until a specified condition is met. Processor interrupts must be disabled to use this type of I/O. See Appendix 10 for programming examples.

Interrupt I/O Control

When using the Immediate Interrupt mechanism, the appropriate Interrupt Service Pointer Table entry must be set up to accommodate a PSW swap. See Appendix 10 for a programming example.

INTERRUPTS

In the Read mode interrupts are always enabled in the Serial Input/Output Port interface. The interface generates an interrupt for the following conditions:

1. In the Read mode when a character is present in the interface (BSY = ZERO).
2. In the Write Enable mode if the interface can accept a character from the processor (BSY = ZERO).
3. EX goes active.

Interrupts pending in the interface may be cleared by:

1. System initialization.
2. Output Command Reset.
3. Read data or Write data instruction.

APPENDIX 1
MODEL 5/16 OP-CODE MAP

LSD \ MSD	0	2	3	4	6	8	9	C	D	E
0		BTBS		STH *			SRLS	BXH *	STM *	
1	BALR	BTFS		BAL *	AHM *		SLLS	BXLE *	LM *	SVC *
2	BTCR	BFBS		BTC *			STBR	LPSW *	STB	SINT
3	BFCR	BFFS		BFC *			LBR	THI	LB	
4	NHR	LIS		NH *	ATL *		EXBR	NHI	CLB	
5	CLHR	LCS		CLH *	ABL *		EPSR	CLHI	AL	
6	OHR	AIS		OH *	RTL *		WBR	OHI	WB *	
7	XHR	SIS		XH *	RBL *		RBR	XHI	RB *	
8	LHR			LH *			WHR	LHI	WH *	
9	CHR			CH *			RHR	CHI	RH *	
A	AHR			AH *			WDR	AHI	WD	RRL
B	SHR			SH *			RDR	SHI	RD	RLL
C	MHR			MH *			MHUR	SRHL	MHU *	SRL
D	DHR			DH *			SSR	SLHL	SS	SLL
E	ACHR			ACH *			OCR	SRHA	OC	SRA
F	SCHR			SCH *			ACKR (AIR)	SLHA	ACK (AI)	SLA

NOTE

\+ INSTRUCTION MAY BE DISABLED BY A STRAP OPTION ON THE CPU.

APPENDIX 2
INSTRUCTION SUMMARY – ALPHABETICAL WITH ATTRIBUTES

Attributes

A: Arithmetic Fault Interrupt can occur
C: Condition Code in the PSW is set to reflect the result
H: Second operand must be on halfword boundary for consistent result
IA: Immediate Interrupt can be initiated

INSTRUCTION	OP-CODE	MNEMONIC	ATTRIBUTES	PAGE NO.
Acknowledge Interrupt	DF	ACK(AI)	C	7-4
Acknowledge Interrupt Register	9F	ACKR(AIR)	C	7-4
Add Halfword	4A	AH	C,H	5-3
Add Halfword Immediate	CA	AHI	C	5-3
Add Halfword to Memory	61	AHM	C,H	5-4
Add Halfword Register	0A	AHR	C	5-3
Add Immediate Short	26	AIS	C	5-3
Add to Bottom of List	65	ABL	C,H	3-25
Add to Top of List	64	ATL	C,H	3-25
Add with Carry Halfword	4E	ACH	C,H	5-6
Add with Carry Halfword Register	0E	ACHR	C	5-6
AND Halfword	44	NH	C,H	3-15
AND Halfword Immediate	C4	NHI	C	3-15
AND Halfword Register	04	NHR	C	3-15
Autoload	D5	AL	C	7-15
Branch and Link	41	BAL	H	4-4
Branch and Link Register	01	BALR		4-4
Branch on False Condition	43	BFC	H	4-3
Branch on False Condition Backward Short	22	BFBS		4-3
Branch on False Condition Forward Short	23	BFFS		4-3
Branch on False Condition Register	03	BFCR		4-3
Branch on Index High	C0	BXH	H	4-6
Branch on Index Low or Equal	C1	BXLE	H	4-5
Branch on True Condition	42	BTC	H	4-2
Branch on True Condition Backward Short	20	BTBS		4-2
Branch on True Condition Forward Short	21	BTFS		4-2
Branch on True Condition Register	02	BTCR		4-2
*Breakpoint	88	BRK		7-15 16
Compare Halfword	49	CH	C,H	5-8
Compare Halfword Immediate	C9	CHI	C	5-8
Compare Halfword Register	09	CHR	C	5-8
Compare Logical Byte	D4	CLB	C	3-14
Compare Logical Halfword	45	CLH	C,H	3-13
Compare Logical Halfword Immediate	C5	CLHI	C	3-13
Compare Logical Halfword Register	05	CLHR	C	3-13
Divide Halfword	4D	DH	H,A	5-11
Divide Halfword Register	0D	DHR	A	5-11
Exchange Byte Register	94	EXBR		3-9
Exchange Program Status Register	95	EPSR	C,IA	6-6
Exclusive OR Halfword	47	XH	C,H	3-17
Exclusive OR Halfword Immediate	C7	XHI	C	3-17
Exclusive OR Halfword Register	07	XHR	C	3-17

* This instruction may be disabled by a strap option on the CPU.

INSTRUCTION	OP-CODE	MNEMONIC	ATTRIBUTES	PAGE NO.
Load Byte	D3	LB		3-8
Load Byte Register	93	LBR		3-8
Load Complement Short	25	LCS	C	3-5
Load Halfword	48	LH	C	3-6
Load Halfword Immediate	C8	LHI	C	3-6
Load Halfword Register	08	LHR	C	3-5
Load Immediate Short	24	LIS	C	3-5
Load Multiple	D1	LM	H	3-7
Load Program Status Word	C2	LPSW	C,IA	6-5
Multiply Halfword	4C	MH	H	5-9
Multiply Halfword Register	0C	MHR		5-9
Multiply Halfword Unsigned	DC	MHUR		5-10
Multiply Halfword Unsigned Register	9C	MHU	H	5-10
OR Halfword	46	OH	C,H	3-16
OR Halfword Immediate	C6	OHI	C	3-16
OR Halfword Register	06	OHR	C	3-16
Output Command	DE	OC	C,IA	7-6
Output Command Register	9E	OCR	C,IA	7-6
Read Block	D7	RB	C,H	7-9
Read Block Register	97	RBR	C	7-10
Read Data	DB	RD	C	7-7
Read Data Register	9B	RDR	C	7-7
Read Halfword	D9	RH	C,H	7-8
Read Halfword Register	99	RHR	C	7-8
Remove from Bottom of List	67	RBL	C,H	3-26
Remove from Top of List	66	RTL	C,H	3-26
Rotate Left Logical	EB	RLL	C	3-23
Rotate Right Logical	EA	RRL	C	3-24
Sense Status	DD	SS	C,P	7-5
Sense Status Register	9D	SSR	C,P	7-5
Shift Left Arithmetic	EF	SLA	C	5-13
Shift Left Halfword Arithmetic	CF	SLHA	C	5-14
Shift Left Halfword Logical	CD	SLHL	C	3-21
Shift Left Logical Short	91	SLLS	C	3-21
Shift Left Logical	ED	SLL	C	3-10
Shift Right Arithmetic	EE	SRA	C	5-15
Shift Right Halfword Arithmetic	CE	SRHA	C	5-16
Shift Right Halfword Logical	CC	SRHL	C	3-22
Shift Right Logical Short	90	SRLS	C	3-22
Shift Right Logical	EC	SRL	C	3-19
Simulate Interrupt	E2	SINT	C,IA	6-7
Store Byte	D2	STB		3-12
Store Byte Register	92	STBR		3-12
Store Halfword	40	STH	H	3-10
Store Multiple	D0	STM	H	3-11

INSTRUCTION	OP-CODE	MNEMONIC	ATTRIBUTES	PAGE NO.
Subtract Halfword	4B	SH	C,H	5-5
Subtract Halfword Immediate	CB	SHI	C	5-5
Subtract Halfword Register	0B	SHR	C	5-5
Subtract Immediate Short	27	SIS	C	5-5
Subtract with Carry Halfword	4F	SCH	C,H	5-7
Subtract with Carry Halfword Register	0F	SCHR	C	5-7
Supervisor Call	E1	SVC	C,H	6-8
Test Halfword Immediate	C3	THI	C	3-18
Write Block	D6	WB	C,H	7-13
Write Block Register	96	WBR	C	7-14
Write Data	DA	WD	C	7-11
Write Data Register	9A	WDR	C	7-11
Write Halfword	D8	WH	C,H	7-12
Write Halfword Register	98	WHR	C	7-12

APPENDIX 3
INSTRUCTION SUMMARY NUMERICAL

OP CODE	MNEMONIC	INSTRUCTION	PAGE NO.
01*	BALR	Branch and Link Register	4-4
02*	BTCR	Branch on True Condition Register	4-2
03*	BFCR	Branch on False Condition Register	4-3
04	NHR	AND Halfword Register	3-15
05	CLHR	Compare Logical Halfword Register	3-13
06	OHR	OR Halfword Register	3-16
07	XHR	Exclusive OR Halfword Register	3-17
08	LHR	Load Halfword Register	3-5
09	CHR	Compare Halfword Register	5-8
0A	AHR	Add Halfword Register	5-3
0B	SHR	Subtract Halfword Register	5-5
0C*	MHR	Multiply Halfword Register	5-9
0D*	DHR	Divide Halfword Register	5-11
0E	ACHR	Add with Carry Halfword Register	5-6
0F	SCHR	Subtract with Carry Halfword Register	5-7
20*	BTBS	Branch on True Condition Backward Short	4-2
21*	BTFS	Branch on True Condition Forward Short	4-2
22*	BFBS	Branch on False Condition Backward Short	4-3
23*	BFFS	Branch on False Condition Forward Short	4-3
24	LIS	Load Immediate Short	3-5
25	LCS	Load Complement Short	3-5
26	AIS	Add Immediate Short	5-3
27	SIS	Subtract Immediate Short	5-5
40*	STH	Store Halfword	3-10
41*	BAL	Branch and Link	4-4
42*	BTC	Branch on True Condition	4-2
43*	BFC	Branch on False Condition	4-3
44	NH	AND Halfword	3-15
45	CLH	Compare Logical Halfword	3-13
46	OH	OR Halfword	3-16
47	XH	Exclusive OR Halfword	3-17
48	LH	Load Halfword	3-6
49	CH	Compare Halfword	5-8
4A	AH	Add Halfword	5-3
4B	SH	Subtract Halfword	5-5
4C*	MH	Multiply Halfword	5-9
4D*	DH	Divide Halfword	5-11
4E	ACH	Add with Carry Halfword	5-6
4F	SCH	Subtract with Carry Halfword	5-7
88	BRK	Breakpoint	7-15

OP CODE	MNEMONIC	INSTRUCTION	PAGE NO.
90	SRLS	Shift Right Logical Short	3-22
91	SLLS	Shift Left Logical Short	3-21
92*	STBR	Store Byte Register	3-12
93*	LBR	Load Byte Register	3-8
94*	EXBR	Exchange Byte Register	3-9
95	EPSR	Exchange Program Status Register	6-6
96	WBR	Write Block Register	7-14
97	RBR	Read Block Register	7-10
98	WHR	Write Halfword Register	7-12
99	RHR	Read Halfword Register	7-8
9A	WDR	Write Data Register	7-11
9B	RDR	Read Data Register	7-7
9C*	MHUR	Multiply Halfword Unsigned Register	5-10
9D	SSR	Sense Status Register	7-5
9E	OCR	Output Command Register	7-6
9F	ACKR (AIR)	Acknowledge Interrupt Register	7-4
C0*	BXH	Branch on Index High	4-6
C1*	BXLE	Branch on Index Low or Equal	4-5
C2	LPSW	Load Program Status Word	6-5
C3	THI	Test Halfword Immediate	3-18
C4	NHI	AND Halfword Immediate	3-15
C5	CLHI	Compare Logical Halfword Immediate	3-13
C6	OHI	OR Halfword Immediate	3-16
C7	XHI	Exclusive OR Halfword Immediate	3-17
C8	LHI	Load Halfword Immediate	3-6
C9	CHI	Compare Halfword Immediate	5-8
CA	AHI	Add Halfword Immediate	5-3
CB	SHI	Subtract Halfword Immediate	5-5
CC	SRHL	Shift Right Halfword Logical	3-22
CD	SLHL	Shift Left Halfword Logical	3-21
CE	SRHA	Shift Right Halfword Arithmetic	5-16
CF	SLHA	Shift Left Halfword Arithmetic	5-14
D0*	STM	Store Multiple	3-11
D1*	LM	Load Multiple	3-7
D2*	STB	Store Byte	3-12
D3*	LB	Load Byte	3-8
D4	CLB	Compare Logical Byte	3-14
D5	AL	Autoload	7-15
D6	WB	Write Block	7-13
D7	RB	Read Block	7-9
D8	WH	Write Halfword	7-12
D9	RH	Read Halfword	7-8

