MODULAR INFORMATION PROCESSOR
SYSTEM REFERENCE MANUAL
FOR
MIP-16 SYSTEMS

SANDERS ASSOCIATES, INC.
95 CANAL STREET
NASHUA, NEW HAMPSHIRE
First Edition (February 1972)

Specifications contained herein are subject to change without notice. Such revisions will be reported in subsequent editions or newsletters.

Communications should be addressed to:

Systems Engineering
Group 1-2706
Sanders Associates, Inc.
95 Canal Street
Nashua, New Hampshire 03060
# TABLE OF CONTENTS

## SECTION 1
### INTRODUCTION

<table>
<thead>
<tr>
<th>Paragraph</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1-1</td>
</tr>
</tbody>
</table>

## SECTION 2
### SYSTEM ARCHITECTURE

<table>
<thead>
<tr>
<th>Paragraph</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0</td>
<td>2-1</td>
</tr>
<tr>
<td>2.1</td>
<td>2-1</td>
</tr>
<tr>
<td>2.2</td>
<td>2-6</td>
</tr>
<tr>
<td>2.3</td>
<td>2-6</td>
</tr>
<tr>
<td>2.4</td>
<td>2-8</td>
</tr>
<tr>
<td>2.5</td>
<td>2-10</td>
</tr>
<tr>
<td>2.6</td>
<td>2-10</td>
</tr>
<tr>
<td>2.7</td>
<td>2-10</td>
</tr>
</tbody>
</table>

## SECTION 3
### PROGRAMMING GUIDE

<table>
<thead>
<tr>
<th>Paragraph</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.0</td>
<td>3-1</td>
</tr>
<tr>
<td>3.1</td>
<td>3-1</td>
</tr>
<tr>
<td>3.2</td>
<td>3-2</td>
</tr>
<tr>
<td>3.3</td>
<td>3-4</td>
</tr>
<tr>
<td>3.4</td>
<td>3-6</td>
</tr>
<tr>
<td>3.5</td>
<td>3-7</td>
</tr>
<tr>
<td>3.6</td>
<td>3-20</td>
</tr>
<tr>
<td>3.7</td>
<td>3-26</td>
</tr>
<tr>
<td>3.8</td>
<td>3-34</td>
</tr>
<tr>
<td>3.9</td>
<td>3-43</td>
</tr>
<tr>
<td>3.10</td>
<td>3-52</td>
</tr>
</tbody>
</table>

## SECTION 4
### PROGRAMMING OF PERIPHERALS

<table>
<thead>
<tr>
<th>Paragraph</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.0</td>
<td>4-1</td>
</tr>
<tr>
<td>4.1</td>
<td>4-2</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS (Cont)

SECTION 5
PERIPHERAL DESCRIPTION

<table>
<thead>
<tr>
<th>Paragraph</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0</td>
<td>Introduction</td>
<td>5-1</td>
</tr>
<tr>
<td>5.1</td>
<td>ASR-33 Teletype</td>
<td>5-1</td>
</tr>
<tr>
<td>5.2</td>
<td>High Speed Paper Tape Reader</td>
<td>5-7</td>
</tr>
<tr>
<td>5.3</td>
<td>High Speed Paper Tape Punch</td>
<td>5-10</td>
</tr>
<tr>
<td>5.4</td>
<td>Real Time Clock (RTC)</td>
<td>5-12</td>
</tr>
<tr>
<td>5.5</td>
<td>Serial Bus Peripheral Addresses</td>
<td>5-14</td>
</tr>
</tbody>
</table>

SECTION 6
OPERATOR'S CONTROL PANEL

<table>
<thead>
<tr>
<th>Paragraph</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.0</td>
<td>Introduction</td>
<td>6-1</td>
</tr>
<tr>
<td>6.1</td>
<td>OCP Displays</td>
<td>6-1</td>
</tr>
<tr>
<td>6.2</td>
<td>OCP Controls</td>
<td>6-3</td>
</tr>
<tr>
<td>6.3</td>
<td>OCP Programming</td>
<td>6-5</td>
</tr>
<tr>
<td>6.4</td>
<td>OCP Interconnections</td>
<td>6-6</td>
</tr>
</tbody>
</table>

SECTION 7
INTERFACING

<table>
<thead>
<tr>
<th>Paragraph</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.0</td>
<td>Introduction</td>
<td>7-1</td>
</tr>
<tr>
<td>7.1</td>
<td>Parallel Asynchronous I/O Bus</td>
<td>7-1</td>
</tr>
<tr>
<td>7.2</td>
<td>Serial I/O Bus</td>
<td>7-17</td>
</tr>
</tbody>
</table>

SECTION 8
SOFTWARE SYSTEMS

<table>
<thead>
<tr>
<th>Paragraph</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.0</td>
<td>Introduction</td>
<td>8-1</td>
</tr>
<tr>
<td>8.1</td>
<td>Basic Software System</td>
<td>8-1</td>
</tr>
<tr>
<td>8.2</td>
<td>Basic Operating System</td>
<td>8-1</td>
</tr>
</tbody>
</table>
### LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>2-2</td>
</tr>
<tr>
<td>2-2</td>
<td>2-11</td>
</tr>
<tr>
<td>2-3</td>
<td>2-13</td>
</tr>
<tr>
<td>2-4</td>
<td>2-14</td>
</tr>
<tr>
<td>5-1</td>
<td>5-3</td>
</tr>
<tr>
<td>6-1</td>
<td>6-2</td>
</tr>
<tr>
<td>7-1</td>
<td>7-3</td>
</tr>
<tr>
<td>7-2</td>
<td>7-6</td>
</tr>
<tr>
<td>7-3</td>
<td>7-6</td>
</tr>
<tr>
<td>7-4</td>
<td>7-8</td>
</tr>
<tr>
<td>7-5</td>
<td>7-9</td>
</tr>
<tr>
<td>7-6</td>
<td>7-9</td>
</tr>
<tr>
<td>7-7</td>
<td>7-11</td>
</tr>
<tr>
<td>7-8</td>
<td>7-12</td>
</tr>
<tr>
<td>7-9</td>
<td>7-13</td>
</tr>
<tr>
<td>7-10</td>
<td>7-14</td>
</tr>
<tr>
<td>7-11</td>
<td>7-14</td>
</tr>
<tr>
<td>7-12</td>
<td>7-15</td>
</tr>
<tr>
<td>7-13</td>
<td>7-16</td>
</tr>
<tr>
<td>7-14</td>
<td>7-17</td>
</tr>
<tr>
<td>7-15</td>
<td>7-18</td>
</tr>
<tr>
<td>7-16</td>
<td>7-19</td>
</tr>
<tr>
<td>7-17</td>
<td>7-20</td>
</tr>
<tr>
<td>7-18</td>
<td>7-21</td>
</tr>
<tr>
<td>7-19</td>
<td>7-22</td>
</tr>
</tbody>
</table>

### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>2-7</td>
</tr>
<tr>
<td>3-1</td>
<td>3-56</td>
</tr>
<tr>
<td>3-2</td>
<td>3-61</td>
</tr>
<tr>
<td>3-3</td>
<td>3-62</td>
</tr>
<tr>
<td>3-4</td>
<td>3-63</td>
</tr>
<tr>
<td>5-2</td>
<td>5-2</td>
</tr>
<tr>
<td>7-1</td>
<td>7-4</td>
</tr>
<tr>
<td>7-2</td>
<td>7-5</td>
</tr>
<tr>
<td>7-3</td>
<td>7-10</td>
</tr>
<tr>
<td>7-4</td>
<td>7-17</td>
</tr>
<tr>
<td>7-5</td>
<td>7-19</td>
</tr>
<tr>
<td>7-6</td>
<td>7-20</td>
</tr>
</tbody>
</table>
MIP-16 LAB AND SEVERE ENVIRONMENT COMPUTERS
SECTION 1
INTRODUCTION

1.0 INTRODUCTION

Sanders Associates Modular Information Processor family of 16 bit computers has been designed to meet the needs of real time control, data acquisition, and analysis. Its small size, ruggedness, light weight and low power permit it to be used where these factors are at a premium, yet it provides extremely high performance at low cost. This performance permits programmable software techniques to be applied to applications which have, in the past, required special purpose designs. Areas of potential application include:

- Optical Countermeasures (OCM)
- Electronic Countermeasures (ECM)
- Electronic Support Measures (ESM)
- Radar Signal Processing
- Engine Control and Health Monitoring
- Airframe Data Management
- Communication and Control
- Navigation Aids
- Fire Control
- Test Equipment
- Antisubmarine Warfare

The MIP-16 family features a modular design which permits a computer to be tailored to fit a particular application. Processor performance, memories, peripherals, and packaging may be selected to provide a cost effective solution to a particular problem.

Both lab and severe environment configurations are available. These are extensible so that a selection of processors, memories and peripherals may be used. The high density severe environment package may be as small as 70 cubic inches and weigh as little as 3 pounds, yet it will operate at temperatures up to 120°C and at vibration levels of 30g's. The lab environment package is a portable, self-contained unit which may be rack or table top mounted.

Two models of the processor are available. The model 116 provides a cost effective processor with a maximum speed of 5 million instructions per second. The model 216
provides a maximum speed of 14 million instructions per second. Both models are software compatible and use the same I/O devices. Both achieve this performance over the full military temperature range and both use off-the-shelf MSI TTL circuit technology to provide high reliability and low cost.

This level of performance is provided by an advanced system architecture utilizing pipelining and instruction fetch overlap. The processors are microprogrammed with an alterable microstore which permits instructions to be added for particular applications.

The MIP-16 family provides facilities for efficient, easy programming. The system architecture has provisions for the efficient implementation of high level languages in an operating system environment. Storage utilization is reduced through the use of a multiregister architecture and reentrant programs.

An instruction set of over 100 instructions is provided with capabilities for operation on byte, single precision fixed and floating point and double precision fixed and floating point data. Interrupt service overhead is minimized by the use of a hardware stack and a priority, vectored, interrupt system.

Programming aids are provided to assist in program development. These include an assembler, mathematical subroutines, diagnostics, and a debugging package.

The MIP-16 family is supported by a range of standard peripherals which include paper tape devices and disk storage. Provisions are made for the addition of a wide range of peripherals of diverse speed and transmission requirements.

The sections of this manual which follow provide comprehensive description of the facilities provided by the MIP-16 family. Additional processor options, peripherals and memory modules will be added in the future.
SECTION 2
SYSTEM ARCHITECTURE

2.0 INTRODUCTION

The MIP-16 family is designed to provide a broad spectrum of capabilities on a modular basis. The general modular architecture is illustrated in figure 2-1. The key system elements that make up a system for a particular application are:

1. Processor
2. Input/Output Facilities
3. Interrupt Facilities
4. Memories
5. Peripherals
6. Control Panel
7. Enclosures

The MIP-16 family provides the user a choice for these key system elements which he can use to configure a system suited for his particular application. In the sequel a description is given of each of these system elements.

2.1 PROCESSOR

The MIP-16 processor is a 16 bit, stored program, general purpose computer utilizing two's complement binary arithmetic. Instructions are variable length and can directly address 32,768 sixteen bit words or 65,536 eight bit bytes. A unified structure is used to address memory and device registers. The processor provides a large, powerful instruction set, 16 general purpose registers, multilevel priority interrupt system and input/output facilities.

The processor is microprogrammed with an extensible microstore. This permits microroutines to be added which are tailored for particular applications.

The MIP-16 processors provide facilities for efficient, easy programming of real time problems. Some of the key features of the processor are discussed below. A detailed description of the instruction set, however, is provided in Section 3 of this manual.
Figure 2-1 General Modular Processor.
2.1.1 PROCESSOR MODELS

Two models of the processor are available. They utilize the same peripherals, memories, and instruction set but differ in internal organization and speed. The model 116 has a 5 MHz instruction rate and the model 216 has a 14 MHz rate and executes some instructions in fewer microcycles.

2.1.2 REGISTERS

The processor provides 16, 16 bit general purpose registers, a program status register and a program address register. The large number of general registers permit many operations to be performed without reference to memory and improves performance while reducing storage requirements. Register 15 is dedicated for use as a system stack pointer.

2.1.2.1 Program Status Register (PSR)

This sixteen bit register is used to indicate the result of previous operations, processor priority, and holds mask bits for internal interrupts. It is structured in the following format:

```
<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>7</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>X</td>
<td>F</td>
<td>M</td>
<td></td>
<td>PR</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>V</td>
</tr>
</tbody>
</table>
```

- Bit 15-P: Privileged Mode
- Bit 14-X: Enables Fixed Point Interrupts
- Bit 13-F: Enables Floating Point Interrupts
- Bit 12-M: Memory Protect Enable
- Bits 11-7: Reserved
- Bit 6-4 PR: Processor Priority
- Bit 3-L: Link-set if result produced a carry
- Bit 2-M: Most-set to most significant bit of result
- Bit 1-N: Not zero-set if result was not zero
- Bit 0-V: Overflow-set if result produced overflow

When the privileged mode bit is off an attempt to address peripheral devices or execute system control instructions will cause an internal interrupt. If the memory protect feature is attached and enabled (M=1), an attempt to access a protected location will cause an internal interrupt.

2.1.2.2 Program Address Register (PC)

This 16 bit register is used to address up to 65,536 bytes for instruction fetch and data access.
2.1.3 INSTRUCTION SET

A total of 98 instructions are provided, broken into the following groups.

- 26 Fixed Point Arithmetic
- 11 Logical
- 15 Shift
- 17 Load-Store
- 20 Program Control
- 9 System Control

This large instruction set reduces the number of instructions required to perform a task and the amount of programming effort required.

Some particularly useful instructions are provided which are not usually found in machines of this class. These include:

- a. Double Precision (32 bits)
- b. One word immediate
- c. Relative branch
- d. Multiple Shift
- e. Multiply and Divide
- f. Load and Store Multiple

The one word immediate instructions permit an eight bit literal to be added to, compared with, or loaded into a register. This saves memory because an eight bit literal is sufficient for most operations of this type. Multiply and Divide can be performed in 1.3 and 2.6 $\mu$s respectively on the model 216 and in 3.6 and 7.2 $\mu$s on the model 116. Multiple shift instructions minimize the number of instructions required to perform scaling and packing and unpacking functions. Load and Store Multiple instructions are useful for saving and restoring the sixteen general purpose registers during subroutine calls.

2.1.4 ADDRESSING

The smallest unit of addressable data is the eight bit byte. Sixteen bit words and thirty-two bit double words may also be addressed.

Seven different memory addressing methods are available. These include:

- a. Indirect Register
- b. Indirect Incremented Register
- c. Indirect Decremented Register
- d. Absolute
- e. Immediate
- f. Relative
- g. Indexed

The utilization of these registers has been carefully selected for maximum frequency use with minimum storage requirements. A sixteen bit address is used which eliminates the problems associated with paging and permits up to 65,536 bytes to be addressed. Programs can be written which can be relocated without modification of internal addresses.
same addressing methods are used for both memory and peripheral device registers.

2.1.5 PROCESSOR OPTIONS

Options available for the processor include memory protection, fast multiply and additional general purpose register banks.

2.1.5.1 Memory Protection

The address space of 32,768 words is divided into 128 blocks of 256 words. Associated with each block are three bits: read protect, write protect, and write detect. A write detect bit is set if a word is written into its block. This is useful during page swapping to detect when a program has changed one of its blocks. The protect bits may be set only in privileged mode. A logical block of 2048 words is defined as a group of eight 256 word blocks.

Memory protect is not disabled by privileged mode. This affords some degree of supervisor self-protection.

Memory protect may be used to indicate blocks of memory that are not available to the system for read-write or write. Memory protect does not inhibit direct memory access. Protection for this is supplied by the supervisor when initiating the access.

2.1.5.2 Fast Multiply

This option performs a 16 x 16 bit multiply in 400 ns. Both sixteen bit rounded and thirty-two bit products are available. No special programming is required to use this option.

2.1.5.3 Register Banks

Up to three additional sets of 16, 16 bit general purpose registers may be installed. Bank selection is performed by the processor priority bits in the PSR. The correspondence is as follows:

<table>
<thead>
<tr>
<th>Bank</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>

The standard model 216 has Banks 0 and 3 installed while the 116 has only Bank 0. Bank 1 provides an extra set of program registers.

The major use of these banks is to reduce the overhead required to save and restore the registers in handling interrupts.
2.2 INTERRUPT FACILITIES

The processor provides a six level priority, vectored interrupt system. A push down stack is used to save machine state information consisting of the PC and PSR. Use of a stack facilitates programming of reentrant interrupt routines.

Multiple priority sources are identified on each level. In the event of the simultaneous request of interrupts the highest priority source on the highest priority level is serviced first. An interrupt request will only be honored if it is at a higher level than the processor priority as determined by the PSR. Table 2-1 lists the sources and their priority levels and transfer vector (T.V.) addresses.

Initiation of interrupt service begins by saving the PSR and PC on the stack. The source is used to address a transfer vector and a new PC and PSR are loaded in that order. The interrupt service routine is then entered. The end of the routine is indicated by a Return From Interrupt (RTI) Instruction which pops the stack and reloads the old PC and PSR.

Typically, the program operates at priority one in the PSR. Upon interrupt from level 6, for example, the new transfer vector will specify a new priority of level 6 which will block further interrupts at that level or below while still permitting interrupts at higher levels. When the PSR is restored by the RTI instruction the priority will be reset to one.

If the transfer vector PSR specifies a priority of zero, the old priority will not be changed when the PSR is loaded at the start of the interrupt sequence. This is used by internal interrupts to handle program interrupts such as overflow at the same level as the program which caused it. Of course, an internal interrupt such as power down can specify a priority of seven which will disable all external interrupts.

Since the priority is set by the transfer vector, levels may be grouped together. In the example above, the T.V. could specify level seven. This would have the effect of grouping levels 7 and 6 together.

The internal interrupts are so arranged so that only one can be active simultaneously. Since they cannot be masked they will always cause an interrupt which will reset the request. The internal interrupt routine may be reentrant however.

In the event of simultaneous internal and external interrupts, the internal interrupt takes precedence. If its T.V. specifies a new priority below that of the active external level the PC and PSR will be saved again and entry made to the external handler routine. At completion of that, the internal routine will be reentered.

This arrangement permits external interrupt routines to cause internal interrupts since the internal request generated at a lower level is saved before entry to the external routine.