OP CODE	MNEMONIC	INSTRUCTION	PAGE NO.
DA	WD	Write Data	7-11
DB	RD	Read Data	7-7
DC*	MHU	Multiply Halfword Unsigned	5-10
DD	SS	Sense Status	7-5
DE	OC	Output Command	7-6
DF	ACK (AI)	Acknowledge Interrupt	7-4
E1	SVC	Supervisor Call	6-8
E2	SINT	Simulate Interrupt	6-7
EA	RRL	Rotate Right Logical	3-24
EB	RLL	Rotate Left Logical	3-23
EC	SRL	Shift Right Logical	3-20
ED	SLL	Shift Left Logical	3-10
EE	SRA	Shift Right Arithmetic	5-15
EF	SLA	Shift Left Arithmetic	5-13

* Condition Code NOT CHANGED.

APPENDIX 4
EXTENDED BRANCH MNEMONICS

INSTRUCTION	OP CODE (M1)	MNEMONIC	OPERAND
Branch on Carry	428	BC	A(X2)
Branch on Carry Register	028	BCR	R2
Branch on No Carry	438	BNC	A(X2)
Branch on No Carry Register	038	BNCR	R2
Branch on Equal	433	BE	A(X2)
Branch on Equal Register	033	BER	R2
Branch on Not Equal	423	BNE	A(X2)
Branch on Not Equal Register	023	BNER	R2
Branch on Low	428	BL	A(X2)
Branch on Low Register	028	BLR	R2
Branch on Not Low	438	BNL	A(X2)
Branch on Not Low Register	038	BNLR	R2
Branch on Minus	421	BM	A(X2)
Branch on Minus Register	021	BMR	R2
Branch on Not Minus	431	BNM	A(X2)
Branch on Not Minus Register	031	BNMR	R2
Branch on Plus	422	BP	A(X2)
Branch on Plus Register	022	BPR	R2
Branch on Not Plus	432	BNP	A(X2)
Branch on Not Plus Register	032	BNPR	R2
Branch on Overflow	424	BO	A(X2)
Branch on Overflow Register	024	BOR	R2
Branch on No Overflow	434	BNO	A(X2)
Branch on No Overflow Register	034	BNOR	R2
Branch Unconditional	430	B	A(X2)
Branch Unconditional Register	030	BR	R2
Branch on Zero	433	BZ	A(X2)
Branch on Zero Register	033	BZR	R2
Branch on Not Zero	423	BNZ	A(X2)
Branch on Not Zero Register	023	BNZR	R2
No Operation	420	NOP	
No Operation Register	020	NOPR	
Branch on Carry Short	208	BCS	A (Backward Reference)
	218	BCS	A (Forward Reference)
Branch on No Carry Short	228	BNCS	A (Backward Reference)
	238	BNCS	A (Forward Reference)
Branch on Equal Short	223	BES	A (Backward Reference)
	233	BES	A (Forward Reference)
Branch on Not Equal Short	203	BNES	A (Backward Reference)
	213	BNES	A (Forward Reference)
Branch on Low Short	208	BLS	A (Backward Reference)
	218	BLS	A (Forward Reference)
Branch on Not Low Short	228	BNLS	A (Backward Reference)
	238	BNLS	A (Forward Reference)

INSTRUCTION	OP CODE (M1)	MNEMONIC	OPERANDS
Branch on Minus Short	201	BMS	A (Backward Reference)
	211	BMS	A (Forward Reference)
Branch on Not Minus Short	221	BNMS	A (Backward Reference)
	231	BNMS	A (Forward Reference)
Branch on Plus Short	202	BPS	A (Backward Reference)
	212	BPS	A (Forward Reference)
Branch on Not Plus Short	222	BNPS	A (Backward Reference)
	232	BNPS	A (Forward Reference)
Branch on Overflow Short	204	BOS	A (Backward Reference)
	214	BOS	A (Forward Reference)
Branch on No Overflow Short	224	BNOS	A (Backward Reference)
	234	BNOS	A (Forward Reference)
Branch Unconditional Short	220	BS	A (Backward Reference)
	230	BS	A (Forward Reference)
Branch on Zero Short	223	BZS	A (Backward Reference)
	233	BZS	A (Forward Reference)
Branch on Not Zero Short	203	BNZS	A (Backward Reference)
	213	BNZS	A (Forward Reference)

APPENDIX 5
ARITHMETIC REFERENCES

TABLE OF POWERS OF TWO

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5

TABLE OF POWERS OF SIXTEEN

16^n							n
						1	0
						16	1
						256	2
					4	096	3
					65	536	4
				1	048	576	5
				16	777	216	6
				268	435	456	7
			4	294	967	296	8
			68	719	476	736	9
		1	099	511	627	776	10
		17	592	186	044	416	11
		281	474	976	710	656	12
	4	503	599	627	370	496	13
	72	057	594	037	927	936	14
1	152	921	504	606	846	976	15

Decimal Values

HEXADECIMAL ADDITION AND SUBTRACTION TABLE

Examples: 5+A = F; 18-D = B; A+B = 15

	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	1
2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	2
3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	3
4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	4
5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	5
6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	6
7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	7
8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	8
9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	9
A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	A
B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	B
C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	C
D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	D
E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	E
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	F
	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

HEXADECIMAL MULTIPLICATION AND DIVISION TABLE

Examples: 5x6 = 1E; 75÷D = 9; 58÷8 = B; 9xC = 6C

	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	1
2	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E	2
3	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D	3
4	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C	4
5	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B	5
6	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A	6
7	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69	7
8	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78	8
9	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87	9
A	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96	A
B	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5	B
C	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4	C
D	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3	D
E	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2	E
F	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1	F
	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

TABLE OF MATHEMATICAL CONSTANTS

CONSTANT	DECIMAL VALUE	HEXADECIMAL VALUE	FLOATING POINT VALUE
π	3.14159 26535 89793 23846	3.243F 6A88 85A3 08D3	4132 43F6 A888 5A31
π^{-1}	0.31830 98861 83790 67154	0.517C C1B7 2722 0A95	4051 7CC1 B727 220B
$\sqrt{\pi}$	1.77245 38509 05516 02730	1.C5BF 891B 4EF6 AA7A	411C 5BF8 91B4 EF6B
$\ln \pi$	1.14472 98858 49400 17414	1.250D 048E 7A1B D0BD	4112 50D0 48E7 A1BD
$\sqrt{3}$	1.73205 08075 68877 29353	1.BB67 AE85 84CA A73B	411B B67A E858 4CAA
e	2.71828 18284 59045 23536	2.B7E1 5162 8AED 2A6B	412B 7E15 1628 AED3
e^{-1}	0.36787 94411 71442 32160	0.5E2D 58D8 B3BC DF1B	405E 2D58 D8B3 BCDF
$\sqrt{e}$	1.64872 12707 00128 14683	1.A612 98E1 E069 BC97	411A 6129 8E1E 069C
$\log_{10}e$	0.43429 44819 03251 82765	0.6F2D EC54 9B94 38CB	406F 2DEC 549B 9439
$\log_{2}e$	1.44269 50408 88963 40736	1.7154 7652 B82F E177	4117 1547 652B 82FE
γ	0.57721 56649 01532 86061	0.93C4 67E3 7DB0 C7A5	4093 C467 E37D B0C8
$\ln \gamma$	- 0.54953 93129 81644 82234	- 0.8CAE 9BC1 1F5A 5FF4	C08C AE9B C11F 5A60
$\sqrt{2}$	1.41421 35623 73095 04880	1.6A09 E667 F3BC C909	4116 A09E 667F 3BCD
$\ln 2$	0.69314 71805 59945 30942	0.B172 17F7 D1CF 79AC	40B1 7217 F7D1 CF7A
$\log_{10}2$	0.30102 99956 63981 19521	0.4D10 4D42 7DE7 FBCC	404D 104D 427D -E7FC
$\sqrt{10}$	3.16227 76601 68379 33199	3.298B 075B 4B6A 5240	4132 98B0 75B4 B6A5
$\ln 10$	2.30258 50929 94045 68402	2.4D76 3776 AAA2 B05C	4124 D763 776A AA2B

Hexadecimal and Decimal Integer Conversion Table

HALFWORD								HALFWORD							
BYTE				BYTE				BYTE				BYTE			
BITS: 0123		4567		0123		4567		0123		4567		0123		4567	
Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	268,435,456	1	16,777,216	1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	536,870,912	2	33,554,432	2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	805,306,368	3	50,331,648	3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	1,073,741,824	4	67,108,864	4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	1,342,177,280	5	83,886,080	5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	1,610,612,736	6	100,663,296	6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	1,879,048,192	7	117,440,512	7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	2,147,483,648	8	134,217,728	8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	2,415,919,104	9	150,994,944	9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	2,684,354,560	A	167,772,160	A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	2,952,790,016	B	184,549,376	B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	3,221,225,472	C	201,326,592	C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	3,489,660,928	D	218,103,808	D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	3,758,096,384	E	234,881,024	E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	4,026,531,840	F	251,658,240	F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15
8		7		6		5		4		3		2		1	

TO CONVERT HEXADECIMAL TO DECIMAL

1. Locate the column of decimal numbers corresponding to the left-most digit or letter of the hexadecimal; select from this column and record the number that corresponds to the position of the hexadecimal digit or letter.
2. Repeat step 1 for the next (second from the left) position.
3. Repeat step 1 for the units (third from the left) position.
4. Add the numbers selected from the table to form the decimal number.

EXAMPLE	
Conversion of Hexadecimal Value	D34
1. D	3328
2. 3	48
3. 4	4
4. Decimal	3380

To convert integer numbers greater than the capacity of table, use the techniques below:

HEXADECIMAL TO DECIMAL

Successive cumulative multiplication from left to right, adding units position.

Example: $D34_{16} = 3380_{10}$

D = 13
x16
208
3 = + 3
211
x16
3376
4 = +4
3380

TO CONVERT DECIMAL TO HEXADECIMAL

1. (a) Select from the table the highest decimal number that is equal to or less than the number to be converted.
   (b) Record the hexadecimal of the column containing the selected number.
   (c) Subtract the selected decimal from the number to be converted.
2. Using the remainder from step 1(c) repeat all of step 1 to develop the second position of the hexadecimal (and a remainder).
3. Using the remainder from step 2 repeat all of step 1 to develop the units position of the hexadecimal.
4. Combine terms to form the hexadecimal number.

EXAMPLE	
Conversion of Decimal Value	3380
1. D	-3328
	52
2. 3	-48
	4
3. 4	-4
4. Hexadecimal	D34

DECIMAL TO HEXADECIMAL

Divide and collect the remainder in reverse order.