2.3 INPUT-OUTPUT FACILITIES

Two types of data transmissions facilities are available. A high speed parallel asynchronous bus is used to link the processor with memories and high speed devices over short distances. The serial bus is utilized for differential transmission for
### TABLE 2-1

**INTERRUPT SOURCES AND PRIORITY LEVELS**

<table>
<thead>
<tr>
<th>Source</th>
<th>Source Description</th>
<th>Priority Level</th>
<th>Transfer Vector Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Power Down</td>
<td>8</td>
<td>0000</td>
</tr>
<tr>
<td>14</td>
<td>Power Up</td>
<td>8</td>
<td>0004</td>
</tr>
<tr>
<td>13</td>
<td>Invalid Instruction</td>
<td>8</td>
<td>0008</td>
</tr>
<tr>
<td>12</td>
<td>Privileged Instruction</td>
<td>8</td>
<td>000C</td>
</tr>
<tr>
<td>11</td>
<td>Protection</td>
<td>8</td>
<td>0010</td>
</tr>
<tr>
<td>10</td>
<td>I/O Error</td>
<td>8</td>
<td>0014</td>
</tr>
<tr>
<td>9</td>
<td>Fixed Point Overflow</td>
<td>8</td>
<td>0018</td>
</tr>
<tr>
<td>8</td>
<td>Floating Point Overflow</td>
<td>8</td>
<td>001C</td>
</tr>
<tr>
<td>7</td>
<td>Floating Point Underflow</td>
<td>8</td>
<td>0020</td>
</tr>
<tr>
<td>6</td>
<td>Floating Point Significance</td>
<td>8</td>
<td>0024</td>
</tr>
<tr>
<td>5</td>
<td>Fixed Point Divide</td>
<td>8</td>
<td>0028</td>
</tr>
<tr>
<td>4</td>
<td>Floating Point Divide</td>
<td>8</td>
<td>002C</td>
</tr>
<tr>
<td>3</td>
<td>Programmed Interrupt</td>
<td>8</td>
<td>0030</td>
</tr>
<tr>
<td>7</td>
<td>External 7</td>
<td>7</td>
<td>0040</td>
</tr>
<tr>
<td>6</td>
<td>External 6</td>
<td>7</td>
<td>0044</td>
</tr>
<tr>
<td>5</td>
<td>External 5</td>
<td>7</td>
<td>0048</td>
</tr>
<tr>
<td>4</td>
<td>External 4</td>
<td>7</td>
<td>004C</td>
</tr>
<tr>
<td>3</td>
<td>External 3</td>
<td>7</td>
<td>0050</td>
</tr>
<tr>
<td>2</td>
<td>External 2</td>
<td>7</td>
<td>0054</td>
</tr>
<tr>
<td>1</td>
<td>External 1</td>
<td>7</td>
<td>0058</td>
</tr>
<tr>
<td>0</td>
<td>External 0</td>
<td>7</td>
<td>005C</td>
</tr>
<tr>
<td>7-0</td>
<td>External 7-0</td>
<td>6</td>
<td>0060-007C</td>
</tr>
<tr>
<td>7-0</td>
<td>External 7-0</td>
<td>5</td>
<td>0080-009C</td>
</tr>
<tr>
<td>7-0</td>
<td>External 7-0</td>
<td>4</td>
<td>00A0-00BC</td>
</tr>
<tr>
<td>15-0</td>
<td>External 15-0</td>
<td>3</td>
<td>00C0-00FC</td>
</tr>
<tr>
<td>Program</td>
<td>Program</td>
<td>2</td>
<td>Used for alternate bank select</td>
</tr>
<tr>
<td></td>
<td>Program</td>
<td>1</td>
<td>Main Program Level</td>
</tr>
<tr>
<td>Retain Priority</td>
<td></td>
<td>0</td>
<td>Used to indicate no change of priority</td>
</tr>
</tbody>
</table>

2-7
communication of over long distance in a high noise environment to peripheral devices. Thirty-two interrupt sources are associated with the parallel bus and sixteen with the serial bus.

A unified programming and interrupt structure is used to control these facilities. Devices are addressed simply as memory locations. Transfer time dependence is transparent to the programmer. Identification of an interrupt source is carried out by the hardware and requires no programming assistance.

Detailed descriptions of these facilities are provided in Section 7 of this manual.

2.3.1 PARALLEL ASYNCHRONOUS BUS

This bus provides for transmission of data over a maximum distance of 5 feet. Direct memory access is available at an 8 MHz rate and instruction fetch at 14 MHz rate using queued memories. Asynchronous transmission is used so that memories and devices of differing transmission rates may be attached to the same bus. This permits a 1μs core memory and a 70 ns bipolar memory to share the same bus facilities. Asynchronous transmission also accommodates the different delays associated with devices located along the fast bus. Total loop delay can be as much as 20 ns which is significant at transmission rates above 5 MHz.

2.3.2 SERIAL BUS

This bus is primarily used to communicate with slower character oriented devices over a maximum distance of fifty feet. It provides for word rates of 250 KHz and byte rates of 350 KHz. The control panel uses this bus for transmission in order to provide for remote location.

For aircraft type system applications this bus may be used for control and program fill of the computer from ground equipment without removal of the computer.

2.4 MEMORIES

Typical application requirements for memories include:

- Non-volatility
- Electrical Programmability
- Speed
- Size
- Cost

Semiconductor and core memories are provided to meet these application requirements. All memories are compatible between different processor models.

2.4.1 SEMICONDUCTOR MEMORIES

Semiconductor read/write memories are provided for use in applications requiring high speed, small to medium size and where volatility is not a problem. The memories are available in both lab and severe environment configurations.
2.4.1.1 1024 WORD, 200 ns, BIPOLAR READ/WRITE MEMORY (MODEL 700)

This memory utilizes 80 ns bipolar integrated circuit memories to provide a cost effective 1024 word memory with the following access times:

<table>
<thead>
<tr>
<th>Processor</th>
<th>Commercial</th>
<th>Military</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temperature Range</td>
<td>Model 116</td>
<td>Model 216</td>
</tr>
<tr>
<td>Commercial</td>
<td>200 ns</td>
<td>160 ns</td>
</tr>
<tr>
<td>Military</td>
<td>250 ns</td>
<td>200 ns</td>
</tr>
</tbody>
</table>

Typical power dissipation is 35 watts and the memory is mounted on a single card.

2.4.1.2 1024 WORD, Instruction Look Ahead Bipolar Read/Write Memory (Model 710)

This memory utilizes 80 ns bipolar integrated circuit memories to provide a high performance 1024 word memory. Instruction fetch times are reduced through the use of two instruction look ahead queuing. Typical power dissipation is 40 watts and the memory is mounted on a single card. Access times are given below.

<table>
<thead>
<tr>
<th>Temperature Range</th>
<th>Data Fetch Processor</th>
<th>Instruction Fetch Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Model 116</td>
<td>Model 216</td>
</tr>
<tr>
<td>Commercial</td>
<td>200 ns</td>
<td>160 ns</td>
</tr>
<tr>
<td>Military</td>
<td>250 ns</td>
<td>200 ns</td>
</tr>
</tbody>
</table>

2.4.2 CORE MEMORIES

Core memories are provided for applications requiring medium to large size, medium speed and where non-volatility is required. A military and commercial version are available.

2.4.2.1 8192 Word, 1 µs, Lab Environment, Core Memory System (Model 750)

This memory is designed specifically for lab application. It consists of a memory controller and a memory unit. One memory controller can control up to 4 memory units for a total capability of 32,768 words. The unit is packaged on 2 circuit cards and the controller on 1 card.

A reduced 4096 word memory unit is also available.

2.4.2.2 4096 Word, 1 µs, Severe Environment, Core Memory System (Model 760)

This system is designed specifically for military applications where small size and low power are prime requirements. The memory uses advanced design techniques in order to provide state-of-the-art performance. Low power and extremely wide operating
temperatures are key performance features. Power strobing is used to reduce power consumption. At 1 μs the typical power consumption is 25 watts but this may be reduced by operating at a lower speed. At 3 μs, for example, power consumption drops to only 15 watts. The complete system requires only 3 cards. An 8192 word version is also available. It operates at similar power levels and requires only 4 cards.

2.5 PERIPHERALS

A complete line of standard peripherals is available. They are generally configured for lab environment but may be used with a severe environment system by proper I/O bus connections. The serial I/O bus is particularly applicable for this. In this manner a processor installed in a remote location can use peripherals.

Standard peripherals available include the following. Additional peripherals will be added in the future:

- ASR-33 Teletype
- 300 cps Paper Tape Reader
- 75 cps Paper Tape Punch
- Disk File
- Real Time Clock

General programming techniques for peripherals are discussed in Section 4. Detailed programming and device descriptions of the peripherals are presented in Section 5.

2.6 CONTROL PANEL

The control panel is designed to control the operation of the computer when located up to fifty feet away in a high noise environment. This is useful in an environment where operator access is limited or where equipment removal is undesirable. The control panel can also read or write into any memory location or device register and can be used as an I/O device itself.

The reader is referred to Section 6 for a fuller discussion of the control panel.

2.7 ENCLOSES

The MIP-16 family is designed to be used in both lab and severe environments. An optimum solution requires two different packaging concepts, however.

2.7.1 LABORATORY CONFIGURATION

MIP-16 systems intended for lab environments use a standard enclosure shown in Figure 2-2. This can be used to house both expansion modules as well as the central processor itself. This enclosure uses a standard card size of 8 x 11 inches with 160 pin connectors.
Figure 2-2 Processor Lab Unit.
The unit offers the following features:

- Rack or table top mounting
- Self contained power supplies and cooling
- Lightweight
- Portable
- Wide range power inputs
- Expandable

The enclosure is available in two basic configurations, although others can be provided.

The main frame configuration includes provisions for the following modules:

- A model 216 or 116 Processor
- Control Panel
- Teletype Interface
- Paper Tape Reader and Punch Interface
- Real Time Clock
- Disk File Controller
- Four Expansion Cards
- Two 1024 Word Bipolar Memory Modules
- Core Memory Controller
- 8192 Word Core Memory

The core memory expansion configuration includes provisions for the following modules:

- One Core Memory Controller
- One to Four 8192 or 4096 Word Core Memory Units

2.7.2 SEVERE ENVIRONMENT CONFIGURATION

MIP-16 systems intended to be used in a severe environment use a modular enclosure. This enclosure is composed of variable number of double card plug clamshell modules which can be attached together to form a complete assembly. The basic module size is approximately .8 x 6.38 x 7.4 inches as is shown in Figure 2-3. Card size including a 160 pin connector is about 6 x 7 inches. Components are cooled by conduction to the sidewalls where a coldplate or finned heat exchanger may be attached.

The size, weight, temperature range, and operating vibration range depend on the number and type of modules used in the complete assembly. A typical assembly, shown in Figure 2-4, provides a guide, however.

This unit consists of modules and contains a model 116 processor, 4K core memory, and one expansion card. The system physical specifications are as follows:

- Size: 2.6 x 6.38 x 7.4 inches (123 cubic inches)
- Weight: 5.5 pounds
- Power: 40 watts
- Vibration: 30g's at 200 cycles
- Operating Temperature: -55°C to +110°C, at sidewall
- Cooling: Conduction

With semiconductor memories smaller systems can be constructed.

2-12
Figure 2-3 Clamshell Module.

2-13
Figure 2-4  Three Module Assembly
SECTION 3
PROGRAMMING GUIDE

3.0 INTRODUCTION

All instructions affecting the processor operations are either one word or two words in length. All instructions must start on an even byte address. There are six types of instructions: fixed-point arithmetic, logical, shifts, load and stores, program control, and system control.

3.1 DATA FORMATS

Data is stored in three types of formats: bytes, words, and double precision words. A word consists of two bytes.

3.1.1 BYTE

A byte contains 8 bits of information. The bits are numbered from right to left with bit 0 considered the least significant bit and bit 7 the most significant bit.

3.1.2 WORD

A word consists of two bytes and contains 16 bits which are numbered from 0 to 15. Bit 0 is the least significant (L. S.) bit.
A negative quantity is always stored in two's complement form. When a word is addressed it is always addressed by the most significant (M. S.) byte. The M. S. byte is always an even numbered byte.

3.1.3 DOUBLE PRECISION WORDS

Consist of four bytes of data - 32 bits. The bits are numbered from right to left (0-31) with bit 0 considered the least significant bit.

A negative quantity is represented as a 32 bit two's complement number. A double precision word is addressed by the most significant byte of the M. S. word.

3.1.4 GENERAL PURPOSE REGISTERS

There are 16 general purpose registers (0-15). Each register consists of two bytes (16 bits) and the bits are numbered from 0 to 15. Bit 0 is the least significant bit.

Negative quantities are stored in two's complement form. When a register byte is referenced it is always the least significant (L. S.) byte - bits 0-7. In byte operations where a sign is considered bit 7 is treated as the sign bit.

All double precision operations involve general purpose registers as operands. Each operand is considered as a 32 bit two's complement quantity.

The programmer always references the most significant register, which can either be odd or even.

3.2 INSTRUCTION FORMATS

3.2.1 ONE WORD INSTRUCTIONS

One word instructions are in one of four formats.
3.2.1.1 Register to Register Operations - RR

The assembler will recognize instructions of the form

\[ \text{OP RA, RB} \]

as register to register operations. RA and RB are defined as one of the 16 general registers. The instruction word will contain 2 four bit fields for the operation codes (opcode) and 1 four bit field for each general register:

15 12 8 4 0

\[ \text{OP RA OP RB} \]

The RA and RB registers usually are the operands with RA being the destination register. With one word Input-Output instructions the definitions of RA and RB changed slightly. Register RA with input instructions is still the destination register but RB now contains the address of the source operand. With output instructions RB contains the address of the destination and RA becomes the source operand.

3.2.1.2 Single Register Operations - RN

The assembler will recognize instructions in the form

\[ \text{OP RA, N} \]

as single register operations. RA is defined as the operand and is one of the 16 general registers. N specifies the number of times the operation is to be performed. The instruction word is similar to register-register operations:

15 12 8 4 0

\[ \text{OP RA OP N} \]

3.2.1.3 Immediate Operations - RI

Instructions of the form

\[ \text{OP RA, LIT} \]

are recognized by the assembler as immediate operand instructions. The instruction word is:

15 12 8 0

\[ \text{OP RA LIT} \]
RA is again one of the 16 general registers. LIT is an eight bit literal. Register RA is always the destination if the operation requires one.

3.2.1.4 Relative Operations - RL

The assembler will recognize instructions of the form

\[
\text{OP EXP}
\]

as a relative instruction and generates a memory word of the following format:

\[
\begin{array}{cccccc}
15 & 12 & 8 & 4 & 0 \\
\hline
\text{OP} & \text{OP} & \text{DISP}
\end{array}
\]

where DISP is the displacement of the expression (EXP) from the program counter (PC). The PC at execution time always points to one instruction ahead. DISP has a range of -128 to +128.

3.2.3 TWO WORD INSTRUCTIONS

The assembler recognizes instructions of the form

\[
\text{OP RA, EXP(RB)} \text{ or } \text{OP EXP(RB)}
\]

as two word instructions and will generate two memory words appearing as:

\[
\begin{array}{cccccc}
15 & 12 & 8 & 4 & 0 \\
\hline
\text{OP} & \text{RA} & \text{OP} & \text{RB} & \text{DISP}
\end{array}
\]

where RA is the general register affected by the operation and RB and DISP are used to calculate an effective byte address. The value of DISP depends on the value of RB and is either equal to the value of EXP or a displacement of the EXP from the Program Counter. The calculation of the effective byte address is discussed below. For immediate addressing modes in arithmetic logical, input-output, and jump and calls the (RB) field of the source code may be left off - the assembler will assume RB = 0.

3.3 BYTE ADDRESS CALCULATIONS

3.3.1 FOR ARITHMETIC, LOGICAL, AND INPUT-OUTPUT INSTRUCTIONS

If the RB field contains a

\[
0 \quad \text{(immediate addressing)}
\]

- The second word of the instruction contains the effective data.
1 (relative addressing) - The effective byte address is calculated by adding the contents of the PC (which at execution time is pointing to the second word) to the contents of the second word - DISP. The addition is performed in two's complement arithmetic. EA = (PC) + DISP

2 (absolute addressing) - The contents of the second word is the effective byte address.
EA = DISP

3-15 (indexed addressing) - The sum of the second word of the instruction and the contents of the specified RB register (3-15) constitutes the effective byte address.
EA = DISP + (Register B)

The effective byte address for logical and arithmetic operation must always be on a word boundary (it must be even). If it is odd the least significant bit of the EA is disregarded. The exception is memory byte operations in Load-Store instructions, where the byte address may be either odd or even.

3.3.2 FOR MEMORY JUMPS (JMPM) AND CALLS (CALM)

The effective byte address (in this case jump or call address) is calculated in the same manner as above except when the RB field is zero (0). If the RB field is zero a one level indirect jump or call address is calculated. The contents of the program counter and the contents of the second word are summed to form an intermediate address. The contents of this intermediate address is the jump or call address. The value of the RA field in JMPM is zero. EA (JA) = ((PC) + DISP)

3.3.3 LA INSTRUCTION

The above only has one address mode - immediate.

3.3.4 TWO WORD BYTE ADDRESS CALCULATION EXAMPLES

3.3.4.1 Example 1

<table>
<thead>
<tr>
<th>15</th>
<th>12</th>
<th>8</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>10AC</td>
<td>0 0 0 0</td>
<td>R A</td>
<td>0 0 0 0</td>
<td>R B</td>
</tr>
<tr>
<td>10AE</td>
<td>0 0 0 0 0 0 1 1 1 0 1 1 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The effective byte address is:
- Immediate: 10AE if RB = 0 (note: 1B0 is the effective data)
- Relative: 125E if RB = 1 (1B0+10AE)
- Absolute: 01B0 if RB = 2
- Indexed: 035C if RB = 4 - if the contents of R4 = 01AC (1AC+1B0=35C)
3.3.4.2 Example 2

<table>
<thead>
<tr>
<th></th>
<th>15</th>
<th>12</th>
<th>8</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>10AC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>10AE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The effective jump address is:
- Relative: $125E$ if $RB = 1$
- Absolute: $01B0$ if $RB = 2$
- Indexed: $035C$ if $RB = 4$
- Indirect: $1000$ if $RB = 0$ and byte $125E$ and $125F$ contain $1000$

$$1B0 + 10AE = 125E$$

3.4 NOTATIONS

Below are notations that will be used in the operation code descriptions that follow:

For Operations:
- $\lor$ - Inclusive OR
- $\oplus$ - Exclusive OR
- $\land$ - AND
- $\neg$ - NOT
- $( )$ - Contents of
- $\rightarrow$ - Is Stored In
- $EA$ - Effective Byte Address

For Condition Codes:
- $X$ - Set Conditionally
- $-\neg$ - Not Affected
- $0$ - Cleared

For Fields:
- EXP - Expression
- RA - Register A
- RB - Register B
- PC - Program Counter
- PSR - Program Status Register
- PIA - Present Instruction Address
- L.S. - Least Significant
- M.S. - Most Significant

For Examples:
- RA: - Means RA Contains
3.5  FIXED POINT ARITHMETIC INSTRUCTIONS

3.5.1  ADD REGISTER

\[
\begin{array}{c|c|c|c|c|c}
15 & 12 & 8 & 4 & 0 \\
\hline
0 & 0 & 0 & 0 & RA \\
0 & 0 & 0 & 1 & RB \\
\end{array}
\]

Operation: \((RA) + (RB) \rightarrow RA\)

The contents of RA are added to the contents of RB and the result is placed in RA.

Condition Codes:
- **N**: Set if result not zero; cleared if result zero
- **V**: Set if overflow; cleared otherwise
- **L**: Set equal to carry
- **M**: Set to sign of result

3.5.2  ADD MEMORY

\[
\begin{array}{c|c|c|c|c|c}
15 & 12 & 8 & 4 & 0 \\
\hline
0 & 0 & 0 & 0 & RA \\
0 & 0 & 0 & 0 & RB \\
\end{array}
\]

Operation: \((EA, EA + 1) + (RA) \rightarrow RA\)

The contents of two bytes starting at the effective byte address are added to the contents of RA and the result is placed in RA.

Condition Codes:
- **N**: Set if result not zero; cleared if result zero
- **V**: Set if overflow; cleared otherwise
- **L**: Set equal to carry
- **M**: Set to sign of result

Example:

Initial Conditions:
- **PIA**: 002E
- **R5**: 01F4
- **R6**: 000E
- **ADR** at byte address 4E and 4F and
- **LOC** at byte address 40

a) **ADDM R5, ADR (R2)**

Assembles as:
- 0502
- 004E

Result:
- R5: 2331
b) ADDM R5, ADR(R1)
Assembles as: 0501
001E
Result: R5: 2331

c) ADDM R5, LOC(R6)
Assembles as: 0506
0040
Result: R5: 2331

d) ADDM R5, ADR(R0)
Assembles as: 0501
004E
Result: R5: 0242

3.5.3 DOUBLE PRECISION ADD

DADD RA,RB

<table>
<thead>
<tr>
<th>15</th>
<th>12</th>
<th>8</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>R A + 1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>R B + 1</td>
</tr>
</tbody>
</table>

Operation: (RA, RA + 1) + (RB, RB + 1) → RA, RA + 1

The 32 bit two's complement contents of RA and RA + 1 is added to the 32 bit two's complement contents of RB and RB +1. The result is placed in RA and RA + 1. RB and RB +1 are unaffected.

Condition Codes:
N: Cleared only if result contents of RA and RA +1 are both zero
V: Set if arithmetic overflow
L: Set equal to carry
M: Set to most significant bit of result RA

Example:
R4: 054F  R5: DE3A
R9: 00F2  R6: 5547

DADD R4, R9
Result: R4: 0642  R5: 3381

3.5.4 ADD LITERAL EXTENDED

ADDL RA,LIT

<table>
<thead>
<tr>
<th>15</th>
<th>12</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>1</td>
</tr>
</tbody>
</table>

Operation: (RA) + LIT → RA
The literal - sign extended - is added to the contents of RA and the result is placed in RA. 
-128 ≤ LIT ≤ 127

Condition Codes:

N: Set if result is non-zero; cleared if zero
V: Set if overflow occurs; cleared otherwise
L: Set equal to carry
M: Set equal to most significant bit of result

Example:
R7: 0300
ADDL R7, -1
R7: 02FF

Programming Note: Useful in incrementing or decrementing a register.

3.5.5 SUBTRACT REGISTER

Operation: (RA) - (RB) → RA

The contents of RB are subtracted from the contents of RA. The result is placed in RA.

Condition Codes:

N: Set if result is not zero; cleared if result zero
V: Set if arithmetic overflow; cleared otherwise
L: Set equal to the carry
M: Set to sign of the result

3.5.6 SUBTRACT MEMORY

Operation: (RA) - (EA, EA +1) → RA

The contents of two bytes starting at the effective byte address are subtracted from the contents of RA, and the result is placed in RA.

Condition Codes:

N: Set if result is not zero; cleared if result zero
V: Set if arithmetic underflow; cleared otherwise
L: Set equal to the carry
M: Set to sign of the result
Examples:

Same conditions (but with SUBM opcode) as in the example for ADDM.
The results would be:

a) R5: E0B7
b) R5: E0B7
c) R5: E0B7
d) R5: 01A6

3.5.7 DOUBLE PRECISION SUBTRACT

```
DSUB RA, RB
```

Operation: (RA, RA +1) - (RB, RB +1) → RA, RA +1

The 32 bit two's complement contents of RB and RB +1 are subtracted from 32 bit two's complement contents of RA and RA +1. The result is placed in RA and RA +1. RB and RB +1 are unaffected.

Condition Codes:
- N: Cleared only if both the result RA and RA +1 are zero
- V: Set if arithmetic overflow
- L: Set equal to carry
- M: Set equal to bit 15 of result RA

Example:

R3: 054F  R4: DE3A
R8: 00F2  R9: 5547

DSUB R3, R8

Result: R3: 045D  R4: 88F7

3.5.8 MULTIPLY

```
MUL RA, RB
```

Operation: (RA) × (RB) → RA, RA +1

The contents of RB are multiplied by the contents of RA. The least significant 16 bits are placed in RA +1. The remaining most significant bits and sign are placed in RA, that is, RA and RA +1 contain a 32 bit two's complement product.
Examples:
R1: 0460  R5: 0500
MUL R1, R5
R1: 0017  R2: 6000

R3: FB40  R6: 0500
MUL R3, R6
R3: FFE8  R4: 4000

3.5.9  MULTIPLY AND ROUND

MULR  RA,RB

\[
\begin{array}{cccc}
15 & 12 & 8 & 4 & 0 \\
\hline
0 & 1 & 0 & 0 & RA \\
0 & 1 & 1 & 0 & RB
\end{array}
\]

Operation: \((RA) \times (RB) \rightarrow RA\)

The 16 bit two's complement of registers RA and RB are multiplied together forming a 32 bit product. The result is left shifted one and bit 15 of the shifted result is used to round bits 16-31. Bits 16-31 are then placed in register RA. Overflow will occur if both register RA and RB contained 8000.

Condition Codes:
- N: Set if result RA is not zero, cleared otherwise
- V: Set if overflow (8000 x 8000)
- L: Cleared
- M: Set to sign of result RA

Example:
R3: 0600  R4: 0C00

MULR  R3, R4

Result:
R3: 0090  R4: 0C00

3.5.10  INTEGER MULTIPLY

IMUL  RA,RB

\[
\begin{array}{cccc}
15 & 12 & 8 & 4 & 0 \\
\hline
0 & 1 & 0 & 0 & RA \\
0 & 1 & 0 & 0 & RB
\end{array}
\]

Operation: \((RA) \times (RB) \rightarrow RA\)

The 16 bit two's complement contents of RA are multiplied by the 16 bit two's complement contents of RB. The result is a 32 bit product, but only the least significant 16 bits (0-15) are placed in RA. Bits 16-31 of the result are lost. RB and RA + 1 are unaffected.
Overflow will occur if the result is greater than 7FFF or less than 8000.

Condition Codes:
- \( N \): Cleared only if \( RA \times RB = 0 \)
- \( V \): Set if result >7FFF or < 8000
- \( L \): Cleared
- \( M \): Set to bit 15 of result \( RA \)

Example:
- \( R3: \) 007A \( R4: \) 0011
  - `IMUL R3, R4`

Result:
- \( R3: \) 081A \( R4: \) 0011

3.5.11 DIVIDE

\[
\begin{array}{c|c|c|c|c|c}
& & & & & \\
15 & 12 & 8 & 4 & 0 \\
\hline
\text{DIV} & \text{RA, RB} & 0 & 1 & 0 & 0 \\
\end{array}
\]

Operation: \( (RA, RA+1)/(RB) \rightarrow RA, RA+1 \)

The contents of \( RA, RA+1 \) are divided by the contents of \( RB \). The dividend \((RA, RA+1)\) is a 32 bit two's complement number. The quotient is placed in \( RA+1 \) with appropriate sign. The absolute value of the remainder is placed in \( RA \).

Condition Codes:
- \( N \): Set if result not zero; cleared if result zero
- \( V \): Set by an attempt to divide a number by a smaller number or zero
- \( L \): Cleared
- \( M \): Set to most significant bit of quotient

Examples:
- \( R3: \) 0366 \( R4: \) 000D \( R6: \) 13F8
  - `DIV R5, R3`
- \( R5: \) 03D9 \( R6: \) 0082

3.5.12 MOVE REGISTER

\[
\begin{array}{c|c|c|c|c|c}
& & & & & \\
15 & 12 & 8 & 4 & 0 \\
\hline
\text{MOV} & \text{RA, RB} & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\end{array}
\]

Operation: \( (RB) \rightarrow RA \)

Contents of \( RB \) are moved to \( RA \). \( RB \) is unaffected.
Condition Codes:

N: Set if RA not zero; cleared if RA = 0
V: Unaffected
L: Unaffected
M: Set to most significant bit of RA

3.5.13 MOVE REGISTER BYTE EXTENDED

\[
\text{MOVX} \quad \text{RA}, \text{RB}
\]

Operation: L. S. 8 bits of RB → RA

The least significant byte of RB is moved to the least significant byte of RA. The bit 7 of the byte is propagated thru the most significant byte of RA. RB is unaffected.

ConditionCodes:

N: Set if byte is not zero; cleared if it is zero
V: Unaffected
L: Unaffected
M: Set equal to bit 7 of transferred byte

3.5.14 MOVE LITERAL EXTENDED

\[
\text{MOVL} \quad \text{RA}, \text{LIT}
\]

Operation: LIT → RA

The eight bit literal (LIT) is moved into the least significant byte of RA. Bit 7 of the literal is extended into the most significant byte of RA.

Condition Codes:

N: Cleared only if the literal is zero
V: Unaffected
L: Unaffected
M: Set equal to most significant bit of the byte
3.5.15 NEGATE

Operation: \(-(RB)\rightarrow RA\)

The two's complement of the contents of RB is placed in RA. The contents of RB are unaffected. Note: the two's complement of 8000 is 8000.

Condition Codes:
- N: Set if result is not zero; cleared if result is zero
- V: Set only if \((RB) = 8000\)
- L: Set equal to carry
- M: Set to bit 15 of result RA

Example:

R3: 5A5A

NEG R3, R3

Result:

R3: B5B6

3.5.16 DOUBLE PRECISION NEGATE

Operation: \(-(RB, RB + 1)\rightarrow RA, RA + 1\)

The 32 bit contents of RB and RB + 1 are two's complemented and the 32 bit result is placed in RA and RA + 1.

Condition Codes:
- N: Set if 32 bit result is not zero, cleared if result is zero
- V: Set if overflow (only if \((RB), (RB)+1 = 8000\ 0000\))
- L: Set equal to carry
- M: Set equal to bit 31 of result
ARITHMETIC COMPARE

Operation:  \((RA) - (RB)\)

The two's complement contents of RA and RB are arithmetically compared. The contents of RA and RB are unaffected.

Condition Codes:
- **N**: Cleared if \((RA) = (RB)\)
- **V**: Set if arithmetic overflow; cleared otherwise
- **L**: Set equal to carry
- **M**: Set equal to bit 15 of result

Example:

\[
\begin{align*}
R3: & \quad 0F5A \\
R4: & \quad 01B3 \\
R5: & \quad 0F5A \\
\end{align*}
\]

\[
\text{CMP \ } R3, R4
\]

A branch will occur if one of the following branch instructions follows the compare:
- BGT, BGE, BHI

\[
\text{CMP \ } R4, R3
\]

A branch will occur if one of the following branch instructions follows the compare:
- BLT, BLE, BLOS

\[
\text{CMP \ } R3, R5
\]

A branch will occur when the compare precedes the following branch instructions:
- BGE, BLE, BLOS

ARITHMETIC COMPARE MEMORY

Operation:  \((RA) - (EA, \cdot EA + 1)\)

The two's complement quantities of RA and the effective word are arithmetically compared. Both operands are unaffected by the compare.
Condition Codes:

N: Cleared if (RA) = (EA, EA + 1)
V: Set if overflow
L: Set equal to carry
M: Set to bit 15 of compare result

3.5.19 DOUBLE PRECISION COMPARE

```
DCMP RA,RB
  15 12  8  4  0
  0 1 1 | R A  + 1 0 0 1 | R B  + 1
```

Operation: (RA, RA+1) - (RB, RB+1)

The 32 bit two's complement contents of RA and RA+1 are compared to the 32 bit two's complement contents of RB and RB+1. RA, RA+1, RB and RB+1 are not affected.

Condition Codes:

Same as DSUB

3.5.20 COMPARE BYTE LITERAL

```
CMPB RA,LIT
  15 12  8  0
  1 0 1 0 | R A  | L I  T|
```

Operation: L. S. byte RA-LIT

Arithmetic compare between the least significant byte of RA and the literal. The most significant byte of RA is not involved in the compare.

Condition Codes:

N: Set if byte not equal to literal; cleared otherwise
V: Unaffected
L: Set equal to carry
M: Set to bit 7 of compare result

Programming Note: Useful in character compares.
3.5.21 SATURATED ADD

\[
\begin{array}{c|c|c|c|c|c}
\text{SADD} & \text{RA, RB} \\
\hline
15 & 12 & 8 & 4 & 0 \\
\hline
0 & 1 & 0 & 0 & \text{RA} \\
0 & 1 & 1 & 0 & \text{RB} \\
\end{array}
\]

Operation: \((RA) + (RB) \rightarrow RA\)
- If overflow set; \(7FFF \rightarrow RA\) or \(8000 \rightarrow RA\)
- If overflow not set; \(RA = RA\)

The contents of Register A and Register B are added and the sum is placed in Register A. If overflow occurred on addition of positive numbers RA is set to 7FFF. If it occurred on addition of negative numbers RA is set to 8000. If no overflow occurred the sum is unaffected.

Condition Codes:
- \(N\): Set if result not zero; cleared otherwise
- \(V\): Cleared
- \(L\): Set equal to zero
- \(M\): Set to bit 15 of result RA

Examples:
- \(R3: 74A5\) \(R6: 0A27\) \(R11: 3000\)
- \(\text{SADD R6, R3}\) \(\text{SADD R3, R11}\)

Results:
- \(R6: 7ECC\) \(R3: 7FFF\)

3.5.22 SATURATED SUBTRACT

\[
\begin{array}{c|c|c|c|c|c}
\text{SSUB} & \text{RA, RB} \\
\hline
15 & 12 & 8 & 4 & 0 \\
\hline
0 & 1 & 0 & 0 & \text{RA} \\
0 & 1 & 1 & 0 & \text{RB} \\
\end{array}
\]

Operation: \((RA) - (RB) \rightarrow RA\)
- If overflow set; \(8000 \rightarrow RA\) or \(7FFF \rightarrow RA\)
- If overflow not set; \(RA = RA\)

The contents of Register B are subtracted from the contents of Register A and the difference is placed in Register A. If overflow occurs when subtracting a positive quantity from a negative quantity RA is set to 8000. If it occurs when subtracting a negative quantity from a positive one RA is set to 7FFF. If no overflow occurs the difference is unaffected.
Condition Codes:
- **N**: Set if result not zero; cleared otherwise
- **V**: Cleared
- **L**: Set equal to zero
- **M**: Set to bit 15 of result RA

Examples:
- R3: 8B5B R6: 0A27 R11: 3000
  - SSUB R3, R6  SSUB R3, R11

Results:
- R3: 8134 R3: 8000

3.5.23 MOVE POSITIVE

```
MOVP RA,RB
```

<table>
<thead>
<tr>
<th>15</th>
<th>12</th>
<th>8</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Operation: If
- \((RB) < 0; -(RB) \rightarrow RA\)
- \((RB) \geq 0; (RB) \rightarrow RA\)

If the contents of Register B are positive they are moved unaffected to Register A. If they are negative the contents of Register B are two's complemented and placed in Register A. Register B is unaffected.

Condition Codes:
- **N**: Set if result RA not zero; cleared otherwise
- **V**: Set only if \((RB)\) is 8000
- **L**: Set to carry
- **M**: Set to bit 15 of result

Example:
- R1: FFA5 R2: 00A5
  - MOV R1, R2 MOV R2, R2

Results:
- R2: 0058 R2: 00A5
### 3.5.24 MOVE NEGATIVE

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>RA</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Operation:**
- If (RB) > 0; (RB) → RA
- If (RB) ≤ 0; -(RB) → RA

If the contents of Register B are negative they are moved unaffected to Register A. If they are positive the contents of Register B are twos complemented and placed in Register A. Register B is unaffected.

**Condition Codes:**
- **N:** Set if result RA not zero; cleared otherwise
- **V:** Unaffected
- **L:** Set equal to carry
- **M:** Set to bit 15 of result RA

**Example:**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>RA</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R1:</td>
<td>FFA5</td>
<td></td>
<td>R2:</td>
<td>00A5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVN</td>
<td>R3, R1</td>
<td>MOVN</td>
<td>R4, R2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Results:**
- R3: FFA5
- R4: FF5B

### 3.5.25 DOUBLE PRECISION MEMORY ADD

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>RA+1</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Operation:**

( RA, RA + 1 ) + ( EA through EA + 3 ) → RA, RA + 1

The double precision word (EA, EA + 1, EA + 2, EA + 3) is added to the contents of Registers RA and RA + 1. The result is placed in Registers RA and RA + 1.

**Condition Codes:**
- **N:** Cleared only if result registers RA and RA + 1 are both zero
- **V:** Set if arithmetic overflow
- **L:** Set equal to carry
- **M:** Set to bit 15 of result RA (MS register)
3.5.26 **DOUBLE PRECISION MEMORY SUBTRACT**

\[
\begin{array}{c|c}
\text{DSUBM} & \text{RA, EXP(RB)} \\
\hline
15 & 12 & 8 & 4 & 0 \\
\hline
0 & 1 & 1 & 1 & \text{RA} + 1 \\
1 & 0 & 1 & 0 & \text{RB} \\
\text{D I S} & \text{P+2} \\
\end{array}
\]

**Operation:** \((\text{RA}, \text{RA} + 1) - (\text{EA} \text{ through EA} + 1) \rightarrow \text{RA}, \text{RA} + 1\)

The double precision word \((\text{EA}, \text{EA} + 1, \text{EA} + 2, \text{EA} + 3)\) is subtracted from the contents of Registers \text{RA} and \text{RA} + 1. The result is placed in Registers \text{RA} and \text{RA} + 1.

**Condition Codes:**
- \(N:\) Cleared only if result Registers \text{RA} and \text{RA} + 1 are both zero
- \(V:\) Set if arithmetic overflow
- \(L:\) Set equal to carry
- \(M:\) Set to bit 15 of result RA (MS register)

### 3.6 LOGICAL INSTRUCTIONS

#### 3.6.1 LOGICAL AND

\[
\begin{array}{c|c}
\text{AND} & \text{RA, RB} \\
\hline
15 & 12 & 8 & 4 & 0 \\
\hline
0 & 0 & 0 & 1 & \text{RA} \\
1 & 1 & 1 & 1 & \text{RB} \\
\end{array}
\]

**Operation:** \((\text{RB}) \land (\text{RA}) \rightarrow \text{RA}\)

The 16 bit contents of RA and RB are logically ANDed together. The result is placed in RA. RB is unaffected. AND is useful in clearing a bit or bits.

**Condition Codes:**
- \(N:\) Set if RA not zero, cleared if zero
- \(V:\) Cleared
- \(L:\) Unaffected
- \(M:\) Set equal to most significant bit of result

**Example:**
- \(\text{R3: 47B5 R7: 3A57}\)
  \(\text{AND R3, R7}\)
  \(\text{R3: 0215 R7: 3A57}\)
3.6.2 MEMORY AND

**ANDM RA, EXP(RB)**

<table>
<thead>
<tr>
<th></th>
<th>15</th>
<th>12</th>
<th>8</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>RA</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>RB</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>I</td>
<td>S</td>
<td>P</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Operation:** \((EA, EA + 1) \land (RA) \rightarrow RA\)

Two bytes of memory starting at the effective byte address are ANDed with the contents of RA. The result is placed in RA.