Example: $3380_{10} = X_{16}$

16 | 3380	→	remainder 4
16 | 211	→	3
16 | 13	→	D

$3380_{10} = D34_{16}$

APPENDIX 5 (Continued)
FRACTION CONVERSION TABLE

Hexadecimal and Decimal Fraction Conversion Table

HALFWORD													
BYTE					BYTE								
BITS 0123		4567			0123				4567				
Hex	Decimal	Hex	Decimal		Hex	Decimal			Hex	Decimal Equivalent			
.0	.0000	.00	.0000	0000	.000	.0000	0000	0000	.0000	.0000	0000	0000	0000
.1	.0625	.01	.0039	0625	.001	.0002	4414	0625	.0001	.0000	1525	8789	0625
.2	.1250	.02	.0078	1250	.002	.0004	8828	1250	.0002	.0000	3051	7578	1250
.3	.1875	.03	.0117	1875	.003	.0007	3242	1875	.0003	.0000	4577	6367	1875
.4	.2500	.04	.0156	2500	.004	.0009	7656	2500	.0004	.0000	6103	5156	2500
.5	.3125	.05	.0195	3125	.005	.0012	2070	3125	.0005	.0000	7629	3945	3125
.6	.3750	.06	.0234	3750	.006	.0014	6484	3750	.0006	.0000	9155	2734	3750
.7	.4375	.07	.0273	4375	.007	.0017	0898	4375	.0007	.0001	0681	1523	4375
.8	.5000	.08	.0312	5000	.008	.0019	5312	5000	.0008	.0001	2207	0312	5000
.9	.5625	.09	.0351	5625	.009	.0021	9726	5625	.0009	.0001	3732	9101	5625
.A	.6250	.0A	.0390	6250	.00A	.0024	4140	6250	.000A	.0001	5258	7890	6250
.B	.6875	.0B	.0429	6875	.00B	.0026	8554	6875	.000B	.0001	6784	6679	6875
.C	.7500	.0C	.0468	7500	.00C	.0029	2968	7500	.000C	.0001	8310	5468	7500
.D	.8125	.0D	.0507	8125	.00D	.0031	7382	8125	.000D	.0001	9836	4257	8125
.E	.8750	.0E	.0546	8750	.00E	.0034	1796	8750	.000E	.0002	1362	3046	8750
.F	.9375	.0F	.0585	9375	.00F	.0036	6210	9375	.000F	.0002	2888	1835	9375
1		2			3				4				

TO CONVERT .ABC HEXADECIMAL TO DECIMAL

Find .A in position 1 .6250

Find .0B in position 2 .0429 6875

Find .00C in position 3 .0029 2968 7500

.ABC Hex is equal to .6708 9843 7500

TO CONVERT .13 DECIMAL TO HEXADECIMAL

1. Find .1250 next lowest to .1300
   subtract −.1250 = .2 Hex
2. Find .0039 0625 next lowest to .0050 0000
   −.0039 0625 = .01
3. Find .0009 7656 2500 .0010 9375 0000
   −.0009 7656 2500 = .004
4. Find .0001 0681 1523 4375 .0001 1718 7500 0000
   −.0001 0681 1523 4375 = .0007
   .0000 1037 5976 5625 = .2147 Hex
5. 13 Decimal is approximately equal to ——————↑

To convert fractions beyond the capacity of table, use techniques below:

HEXADECIMAL FRACTION TO DECIMAL

Convert the hexadecimal fraction to its decimal equivalent using the same technique as for integer numbers. Divide the results by 16^n (n is the number of fraction positions).

Example: $.8A7 = .540771_{10}$

$8A7_{16} = 2215_{10}$

$16^3 = 4096$

$$4096\overline{)2215.000000} \quad .540771$$

DECIMAL FRACTION TO HEXADECIMAL

Collect integer parts of product in the order of calculation.

Example: $.5408_{10} = .8A7_{16}$

.5408

8 ← [8].6528 ×16

A ← [10].4448 ×16

7 ← [7].1168 ×16

APPENDIX 6
INSTRUCTION TIMING

INSTRUCTION	RR/SF	RI	RX	COMMENTS
ABL	–	–	4.9/10.8/11.4	OVF/NORM/WRAP
ACH	1.5	–	3.0	
ACK (AI)	4.2(5.4)	–	5.1(6.3)	NOTE 1
AH	1.2	2.1	2.7	
AHM	–	–	3.3	
AIS	1.5	–	–	
AL	–	–	5.7+3.9n(6.9+5.1n)	NOTE 1
ATL	–	–	4.9/10.8/11.1	OVF/NORM/WRAP
BAL	1.5	–	2.4	
BFBS	1.5/3.0	–	–	NO BRANCH/BRANCH
BFC	1.5/1.5	–	1.8/2.4	NO BRANCH/BRANCH
BFFS	1.5/3.0	–	–	NO BRANCH/BRANCH
BTBS	1.5/3.0	–	–	NO BRANCH/BRANCH
BTC	1.5/1.5	–	1.8/2.4	NO BRANCH/BRANCH
BTFS	1.5/3.0	–	–	NO BRANCH/BRANCH
BXH	–	–	4.8/5.1	NO BRANCH/BRANCH
BXLE	–	–	4.8/5.1	NO BRANCH/BRANCH
CH	2.1/2.4	3.0/3.3	3.6/3.9	SIGNS ALIKE/DIFFER
CLB	–	–	3.9	
CLH	1.2	2.1	2.7	
DH	33.3/36.9/37.5	–	34.8/38.4/42.0	MIN/AVG/MAX
EPSR	3.9	–	–	NOTE 2
EXBR	1.5	–	–	
LB	1.8	–	3.6	
LCS	1.5	–	–	
LH	1.2	2.1	2.7	
LIS	1.5	–	–	
LM	–	–	2.1+1.5n	n = registers
LPSW	–	–	6.0	NOTE 2
MH	24.6/27.0/31.2	–	26.1/28.5/32.7	MIN/AVG/MAX
MHU	23.7/26.1/28.5	–	25.2/27.6/30.0	MIN/AVG/MAX
NH	1.2	2.1	2.7	
OC	3.0(3.6)	–	4.5(5.1)	NOTE 1
OH	1.2	2.1	2.7	
RB	5.4+3.9n(5.4+5.1n)	–	5.4+3.9n(5.4+5.1n)	NOTES 1, 3
RBL	–	–	4.2/11.1/11.4	UNF/NORM/WRAP
RD	3.0(3.6)	–	4.2(4.8)	NOTE 1
RH	3.0/4.5(6.3)	–	4.5/5.4(7.2)	NOTES 1, 4
RLL	–	6.0+0.9n	–	n = shifts
RRL	–	6.0+0.9n	–	n = shifts
RTL	–	–	4.2/11.7/11.7	UNF/NORM/WRAP
SCH	1.5	–	3.0	
SH	1.2	2.1	2.7	
SINT	–	7.2	–	
SIS	1.5	–	–	
SLA	–	6.9+0.9n	–	n = shifts
SLHA	–	5.4+0.9n	–	n = shifts
SLHL	–	4.5+0.9n	–	n = shifts
SLL	–	6.0+0.9n	–	n = shifts

SLLS	3.0+0.9n	–	–	n = shifts
SRA	–	7.2+1.2n	–	n = shifts
SRHA	–	5.7+1.2n	–	n = shifts
SRHL	–	4.5+0.9n	–	n = shifts
SRL	–	6.0+0.9n	–	n = shifts
SRLS	3.0+0.9n	–	–	n = shifts
SS	3.6(4.2)	–	4.8(5.1)	NOTE 1
STB	2.7	–	4.2	
STH	–	–	3.3	
STM	–	–	1.8+1.5n	n = no. of registers
SVC	–	–	7.2+0.6(15-r)	r = R1 field
THI	–	2.1	–	
WB	5.7+3.6n(5.7+4.8n)	–	5.7+3.6n(5.7+4.8n)	NOTES 1,3
WD	2.4(3.6)	–	3.9(5.1)	NOTE 1
WH	2.7/3.6(4.8)	–	4.2/5.1(6.3)	NOTES 1, 4
XH	1.2	2.1	2.7	

NOTE 1 Times in parenthesis are for Micro I/O Bus devices

NOTE 2 Add 2.7 us if Queue Service Enabled in new PSW
Add 3.9 us if System Queue not empty

NOTE 3 n = number of bytes. In RR format, add 0.9r where r is the value of the R2 field.

NOTE 4 Halfword device/byte device

OTHER PERTINENT PROCESSOR TIMES

Normal I/O Interrupt	7.8	NOTE 2
Immediate Interrupt	8.4(9.0)	NOTES 1, 2
Illegal Instruction Interrupt	6.6	NOTE 2

Auto-Load Option (Boot Load)

DMA	Write one HW	9.6	
	Write Burst	2.1n	n = Halfwords
DMA	Read one HW	9.6	
	Read Burst	2.1n	n = Halfwords

APPENDIX 7

I/O REFERENCES

TELETYPE/ASCII HEX CONVERSION TABLE

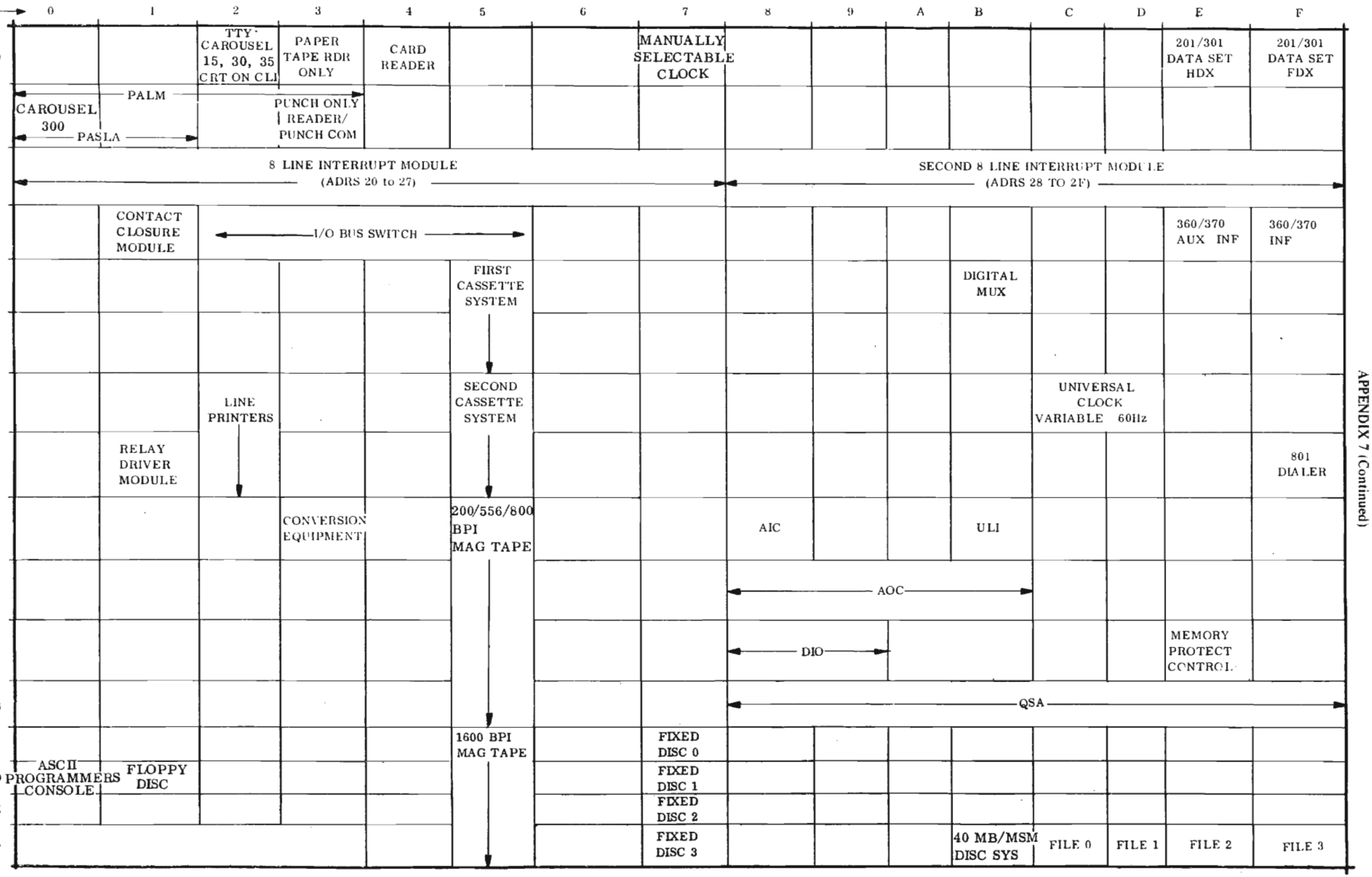
HEX (MSD) →					0	1	2	3	4	5	6	7
(LSD) ↓	Teletype Tape Channels →			8	DEPENDS UPON PARITY*							
				7	0	0	0	0	1	1	1	1
				6	0	0	1	1	0	0	1	1
				5	0	1	0	1	0	1	0	1
	4	3	2	1								
0	0	0	0	0	NULL	DC_0	SPACE	0	@	P		
1	0	0	0	1	SOM	X-ON	!	1	A	Q		
2	0	0	1	0	EOA	TAPE ON	"	2	B	R		
3	0	0	1	1	EOM	X-OFF	#	3	C	S		
4	0	1	0	0	EOT	TAPE OFF	$	4	D	T		
5	0	1	0	1	WRU	ERR	%	5	E	U		
6	0	1	1	0	RU	SYNC	&	6	F	V		
7	0	1	1	1	BELL	LEM	'	7	G	W		
8	1	0	0	0	FE_0	S_0	(	8	H	X		
9	1	0	0	1	HT/SK	S_1	)	9	I	Y		
A	1	0	1	0	LF	S_2	*	:	J	Z		
B	1	0	1	1	VT	S_3	+	;	K	[		
C	1	1	0	0	FF	S_4	,	<	L	\		ACK
D	1	1	0	1	CR	S_5	-	=	M	]		ALT. MODE
E	1	1	1	0	SO	S_6	.	>	N	↑		ESC
F	1	1	1	1	SI	S_7	/	?	O	←		DEL

*Parity bit adjusted for even parity (even number of 1's) on input from Teletype keyboard. Parity bit is ignored on output to Teletype printer.

ASCII CARD CODE CONVERSION TABLE
(029 EBCDIC)

GRAPHIC	7-BIT ASCII CODE	CARD CODE	GRAPHIC	7-BIT ASCII CODE	CARD CODE
SPACE	20	BLANK	@	40	8-4
!	21	12-8-7	A	41	12-1
"	22	8-7	B	42	12-2
#	23	8-3	C	43	12-3
$	24	11-8-3	D	44	12-4
%	25	0-8-4	E	45	12-5
&	26	12	F	46	12-6
'	27	8-5	G	47	12-7
(	28	12-8-5	H	48	12-8
)	29	11-8-5	I	49	12-9
*	2A	11-8-4	J	4A	11-1
+	2B	12-8-6	K	4B	11-2
,	2C	0-8-3	L	4C	11-3
-	2D	11	M	4D	11-4
.	2E	12-8-3	N	4E	11-5
/	2F	0-1	O	4F	11-6
0	30	0	P	50	11-7
1	31	1	Q	51	11-8
2	32	2	R	52	11-9
3	33	3	S	53	0-2
4	34	4	T	54	0-3
5	35	5	U	55	0-4
6	36	6	V	56	0-5
7	37	7	W	57	0-6
8	38	8	X	58	0-7
9	39	9	Y	59	0-8
:	3A	8-2	Z	5A	0-9
;	3B	11-8-6	[	5B	12-8-2
<	3C	12-8-4	\	5C	11-8-1
=	3D	8-6	]	5D	11-8-2
>	3E	0-8-6	↑	5E	11-8-7
?	3F	0-8-7	←	5F	0-8-5

STANDARD-PREFERRED ADDRESS TABLE 5/16



AIC = ANALOG INPUT CONTROLLER
AOC = ANALOG OUTPUT CONTROLLER
DIO = DIGITAL I/O CONTROLLER
QSA = QUAD SYNCHRONOUS ADAPTER
ULI = UNIVERSAL LOGIC INTERFACE

CAROUSEL ASCII/HEX CONVERSION TABLE

BITS				b_6 b_5 b_4	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_3	b_2	b_1	b_0	MSD / LSD	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SPACE	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	A	LF	SUB	*	:	J	Z	j	z
1	0	1	1	B	VT	ESC	+	;	K	[	k	{
1	1	0	0	C	FF	FS	,	<	L	\	l	¦
1	1	0	1	D	CR	GS	–	=	M	]	m	}
1	1	1	0	E	SO	RS	.	>	N	^	n	~
1	1	1	1	F	SI	US	/	?	O	—	o	DEL

NUL	Null	DLE	Data link escape
SOH	Start of heading	DC1-3	Device control
STX	Start of text	DC4	Device stop
ETX	End of text	NAK	Negative acknowledge
EOT	End of transmission	SYN	Synchronous idle
ENQ	Enquiry	ETB	End of transmission block
ACK	Acknowledge	CAN	Cancel
BEL	Audible signal	EM	End of medium
BS	Backspace	SUB	Start of special sequence
HT	Horizontal tabulation	ESC	Escape
LF	Line feed	FS	File separator
VT	Vertical tabulation	GS	Group separator
FF	Form feed	RS	Record separator
CR	Carriage return	US	Unit separator
SO	Shift out	SP	Space
SI	Shift in	DEL	Delete/Idle

APPENDIX 8
BOOTSTRAP LOADER OPTION

INTRODUCTION

The M51-101 or M51-102 Bootstrap Loader option provides a simple, single switch, initial program load capability. Associated with either option is the Turnkey Console shown in Figure A8-1.

The Turnkey Console provides a means of providing power to the system, initializing the system, interrupting execution, and loading the fixed initial program sequence.

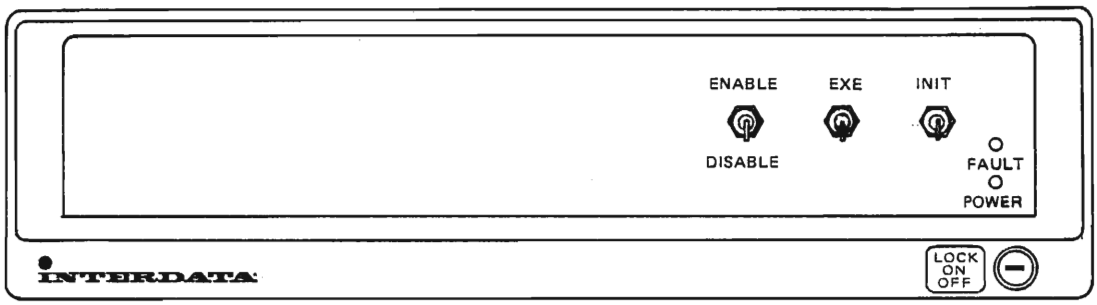


Figure A8-1. Turnkey Console

Key Operated Security Lock

This is a three position, OFF-ON-LOCK, key-operated switch. It controls primary power to the system. It can also disable (LOCK) the Initialize, Console, and Program Load switches, thereby preventing any accidental manual input to the system. The power indicator lamp (POWER) is on when the Key Lock is in the ON or LOCK position.

Control Switches

The momentary contact control switches are only active when the key-operated locking switch is in the ON position.

INITIALIZE (INIT) — The Initialize Switch (INIT) causes the system to be initialized. After the initialize sequence, all device controllers on the system I/O Bus are cleared and certain functions in the processor are reset. The Fault lamp comes on with initialization and is extinguished when the microcode diagnostic routine is completed.

NOTE

A function of the micro-diagnostic routine is to write known data into the first 8KB of main memory. Since this overwrites any data that may have been there, operation of the Initialize Switch should be carefully considered.

EXECUTE (EXE) — The Execute Switch (EXE) causes the running program to be interrupted. Control is given to the microcode driven ASCII Console Device. If the processor was already in the ASCII Console routine, the Run mode is entered.

ENABLE/DISABLE — The Enable/Disable switch provides 12 VAC, 50/60 HRz, to the Non-Maskable Interrupt logic in the processor. In the Enable (UP) position the Real Time Clock feature is enabled.

CONFIGURATION

The M51-101 Bootstrap Loader option has space on the basic Model 5/16 Processor card for two 1024 x 8 bit or one 2048 x 8 bit customer ROM devices.

In the M51-102 Bootstrap Loader option, this space may be occupied by the standard OS/16 MT2 bootstrap loader chip set.

OPERATION

The presence of the Bootstrap Loader is indicated to the microprogram by a testable strap (ALO). On power up or after an initalize sequence, the microcode diagnostic is executed. Then, if there is no Memory Addressable ROM, the Bootstrap Loader option is tested. If equipped and enabled, the microcode reads consecutive bytes from the Bootstrap Loader ROM. These reads occur over the I/O Bus. The Bootstrap Loader selects itself on initialize and is de-selected when any I/O device is accessed.

Figure A8-2 shows the data format for the Bootstrap Loader ROM devices. The first two bytes from the Bootstrap Loader correspond to a new PSW value. The second two bytes correspond to a new Location Counter value. The third and fourth byte pairs are the main memory start and end addresses, respectively, of the area of memory that is to receive the remaining Bootstrap Loader data bytes. The start address must be less than the end address. If not, the ASCII Console routine is defaulted to. Otherwise, the remaining bytes from the Bootstrap Loader are copied into the area of memory defined by the start and end addresses.

On completion of the Bootstrap Load, an Output Command Reset is issued to the ASCII Console Device. Depending on PSW bit 0, the processor then either enters the wait state or begins program execution at the address specified by the new Location Counter value.