**Condition Codes:**
- **N:** Set if RA not zero, cleared if zero
- **V:** Cleared
- **L:** Unaffected
- **M:** Set equal to most significant bit of result

3.6.3 AND BYTE LITERAL

**ANDB RA, LIT**

<table>
<thead>
<tr>
<th></th>
<th>15</th>
<th>12</th>
<th>8</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>RA</td>
</tr>
<tr>
<td>L</td>
<td>I</td>
<td>T</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Operation:** L. S. Byte \(RA \land LIT \rightarrow RA\)

The least significant byte of RA is ANDed with the literal and the result is placed in RA. The most significant byte of RA is unaffected.

**Condition Codes:**
- **N:** Set if byte result not zero; cleared otherwise
- **V:** Unaffected
- **L:** Unaffected
- **M:** Set to bit 7 of result

**Programming Note:** Useful to clear a bit or bits in the L. S. byte of a register.

3.6.4 INCLUSIVE OR

**OR RA, RB**

<table>
<thead>
<tr>
<th></th>
<th>15</th>
<th>12</th>
<th>8</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>RA</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>RB</td>
<td></td>
</tr>
</tbody>
</table>

**Operation:** \((RB) \lor (RA) \rightarrow RA\)
Contents of RB are inclusive OR'd with the contents of RA. The result is placed in RA. The contents of RB are unaffected. OR is useful in setting a bit or bits.

Condition Codes:
- N: Set if RA is not zero; cleared if RA is zero
- V: Unaffected
- L: Unaffected
- M: Set equal to most significant bit of RA

Example:
- R3: 47B5  R7: 3A57
- OR R3, R7
- R3: 7FF7  R7: 3A57

3.6.5 MEMORY INCLUSIVE OR

<table>
<thead>
<tr>
<th>ORM</th>
<th>RA, EXP(RB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>RA</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>RB</td>
</tr>
</tbody>
</table>

Operation: (EA, EA+1) v (RA) → RA

The contents of the effective address is inclusive OR'd with the contents of RA. The result is placed in RA. The contents of the two bytes of the effective address are unaffected.

Condition Codes:
- N: Set if result RA is not zero; cleared otherwise
- V: Unaffected
- L: Unaffected
- M: Set to bit 15 of result RA

3.6.6 EXCLUSIVE OR

<table>
<thead>
<tr>
<th>XOR</th>
<th>RA, RB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>RA</td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>RB</td>
</tr>
</tbody>
</table>

Operation: (RB) ⊕ (RA) → RA

The contents of RB are exclusive OR'd with the contents of RA. The result is placed in RA. The contents of RB are unaffected.
Condition Codes:

N: Set if RA is not zero; cleared if RA is zero
V: Unaffected
L: Unaffected
M: Set equal to most significant bit of RA

Example:
R3: 47B5  R5: 59CB
XOR R3, R5
R3: 1E7E

Programming Note: The sequence RA exclusive OR'd RB, RB exclusive OR'd RA, RA exclusive OR'd RB results in the exchange of the contents of RA and RB without the use of temporary storage words.

Exclusive OR may be used to invert bits. Any field exclusive OR'd with itself becomes zero.

3.6.7 MEMORY EXCLUSIVE OR

XORM RA, EXP(RB)

Operation: (EA, EA 1) ⊕ (RA) → RA

The contents of the effective address is exclusive ORd with the contents of RA. The result is placed in RA. The contents of the EA are unaffected.

Condition Codes:
N: Set if result RA not zero; cleared otherwise
V: Unaffected
L: Unaffected
M: Set to bit 15 of result RA

3.6.8 ONE'S COMPLEMENT

COM RA, RB

Operation: -(RB)-1 → RA

One's complement RB - each bit of RB equal to 1 is cleared and each bit equal to zero is set to 1. The result is placed in RA. RB is unaffected.
Condition Codes:
N: Cleared only if original RB was FFFF
V: Unaffected
L: Cleared
M: Set to most significant bit of result RA

Example:
R3: 47B5
COM R3, R3
R3: B84A

3.6.9 ONE'S COMPLEMENT BYTE

<table>
<thead>
<tr>
<th>COMB</th>
<th>RA, RB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 1</td>
<td>R_A</td>
</tr>
</tbody>
</table>

Operation: -(RB)-1 byte → RA byte

The least significant byte of RB is one's complemented and placed in the least significant byte of RA. RB and the most significant byte of RA are unaffected.

Condition Codes:
N: Cleared only if source RB was XXFF
V: Unaffected
L: Cleared
M: Set to bit 7 of result RA

3.6.10 LOGICAL COMPARE

<table>
<thead>
<tr>
<th>LC</th>
<th>RA, RB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>R_A</td>
</tr>
</tbody>
</table>

Operation: (RA) ^ (RB)

The contents of RA and RB are logically compared - they are ANDed together.

Condition Codes:
N: Set if any bits match; cleared otherwise
V: Unaffected
L: Unaffected
M: Set to most significant bit of the AND result

Programming Note: The logical compare is useful to compare bits of RA and RB. RA and RB are unaffected.
3.6.11 LOGICAL COMPARE MEMORY

\[
\begin{array}{c|c|c|c|c|c}
15 & 12 & 8 & 4 & 0 \\
\hline
0 & 0 & 0 & 0 & RA & 1 & 0 & 0 & 0 & RB \\
0 & 0 & 0 & 0 & D & I & S & P & \\
\end{array}
\]

Operation: \((RA) \land (EA, EA + 1)\)

Same as LC except RA is logically compared to the contents of the EA and EA + 1 instead of RB.

Condition Codes:
- **N:** Set if any bits match; cleared otherwise
- **V:** Unaffected
- **L:** Unaffected
- **M:** Set to bit 15 of the compare result
3.7 SHIFT INSTRUCTIONS

3.7.1 SHIFT LEFT LOGICAL AND INSERT LINK

**SLL RA,N**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>12</td>
<td>8</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>RA</td>
</tr>
</tbody>
</table>

Operation:

Shift all 16 bits of Register A left N (≤ 16) places. On each shift, the link is shifted into bit 0 and bit 15 is shifted into the link. A shift of zero (as in all shift operations) is illegal.

Condition Codes:

- **N**: Set if result not zero; cleared otherwise
- **V**: Unaffected
- **L**: Set to bit (16-N) of original data
- **M**: Set to bit 15 of result RA

3.7.2 SHIFT LEFT LOGICAL-LEAST SIGNIFICANT BYTE-AND INSERT LINK

**SLLB RA,N**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>12</td>
<td>8</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>RA</td>
</tr>
</tbody>
</table>

Operation:

Shift the least significant 8 bits of Register A left N (≤ 8) places. On each shift the link is shifted into bit 0 and bit 7 of RA is shifted into the link.

Condition Codes:

- **N**: Set if result LSB is not zero; cleared otherwise
- **V**: Unaffected
- **L**: Set to bit (8-N) of original data
- **M**: Set to bit 7 of result RA
3.7.3  SHIFT RIGHT LOGICAL AND INSERT LINK

SRL  RA,N

Operation:

Shift all 16 bits of Register A right $N$ (≤ 16) places. On each shift the link is shifted into bit 15 and bit 0 is shifted into the link.

Condition Codes:
- **N**: Set if results not zero; cleared otherwise
- **V**: Unaffected
- **L**: Set equal to bit (N-1) of original data
- **M**: Set to bit 15 of result RA

Programming Note: Useful in conjunction with SLL to reverse bits in a register.

3.7.4  SHIFT RIGHT LOGICAL-LEAST SIGNIFICANT BYTE-AND INSERT LINK

SRLB  RA,N

Operation:

Shift the least significant 8 bits of Register A right $N$ (≤ 8) places. On each shift the link is shifted into bit 7 and bit 0 is shifted into the link.

Condition Codes:
- **N**: Set if result LSB is not equal to zero; cleared otherwise
- **V**: Unaffected
- **L**: Set equal to bit (N-1) of original data
- **M**: Set to bit 7 of result RA

3-27
3.7.5  SHIFT LEFT LOGICAL

Operation:

Shift all 16 bits of Register A left N (≤ 16) bits. On each shift a zero is shifted into bit 0 and bit 15 is shifted into link.

Condition Codes:
N: Set if result is not zero; cleared otherwise
V: Unaffected
L: Set to bit (16-N) of original data
M: Set to bit 15 of result RA

3.7.6  SHIFT LEFT LOGICAL-LEAST SIGNIFICANT BYTE

Operation:

Shift the least significant 8 bits of Register A left N (≤ 8) times. On each shift a zero is shifted into bit 0 of RA and bit 7 is shifted into the link.

Condition Codes:
N: Set if result LSB is not zero; cleared otherwise
V: Unaffected
L: Set to bit (8-N) of original data
M: Set to bit 7 of result RA
3.7.7  SHIFT LEFT DOUBLE LOGICAL

DSL  RA,N

\[
\begin{array}{c|c|c|c|c|c}
15 & 12 & 8 & 4 & 0 \\
0 & 1 & 1 & 0 & R_A+1 \\
1 & 1 & 1 & 0 & N-1 \\
\end{array}
\]

Operation:

Shift the 32 bits of Register A and Register A+1 left N (16) places. On each shift a zero is shifted into bit 0 of RA+1 and bit 15 of RA is shifted into the link. If RA = 15 is specified in the symbolic language then the two registers shifted are R15 (RA) and RO (RA+1).

Note: Field 0-3 is always N(Mod 16)

Condition Codes:
- N: Set if result (RA, RA+1) is not zero; cleared otherwise
- V: Unaffected
- L: Set equal to bit (16-N) of original RA data
- M: Set equal to bit 15 of result RA

3.7.8  SHIFT RIGHT LOGICAL

SRZ  RA,N

\[
\begin{array}{c|c|c|c|c|c}
15 & 12 & 8 & 4 & 0 \\
0 & 1 & 1 & 0 & R_A \\
1 & 0 & 0 & 1 & N-1 \\
\end{array}
\]

Operation:

All 16 bits of Register A are right shifted N (≤16) places. On each shift bit 0 is shifted into the link and a zero is shifted into bit 15.

Condition Codes:
- N: Set if result is not zero; cleared otherwise
- V: Unaffected
- L: Set equal to bit N-1 of result
- M: Set equal to bit 15 of result RA
3.7.9  SHIFT RIGHT CIRCULAR

SRC  RA,N

<table>
<thead>
<tr>
<th>15</th>
<th>12</th>
<th>8</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>N-1</td>
</tr>
</tbody>
</table>

Operation:

Shift the 16 bits of Register A right circular N places. On each shift bit 15 is shifted into bit 14 and bit 0 is shifted into bit 15.

Condition Codes:
N: Cleared only if original RA was zero; set otherwise
V: Unaffected
L: Unaffected
M: Set to bit 15 of result RA

3.7.10  SHIFT RIGHT DOUBLE LOGICAL

DSR  RA,N

<table>
<thead>
<tr>
<th>15</th>
<th>12</th>
<th>8</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>N-1</td>
</tr>
</tbody>
</table>

Operation:

Shift the 32 bits of RA and RA +1 right N (≤ 16) places. On each shift bit 0 of RA +1 is shifted into the link and a zero is shifted into bit 15 of RA. If RA =15 is specified the two registers operated on are R15 (RA) and R0 (RA +1).

Condition Codes:
N: Set if result (RA, RA +1) is zero; cleared otherwise
V: Unaffected
L: Set to bit (N-1) of original RA +1 data
M: Set to bit 15 of result RA
3.7.11  SHIFT LEFT ARITHMETIC

**SLA** RA, N

<table>
<thead>
<tr>
<th>15</th>
<th>12</th>
<th>8</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RA</td>
<td></td>
<td>N-1</td>
</tr>
</tbody>
</table>

Operation:

Shift all 16 bits of register A left \( N \) (\( \leq 16 \)) places. On each shift bit 15 is shifted into the link and a zero is shifted into bit 0. Same as SLL except overflow is detected. (L \( \neq \) M after shift).

Condition Codes:
- **N**: Set if result RA not zero; cleared otherwise
- **V**: Set if arithmetic overflow (L \( \neq \) M)
- **L**: Set to bit (16-\( N \)) of original RA data
- **M**: Set to bit 15 of result

3.7.12  SHIFT LEFT ARITHMETIC-LEAST SIGNIFICANT BYTE

**SLAB** RA, N

<table>
<thead>
<tr>
<th>15</th>
<th>12</th>
<th>8</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RA</td>
<td></td>
<td>N-1</td>
</tr>
</tbody>
</table>

Operation:

Shift the least significant 8 bits of RA left \( N \) (\( \leq 8 \)) places. On each shift bit 7 of RA is shifted into the link and a zero is shifted into bit 0. Same as SLB except that overflow is detected.

Condition Codes:
- **N**: Set if LSB of result RA is not zero; cleared otherwise
- **V**: Set if overflow (L \( \neq \) M)
- **L**: Set to bit (8-\( N \)) of original RA data
- **M**: Set to bit 7 of result RA
3.7.13  SHIFT LEFT DOUBLE ARITHMETIC

DSL A RA, N

Operation:

Shift the 32 bits of RA and RA + 1 left N (≤ 16) places. On each shift bit 15 of RA is shifted into the link and a zero is shifted into bit 0 of RA + 1. Same as DSL except that arithmetic overflow is detected.

Condition Codes:
- N: Set if (RA) and (RA + 1) are both not zero; cleared otherwise
- V: Set if overflow (L ≠ M)
- L: Set to bit (16-N) of original RA data
- M: Set to bit 15 of result RA

3.7.14  SHIFT RIGHT ARITHMETIC

SRA RA, N

Operation:

Register A is right shifted N (≤ 16) places. On each shift bit zero of RA is shifted into the link and bit 15 is shifted into bit 14. Bit 15 same as before shift.

Condition Codes:
- N: Set if result RA is not zero; cleared otherwise
- V: Unaffected
- L: Set to bit (N-1) of original RA data
- M: Set to bit 15 of RA
3.7.15  SHIFT RIGHT DOUBLE ARITHMETIC

DSRA RA,N

<table>
<thead>
<tr>
<th>15</th>
<th>12</th>
<th>8</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>R_A</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>N-1</td>
</tr>
</tbody>
</table>

Operation:

Shift RA and RA+1 right N (≤ 16) places. On each shift bit 0 of RA+1 is shifted into the link and bit 15 of RA is shifted into bit 14 of RA. Bit 15 of RA does not change.

Condition Codes:
- N: Set if both (RA) and (RA+1) not zero; cleared otherwise
- V: Unaffected
- L: Set to bit (N-1) of original RA+1 data
- M: Set to bit 15 of RA

SHIFT INSTRUCTION EXAMPLES:

Initial Condition: R5: 5A5A   L = 1 (for each instruction)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLL R5, 2</td>
<td>R5: 696A L = 1</td>
</tr>
<tr>
<td>SLLB R5, 2</td>
<td>R5: 5A6A L = 1</td>
</tr>
<tr>
<td>SRL R5, 2</td>
<td>R5: 5696 L = 1</td>
</tr>
<tr>
<td>SRLB R5, 2</td>
<td>R5: 5A56 L = 1</td>
</tr>
<tr>
<td>SLZ R5, 2</td>
<td>R5: 6968 L = 1</td>
</tr>
<tr>
<td>SLZB R5, 2</td>
<td>R5: 5A68 L = 1</td>
</tr>
<tr>
<td>SRZ R5, 2</td>
<td>R5: 1696 L = 1</td>
</tr>
<tr>
<td>SRC R5, 2</td>
<td>R5: 9696 L = 1</td>
</tr>
</tbody>
</table>

Initial Condition: R5: E4E3   R6: C2B1   L = 0

<table>
<thead>
<tr>
<th>Operation</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLA R5, 2</td>
<td>R5: 938C L = 1</td>
</tr>
<tr>
<td>SLAB R5, 2</td>
<td>R5: E48C L = 1</td>
</tr>
<tr>
<td>SRA R5, 2</td>
<td>R5: F938 L = 1</td>
</tr>
<tr>
<td>DSL R5, 2</td>
<td>R5: 938F R6: 0AC4 L = 1</td>
</tr>
<tr>
<td>DSR R5, 2</td>
<td>R5: 3938 R6: F0AC L = 0</td>
</tr>
<tr>
<td>DSLA R5, 2</td>
<td>R5: 9384 R6: 0AC4 L = 1</td>
</tr>
<tr>
<td>DSRA R5, 2</td>
<td>R5: F938 R6: F0AC L = 0</td>
</tr>
</tbody>
</table>
3.8 LOAD-STORE INSTRUCTIONS

All Load-Store instructions may reference memory or any of the devices on either the serial or parallel bus.

3.8.1 LOAD REGISTER

<table>
<thead>
<tr>
<th>L</th>
<th>(RA, RB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Operation: \(((RB), (RB) + 1) \rightarrow RA\)

Load Register A with two bytes starting with the byte addressed by Register B.

Condition Codes:
- N: Set only if input data is not zero
- V: Unaffected
- L: Unaffected
- M: Set to most significant bit of input data

3.8.2 LOAD REGISTER AND INCREMENT

<table>
<thead>
<tr>
<th>LI</th>
<th>(RA, RB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Operation: \(((RB), (RB) + 1) \rightarrow RA\) \n\((RB) + 2 \rightarrow RB\)

RA is loaded with two bytes starting with the byte addressed by RB. RB is then incremented by two.

Condition Codes:
- N: Set only if input word is not zero
- V: Unaffected
- L: Unaffected
- M: Set to bit 15 of input word

Programming Note: Useful to sequentially access data in a buffer or stack.

3.8.3 LOAD REGISTER AND DECREMENT

<table>
<thead>
<tr>
<th>LD</th>
<th>(RA, RB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Operation:  
\[ ((RB), (RB + 1)) \rightarrow RA \]
\[ (RB) - 2 \rightarrow RB \]

RA is loaded with two bytes starting with the byte addressed by RB. RB is then decremented by two.

Condition Codes:
- **N**: Set only if input word not zero
- **V**: Unaffected
- **L**: Unaffected
- **M**: Set to bit 15 of input word
3.8.4 LOAD REGISTER

```
<table>
<thead>
<tr>
<th>15</th>
<th>12</th>
<th>8</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>RA</td>
<td></td>
</tr>
</tbody>
</table>
```

```
L M RA, EXP(RB)
```

```
Operation: (EA, EA+1) → RA
```

RA is loaded with 2 bytes starting at the effective byte address.

Condition Codes:
- N: Set only if input word not zero
- V: Unaffected
- L: Unaffected
- M: Set to bit 15 of input word

3.8.5 LOAD REGISTER BYTE

```
<table>
<thead>
<tr>
<th>15</th>
<th>12</th>
<th>8</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>RA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```

```
L M B RA, EXP(RB)
```

```
Operation: (EA) → RA
```

The least significant byte of RA is loaded with the contents of the effective byte address. The most significant byte of RA is unaffected.

Condition Codes:
- N: Set only if input byte not zero
- V: Unaffected
- L: Unaffected
- M: Set to bit 7 of input byte

3.8.6 LOAD ADDRESS

```
<table>
<thead>
<tr>
<th>15</th>
<th>12</th>
<th>8</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>RA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```

```
L A RA, EXP
```

```
Operation: (PC) + DISP → RA
```

The contents of the Program counter is added to the displacement and placed in Register A. This instruction is useful in programs that are meant to be relocatable.
Condition Codes:
N: Cleared only if DISP= 0
V: Unaffected
L: Unaffected
M: Unaffected

3.8.7 STORE REGISTER

<table>
<thead>
<tr>
<th>S</th>
<th>RA, RB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0101</td>
<td>R A 0110</td>
</tr>
</tbody>
</table>

Operation: (RA)\(\rightarrow\)(RB), (RB) + 1

The contents of RA are stored into memory starting at the byte address addressed by RB.

Condition Codes:
N: Set only if (RA) not zero
V: Unaffected
L: Unaffected
M: Set to bit 15 of RA

3.8.8 STORE REGISTER AND INCREMENT

<table>
<thead>
<tr>
<th>SI</th>
<th>RA, RB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0101</td>
<td>R A 1000</td>
</tr>
</tbody>
</table>

Operation: (RA)\(\rightarrow\)(RB), (RB) + 1
(RB)+2 \(\rightarrow\)RB

The 2 bytes in Register A are stored starting at the byte addressed by RB. RB is then incremented by two.

Condition Codes:
N: Set only if (RA) not zero
V: Unaffected
L: Unaffected
M: Set to bit 15 of RA

3.8.9 STORE AND DECREMENT

<table>
<thead>
<tr>
<th>SD</th>
<th>RA, RB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0101</td>
<td>R A 1010</td>
</tr>
</tbody>
</table>

3-37
Operation: \[(RA) \rightarrow (RB), (RB) + 1\]
\[(RB) - 2 \rightarrow RB\]

The two bytes of Register A are stored starting at the byte addressed by Register B. The contents of RB are then decremented by 2.

Condition Codes:
- N: Set only if (RA) not zero
- V: Unaffected
- L: Unaffected
- M: Set to bit 15 of RA

### 3.8.10 STORE REGISTER

\[
\begin{array}{c|cc|c|c|cc}
15 & 12 & 8 & 4 & 0 \\
\hline
0 & 0 & 0 & 1 & RA & 0 & 1 & 1 & 0 & RB \\
\hline
\end{array}
\]

Operation: \[(RA) \rightarrow EA, EA + 1\]

The contents of RA are stored into the effective byte address and effective byte address plus one. This instruction has no immediate addressing mode. RB = 0 is treated as indexing.

Condition Codes:
- N: Set only if RA not zero
- V: Unaffected
- L: Unaffected
- M: Set to bit 15 of RA

### 3.8.11 STORE REGISTER BYTE

\[
\begin{array}{c|cc|c|c|cc}
15 & 12 & 8 & 4 & 0 \\
\hline
0 & 0 & 0 & 1 & RA & 1 & 0 & 0 & 0 & RB \\
\hline
\end{array}
\]

Operation: \[\text{L. S. Byte of RA} \rightarrow \text{Effective byte address}\]

Condition Codes:
- N: Set only if L. S. byte of RA not zero
- V: Unaffected
- L: Unaffected
- M: Set to bit 7 of RA

3-38
3.8.12 DOUBLE PRECISION LOAD

DLM RA, EXP(RB)

Operation: \((EA \text{ through } EA + 3) \rightarrow RA, RA + 1\)

The double precision contents of effective byte address are placed in Registers RA and RA + 1.

Condition Codes:
- **N**: Cleared only if \((EA \text{ through } EA + 3)\) are zero
- **V**: Unaffected
- **L**: Unaffected
- **M**: Set equal to bit 7 of \((EA)\).

3.8.13 DOUBLE PRECISION STORE

DSM RA, EXP(RB)

Operation: \((RA, RA + 1) \rightarrow EA \text{ through } EA + 3\)

The contents of Registers RA and RA + 1 are stored in byte addresses EA through EA + 3.

Condition Codes:
- **N**: Cleared only if Registers RA and RA + 1 are both zero
- **V**: Unaffected
- **L**: Unaffected
- **M**: Set equal to bit 15 of Register RA

3.8.14 POP (LOAD MULTIPLE)

POP RA, N

Operation: Words starting at \((R15) - 2\) to \((R15) - 2N\) are loaded into register A to register A - N + 1.

The contents of the stack pointer \((R15)\) is decremented by two. Register A is then loaded with 2 bytes starting with the byte addressed by R15. The count \((N)\) is then decremented by one. If the count is not zero, R15 is again decremented by two and the next lower
numbered register is loaded. This process is performed N times. R15 will point to last word loaded.

Condition Codes:
None affected

Example:
If R15 contains: 03FA then
POP R9, 4 will load

R9 with contents of bytes 3F8 and 3F9
R8 with contents of bytes 3F6 and 3F7
R7 with contents of bytes 3F4 and 3F5
R6 with contents of bytes 3F2 and 3F3 and
R15 will contain 03F2

Programming Note: Useful in restoring registers before exiting an interrupt handling routine.

3.8.15 PUSH (STORE MULTIPLE)

| Operation: (RA) to (RA) + N-1 are stored into bytes starting at (R15) to (R15)+2N-2 |
|---|---|---|---|---|
| 15 | 12 | 8 | 4 | 0 |
| PUSH RA,N |

Register A is stored into two bytes starting with the one addressed by the stack pointer (R15). The contents of R15 is then incremented by two. The count (N) is then decremented by one. If the count is not zero the next higher numbered register is then stored into two bytes starting with the byte addressed by R15. The contents of R15 are again incremented by two. This process is performed N times. R15 will point to last word stored plus two.

Condition Codes:
None affected

Example:
If R15 contains: 03F2 then
PUSH R6, 4 will store

Contents of R6 into bytes 3F2 and 3F3
Contents of R7 into bytes 3F4 and 3F5
Contents of R8 into bytes 3F6 and 3F7
Contents of R9 into bytes 3F8 and 3F9 and
R15 will contain 03FA

Programming Note: Useful in saving registers in an interrupt handling routine.
3.8.16 LOAD MULTIPLE

<table>
<thead>
<tr>
<th>15</th>
<th>12</th>
<th>8</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>RA</td>
<td>A</td>
</tr>
</tbody>
</table>

Operation: Words starting at (R0) - 2 to (R0) - 2N are loaded into register A to register A - N + 1.

The contents of the stack pointer (R0) is decremented by two. Register A is then loaded with 2 bytes starting with the byte addressed by R0. The count (N) is then decremented by one. If the count is not zero, R0 is again decremented by two and the next lower numbered register is loaded. This process is performed N times. R0 will point to last word loaded.

Condition Codes:
None affected

Example:
If R0 contains: 03FA then
LDM R9, 4 will load

R9 with contents of bytes 3F8 and 3F9
R8 with contents of bytes 3F6 and 3F7
R7 with contents of bytes 3F4 and 3F5
R6 with contents of bytes 3F2 and 3F3 and
R0 will contain 03F2
3.8.17 STORE MULTIPLE

Operation: (RA) to (RA) + N-1 are stored into bytes starting at (RO) to (RO) + 2N-2

Register A is stored into two bytes starting with the one addressed by the stack pointer (RO). The contents of RO is then incremented by two. The count (N) is then decremented by one. If the count is not zero the next higher numbered register is then stored into two bytes starting with the byte addressed by RO. The contents of RO are again incremented by two. This process is performed N times. RO will point to last word stored plus two.

Condition Codes:
None affected

Example:

If RO contains: 03F2 then
STM R6, 4 will store

Contents of R6 into bytes 3F2 and 3F3
Contents of R7 into bytes 3F4 and 3F5
Contents of R8 into bytes 3F6 and 3F7
Contents of R9 into bytes 3F8 and 3F9 and
RO will contain 03FA
3.9 PROGRAM CONTROL INSTRUCTIONS

3.9.1 SUBROUTINE CALLS AND UNCONDITIONAL JUMPS

3.9.1.1 SUBROUTINE CALL

CAL  RA, RB

Operation:  (PC) → RA, (RB) → PC

The contents of the program counter (return address) are saved in RA and then the program counter assumes the value (in bytes) contained in RB.

Condition Codes:
None affected

3.9.1.2 JUMP

JMP  RB

Operation:  (RB) → PC

The contents of the PC are replaced by the contents of RB. The original contents of PC are lost.

Condition Codes:
None affected

3.9.1.3 SUBROUTINE CALL

CALM  RA, EXP(RB)

Operation:  (PC) → RA, JA → PC

The contents of the PC (return address) are saved in RA. The calculated jump address (depends on RB field) now replaces the contents of PC.

Condition Codes:
None affected
3.9.1.4 JUMP

Operation: JA → PC

The contents of the PC are replaced by the calculated jump address.

Condition Codes:
None affected

Examples:

1) JMPM ADR(R2): ADR at byte address 114
   Assembles as: 4092
                   0114
   Jump Address 114, i.e., 114→PC

2) JMPM ADR(R4)
   Assembles as: 4094
                   0114
   If R4: 0100 Jump Address is 214, i.e., 214→PC

3) JMPM ADR(R1) instruction at byte address 100 and ADR at byte address 114
   Assembles as: 4091
                   0012
   Jump Address is 114, 114→PC

4) JMPM ADR(R0) same condition as (3) and byte address ADR contains 02 and ADR + 1 contains 20
   Assembles as: 4090
                   0012
   Jump Address is (114, 115) = 0220, 220→PC

3.9.2 RELATIVE BRANCHES

In the description of relative branches the DISP (displacement) is equal to EXP-current instruction address-2. EXP can be an address or any valid expression.
3.9.2.1 BRANCH UNCONDITIONALLY

\[
\begin{array}{c|c|c|c|c}
15 & 12 & 8 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & D & I & S & P \\
\end{array}
\]

Operation: \( \text{PC + DISP} \rightarrow \text{PC} \)

The program counter is replaced by the sum of the present PC (which is at present byte location plus two) and the value of the displacement sign extended. This permits relative branching in the range of +64 words or -63 words (+128 and -126 bytes).

Condition Codes:
None affected

The following branch instructions all examine the Program Status Register (PSR). Whether or not a certain bit \((N, V, L, M)\) or combination of bits is set determines if the branch will occur. BL, BNL, BM, BNM, BNZ, BZ, BV and BNV all test just one bit in the PSR. These bits may be set or cleared by any instruction affecting the condition codes. The symbol \(Z\) will be used for \(\sim N\), i.e., \(Z = 1\) if result is zero, \(Z = 0\) if result is not zero.

BGT, BLE, BGE and BLT are used to test the results of instructions in which the operands were signed (two's complement) values.

BHI and BLOS allow the programmer to test the result of an operation on unsigned operations—where the sixteen bits of each operand are considered as an unsigned quantity.

At execution time the PC is equal to the current instruction address plus two.

3.9.2.2 BRANCH IF LINK SET

\[
\begin{array}{c|c|c|c|c}
15 & 12 & 8 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & D & I & S & P \\
\end{array}
\]

Operation: (1) if Link = 1; \( \text{PC + DISP} \rightarrow \text{PC} \)
(2) if Link = 0; step

If the link is set, the program counter is replaced by the sum of the present PC and the DISP sign extended. If the link is clear the next sequential instruction is executed.

Condition Codes:
None affected

3-45
3.9.2.3 BRANCH IF LINK CLEAR

BNL EXP

| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | D | I | S | P |

Operation:
1. if Link = 1; step
2. if Link = 0; PC + DISP → PC

If the Link is set, the next sequential instruction is executed. If the Link is clear, the Program Counter is replaced by the sum of the present PC and the DISP sign extended.

Condition Codes:
None affected

3.9.2.4 BRANCH IF MOST SET
BRANCH IF NEGATIVE

BM EXP

| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | D | I | S | P |

BN EXP

| 1 | 0 | 0 | 0 | 0 | 1 | 0 | D | I | S | P |

Operation:
1. if M = 1; PC + DISP → PC
2. if M = 0; step

If the most significant result bit in the status word is set, the Program Counter is replaced by the sum of the present PC and the DISP sign extended. If it is clear, the next sequential instruction is executed.

Condition Codes:
None affected

3.9.2.5 BRANCH IF MOST CLEAR
BRANCH IF POSITIVE

BNM EXP

| 1 | 0 | 0 | 0 | 0 | 1 | 1 | D | I | S | P |

BP EXP

| 1 | 0 | 0 | 0 | 0 | 1 | 1 | D | I | S | P |

Operation:
1. if M = 1; step
2. if M = 0; PC + DISP → PC

If the most significant result bit in the status word is clear, the Program Counter is replaced by the sum of the present PC and the DISP sign extended. If it is set the next sequential instruction is executed.
Condition Codes:
None affected

3.9.2.6  BRANCH IF NOT ZERO SET
BRANCH NOT EQUAL

BNZ  EXP
BNE  EXP

Operation:  (1) if \( N = 1 \); \( PC + \text{DISP} \rightarrow PC \)
            (2) if \( N = 0 \); \( \text{step} \)

If the Not Zero indicator is set, the Program Counter is replaced by the sum of the present PC and the DISP sign extended. If Not Zero is clear, the next sequential instruction is executed.

Condition Codes:
None affected

3.9.2.7  BRANCH IF NOT ZERO CLEAR
BRANCH EQUAL

BZ  EXP
BE  EXP

Operation:  (1) if \( N = 1 \); \( \text{step} \)
            (2) if \( N = 0 \); \( PC + \text{DISP} \rightarrow PC \)

If the Not Zero indicator is clear, the Program Counter is replaced by the sum of the present PC and the DISP sign extended. If it is set, the next sequential instruction is executed.

Condition Codes:
None affected

3.9.2.8  BRANCH IF OVERFLOW SET

BV  EXP

Operation:  (1) if \( V = 1 \); \( PC + \text{DISP} \rightarrow PC \)
            (2) if \( V = 0 \); \( \text{step} \)
If the overflow bit in the status word is set, the Program Counter is replaced by the sum of the present PC and the DISP sign extended. If it is clear, the next sequential instruction is executed.

Condition Codes:
None affected

3.9.2.9 BRANCH IF OVERFLOW CLEAR

<table>
<thead>
<tr>
<th>BNV EXP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0 0 1 1 1 D I S P</td>
</tr>
</tbody>
</table>

Operation:  
(1) if V = 1; step  
(2) if V = 0; PC + DISP → PC

If the overflow indicator is not set, the Program Counter is replaced by the sum of the present PC and the DISP sign extended. If it is set, the next sequential instruction is executed.

Condition Codes:
None affected

3.9.2.10 BRANCH GREATER THAN

<table>
<thead>
<tr>
<th>BGT EXP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0 1 0 0 0 D I S P</td>
</tr>
</tbody>
</table>

Operation:  
(1) if Z V (MeV) = 0; PC + DISP → PC  
(2) if Z V (MeV) = 1; step

The Program Counter is replaced by the sum of the present PC and the DISP if a previous result was greater than zero. Otherwise, the next instruction is executed.

Condition Codes:
None affected

3.9.2.11 BRANCH LESS THAN OR EQUAL

<table>
<thead>
<tr>
<th>BLE EXP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0 1 0 0 1 D I S P</td>
</tr>
</tbody>
</table>

Operation:  
(1) if Z V (MeV) = 1; PC + DISP → PC  
(2) if Z V (MeV) = 0; step
The Program Counter is replaced by the sum of the present PC and the DISP sign extended, if a previous operation result was less than or equal to zero. Otherwise, the next sequential instruction is executed.

Condition Codes:
None affected

3.9.2.12 BRANCH GREATER THAN OR EQUAL

**Operation:**
1. if \((M\&V) = 0\); PC + DISP → PC
2. if \((M\&V) = 1\); step

The Program Counter is replaced by the sum of the present PC and the DISP sign extended, if a previous result was greater than or equal to zero. Otherwise, the next sequential instructions are executed.

Condition Codes:
None affected

3.9.2.13 BRANCH LESS THAN

**Operation:**
1. if \(M\&V = 1\); PC + DISP → PC
2. if \(M\&V = 0\); step

The Program Counter is replaced by the sum of the present PC and the DISP sign extended, if a previous result was less than zero. Otherwise, the next sequential instruction is executed.

Condition Codes:
None affected
3.9.2.14 BRANCH HIGH

**BHI**  EXP

<table>
<thead>
<tr>
<th>15</th>
<th>12</th>
<th>8</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Operation:
1. if Z V L = 0; PC + DISP → PC
2. if Z V L = 1; step

The branch will occur if a previous operation, such as CMP, caused neither a carry or zero result. If the instructions were CMP R5, R6 a branch will occur if R5 had a higher unsigned value than R6.

**Condition Codes:**
None affected

3.9.2.15 BRANCH LOW OR SAME

**BLOS**  EXP

<table>
<thead>
<tr>
<th>15</th>
<th>12</th>
<th>8</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Operation:
1. if Z V L = 1; PC + DISP → PC
2. if Z V L = 0; step

The branch will occur if a previous operation caused either a carry or zero result. If the operation were CMP R5, R6 a branch will occur if R5 ≤ R6 - both being considered as unsigned quantities.

**Condition Codes:**
None affected

3.9.2.16 BRANCH ON COUNT

**BCT**  RA,EXP

<table>
<thead>
<tr>
<th>15</th>
<th>12</th>
<th>8</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>RA</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Operation:
1. if RA ≠ 0; (PC) + DISP → PC
2. if RA = 0; step

The contents of Register A are decremented by one. If the result is not zero the PC is replaced by EXP (PC + DISP). If the result is zero the next sequential instruction is executed.

3-50
Condition Codes:

N: Set if RA not equal to zero; cleared otherwise
V: Set if arithmetic overflow
L: Set equal to carry
M: Set to bit 15 of RA
3.10  SYSTEM CONTROL INSTRUCTIONS

3.10.1  PROGRAMMED INTERRUPT

```
TRP  LIT
  15 12 8  4  0
  0 1 1 1 | LITMS 1 1 0 0 | LITLS
```

Operation:  
(R0) → (R15)  
(PSR) → (R15)  
(PC) → (R15)  
2*LIT - L.S. Byte Of R0

30₁₆  = Address of Transfer Vector

TRP is a programmed interrupt or supervisor call. The contents of R0, PSR and PC, in that order, are saved in a stack table pointed to by R15. R0 is then loaded with the LIT. A new PC and PSR, in that order, are obtained from the transfer vector at byte address 30₁₆. R15 at completion will be six greater than before TRP was executed.

Condition Codes:
As in new PSR

Programming Note: The actual trap routine number LIT (≤ 255) is divided into 2 four bit fields in the instruction.

The TRP handling routine examines R0 (containing the 8 bit literal - LIT) and determines if the pointer (LIT) to the branch table is valid. If it is, the branch to the specific TRP routine is taken. If not such TRP address exists the TRP handling routine will generate an error message.

3.10.2  RETURN FROM INTERRUPT

```
RTI
  15 12 8  4  0
  0 1 1 1 0 0 0 0 | 1 1 0 1 0 0 0 0
```

Operation:  
(R15) → R15  
((R15)) → PC  
((R15)) → PSR

RTI is a return from a machine interrupt. The original contents of PC and PSR, in that order, are restored from the stack table pointed to by R15. At completion R15 points to stack word containing original PC. To be used in privileged mode only.