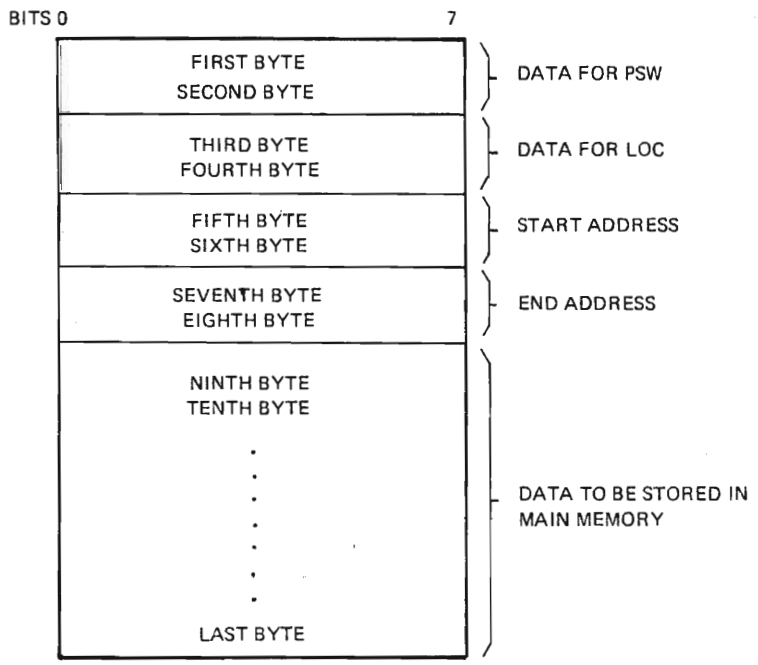
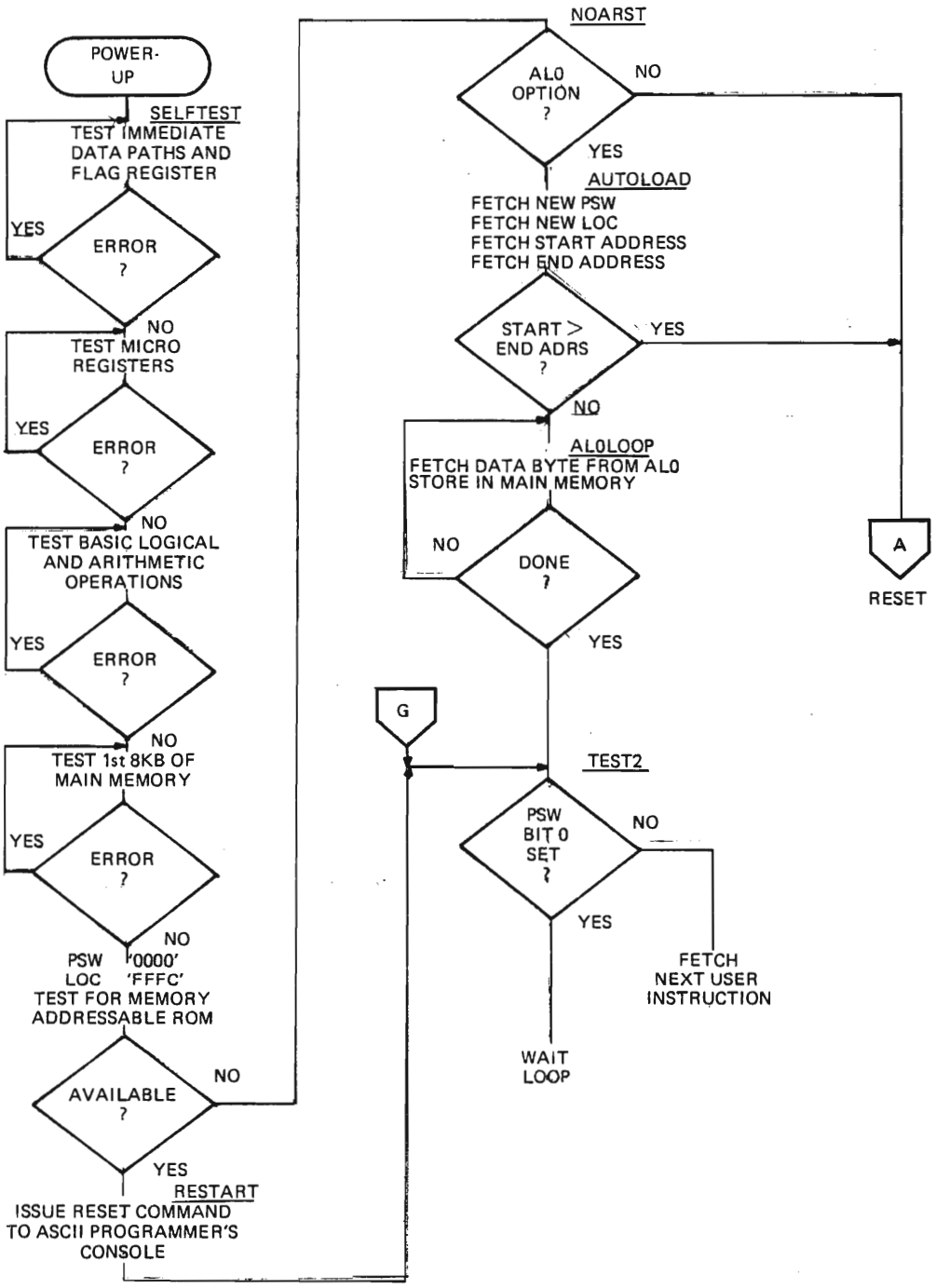
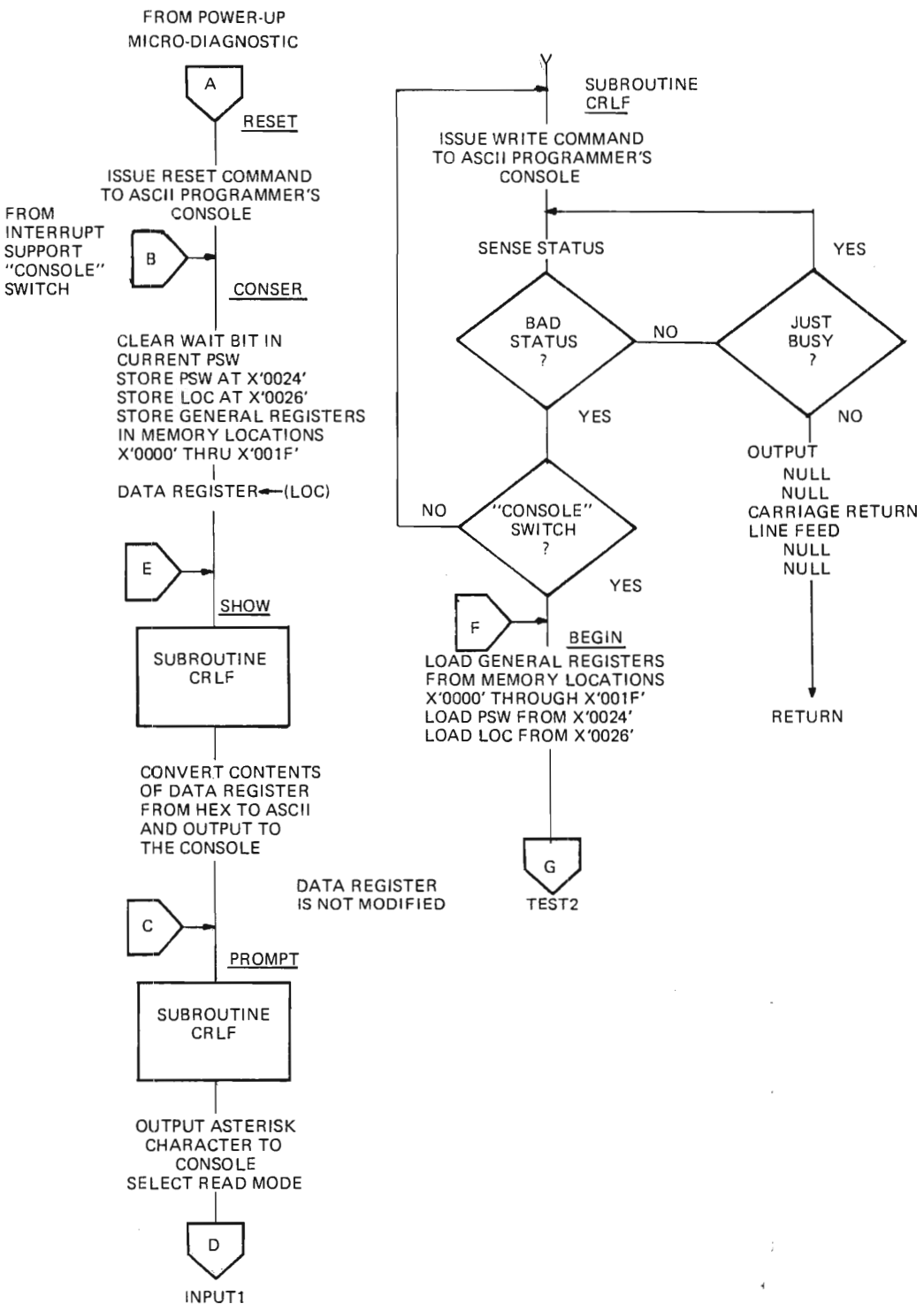
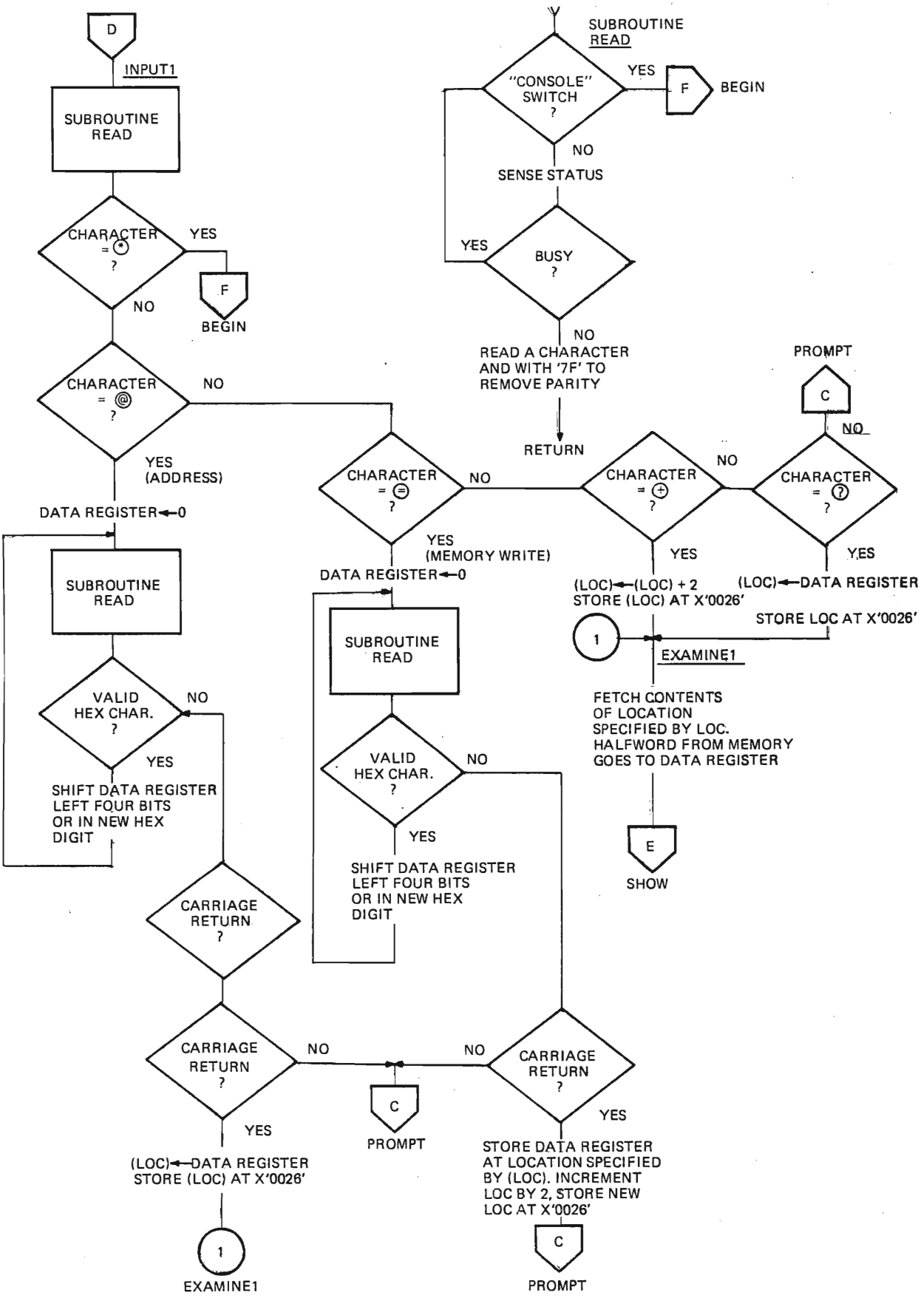


Figure A8-2. Bootstrap Loader Data Format

APPENDIX 9
INITIALIZATION AND CONSOLE ROUTINE FLOWCHART




FROM POWER-UP
MICRO-DIAGNOSTIC
A
RESET
ISSUE RESET COMMAND
TO ASCII PROGRAMMER'S
CONSOLE
FROM
INTERRUPT
SUPPORT
"CONSOLE"
SWITCH
B
CONSER
CLEAR WAIT BIT IN
CURRENT PSW
STORE PSW AT X'0024'
STORE LOC AT X'0026'
STORE GENERAL REGISTERS
IN MEMORY LOCATIONS
X'0000' THRU X'001F'
DATA REGISTER←(LOC)
E
SHOW
SUBROUTINE
CRLF
CONVERT CONTENTS
OF DATA REGISTER
FROM HEX TO ASCII
AND OUTPUT TO
THE CONSOLE
C
PROMPT
SUBROUTINE
CRLF
OUTPUT ASTERISK
CHARACTER TO
CONSOLE
SELECT READ MODE
D
INPUT1
SUBROUTINE
CRLF
ISSUE WRITE COMMAND
TO ASCII PROGRAMMER'S
CONSOLE
SENSE STATUS
YES
BAD
STATUS
?
NO
JUST
BUSY
?
YES
NO
OUTPUT
NULL
NULL
CARRIAGE RETURN
LINE FEED
NULL
NULL
RETURN
NO
"CONSOLE"
SWITCH
?
YES
F
BEGIN
LOAD GENERAL REGISTERS
FROM MEMORY LOCATIONS
X'0000' THROUGH X'001F'
LOAD PSW FROM X'0024'
LOAD LOC FROM X'0026'
G
TEST2
DATA REGISTER
IS NOT MODIFIED