Condition Codes:
As contained in the PSR in the stack
3.10.3  **OR INTO PROGRAM STATUS REGISTER**

<table>
<thead>
<tr>
<th></th>
<th>LIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0, 1, 1, 1</td>
</tr>
<tr>
<td>12</td>
<td>0, 0, 0, 0</td>
</tr>
<tr>
<td>8</td>
<td>0, 0, 0, 0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**OSR**

Operation:  \((\text{PSR}) \lor \text{LIT}) \rightarrow \text{PSR}\)

The four bit literal is OR'd with the status bits \((L, M, N, V)\) of the PSR and the result placed into the status bits. The remaining bits of the PSR are unaffected.

**Condition Codes:**
As set by **PSR**

3.10.4  **AND INTO PROGRAM STATUS REGISTER**

<table>
<thead>
<tr>
<th></th>
<th>LIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0, 1, 0, 1</td>
</tr>
<tr>
<td>12</td>
<td>0, 0, 0, 0</td>
</tr>
<tr>
<td>8</td>
<td>1, 1, 1, 1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**ASR**

Operation:  \((\text{PSR}) \land \text{LIT}) \rightarrow \text{PSR}\)

The four bit literal is ANDed with the status bits \((L, M, N, V)\) of the PSR and the result is placed into the status bits. The remaining bits of the PSR are unaffected.

**Condition Codes:**
As set by **NSR**

3.10.5  **LOAD PROGRAM STATUS REGISTER**

<table>
<thead>
<tr>
<th></th>
<th>RA</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0, 1, 0, 1</td>
</tr>
<tr>
<td>12</td>
<td>R, A</td>
</tr>
<tr>
<td>11</td>
<td>1, 1, 1, 0</td>
</tr>
<tr>
<td>8</td>
<td>0, 0, 0, 0</td>
</tr>
</tbody>
</table>

**LSR**

Operation:  \((\text{RA}) \rightarrow \text{PSR}\)

The contents of RA replace the contents of the PSR. To be used in the privileged mode only.

**Condition Codes:**
As set by **LSR**
3.10.6 STORE PROGRAM STATUS REGISTER

SSR RA

\[
\begin{array}{cccccc}
15 & 12 & 8 & 4 & 0 \\
0 & 1 & 1 & 1 & \text{RA} & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
\end{array}
\]

Operation: \((PSR) \rightarrow RA\)

The contents of the Program Status Register are stored in register RA.

Condition Codes:
Unaffected

3.10.7 HALT

HALT

\[
\begin{array}{cccccc}
15 & 12 & 8 & 4 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
\end{array}
\]

The HALT instruction causes the processor to stop at the current address of the HALT (PC will be at current instruction address plus 2). To continue the RUN switch on the processor console must be placed in the HALT position, then it can be either placed in the RUN mode to continue executing the program or in the SIE position to single step through the program. To be used in privilidge mode only.

Condition Codes:
None affected

3.10.8 WAIT

WAIT

\[
\begin{array}{cccccc}
15 & 12 & 8 & 4 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
\end{array}
\]

The WAIT instruction causes the processor to wait at the current instruction address for an interrupt to occur. While in WAIT the processor does not access memory thus freeing the bus.

Condition Codes:
None affected

3-54
3.10.9  RESET

RESET issues a reset simultaneously to both the serial and parallel I/O busses. The effect of the RESET is the same as power up.

Condition Codes:
None affected
<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Type</th>
<th>Opcode</th>
<th>Description</th>
<th>Condition Codes</th>
<th>MIPI16/216</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>RA, RB</td>
<td>RR</td>
<td>Add Registers</td>
<td>XXXX</td>
<td>1</td>
<td>3-7</td>
</tr>
<tr>
<td>ADDL</td>
<td>RA, LIT</td>
<td>RI</td>
<td>Add literal extended to register</td>
<td>XXXX</td>
<td>1</td>
<td>3-8</td>
</tr>
<tr>
<td>ADDM</td>
<td>RA, EXP(RB)</td>
<td>MR</td>
<td>Add memory to register</td>
<td>XXXX</td>
<td>1</td>
<td>3-7</td>
</tr>
<tr>
<td>AND</td>
<td>RA, RB</td>
<td>RR</td>
<td>AND registers</td>
<td>-XX-</td>
<td>1</td>
<td>3-20</td>
</tr>
<tr>
<td>ANDB</td>
<td>RA, LIT</td>
<td>RI</td>
<td>AND literal with register</td>
<td>-XX-</td>
<td>1</td>
<td>3-21</td>
</tr>
<tr>
<td>ANDM</td>
<td>RA, EXP(RB)</td>
<td>MR</td>
<td>AND memory with register</td>
<td>-XX-</td>
<td>3^1</td>
<td>3-21</td>
</tr>
<tr>
<td>ASR</td>
<td>LIT</td>
<td>RR</td>
<td>AND literal into PSR</td>
<td>XXXX</td>
<td>1</td>
<td>3-53</td>
</tr>
<tr>
<td>BCT</td>
<td>RA, EXP</td>
<td>RL</td>
<td>Branch on count</td>
<td>XXXX</td>
<td>3</td>
<td>3-50</td>
</tr>
<tr>
<td>BE</td>
<td>EXP</td>
<td>RL</td>
<td>Branch on equal (≡ BZ)</td>
<td>---</td>
<td>3^2</td>
<td>3-47</td>
</tr>
<tr>
<td>BGE</td>
<td>EXP</td>
<td>RL</td>
<td>Branch if greater than or equal</td>
<td>---</td>
<td>3^2</td>
<td>3-49</td>
</tr>
<tr>
<td>BGT</td>
<td>EXP</td>
<td>RL</td>
<td>Branch if greater than</td>
<td>---</td>
<td>3^2</td>
<td>3-48</td>
</tr>
<tr>
<td>BHI</td>
<td>EXP</td>
<td>RL</td>
<td>Branch if high</td>
<td>---</td>
<td>3^2</td>
<td>3-50</td>
</tr>
<tr>
<td>BL</td>
<td>EXP</td>
<td>RL</td>
<td>Branch if link set</td>
<td>---</td>
<td>3^2</td>
<td>3-45</td>
</tr>
<tr>
<td>BLE</td>
<td>EXP</td>
<td>RL</td>
<td>Branch if less than or equal</td>
<td>---</td>
<td>3^2</td>
<td>3-48</td>
</tr>
<tr>
<td>BLOS</td>
<td>EXP</td>
<td>RL</td>
<td>Branch if low or same</td>
<td>---</td>
<td>3^2</td>
<td>3-50</td>
</tr>
<tr>
<td>BLT</td>
<td>EXP</td>
<td>RL</td>
<td>Branch if less than</td>
<td>---</td>
<td>3^2</td>
<td>3-49</td>
</tr>
<tr>
<td>BM</td>
<td>EXP</td>
<td>RL</td>
<td>Branch if most set</td>
<td>---</td>
<td>3^2</td>
<td>3-46</td>
</tr>
<tr>
<td>BN</td>
<td>EXP</td>
<td>RL</td>
<td>Branch if negative (≡ BM)</td>
<td>---</td>
<td>3^2</td>
<td>3-46</td>
</tr>
<tr>
<td>BNE</td>
<td>EXP</td>
<td>RL</td>
<td>Branch if not equal (≡ BNZ)</td>
<td>---</td>
<td>3^2</td>
<td>3-47</td>
</tr>
<tr>
<td>BNL</td>
<td>EXP</td>
<td>RL</td>
<td>Branch if link not set</td>
<td>---</td>
<td>3^2</td>
<td>3-46</td>
</tr>
<tr>
<td>BNM</td>
<td>EXP</td>
<td>RL</td>
<td>Branch if most not set</td>
<td>---</td>
<td>3^2</td>
<td>3-46</td>
</tr>
<tr>
<td>BNV</td>
<td>EXP</td>
<td>RL</td>
<td>Branch if overflow not set</td>
<td>---</td>
<td>3^2</td>
<td>3-48</td>
</tr>
<tr>
<td>BNZ</td>
<td>EXP</td>
<td>RL</td>
<td>Branch if overflow set</td>
<td>---</td>
<td>3^2</td>
<td>3-47</td>
</tr>
<tr>
<td>BP</td>
<td>EXP</td>
<td>RL</td>
<td>Branch if positive (≡ BNM)</td>
<td>---</td>
<td>3^2</td>
<td>3-46</td>
</tr>
<tr>
<td>BR</td>
<td>EXP</td>
<td>RL</td>
<td>Unconditional branch</td>
<td>---</td>
<td>3</td>
<td>3-45</td>
</tr>
<tr>
<td>BV</td>
<td>EXP</td>
<td>RL</td>
<td>Branch if overflow set</td>
<td>---</td>
<td>3^2</td>
<td>3-47</td>
</tr>
<tr>
<td>BZ</td>
<td>EXP</td>
<td>RL</td>
<td>Branch if not zero clear</td>
<td>---</td>
<td>3^2</td>
<td>3-47</td>
</tr>
<tr>
<td>Mnemonic</td>
<td>Type</td>
<td>Opcode</td>
<td>Description</td>
<td>Condition Codes</td>
<td>MIP16/216</td>
<td>Reference</td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
<td>--------</td>
<td>-------------------------------------------------</td>
<td>-----------------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>CAL</td>
<td>RA, RB</td>
<td>RR</td>
<td>1aBb Call Subroutine</td>
<td>----</td>
<td>2</td>
<td>3-43</td>
</tr>
<tr>
<td>CALM</td>
<td>RA, EXP(RB)</td>
<td>MR</td>
<td>1aAb Call Subroutine</td>
<td>----</td>
<td>3</td>
<td>3-43</td>
</tr>
<tr>
<td>CMP</td>
<td>RA, RB</td>
<td>RR</td>
<td>0aBb Arithmetic register compare</td>
<td>XXXX</td>
<td>1</td>
<td>3-15</td>
</tr>
<tr>
<td>CMPB</td>
<td>RA, LIT</td>
<td>RL</td>
<td>AaLL Arithmetic literal compare</td>
<td>XXXX</td>
<td>1</td>
<td>3-16</td>
</tr>
<tr>
<td>CMPM</td>
<td>RA, EXP(RB)</td>
<td>MR</td>
<td>0aAb Arithmetic memory compare</td>
<td>XXXX</td>
<td>3</td>
<td>3-15</td>
</tr>
<tr>
<td>COM</td>
<td>RA, RB</td>
<td>RR</td>
<td>1a1b One's complement register</td>
<td>-XX-</td>
<td>1</td>
<td>3-23</td>
</tr>
<tr>
<td>COMB</td>
<td>RA, RB</td>
<td>RR</td>
<td>1a3b One's complement register byte</td>
<td>-XX-</td>
<td>1</td>
<td>3-24</td>
</tr>
<tr>
<td>DADD</td>
<td>RA, RB</td>
<td>RR</td>
<td>7a2b Double precision add</td>
<td>XXXX</td>
<td>2</td>
<td>3-8</td>
</tr>
<tr>
<td>DADDM</td>
<td>RA, EXP(RB)</td>
<td>MR</td>
<td>7a1b Double precision memory add</td>
<td>XXXX</td>
<td>6/4</td>
<td>3-19</td>
</tr>
<tr>
<td>DCMP</td>
<td>RA, RB</td>
<td>RR</td>
<td>7a3b Double precision arithmetic compare</td>
<td>XXXX</td>
<td>3</td>
<td>3-16</td>
</tr>
<tr>
<td>DIV</td>
<td>RA, RB</td>
<td>RR</td>
<td>4a1b Divide registers</td>
<td>XXXX</td>
<td>34</td>
<td>3-12</td>
</tr>
<tr>
<td>DLM</td>
<td>RA, EXP(RB)</td>
<td>MR</td>
<td>5a1b Double precision register load</td>
<td>-XX-</td>
<td>4</td>
<td>3-39</td>
</tr>
<tr>
<td>DNEG</td>
<td>RA, RB</td>
<td>RR</td>
<td>7a4b Double precision two's complement</td>
<td>XXXX</td>
<td>3</td>
<td>3-14</td>
</tr>
<tr>
<td>DSL</td>
<td>RA, N</td>
<td>RN</td>
<td>6aEn Double register logical left shift</td>
<td>X-XX</td>
<td>4N/N+3</td>
<td>3-29</td>
</tr>
<tr>
<td>DSLA</td>
<td>RA, N</td>
<td>RN</td>
<td>6a8n Double register arithmetic left shift</td>
<td>XXXX</td>
<td>4N/N+3</td>
<td>3-32</td>
</tr>
<tr>
<td>DSM</td>
<td>RA, EXP(RB)</td>
<td>MR</td>
<td>5a9b Double precision register store</td>
<td>-XX-</td>
<td>4</td>
<td>3-39</td>
</tr>
<tr>
<td>DSR</td>
<td>RA, N</td>
<td>RN</td>
<td>6aBn Double register logical right shift</td>
<td>XXX-</td>
<td>4N/N+3</td>
<td>3-30</td>
</tr>
<tr>
<td>DSRA</td>
<td>RA, N</td>
<td>RN</td>
<td>6a5n Double register arithmetic right shift</td>
<td>XXX-</td>
<td>N+3</td>
<td>3-32</td>
</tr>
<tr>
<td>DSUB</td>
<td>RA, RB</td>
<td>RR</td>
<td>7a5b Double precision subtract</td>
<td>XXXX</td>
<td>2</td>
<td>3-10</td>
</tr>
<tr>
<td>DSUBM</td>
<td>RA, EXP(RB)</td>
<td>MR</td>
<td>7aAb Double precision memory subtract</td>
<td>XXXX</td>
<td>6/4</td>
<td>3-20</td>
</tr>
<tr>
<td>HALT</td>
<td></td>
<td>RR</td>
<td>70F0 Halt</td>
<td>----</td>
<td>1</td>
<td>3-54</td>
</tr>
<tr>
<td>IMUL</td>
<td>RA, RB</td>
<td>RR</td>
<td>4a4b Integer multiply</td>
<td>0XXX</td>
<td>18</td>
<td>3-11</td>
</tr>
<tr>
<td>JMP</td>
<td>RB</td>
<td>RR</td>
<td>10Db Jump</td>
<td>----</td>
<td>2</td>
<td>3-43</td>
</tr>
<tr>
<td>JMPM</td>
<td>EXP(RB)</td>
<td>MR</td>
<td>10Cb Jump</td>
<td>----</td>
<td>3</td>
<td>3-44</td>
</tr>
<tr>
<td>Mnemonic</td>
<td>Type</td>
<td>Opcode</td>
<td>Description</td>
<td>Condition Codes</td>
<td>MPl16/216 Microcycles</td>
<td>Reference</td>
</tr>
<tr>
<td>----------</td>
<td>-------</td>
<td>--------</td>
<td>-------------------------------------------------</td>
<td>-----------------</td>
<td>------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>L</td>
<td>RA, RB</td>
<td>RR</td>
<td>5a0b Load register</td>
<td>-XX-</td>
<td>3</td>
<td>3-34</td>
</tr>
<tr>
<td>LA</td>
<td>RA, EXP</td>
<td>MR</td>
<td>1a51 Load register with address</td>
<td>-XX-</td>
<td>2</td>
<td>3-36</td>
</tr>
<tr>
<td>LC</td>
<td>RA, RB</td>
<td>RR</td>
<td>0a9b Logically compare registers</td>
<td>XXXX</td>
<td>1</td>
<td>3-24</td>
</tr>
<tr>
<td>LCM</td>
<td>RA, EXP(RB)</td>
<td>MR</td>
<td>0a8b Logically compare memory with register</td>
<td>XXXX</td>
<td>3</td>
<td>3-25</td>
</tr>
<tr>
<td>LD</td>
<td>RA, RB</td>
<td>RR</td>
<td>5a4b Load register and decrement</td>
<td>XXXX</td>
<td>3</td>
<td>3-34</td>
</tr>
<tr>
<td>LDM</td>
<td>RA, N</td>
<td>RN</td>
<td>7a8n Load multiple registers</td>
<td>----</td>
<td>N +3</td>
<td>3-41</td>
</tr>
<tr>
<td>LI</td>
<td>RA, RB</td>
<td>RR</td>
<td>5a2b Load register and increment</td>
<td>XXXX</td>
<td>3</td>
<td>3-34</td>
</tr>
<tr>
<td>LM</td>
<td>RA, EXP(RB)</td>
<td>MR</td>
<td>1a2b Load register</td>
<td>-XX-</td>
<td>3</td>
<td>3-36</td>
</tr>
<tr>
<td>LMB</td>
<td>RA, EXP(RB)</td>
<td>MR</td>
<td>1a4b Load register byte</td>
<td>-XX-</td>
<td>3</td>
<td>3-36</td>
</tr>
<tr>
<td>LSR</td>
<td>RA</td>
<td>RR</td>
<td>5aE0 Load PSR</td>
<td>XXXX</td>
<td>1</td>
<td>3-53</td>
</tr>
<tr>
<td>MOV</td>
<td>RA, RB</td>
<td>RR</td>
<td>1a0b Move register</td>
<td>-XX-</td>
<td>1</td>
<td>3-12</td>
</tr>
<tr>
<td>MOVL</td>
<td>RA, LIT</td>
<td>RI</td>
<td>DaLL Move literal extended to register</td>
<td>-XX-</td>
<td>1</td>
<td>3-13</td>
</tr>
<tr>
<td>MOVN</td>
<td>RA, RB</td>
<td>RR</td>
<td>7a6b Move register negative</td>
<td>XXXX</td>
<td>1</td>
<td>3-19</td>
</tr>
<tr>
<td>MOVP</td>
<td>RA, RB</td>
<td>RR</td>
<td>7a7b Move register positive</td>
<td>XXXX</td>
<td>1</td>
<td>3-18</td>
</tr>
<tr>
<td>MOVX</td>
<td>RA, RB</td>
<td>RR</td>
<td>4a2b Move register byte extended</td>
<td>-XX-</td>
<td>1</td>
<td>3-13</td>
</tr>
<tr>
<td>MUL</td>
<td>RA, RB</td>
<td>RR</td>
<td>4a0b Multiply registers</td>
<td>0XX-</td>
<td>18</td>
<td>3-10</td>
</tr>
<tr>
<td>MULR</td>
<td>RA, RB</td>
<td>RR</td>
<td>4a6b Multiply registers and round</td>
<td>0XXX</td>
<td>20</td>
<td>3-11</td>
</tr>
<tr>
<td>NEG</td>
<td>RA, RB</td>
<td>RR</td>
<td>4a4b Two's complement register</td>
<td>XXXX</td>
<td>1</td>
<td>3-14</td>
</tr>
<tr>
<td>NOP</td>
<td></td>
<td>RL</td>
<td>8E00 No operation (BR *+2)</td>
<td>----</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>RA, RB</td>
<td>RR</td>
<td>0a7b Inclusive OR registers</td>
<td>-XX-</td>
<td>1</td>
<td>3-21</td>
</tr>
<tr>
<td>ORM</td>
<td>RA, EXP(RB)</td>
<td>MR</td>
<td>0a6b Inclusive OR with memory</td>
<td>-XX-</td>
<td>3</td>
<td>3-22</td>
</tr>
<tr>
<td>OSR</td>
<td>LIT</td>
<td>RR</td>
<td>700L Inclusive OR into PSR</td>
<td>XXXX</td>
<td>2</td>
<td>3-53</td>
</tr>
<tr>
<td>POP</td>
<td>RA, N</td>
<td>RN</td>
<td>5aCm Pop through R15</td>
<td>----</td>
<td>N +3</td>
<td>3-39</td>
</tr>
<tr>
<td>PUSH</td>
<td>RA, N</td>
<td>RN</td>
<td>5aDm Push through R15</td>
<td>----</td>
<td>N +3</td>
<td>3-40</td>
</tr>
<tr>
<td>Mnemonic</td>
<td>Type</td>
<td>Opcode</td>
<td>Description</td>
<td>Condition Codes</td>
<td>MIP16/216</td>
<td>Reference</td>
</tr>
<tr>
<td>----------</td>
<td>-------</td>
<td>--------</td>
<td>--------------------------------------------</td>
<td>-----------------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>RSET</td>
<td>RR</td>
<td>2010</td>
<td>Reset</td>
<td>-XX-</td>
<td>XXX</td>
<td>1</td>
</tr>
<tr>
<td>RTI</td>
<td>RR</td>
<td>7FDF</td>
<td>Return from interrupt</td>
<td>XXXX</td>
<td>3</td>
<td>3-52</td>
</tr>
<tr>
<td>S</td>
<td>RA, RB</td>
<td>RR</td>
<td>Store register</td>
<td></td>
<td>N + 1</td>
<td>3-31</td>
</tr>
<tr>
<td>SADD</td>
<td>RA, RB</td>
<td>RR</td>
<td>Saturated add</td>
<td></td>
<td>N + 1</td>
<td>3-17</td>
</tr>
<tr>
<td>SD</td>
<td>RA, RB</td>
<td>RR</td>
<td>Store register and decrement</td>
<td>XXXX</td>
<td>3</td>
<td>3-37</td>
</tr>
<tr>
<td>SI</td>
<td>RA, RB</td>
<td>RR</td>
<td>Store register and increment</td>
<td>XXXX</td>
<td>3</td>
<td>3-37</td>
</tr>
<tr>
<td>SLA</td>
<td>RA, N</td>
<td>RN</td>
<td>Shift left arithmetic</td>
<td>XXXX</td>
<td>N + 1</td>
<td>3-31</td>
</tr>
<tr>
<td>SLAB</td>
<td>RA, N</td>
<td>RN</td>
<td>Shift byte left arithmetic</td>
<td>XXXX</td>
<td>N + 1</td>
<td>3-31</td>
</tr>
<tr>
<td>SLL</td>
<td>RA, N</td>
<td>RN</td>
<td>Shift left logical and insert link</td>
<td>XXXX</td>
<td>N + 1</td>
<td>3-26</td>
</tr>
<tr>
<td>SLLB</td>
<td>RA, N</td>
<td>RN</td>
<td>Shift byte left logical and insert link</td>
<td>XXXX</td>
<td>N + 1</td>
<td>3-26</td>
</tr>
<tr>
<td>SLZ</td>
<td>RA, N</td>
<td>RN</td>
<td>Shift left logical</td>
<td>XXXX</td>
<td>N + 1</td>
<td>3-28</td>
</tr>
<tr>
<td>SLZB</td>
<td>RA, N</td>
<td>RN</td>
<td>Shift byte left logical</td>
<td>XXXX</td>
<td>N + 1</td>
<td>3-28</td>
</tr>
<tr>
<td>SM</td>
<td>RA, EXP(RB)</td>
<td>MR</td>
<td>Store register</td>
<td>-XX-</td>
<td>3</td>
<td>3-38</td>
</tr>
<tr>
<td>SMB</td>
<td>RA, EXP(RB)</td>
<td>MR</td>
<td>Store register byte</td>
<td>-XX-</td>
<td>3</td>
<td>3-38</td>
</tr>
<tr>
<td>SRA</td>
<td>RA, N</td>
<td>RN</td>
<td>Shift right arithmetic</td>
<td>XXXX</td>
<td>N + 1</td>
<td>3-32</td>
</tr>
<tr>
<td>SRC</td>
<td>RA, N</td>
<td>RN</td>
<td>Shift right circular</td>
<td>XXXX</td>
<td>N + 1</td>
<td>3-30</td>
</tr>
<tr>
<td>SRL</td>
<td>RA, N</td>
<td>RN</td>
<td>Shift right logical and insert link</td>
<td>XXXX</td>
<td>N + 1</td>
<td>3-27</td>
</tr>
<tr>
<td>SRLB</td>
<td>RA, N</td>
<td>RN</td>
<td>Shift byte right logical and insert link</td>
<td>XXXX</td>
<td>N + 1</td>
<td>3-27</td>
</tr>
<tr>
<td>SRZ</td>
<td>RA, N</td>
<td>RN</td>
<td>Shift right logical</td>
<td>XXXX</td>
<td>N + 1</td>
<td>3-29</td>
</tr>
<tr>
<td>SSR</td>
<td>RA</td>
<td>RR</td>
<td>Store PSR</td>
<td>-XX-</td>
<td>1</td>
<td>3-54</td>
</tr>
<tr>
<td>SSUB</td>
<td>RA, RB</td>
<td>RR</td>
<td>Saturated subtract</td>
<td>XXXX</td>
<td>4</td>
<td>3-17</td>
</tr>
<tr>
<td>STM</td>
<td>RA, N</td>
<td>RN</td>
<td>Store multiple registers</td>
<td>-XX-</td>
<td>N + 3</td>
<td>3-42</td>
</tr>
<tr>
<td>SUB</td>
<td>RA, RB</td>
<td>RR</td>
<td>Subtract register</td>
<td>XXXX</td>
<td>1</td>
<td>3-9</td>
</tr>
<tr>
<td>SUBM</td>
<td>RA, EXP(RB)</td>
<td>MR</td>
<td>Subtract memory</td>
<td>XXXX</td>
<td>3¹</td>
<td>3-9</td>
</tr>
<tr>
<td>TRP</td>
<td>LIT</td>
<td>RN</td>
<td>Programmed interrupt</td>
<td>XXXX</td>
<td>8</td>
<td>3-52</td>
</tr>
<tr>
<td>WAIT</td>
<td>RR</td>
<td>70E0</td>
<td>Wait</td>
<td>-XX-</td>
<td>1</td>
<td>3-54</td>
</tr>
</tbody>
</table>
TABLE 3-1 (Cont)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Type</th>
<th>Opcode</th>
<th>Description</th>
<th>Condition Codes</th>
<th>MIP16/216</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOR</td>
<td>RA, RB</td>
<td>RR</td>
<td>Exclusive OR registers</td>
<td>-XX-</td>
<td>1</td>
<td>3-22</td>
</tr>
<tr>
<td>XORM</td>
<td>RA, EXP(RB)</td>
<td>MR</td>
<td>Exclusive OR memory with register</td>
<td>-XX-</td>
<td>3</td>
<td>3-23</td>
</tr>
</tbody>
</table>