D
INPUT1
SUBROUTINE READ
CHARACTER = (*) ?
YES
F
BEGIN
NO
CHARACTER = (@) ?
NO
YES (ADDRESS)
DATA REGISTER←0
SUBROUTINE READ
VALID HEX CHAR. ?
NO
YES
SHIFT DATA REGISTER LEFT FOUR BITS OR IN NEW HEX DIGIT
CARRIAGE RETURN ?
CARRIAGE RETURN ?
NO
YES
(LOC)←DATA REGISTER STORE (LOC) AT X'0026'
1
EXAMINE1
C
PROMPT
SUBROUTINE READ
"CONSOLE" SWITCH ?
YES
F
BEGIN
NO
SENSE STATUS
YES
BUSY ?
NO
READ A CHARACTER AND WITH '7F' TO REMOVE PARITY
RETURN
CHARACTER = (=) ?
NO
YES (MEMORY WRITE)
DATA REGISTER←0
SUBROUTINE READ
VALID HEX CHAR. ?
NO
YES
SHIFT DATA REGISTER LEFT FOUR BITS OR IN NEW HEX DIGIT
CARRIAGE RETURN ?
NO
YES
STORE DATA REGISTER AT LOCATION SPECIFIED BY (LOC). INCREMENT LOC BY 2, STORE NEW LOC AT X'0026'
C
PROMPT
CHARACTER = (+) ?
NO
YES
(LOC)←(LOC) + 2 STORE (LOC) AT X'0026'
1
EXAMINE1
FETCH CONTENTS OF LOCATION SPECIFIED BY LOC. HALFWORD FROM MEMORY GOES TO DATA REGISTER
E
SHOW
PROMPT
C
NO
CHARACTER = (?) ?
YES
(LOC)←DATA REGISTER
STORE LOC AT X'0026'

```
PROG= *NONE*      ASSEMBLED BY CAL 03-066R04-01 (32-BIT)

                                    1          SCRAT
                                    2          CROSS
                                    3          TARGT  16
                                    4  *
                                    5  * SAMPLE PROGRAM FOR ASCII PROGRAMMERS CONSOLE ON THE
                                    6  * SERIAL I/O PORT CURRENT LOOP INTERFACE
                                    7  *
                                    8  * PROGRAM USES SENSE STATUS LOOPS TO READ TEN CHARACTERS
                                    9  * THEN OUTPUT A CARRIAGE RETURN, LINE FEED, THE ORIGINAL
                                   10  * TEN INPUT CHARACTERS AND ANOTHER CARRIGE RETURN AND
                                   11  * LINE FEED SEQUENCE.
                                   12  *
                                   13  * REGISTER ASSIGNMENTS
                                   14  *
          0000 0000                15  R0,LINK   EQU    0              LINK REGISTER
          0000 0001                16  R1,DEV    EQU    1              DEVICE ADDRESS REGISTER
          0000 0002                17  R2,STAT   EQU    2              DEVICE STATUS REGISTER
          0000 0003                18  R3,INDX   EQU    3              BUFFER INDEX REGISTER
          0000 0004                19  R4,DATA   EQU    4              DEVICE DATA REGISTER
                                   20  *
                                   21  *
0000R     4810 0084R               22          LH     R1,DEV,DEVICE  LOAD R1,DEV WITH DEVICE NUMBER
0004R     DE10 0087R               23  OC1     OC     R1,DEV,READ    SELECT READ MODE
0008R     9B14                     24          RDR    R1,DEV,R4,DATA DUMMY READ TO FORCE BUSY
000AR     9D12                     25          SSR    R1,DEV,R2,STAT SENSE STATUS
000CR     4210 0004R               26          BTC    1,OC1          RE-ISSUE COMMAND IF DU
0010R     0733                     27          XHR    R3,INDX,R3,INDX CLEAR INDEX REGISTER
0012R     4100 006ER               28  INPUT   BAL    R0,LINK,GETBYT INPUT A CHARACTER
0016R     D243 008AR               29          STB    R4,DATA,BUFFER(R3,INDX)STORE IN BUFFER
001AR     2631                     30          AIS    R3,INDX,1      INCREMENT INDEX
001CR     C530 000A                31          CLHI   R3,INDX,10     TEN CHARACTERS YET?
0020R     2087                     32          BLS    INPUT          LOOP
                                   33  *
                                   34  *
0022R     DE10 0088R               35          OC     R1,DEV,WRITE   SELECT WRITE MODE
0026R     2440                     36          LIS    R4,DATA,0      NULL CHARACTER
0028R     4100 007CR               37          BAL    R0,LINK,OUTPUT OUTPUT IT
002CR     244D                     38          LIS    R4,DATA,X'0D'  CARRIAGE RETURN
002ER     4100 007CR               39          BAL    R0,LINK,OUTPUT OUTPUT IT
0032R     244A                     40          LIS    R4,DATA,X'0A'  LINE-FEED
0034R     4100 007CR               41          BAL    R0,LINK,OUTPUT OUTPUT IT
0038R     2440                     42          LIS    R4,DATA,0      NULL CHARACTER
003AR     4100 007CR               43          BAL    R0,LINK,OUTPUT OUTPUT IT
003ER     0733                     44          XHR    R3,INDX,R3,INDX RESET INDEX
0040R     D343 008AR               45  OUTPTL  LB     R4,DATA,BUFFER(R3,INDX)
0044R     4100 007CR               46          BAL    R0,LINK,OUTPUT OUTPUT CHARACTER
0048R     2631                     47          AIS    R3,INDX,1      INCREMENT INDEX
004AR     C530 000A                48          CLHI   R3,INDX,10     DONE TEN CHARACTERS?
004ER     2087                     49          BLS    OUTPTL         LOOP
                                   50  *
0050R     244D                     51          LIS    R4,DATA,X'0D'  CARRIAGE RETURN
0052R     4100 007CR               52          BAL    R0,LINK,OUTPUT OUTPUT IT
0056R     244A                     53          LIS    R4,DATA,X'0A'  LINE-FEED
0058R     4100 007CR               54          BAL    R0,LINK,OUTPUT OUTPUT IT
```

```
005CR  2440                55          LIS    R4,DATA,0          NULL CHARACTER
005ER  4100 007CR          56          BAL    R0,LINK,OUTPUT
0062R  DE10 0087R          57          OC     R1,DEV,READ        SWITCH BACK TO READ MODE
0066R  C200 006AR          58  WAIT    LPSW   WAIT1              HALT
                           59  *
006AR  8000                60  WAIT1   DC     X'8000',WAIT
006CR  0066R
                           61  *
                           62  *                                 SUBROUTINE "GETBYT"
006ER  9D12                63  GETBYT  SSR    R1,DEV,R2,STAT     SENSE STATUS
0070R  2091                64          BTBS   9,1                LOOP ON BUSY OR DU
0072R  9B14                65          RDR    R1,DEV,R4,DATA     INPUT DATA
0074R  9A14                66          WDR    R1,DEV,R4,DATA     ECHO BACK
0076R  C440 007F           67          NHI    R4,DATA,X'7F'      FORCE PARITY BIT RESET
007AR  0300                68          BR     R0,LINK            RETURN TO CALL
                           69  *
                           70  *                                 SUBROUTINE "OUTPUT"
007CR  9D12                71  OUTPUT  SSR    R1,DEV,R2,STAT     SENSE STATUS
007ER  2091                72          BTBS   9,1                LOOP ON BUSY OR DU
0080R  9A14                73          WDR    R1,DEV,R4,DATA     OUTPUT DATA
0082R  0300                74          BR     R0,LINK
                           75  *
                           76  *
                           77  *
0084R  00C0                78  DEVICE  DC     X'00C0'            SERIAL I/O PORT DEVICE NUMBER
                           79  *
0086R  03                  80  RESET   DB     X'03'              RESET COMMAND
0087R  82                  81  READ    DB     X'82'              READ COMMAND
0088R  02                  82  WRITE   DB     X'02'              DISABLE, WRITE
0089R  00                  83          DB     *
008AR                      84  BUFFER  DS     10                 TEN BYTE STORAGE BUFFER
0094R                      85          END
```

NO ERRORS 0 SQUEZ PASSES

CAL 04-01

ABSTOP	0000												
ADC	0002												
BUFFER	008AR	29	45										
DEVICE	0084R	22											
GETBYT	006ER	20											
IMPTOP	0094R												
INPUT	0012R	32											
LADC	0001												
OC1	0004R	26											
OUTPTL	0040R	49											
OUTPUT	007CR	37	39	41	43	46	52	54	56				
PURETOP	0000R												
R0,LINK	0000	28	37	39	41	43	46	52	54	56	68	74	
R1,DEV	0001	22	23	24	25	35	57	63	65	66	71	73	
R2,STAT	0002	25	63	71									
R3,INDX	0003	27	27	29	30	31	44	44	45	47	48		
R4,DATA	0004	24	29	36	38	40	42	45	51	53	55	65	66 67
		73											
READ	0087R	23	57										
RESET	0086R												
WAIT	0066R	60											
WAIT1	006AR	58											
WRITE	0088R	35											