a - Register RA  
b - Register RB  
L - 4 bit literal  
LL - 8 bit literal  
n - Number of shifts minus one (N-1)  
m - Number of registers to be loaded/stored  
dd - Displacement  
X - Condition code affected  
- - Condition code unaffected  
0 - Condition code set to zero

1 - 2 cycles in the immediate mode  
2 - 2 cycles if path is step
### TABLE 3-2

#### INSTRUCTION FORMATS

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-12</td>
<td>OP</td>
</tr>
<tr>
<td>8</td>
<td>RA</td>
</tr>
<tr>
<td>4-0</td>
<td>OP</td>
</tr>
<tr>
<td></td>
<td>RB</td>
</tr>
</tbody>
</table>

- **RR**
- **RI**
- **RN**
- **RL**
- **MR**

3-61
<table>
<thead>
<tr>
<th>n</th>
<th>(2^{-n})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>2</td>
<td>0.05</td>
</tr>
<tr>
<td>4</td>
<td>0.025</td>
</tr>
<tr>
<td>8</td>
<td>0.125</td>
</tr>
<tr>
<td>16</td>
<td>0.0625</td>
</tr>
<tr>
<td>32</td>
<td>0.03125</td>
</tr>
<tr>
<td>64</td>
<td>0.015625</td>
</tr>
<tr>
<td>128</td>
<td>0.0078125</td>
</tr>
<tr>
<td>256</td>
<td>0.00390625</td>
</tr>
<tr>
<td>512</td>
<td>0.001953125</td>
</tr>
<tr>
<td>1024</td>
<td>0.0009765625</td>
</tr>
<tr>
<td>2048</td>
<td>0.00048828125</td>
</tr>
<tr>
<td>4096</td>
<td>0.000244140625</td>
</tr>
<tr>
<td>8192</td>
<td>0.0001220703125</td>
</tr>
<tr>
<td>16384</td>
<td>0.00006103515625</td>
</tr>
<tr>
<td>32768</td>
<td>0.000030517578125</td>
</tr>
<tr>
<td>65536</td>
<td>0.000015258089875</td>
</tr>
<tr>
<td>131072</td>
<td>0.000007629394531</td>
</tr>
<tr>
<td>262144</td>
<td>0.000003814697265</td>
</tr>
<tr>
<td>524288</td>
<td>0.000001907348632</td>
</tr>
<tr>
<td>1048576</td>
<td>0.000000953789625</td>
</tr>
<tr>
<td>2097152</td>
<td>0.000000476837158</td>
</tr>
<tr>
<td>4194304</td>
<td>0.000000238418579</td>
</tr>
<tr>
<td>8388608</td>
<td>0.000000119209289</td>
</tr>
<tr>
<td>16777216</td>
<td>0.000000059604775</td>
</tr>
<tr>
<td>33554432</td>
<td>0.000000029802322</td>
</tr>
<tr>
<td>67108864</td>
<td>0.000000014901161</td>
</tr>
<tr>
<td>134217728</td>
<td>0.000000007450580</td>
</tr>
<tr>
<td>268435456</td>
<td>0.000000003725290</td>
</tr>
<tr>
<td>536870912</td>
<td>0.000000001862645</td>
</tr>
<tr>
<td>1073741824</td>
<td>0.000000000931322</td>
</tr>
<tr>
<td>2147483648</td>
<td>0.000000000465661</td>
</tr>
<tr>
<td>4294967296</td>
<td>0.000000000232830</td>
</tr>
<tr>
<td>8589934592</td>
<td>0.000000000116415</td>
</tr>
<tr>
<td>17179869184</td>
<td>0.000000000058207</td>
</tr>
<tr>
<td>34359738368</td>
<td>0.000000000029103</td>
</tr>
<tr>
<td>68719476736</td>
<td>0.000000000014551</td>
</tr>
<tr>
<td>137438953472</td>
<td>0.000000000007275</td>
</tr>
<tr>
<td>274877906944</td>
<td>0.000000000003637</td>
</tr>
<tr>
<td>549755813888</td>
<td>0.000000000001818</td>
</tr>
</tbody>
</table>

3-62
# TABLE 3-4

## DECIMAL AND HEXADECIMAL CONVERSIONS

### POWERS OF 16

<table>
<thead>
<tr>
<th>n</th>
<th>n.16⁰</th>
<th>n.16¹</th>
<th>n.16²</th>
<th>n.16³</th>
<th>n.16⁴</th>
<th>n.16⁵</th>
<th>n.16⁶</th>
<th>n.16⁷</th>
<th>n.16⁸</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEX</td>
<td>DEC</td>
<td>HEX</td>
<td>DEC</td>
<td>HEX</td>
<td>DEC</td>
<td>HEX</td>
<td>DEC</td>
<td>HEX</td>
<td>DEC</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>2</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>3</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>4</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
<td>1024</td>
</tr>
<tr>
<td>5</td>
<td>2048</td>
<td>2048</td>
<td>2048</td>
<td>2048</td>
<td>2048</td>
<td>2048</td>
<td>2048</td>
<td>2048</td>
<td>2048</td>
</tr>
<tr>
<td>6</td>
<td>4096</td>
<td>4096</td>
<td>4096</td>
<td>4096</td>
<td>4096</td>
<td>4096</td>
<td>4096</td>
<td>4096</td>
<td>4096</td>
</tr>
<tr>
<td>7</td>
<td>8192</td>
<td>8192</td>
<td>8192</td>
<td>8192</td>
<td>8192</td>
<td>8192</td>
<td>8192</td>
<td>8192</td>
<td>8192</td>
</tr>
<tr>
<td>8</td>
<td>16384</td>
<td>16384</td>
<td>16384</td>
<td>16384</td>
<td>16384</td>
<td>16384</td>
<td>16384</td>
<td>16384</td>
<td>16384</td>
</tr>
<tr>
<td>9</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
<td>32768</td>
</tr>
<tr>
<td>A</td>
<td>65536</td>
<td>65536</td>
<td>65536</td>
<td>65536</td>
<td>65536</td>
<td>65536</td>
<td>65536</td>
<td>65536</td>
<td>65536</td>
</tr>
<tr>
<td>B</td>
<td>131072</td>
<td>131072</td>
<td>131072</td>
<td>131072</td>
<td>131072</td>
<td>131072</td>
<td>131072</td>
<td>131072</td>
<td>131072</td>
</tr>
<tr>
<td>C</td>
<td>262144</td>
<td>262144</td>
<td>262144</td>
<td>262144</td>
<td>262144</td>
<td>262144</td>
<td>262144</td>
<td>262144</td>
<td>262144</td>
</tr>
<tr>
<td>D</td>
<td>524288</td>
<td>524288</td>
<td>524288</td>
<td>524288</td>
<td>524288</td>
<td>524288</td>
<td>524288</td>
<td>524288</td>
<td>524288</td>
</tr>
<tr>
<td>E</td>
<td>1048576</td>
<td>1048576</td>
<td>1048576</td>
<td>1048576</td>
<td>1048576</td>
<td>1048576</td>
<td>1048576</td>
<td>1048576</td>
<td>1048576</td>
</tr>
<tr>
<td>F</td>
<td>2097152</td>
<td>2097152</td>
<td>2097152</td>
<td>2097152</td>
<td>2097152</td>
<td>2097152</td>
<td>2097152</td>
<td>2097152</td>
<td>2097152</td>
</tr>
</tbody>
</table>

To translate a decimal number to a hex number, find the next lower decimal number and its hex equivalent. Subtract this number from the decimal number. The next hex number is obtained from the difference. Continue until the entire hex number is developed.

To translate a hex number to its decimal equivalent, find the hex number and its decimal equivalent for each hex position. Sum these to obtain the decimal number.
SECTION 4
PROGRAMMING OF PERIPHERALS

4.0 INTRODUCTION

The MIP is designed to use a wide variety of peripherals ranging from teletypes to real
time signal processing interfaces. A broad range of transfer rates, interrupt requirements
and data transmission requirements may be accommodated. Programming is simplified by
using a unified addressing structure and common interrupt programming techniques.

This flexibility is obtained by providing both a parallel and a serial data bus. The serial
bus utilizes differential transmission and is suited for character type devices located up to
fifty feet away in a high noise environment. It has a character rate of 500 KHz. The
parallel bus is intended for high speed word oriented devices located within the processor
enclosure. Transfer rates may be as high as 5 MHz on this bus for data and 5-14 MHz for
instruction fetches.

Programming is simplified by using the same instructions for both busses as well as
memory reference. Different addresses are used for devices on the serial or parallel
busses. The upper 1024 byte addresses of page zero form the device address space of
which the upper 256 byte addresses form the serial device address space.

Interrupt functions the same way for both busses. A device generates an interrupt request
(one level for serial I/O; three levels for parallel I/O). The processor saves machine
status and polls the highest level. It determines which of the requests has the highest
priority and fetches a transfer vector to the appropriate handler. The transfer of control
can take as little as 1 μs for parallel requests or 5 μs for serial requests.

Standard peripherals are available to provide input/output capability. These may be
augmented by special application oriented peripherals. The standard peripherals include:

a. ASR-33 Teletype (serial bus)
b. 300 cps Paper Tape Reader (serial bus)
c. 75 cps Paper Tape Punch (serial bus)
d. Removable Cartridge DISK File (parallel bus)
e. Real Time Clock (serial bus)
Programming examples illustrating the use of these peripherals and interfaces are provided in section five, Peripheral Descriptions. Also provided in that section is a sample interrupt handling routine.

4.1 GENERAL PERIPHERAL INTERFACE OPERATION

A peripheral will be attached to one of the I/O busses by an interface. This interface will contain a status register and buffer registers for input or output. The processor addresses these registers and transfers data into or from them to control a peripheral. Data transfers may be either byte or full word depending on the register.

4.1.1 STATUS REGISTERS

This register usually contains bits indicating the status of the peripheral. A bit is usually provided to enable an interrupt request on the basis of certain status bits going from low to high as well as a DONE bit. The DONE bit is usually used to indicate completion of a peripheral operation initiated by loading an output buffer or by setting a function bit in the status register. Errors in peripheral operation are usually indicated by ERROR bits in the status register. If an interface controls multiple devices the status register may contain unit selection bits.

4.1.2 PERIPHERAL INTERRUPTS

The interface usually contains an interrupt request bit which is set by the low to high transition of various status bits; e.g., the DONE bit. Once set, it generates a request at a predetermined level and permits the interface to respond to polls of that level. The interface’s poll position and request level and hence its transfer vector address are usually set by backpanel wiring. The request bit is reset when the status register is read by the processor.
SECTION 5
PERIPHERAL DESCRIPTIONS

5.0 INTRODUCTION

Standard peripherals are provided with the MIP to provide standardized input-output capabilities. These units are interfaced to the MIP using the techniques outlined in Section 4, Programming of Peripherals.

In the paragraphs which follow these devices and their interfaces are described. Programming examples are provided for each device to illustrate their operation. Device addresses and transfer vectors are given in Section 5.5.

5.1 ASR-33 TELETYPEx

This device contains two separate subunits - a printer/punch and a keyboard/reader-capable of handling 10 characters per second to and from the computer simultaneously. The teletype unit is attached to the serial I/O bus through a teletype interface.

5.1.1 TELETYPE DEVICE CHARACTERISTICS

5.1.1.1 Codes

The 8 bit code generated by the keyboard and printed by the printer corresponds to the seven bit American Standard Code for Information Interchange (ASCII) modified. To convert ASCII to teletype code set bit 8 to a one. The teletype code is shown in table 5-1 below. The eighth bit corresponds to a parity bit. Most teletypes keyboards set this to a one for all data; however, some teletypes set this to an odd parity bit (i.e., a one if odd number of data bits). This bit is ignored by the printer.

The reader/punch will transmit/receive characters exactly as they appear on the tape or as they are sent from the computer. The keyboard can generate all characters except E0₁₆ to FC₁₆ and FE₁₆ (lower case letters).