Appendix 10 Continued

```
PROG= *NONE*      ASSEMBLED BY CAL 03-066R04-01 (32-BIT)

                                    1          SCRAT
                                    2          CROSS
                                    3          TARGT  16
                                    4  *
                                    5  * SAMPLE PROGRAM FOR ASCII PROGRAMMERS CONSOLE ON THE
                                    6  * SERIAL I/O PORT CURRENT LOOP INTERFACE
                                    7  *
                                    8  * PROGRAM USES INTERRUPT MECHANISM TO READ TEN CHARACTERS
                                    9  * THEN OUTPUT A CARRIAGE RETURN, LINE FEED, THE ORIGINAL
                                   10  * TEN INPUT CHARACTERS AND ANOTHER CARRIAGE RETURN AND
                                   11  * LINE FEED SEQUENCE.
                                   12  *
                                   13  * REGISTER ASSIGNMENTS
                                   14  *
                0000 0000          15  R0         EQU    0                ACCUMULATOR
                0000 0002          16  R2,DEV     EQU    2                ASCII CONSOLE DEVICE NUMBER
                0000 0004          17  R4,COUNT   EQU    4                INDEX
                0000 0005          18  R5,DATA    EQU    5                ACCUMULATOR
                0000 0006          19  R6,IDEV    EQU    6                INTERRUPTING DEVICE NUMBER
                0000 0007          20  R7,STAT    EQU    7                DEVICE STATUS
                0000 000E          21  R14        EQU    14               ACCUMULATOR
                                   22  *
                                   23  *
0000R  4820 00BCR                  24             LH     R2,DEV,DEVICE
0004R  2451                        25             LIS    R5,DATA,1
0006R  4050 00BAR                  26             STH    R5,DATA,FLAG     SET READ FLAG
                                   27  *
                                   28  *                                  SET-UP LOW MEMORY
000AR  24E0                        29             LIS    R14,0
000CR  40E0 0040                   30             STH    R14,X'40'        OLD PSW, I/O INTERRUPT
0010R  40E0 0042                   31             STH    R14,X'42'
0014R  40E0 0044                   32             STH    R14,X'44'        NEW PSW, I/O INTERRUPT
0018R  C850 0042R                  33             LHI    R5,DATA,INTERUPT
001CR  4050 0046                   34             STH    R5,DATA,X'46'    NEW LOC, I/O INTERRUPT
0020R  950E                        35             EPSR   R0,R14           DISABLE INTERRUPTS IN PSW
0022R  DE20 00C0R                  36             OC     R2,DEV,DISWRITE  DISABLE WRITE
0026R  DE20 00BER                  37             OC     R2,DEV,READENBL  COMMAND READ
002AR  2440                        38             LIS    R4,COUNT,0       RESET INDEX
002CR  C200 0030R                  39             LPSW   INTRPTW          ENABLE INTERRUPTS
0030R  4000                        40  INTRPTW    DC     X'4000',CONTINUE
0032R  0034R
0034R  2452                        41  CONTINUE   LIS    R5,DATA,2
0036R  6150 00BAR                  42             AHM    R5,DATA,FLAG     FLAG GETS 2 OR 3
003AR  C200 003ER                  43  WAIT1      LPSW   INTRPTX
003ER  C000                        44  INTRPTX    DC     X'C000',WAIT1    HALT, WAIT FOR INTERRUPT
0040R  003AR
                                   45  *
                                   46  *                                  INTERRUPT ROUTINE
                                   47  *
0042R  9F67                        48  INTERUPT   ACKR   R6,IDEV,R7,STAT  ACKNOWLEDGE INTERRUPTS
0044R  0562                        49             CLHR   R6,IDEV,R2,DEV   SEE IF ASCII CONSOLE
0046R  4230 00A4R                  50             BNE    LOOK             SKIP IF NO
004AR  C370 000D                   51             THI    R7,STAT,X'0D'    TEST RETURNED STATUS
004ER  4230 00A6R                  52             BNZ    STSERR           ERROR IF BAD STATUS
```

```
0052R  4850 00BAR          53           LH      R5,DATA,FLAG        TEST READ/WRITE FLAG
0056R  2751                54           SIS     R5,DATA,1
0058R  4330 0074R          55           BZ      ISRETURN            IGNORE FIRST READ INTERRUPT
005CR  C350 0001           56           THI     R5,DATA,1           WRITE?
0060R  4230 0088R          57           BNZ     WRITEBR             BRANCH IF YES
                           58  *
0064R  9B65                59  READBR   RDR     R6,IDEV,R5,DATA     READ DATA
0066R  9A65                60           WDR     R6,IDEV,R5,DATA     ECHO BACK
0068R  D254 00ABR          61           STB     R5,DATA,BUFFER(R4,COUNT)STORE
006CR  2641                62           AIS     R4,COUNT,1          INCREMENT INDEX
006ER  C540 000A           63           CLHI    R4,COUNT,10         LIMIT TEN CHARACTERS
0072R  2383                64           BNLS    GOWRITE
0074R  C200 0040           65  ISRETURN LPSW    X'40'               RETURN TO PREVIOUS STATE
                           66  *
0078R  2450                67  GOWRITE  LIS     R5,DATA,0
007AR  4050 00BAR          68           STH     R5,DATA,FLAG        SET WRITE FLAG
007ER  2440                69           LIS     R4,COUNT,0
0080R  DE60 00BFR          70           OC      R6,IDEV,WRITE       SWITCH TO WRITE MODE
0084R  C200 0040           71           LPSW    X'40'               WAIT FOR INTERRUPT
                           72  *
                           73  *
0088R  DA64 00A8R          74  WRITEBR  WD      R6,IDEV,W,BUFF(R4,COUNT)OUTPUT CHARACTER
008CR  2641                75           AIS     R4,COUNT,1          INCREMENT INDEX
008ER  C540 0010           76           CLHI    R4,COUNT,16
0092R  2383                77           BNLS    DONE
0094R  C200 0040           78           LPSW    X'40'               WAIT FOR NEXT INTERRUPT
                           79  *
       0000 0098R          80  DONE     EQU     *
0098R  DE20 00BER          81           OC      R2,DEV,READ
009CR  C200 00A0R          82           LPSW    EOJ
                           83  *
00A0R  8000                84  EOJ      DC      X'8000',DONE
00A2R  0098R
                           85  *
                           86  *
       0000 00A4R          87  LOOK     EQU     *
                           88  *
00A4R  2200                89           BS      *
                           90  *
       0000 00A6R          91  STSERR   EQU     *
00A6R  2200                92           BS      *
                           93  *
                           94  *
00A8R  000D0A              95  W.BUFF   DB      X'00',X'0D',X'0A'
00ABR                      96  BUFFER   DS      10
00B5R  0D0A0000            97           DB      X'0D',X'0A',0,0
00B9R  00                  98           DB      *
00BAR  0000                99  FLAG     DCX     0
00BCR  00C0               100  DEVICE   DC      X'00C0'
       0000 00BER         101  READ     EQU     *
00BER  82                 102  READENBL DB      X'82'
00BFR  22                 103  WRITE    DB      X'22'
00C0R  02                 104  DISWRITE DB      X'02'
00C1R  00                 105           DB      *
00C2R                     106           END
```

Appendix 10 Continued

NO ERRORS 0 SQUEZ PASSES

CAL 04-01

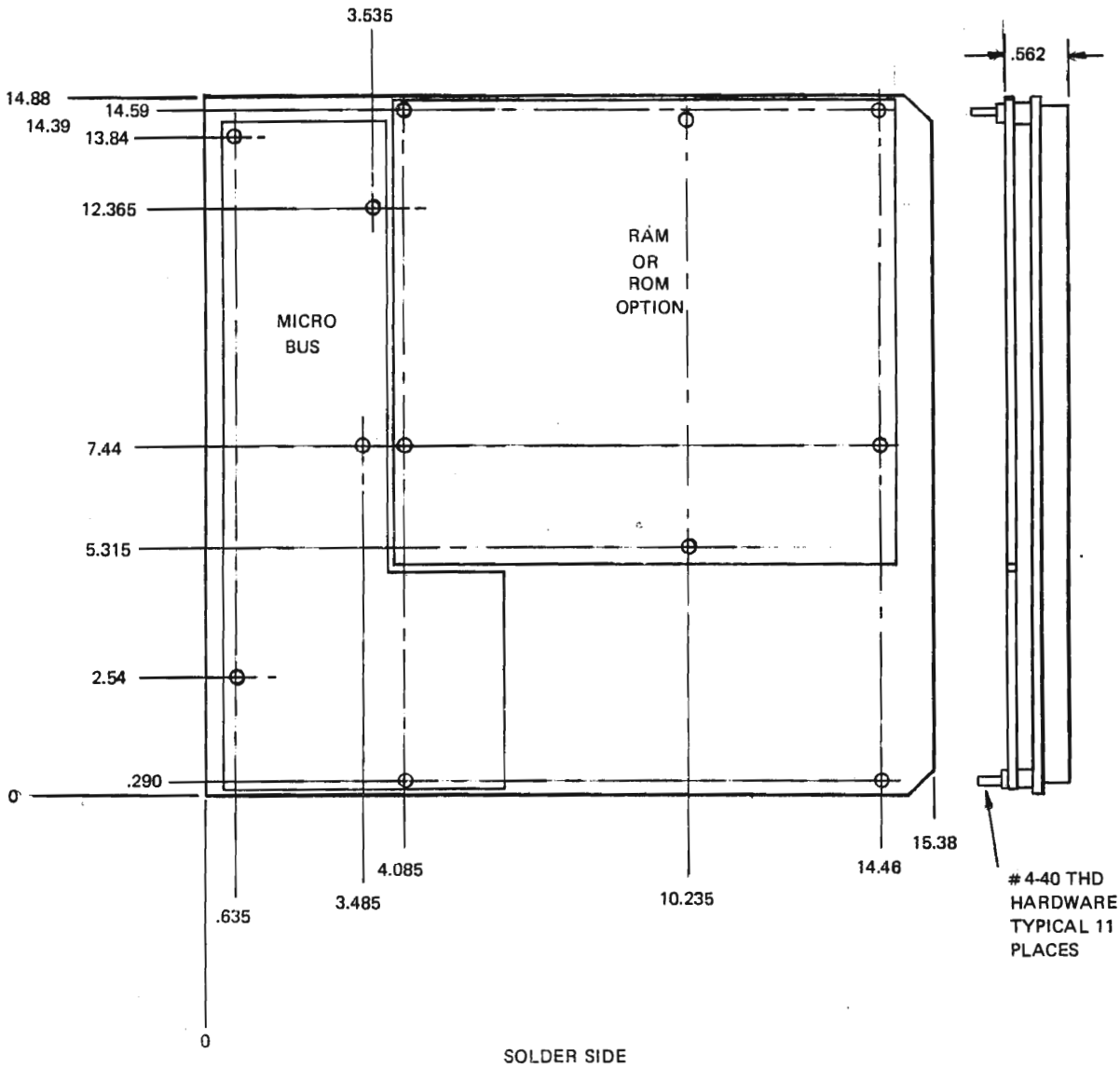
ABSTOP	0000												
ADC	0002												
BUFFER	00ABR	61											
CONTINUE	0034R	40											
DEVICE	00BCR	24											
DISWRITE	00C0R	36											
DONE	0098R	77	84										
EOJ	00A0R	82											
FLAG	00BAR	26	42	53	68								
GOWRITE	0078R	64											
IMPTOP	00C2R												
INTERUPT	0042R	33											
INTRPTW	0030R	59											
INTRPTX	003ER	43											
ISRETURN	0074R	55											
LADC	0001												
LOOK	00A4R	50											
PURETOP	0000R												
R0	0000	35											
R14	000E	29	30	31	32	35							
R2,DEV	0002	24	36	37	49	81							
R4,COUNT	0004	38	61	62	63	69	74	75	76				
R5,DATA	0005	25	26	33	34	41	42	53	54	56	59	60	61	67	68
R6,IDEV	0006	48	49	59	60	70	74								
R7,STAT	0007	48	51												
READ	00BER	81													
READBR	0064R														
READENBL	00BER	37													
STSERR	00A6R	52													
W,BUFF	00A8R	74													
WAIT1	003AR	44													
WRITE	00BFR	70													
WRITEBR	0088R	57													

Appendix 10 Continued

APPENDIX 11
PHYSICAL AND ELECTRICAL CHARACTERISTICS

This appendix contains information for using the Model 5/16 Processor when it is not packaged in a standard INTERDATA chassis.

A. PROCESSOR BOARD MOUNTING AND COOLING INFORMATION



SOLDER SIDE

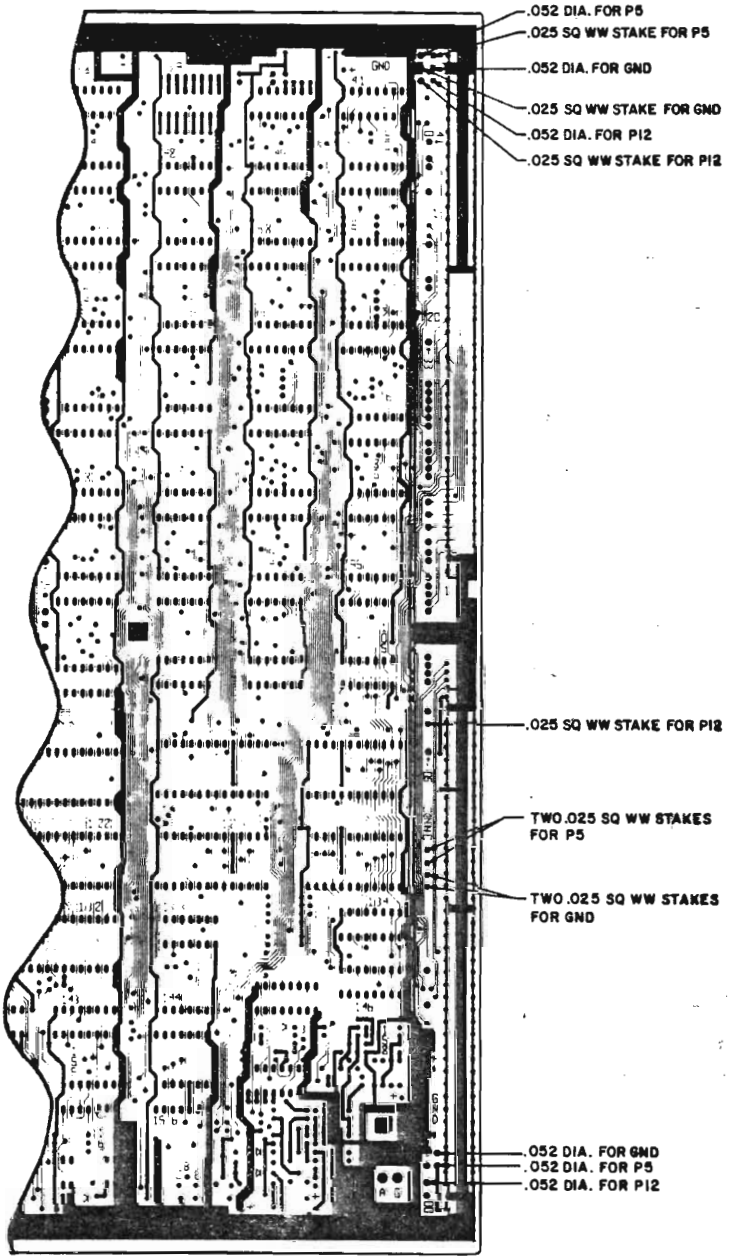
NOTE

1. THE STUD POSITIONS SHOWN MAY BE USED TO MOUNT BOARD WHEN THE STANDARD CARD FILE IS NOT USED. POSITIONS USED ARE OPTIONAL PER CUSTOMER REQUIREMENTS.
2. ADEQUATE COOLING MUST BE PROVIDED TO MAINTAIN A MAXIMUM CASE TEMPERATURE OR ANY COMPONENT TO A MAXIMUM OF 70° C. AT AN AMBIENT TEMPERATURE OF 50° C, A MINIMUM AIR FLOW OF 10CFM MUST BE DELIVERED EVENLY ACROSS THE BOARD.

B. 5/16 CURRENT MEASUREMENTS (P5=5.25V, P12=13.0V):

ACTIVE		STATIC		CONFIGURATION		
IP5	IP12	IP5	IP12	MICRO I/O BUS	RAM OPTION	TEST AID
5.85a	318 ma	5.80a	214 ma	YES	YES	YES
5.39a	318 ma	5.25a	214 ma	YES	YES	NO
5.45a	248 ma	5.40a	153 ma	YES	NO	YES
4.95a	2 44 ma	4.95a	148 ma	YES	NO	NO
–	–	5.40a	170 ma	NO	YES	YES
–	–	4.90a	170 ma	NO	YES	NO
–	–	5.00a	140 ma	NO	NO	YES
–	–	4.55a	140 ma	NO	NO	NO
4.60a	200 ma	4.55a	140 ma	CENTRAL PROCESSING UNIT (CPU)		
400 ma	44 ma	400 ma	44 ma	MICRO I/O BUS		
440 ma	74 ma	400 ma	30 ma	RAM OPTION		
500 ma	–	550 ma	–	TEST AID		
5.94a	318 ma	5.90a	214 ma	TOTALS		

C. OPTIONAL POWER CONNECTIONS



APPARATUS SIDE OF 5/16 PROCESSOR BOARD

INDEX

Acknowledge Interrupt, 8-16
Address and Data Transfer (Figures 8-28 and 8-29), 8-34, 8-35
Address Lines, 8-1
Arithmetic References (Appendix 5), A5-1
ASCII Character U, Eleven Bit Code (Figure 9-3), 9-2
ASCII Console, 9-3
ASCII Console Command Summary (Table 9-1), 9-2
ASCII Programmers Console (Chapter 9), 9-1
Automatic Vectoring, 7-18

Block I/O, 8-16
15 Inch Boards, 8-36
7 Inch Boards, 8-36
Boolean Operations, 3-2
Bootstrap Loader Turnkey Console (Figures A8-1), A8-1
Bootstrap Loader (Figure A8-2), A8-2
Bootstrap Option (Appendix 8), A8-1
Branch Instructions, 4-1
Branch Instruction Formats, 4-1, 2-7
Branching (Chapter 4), 4-1
Bus Sequence for Byte or Halfword Device (Figure 8-26), 8-33
Byte-Oriented Device Controller Design, 8-32

Centronics Line Printer Interface (Figure 8-11), 8-14
Circular List Definition (Figure 3-2), 3-3
Circular List (Figure 3-3), 3-3
Command, 9-6
Command Byte (Figure 8-25), 8-31
Configuration, 9-1
Console Routine Flowchart (Appendix 9), A9-1
Control of I/O Operations, 7-17
Control Lines, 8-3
Control Switches, 9-3

Data and Command Output, 8-32
Data Formats, 3-2, 5-1, 9-6
Data and Status Input, 8-32
Data Lines, 8-1
Data Output Multiplexor (Figure 8-8), 8-11
Data Transfer, 8-33
DC/DC Converter (Figure 8-23), 8-29
DC/DC Converter Specifications, 8-29
Device Controller Addressing, 8-24
Direct Memory Access, 7-20
DMA Devices, 7-20
DMA Operation, 7-20

ENABLE Timing (Figure 8-12), 8-14
Extended Branch Mnemonics (Appendix 4), 4-7, A4-1
External Interrupt, 6-3

Fixed Point Arithmetic, (Chapter 5), 5-1
Fixed Point Data Words Format (Figure 5-1), 5-1
Fixed Point Fault Interrupt, 6-3
Fixed Point Format Relations (Tables 5-1), 5-1
Fixed Point Instructions, 5-1
Fixed Point Instruction Formats, 5-2
Fixed Point Number Range, 5-1
Functions Common to the 7 Inch and 15 Inch I/O Interface Boards, 8-37
Functions of the C/D Data Line (Table 8-2), 8-3

General Interface to Multiplexor Bus (Figure 8-20), 8-24
General Multiplexor Bus Interface, 8-28
General Register, 2-3
General Register Examination and Modifications, 9-4

Half Board Adapter Kit (Figure 8-3), 8-36

Illegal Instruction Interrupt, 6-4
Implementation, 2-5
Initialize Line, 8-4
I/O Bus Sequence and Timing, 8-34
Input/Output Operations (Chapter 7), 7-1
Input/Output System, 1-2
Instruction Formats (Figure 2-5), 2-6
Instruction Summary - Alphabetical with Attributes (Appendix 2), A2-1
Instruction Summary - Numerical (Appendix 3), A3-1
Instruction Timing (Appendix 6), A6-1
Interpretation of PSW Bits (Table 2-1), 2-2
Interrupt Control, 8-24
Interrupt Driven I/O, 7-18
Interrupt I/O Control, 8-23, 9-7/9-8
Interrupt System, 6-2
Interrupt Timing (Figure 8-30), 8-35
Interrupts, 9-7/9-8
Introduction (Chapter 1), 1-1
I/O Back Panel Connections (Figure 8-33), 8-38
I/O Instruction Formats, 7-3
I/O Interface Transmit and Receive Characteristics (Figure 8-19), 8-23
I/O References (Appendix 7), A7-1

Keyboard Layout (Figure 9-2), 9-1
Key Operated Security Lock, 9-2

List Processing, 3-3
Logic and Timing for Model 5/16 Micro DMA (Figure 8-18), 8-19
Logical Data (Figure 3-1), 3-2
Logical Instruction Formats, 3-4
Logical Instructions, 3-4

Memory, 1-1, 2-4
Memory Addressable ROM, 2-5
Memory Expansion, 2-4
Memory Initialization, 9-5
Micro DMA Bus, 8-17
Micro I/O Bus, 8-1, 8-4
Micro I/O Bus Data Line Buffering, 8-9
Micro I/O Device Addressing, 8-8
Micro I/O Bus Operations, 8-4
Micro I/O Bus Timing, 8-23
Micro I/O Interface Design, 8-4
Modify Open Cell, 9-4
Multiplexor Bus, 8-20
Multiplexor Bus Input Timing (Figure 8-22), 8-27
Multiplexor Bus Length Restrictions, 8-24
Multiplexor Bus Lines (Table 8-7), 8-20
Multiplexor Bus Loading Rules, 8-22
Multiplexor Bus Output Timing (Figure 8-21), 8-27
Multiplexor Bus Timing, 8-26
Multiplexor Bus Terminators, 8-24
Multiplexor Bus Wiring, 8-26
Multiplexor I/O Device Controller Logic Design, 8-22
Multiplexor I/O Interface Design (Programming Characteristics), 8-31
Multiplexor I/O Interface Physical Packaging, Cabling and Connections, 8-36

Open and Examine Location Shown, 9-4
Open and Examine Next Cell, 9-4
Open Cell and Examine, 9-3
Operations, 4-1
Operation Procedures, 9-3
Output Command (OC or OCR), 9-2

Physical and Electrical Characteristics (Appendix 11), A11-1/A11-2
Power Up, 9-3
Printer Circuit Board (Figure 8-32), 8-37
Processor, 2-1
Processor/Controller Communication, 7-2
Processor Interrupts, 2-2
Processor Operations, 2-5
ProcessorsOptions and Peripherals, 1-2
Program Execution, 9-4
Program Status Word, 6-2, 2-22
Program Status Word Examination and Modification, 9-4
Program Status Word Format (Figure 2-2) (Figure 6-1), 2-2, 6-1
Program Termination, 9-4
Programming Examples, 2-7
Programming Instructions, 9-2
Programming Notes, 9-7/9-8
Programming Sequences, 9-3
Programming Switches, 1-7

Read Data or Sense Status, 8-15
Read/Write Data Transfer Bus Sequence (Figure 8-27), 8-34
Receiver Characteristics (Table 8-5), 8-6
Register and Immediate Storage (RI) Format, 2-8
Register and Indexed Storage (RX) Format, 2-7
Register to Register (RR) Format, 2-7
Relationship Between Micro I/O Bus and Motorola and Intel Chips (Table 8-6), 8-7
Reserved Memory Locations, 2-3

Sample Program (Appendix 10), A10-1
Sense Status (SS or SSR), 9-2
Serial Input/Output Port Status and Command Data (Table 9-2), 9-66
Short Form (SF) Format, 2-7
Signals Carried by Micro I/O Bus Cables (Table 8-3), 8-6
Simulated Interrupt, 6-4
Software, 1-2
Software Vectoring, 7-19
Status, 9-6
Status and Command Bytes, 9-6
Status Byte (Figure 8-24), 8-31
Status Monitoring I/O, 7-17, 9-7/9-8
Status Switching and Interrupts (Chapter 6), 6-1
Status Switching Instruction Formats, 6-4
Status Switching Instructions, 6-4
Subroutine Linkage, 4-1
Supervisor Call Interrupt, 6-4
System Block Diagram (Figure 2-1), 2-1
System Queue Interrupt, 6-4

Tape Recorder Interface (Figure 8-10), 8-13
Teletype Controller (Figure 8-9), 8-12
Transmitter Characteristics (Table 8-4), 8-6
Typical Intel Peripheral Chip Control (Figure 8-4), 8-8
Typical Motorola Peripheral Chip Control (Figure 8-3), 8-7

Wait State, 6-2

PUBLICATION COMMENT FORM

Please use this postage-paid form to make any comments, suggestions, criticisms, etc. concerning this publication.

From _____ Date _____

Title _____ Publication Title _____

Company _____ Publication Number _____

Address _____

_____

_____

FOLD FOLD

Check the appropriate item.

☐ Error Page No. _____ Drawing No. _____

☐ Addition Page No. _____ Drawing No. _____

☐ Other Page No. _____ Drawing No. _____

Explanation:

FOLD FOLD

CUT ALONG LINE

Fold and Staple

No postage necessary if mailed in U.S.A.

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT No. 22
OCEANPORT, N. J.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY:

INTERDATA®

Subsidiary of PERKIN-ELMER
Oceanport, New Jersey 07757, U.S.A.

TECH PUBLICATIONS DEPT. MS 229

FOLD

FOLD

STAPLE

STAPLE