5.1.1.2 Teletype Serial Code

Data is transmitted to and from the interface to the teletype using serial, 11 unit code, at 10 characters per second. This code is illustrated in figure 5-1. Twenty milliampere,
<table>
<thead>
<tr>
<th>Teletype Character</th>
<th>Teletype Code</th>
<th>Teletype Character</th>
<th>Teletype Code</th>
<th>Teletype Character</th>
<th>Teletype Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>B0</td>
<td>blank</td>
<td>A0</td>
<td>BEL</td>
<td>87</td>
</tr>
<tr>
<td>1</td>
<td>B1</td>
<td>!</td>
<td>A1</td>
<td>FE</td>
<td>88</td>
</tr>
<tr>
<td>2</td>
<td>B2</td>
<td>&quot;</td>
<td>A2</td>
<td>HTAB</td>
<td>89</td>
</tr>
<tr>
<td>3</td>
<td>B3</td>
<td>#</td>
<td>A3</td>
<td>LINE FEED</td>
<td>8A</td>
</tr>
<tr>
<td>4</td>
<td>B4</td>
<td>$</td>
<td>A4</td>
<td>V TAB</td>
<td>8B</td>
</tr>
<tr>
<td>5</td>
<td>B5</td>
<td>%</td>
<td>A5</td>
<td>FORM</td>
<td>8C</td>
</tr>
<tr>
<td>6</td>
<td>B6</td>
<td>&amp;</td>
<td>A6</td>
<td>RETURN</td>
<td>8D</td>
</tr>
<tr>
<td>7</td>
<td>B7</td>
<td></td>
<td>A7</td>
<td>SO</td>
<td>8E</td>
</tr>
<tr>
<td>8</td>
<td>B8</td>
<td>(</td>
<td>A8</td>
<td>SI</td>
<td>8F</td>
</tr>
<tr>
<td>9</td>
<td>B9</td>
<td>)</td>
<td>A9</td>
<td>DCO</td>
<td>90</td>
</tr>
<tr>
<td>A</td>
<td>C1</td>
<td>*</td>
<td>AA</td>
<td>X-ON</td>
<td>91</td>
</tr>
<tr>
<td>B</td>
<td>C2</td>
<td>+</td>
<td>AB</td>
<td>TAPE ON</td>
<td>92</td>
</tr>
<tr>
<td>C</td>
<td>C3</td>
<td>-</td>
<td>AC</td>
<td>X-OFF</td>
<td>93</td>
</tr>
<tr>
<td>D</td>
<td>C4</td>
<td>.</td>
<td>AE</td>
<td>ERROR</td>
<td>95</td>
</tr>
<tr>
<td>E</td>
<td>C5</td>
<td>/</td>
<td>AF</td>
<td>SYNC</td>
<td>96</td>
</tr>
<tr>
<td>F</td>
<td>C6</td>
<td>:</td>
<td>BA</td>
<td>LEM</td>
<td>97</td>
</tr>
<tr>
<td>G</td>
<td>C7</td>
<td>;</td>
<td>BB</td>
<td>S0</td>
<td>98</td>
</tr>
<tr>
<td>H</td>
<td>C8</td>
<td>&lt;</td>
<td>BC</td>
<td>S1</td>
<td>99</td>
</tr>
<tr>
<td>I</td>
<td>C9</td>
<td></td>
<td>BD</td>
<td>S2</td>
<td>9A</td>
</tr>
<tr>
<td>J</td>
<td>CA</td>
<td>=</td>
<td>BE</td>
<td>S3</td>
<td>9B</td>
</tr>
<tr>
<td>K</td>
<td>CB</td>
<td>&gt;</td>
<td>BF</td>
<td>S4</td>
<td>9C</td>
</tr>
<tr>
<td>L</td>
<td>CC</td>
<td>?</td>
<td>CO</td>
<td>S5</td>
<td>9D</td>
</tr>
<tr>
<td>M</td>
<td>CD</td>
<td>@</td>
<td>DB</td>
<td>S6</td>
<td>9E</td>
</tr>
<tr>
<td>N</td>
<td>CE</td>
<td></td>
<td>DC</td>
<td>S7</td>
<td>9F</td>
</tr>
<tr>
<td>O</td>
<td>CF</td>
<td></td>
<td>DD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>D0</td>
<td></td>
<td>DE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>D1</td>
<td></td>
<td>DF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>D2</td>
<td>RUBOUT</td>
<td>FF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>D3</td>
<td>NUL</td>
<td>80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>D4</td>
<td>SOM</td>
<td>81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>D5</td>
<td>EOA</td>
<td>82</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>D6</td>
<td>EOM</td>
<td>83</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>D7</td>
<td>EOT</td>
<td>84</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>D8</td>
<td>WRU</td>
<td>85</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>D9</td>
<td>RU</td>
<td>86</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>DA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MARK IS A PUNCH OR A 1 IN THE BINARY CODE
SPACE IS A NO PUNCH OR A 0 IN THE BINARY CODE

Figure 5-1 Teletype Serial Code.
full duplex current loops are used. A cable of up to ten feet in length is attached to an accessory plug at the rear of the enclosure.

5.1.1.3 Operating Power

Approximately 500 watts of power are required. This is supplied from the accessory plug at the rear of the computer enclosure so that the main computer power switch will control the teletype power. Depending on the model either 115V ± 10%, 60 Hz ± 0.45 Hz or 230V ± 10%, 50 ± 0.75 Hz power may be used.

5.1.1.4 Installation

The unit is mounted on a stand and is 33 inches in height. It requires floor space 22-1/4” wide x 18-1/2” deep. The unit weight is 56 pounds with stand.

5.1.1.5 Environment

Commercial Lab Environment:
0 to 40°C continuous temperature operating
0 to 90% relatively humidity without condensation

5.1.2 TELETYPE INTERFACE CHARACTERISTICS

The interface between the computer and the teletype is located on a standard 8 x 11 inch card along with the punch and reader interfaces. The interface is attached to the serial bus and uses interrupt poll positions, 2 and 1 for the punch and reader respectively. This corresponds to transfer vector addresses of 244-247 (Printer) and 248-251 (Keyboard). The poll position may be altered by backpanel wiring but should be below the reader and punch.

5.1.2.1 Interface Registers

The interface contains teletype keyboard/reader input buffer (TKB), a printer/punch output buffer TPB, a keyboard/reader status register (TKS), and a printer/punch status register (TPS). These have the following format,

Input Buffer (TKB) 7 0 Address FFF2

Output Buffer (TPB) 7 0 Address FFF6

5-4
Teletype Keyboard/Reader Status Register (TKS) Address FFF0

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Interrupt enable</td>
</tr>
<tr>
<td>6</td>
<td>Power up or reset sets to zero</td>
</tr>
<tr>
<td>5</td>
<td>Read/write</td>
</tr>
<tr>
<td>4</td>
<td>ERROR - read only</td>
</tr>
<tr>
<td>3</td>
<td>Power up or reset sets to zero</td>
</tr>
<tr>
<td>2</td>
<td>Cleared by reading input buffer</td>
</tr>
<tr>
<td>1</td>
<td>DONE - read only</td>
</tr>
<tr>
<td>0</td>
<td>Power up or reset sets to zero</td>
</tr>
<tr>
<td></td>
<td>Cleared by reading input buffer</td>
</tr>
</tbody>
</table>

Teletype Printer/Punch Status (TPS) Address FFF4

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Interrupt enable</td>
</tr>
<tr>
<td>6</td>
<td>Read/write</td>
</tr>
<tr>
<td>5</td>
<td>Power up or reset sets to zero</td>
</tr>
<tr>
<td>4</td>
<td>DONE</td>
</tr>
<tr>
<td>3</td>
<td>Read only</td>
</tr>
<tr>
<td>2</td>
<td>Power up or reset sets to one</td>
</tr>
<tr>
<td>1</td>
<td>Cleared by writing output buffer</td>
</tr>
</tbody>
</table>

5.1.2.2 Reader/Keyboard Interface Operation

Keying in a character or receiving a character from the reader causes the teletype to transmit a character into the interface input buffer. Receipt of the complete character causes DONE to be set in the TKS. Reading the input buffer will cause DONE and ERROR to be reset. If the teletype transmits a character before DONE has been reset the ERROR bit will be set. The input buffer must be read within 100 milliseconds of DONE to insure no loss of data.

DONE or ERROR going to a one will set an internal interrupt request bit, if interrupt is enabled. This will cause an interrupt through the serial I/O level. This bit may be reset by reading the status register.
5.1.3 PRINTER/PUNCH INTERFACE OPERATION

The computer initiates printing or punching of a character by sending it to the output buffer (TPB). This resets the DONE bit in TPS. The bit is set when the character has been transmitted and the buffer reloaded. If the interrupt is enabled the 0 to 1 transition of DONE will cause the interrupt request bit to be set. This bit is reset by reading the status register (TPS).

5.1.4 PROGRAMMING EXAMPLES

5.1.4.1 Interrupt Disabled

The following routine echoes what is entered on the keyboard onto the printer.

<table>
<thead>
<tr>
<th>Label</th>
<th>Opcode</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPS</td>
<td>EQU</td>
<td>'FFF4</td>
<td></td>
</tr>
<tr>
<td>TKS</td>
<td>EQU</td>
<td>'FFF0</td>
<td></td>
</tr>
<tr>
<td>TPB</td>
<td>EQU</td>
<td>'FFF6</td>
<td></td>
</tr>
<tr>
<td>TKB</td>
<td>EQU</td>
<td>'FFF2</td>
<td></td>
</tr>
<tr>
<td>ECHO</td>
<td>LB</td>
<td>R0, TKS(2)</td>
<td>Get Status</td>
</tr>
<tr>
<td></td>
<td>BNM</td>
<td>ECHO</td>
<td>WAIT until character entered</td>
</tr>
<tr>
<td></td>
<td>LB</td>
<td>R1, TPB(2)</td>
<td></td>
</tr>
<tr>
<td>WAIT</td>
<td>LB</td>
<td>R0, TPS(2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BNM</td>
<td>WAIT</td>
<td>WAIT until punch done</td>
</tr>
<tr>
<td>SB</td>
<td>R1, TTB(2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BR</td>
<td>ECHO</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.1.4.2 Interrupt Enabled

The following routine transfers a block of characters to the teleprinter using interrupt. The printer routine is initiated by establishing unit control block containing the count and the address of the data to be transmitted. The setup routine starts I/O and then returns to the program. As each character is transmitted it interrupts the computer. The handler reloads the buffer until the count is exhausted.

<table>
<thead>
<tr>
<th>Label</th>
<th>Opcode</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORG</td>
<td>244</td>
<td></td>
<td>TTY Printer T.V.</td>
</tr>
<tr>
<td>DATA</td>
<td>TTYP</td>
<td></td>
<td>PC</td>
</tr>
<tr>
<td>DATA</td>
<td>0</td>
<td></td>
<td>PSR</td>
</tr>
<tr>
<td>TTYP</td>
<td>STM</td>
<td>R0, 2</td>
<td>Begin Handler</td>
</tr>
<tr>
<td></td>
<td>LB</td>
<td>R0, TPS(2)</td>
<td>Turn off Request</td>
</tr>
<tr>
<td>TPS</td>
<td>EQU</td>
<td>'FFF4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>R0, UCB(1)</td>
<td>Load Count</td>
</tr>
<tr>
<td>ADDL</td>
<td>R0, -1</td>
<td></td>
<td>Decrement</td>
</tr>
<tr>
<td>BGT</td>
<td>MORE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Label</td>
<td>Opcode</td>
<td>Operand</td>
<td>Comments</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
<td>---------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>DONE</td>
<td>LDM</td>
<td>R1, 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RTI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MORE</td>
<td>S</td>
<td>R0, UCB(1)</td>
<td>Update Count</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>R0, UCB+2(1)</td>
<td>Get Address</td>
</tr>
<tr>
<td></td>
<td>LIB</td>
<td>R1, R0</td>
<td>Get Character</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>R0, UCB+2(1)</td>
<td>Update Address</td>
</tr>
<tr>
<td></td>
<td>SB</td>
<td>R0, TPB(2)</td>
<td>Print Char and turn off DONE</td>
</tr>
<tr>
<td>TPB</td>
<td>EQU</td>
<td>'FFF6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BR</td>
<td>DONE</td>
<td></td>
</tr>
<tr>
<td>UCB</td>
<td>DATA</td>
<td>0, 0</td>
<td></td>
</tr>
<tr>
<td>PRNT</td>
<td>L</td>
<td>R0, BLK</td>
<td>Get Count Address</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>R1, R0</td>
<td>Get Count</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>R1, UCB(2)</td>
<td>Store Count</td>
</tr>
<tr>
<td>CHK</td>
<td>LB</td>
<td>R1, TPS(2)</td>
<td>Check if Printer DONE</td>
</tr>
<tr>
<td></td>
<td>BNM</td>
<td>CHK</td>
<td>WAIT if not</td>
</tr>
<tr>
<td></td>
<td>LIB</td>
<td>R1, R0</td>
<td>Get Character</td>
</tr>
<tr>
<td></td>
<td>SB</td>
<td>R1, TPB(2)</td>
<td>Print and turn off DONE</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>R0, UCB+2(2)</td>
<td>Store Text Address</td>
</tr>
<tr>
<td></td>
<td>BR</td>
<td>PROG</td>
<td>Do rest of program</td>
</tr>
<tr>
<td>BLK</td>
<td>DATA</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>TEXT</td>
<td>/EXAMPLE/</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.2 **HIGH SPEED PAPER TAPE READER**

5.2.1 **READER CHARACTERISTICS**

5.2.1.1 General Characteristics

This unit senses perforated paper tape photoelectrically at 300 characters per second using sprocket feet. Tapes may be 5 to 8 channel, oiled, unoiled, or mylar tape with less than 60% transmissivity. Fan fold boxes are provided on the 7 inch front panel with a capacity of 110 feet. The unit is attached to the serial bus through a paper tape reader interface.

5.2.1.2 Codes

Any standard codes may be used as the unit simply transfers the data punched on tape to the interface buffer.
5.2.1.3 Operating Power

Approximately 250 watts of power are required. This may be supplied from either 115V, 47-400 Hz or 230V, 50 Hz single phase AC. The power switch is located on the front panel.

5.2.1.4 Environment

Commercial Lab Environment:

Operating 0 to 55°C ambient, 90% relative humidity, 0-10K feet altitude

Non-Operating -20°C to +85°C ambient, 100% relative humidity without condensation, 0-10K feet altitude

5.2.1.5 Installation

Unit is designed for a 19 inch rack mounting. The environment in the rack must not exceed the limits of 5.2.1.4. Panel height is 7 inches and mounting depth is 8-1/2 inches. The unit weight is 19 pounds.

5.2.1.6 Interconnection

Unit signal and power connections are provided through an accessory plug in the rear of the computer enclosure. The reader is attached by a cable of up to 10 feet in length.

5.2.2 INTERFACE CHARACTERISTICS

The interface between the reader and the computer is mounted on a standard 8 × 11 inch card along with the punch and teletype interfaces. The interface is attached to the serial bus and uses interrupt poll position 3 corresponding to a transfer vector address of 232-235. The poll position and register addresses may be altered by backpanel wiring but should be above the punch and teletype.

5.2.2.1 Interface Registers

The interface contains a reader buffer (PRB) and status register (PRS) with the following formats.

<table>
<thead>
<tr>
<th>Paper Tape Reader Buffer (PRB)</th>
<th>Address FFEA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

7-8
5.2.3 INTERFACE OPERATION

A read is initiated under program control by reading the input buffer. This causes the DONE bit to be reset and the reader to advance one character. When the character has been read the DONE bit is set. The load switch on the front panel, if high, disables the read advance and sets the load bit in the interface. Lowering the load switch enables the reader and resets the load bit in the interface. DONE may not be set after load goes high and is reset by load going high. Lowering the load switch causes the input buffer to be filled and DONE set.

If interrupt is enabled, the low to high transition of DONE or LOAD will cause an internal interrupt request bit to be set. This bit is cleared by reading the status register.

Reading the last character of a record will cause the tape to advance and a new character to be read. If no interrupt is desired, the interrupt enable should be turned off before the last read.

5.2.4 PROGRAMMING EXAMPLE

To read data until a rubout (FF) is encountered and store it in memory without interrupt.

<table>
<thead>
<tr>
<th>Label</th>
<th>Opcode</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEM</td>
<td>EQU</td>
<td>1000</td>
</tr>
<tr>
<td>PRS</td>
<td>EQU</td>
<td>'FFE8</td>
</tr>
<tr>
<td>PRB</td>
<td>EQU</td>
<td>'FFEA</td>
</tr>
</tbody>
</table>
Label | Opcode | Operand | Comments
--- | --- | --- | ---
STRT | L | R0, MEM | : Load Address
READ | LB | R1, PRS(2) | : Get Status
| BNM | READ | : Wait until done set
| LB | R1, PRB(2) | : Get Data, turn off DONE and get NEXT
| CMPB | R1, 'FF | : Check if RUBOUT
DONE | BZ | * | 
| SI | R1, R0 | : Store Data
| BR | READ | : Go to get new character

5.3 **HIGH SPEED PAPER TAPE PUNCH**

5.3.1 **DEVICE CHARACTERISTICS**

5.3.1.1 **General Characteristics**

This unit is capable of perforating standard fanfold tapes at a rate of 75 characters per second. The unit contains its own power supplies and is designed for rack mounting. The punch is attached to the serial I/O bus by an interface located within the computer enclosure.

5.3.1.2 **Tape Characteristics**

Five or eight channel, fanfold, oiled, unoiled, or mylar tapes may be used. They should have a thickness between 3 and 4.3 mils.

5.3.1.3 **Power**

Requires 175 VA of power from either 115V, 47-440 Hz or 230V, 50 Hz single phase AC.

5.3.1.4 **Installation**

The unit is designed for rack mounting in a standard 19 inch rack. Panel height is 10-1/2 inches and depth behind panel is 12 inches. The unit weight is approximately 38 pounds. The unit is mounted on chassis slides for access to load paper tape.

5.3.1.5 **Environment**

Commercial Lab Environment
Temperature: -5°C to +55°C operating ambient
Humidity: 10% to 90% relative humidity without condensation

5.3.1.6 **Operator Controls**

a. **Power On/Off Control**

b. **Tape Feed Control** - momentary switch to cause punching of loader at 75 cps	

5-10
c. Run/Load Control - RUN position enables punch operation LOAD position enables tape to be loaded and causes Tape Error

d. Tape Low Indication - Indicates tape supply nearly exhausted

5.3.1.7 Interconnection

Unit is attached to interface by a ten foot signal and power cable. This mates with an accessory plug at the rear of the computer enclosure.

5.3.2 INTERFACE CHARACTERISTICS

The interface between the punch and the computer is mounted on a standard 8 x 11 inch card along with the reader and teletype interfaces. The interface is attached to the bus and uses interrupt poll position 2 corresponding to a transfer vector address of 240-243. The poll position and register addresses may be altered by backpanel wiring but the poll position should be below the reader and above the teletype.

5.3.2.1 Interface Registers

The interface contains a punch buffer (PPB) and status register (PPS) with the following formats:

Paper Tape Punch Buffer (PPB) - Address FFEE

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>R</td>
<td>A</td>
<td>I</td>
</tr>
</tbody>
</table>

**DONE**
Read only
Indicates that character punched
Set by power up or reset
Cleared by loading PPB

**Interrupt Enable**
Set to zero by power up or reset
Read/Write

**Unit not available**
Read only;
Set to one to indicate punch off line or not physically attached

**ERROR**
Read only
Set to one to indicate tape feed error or run/load switch in load position

5-11
5.3.3 INTERFACE OPERATION

The computer initiates a punch cycle by loading a character into the buffer. This resets the DONE bit which is set again at the end of the punch cycle. The status register should be checked before the write cycle to insure that the punch is available, has no errors and has finished its last cycle.

If the unit is not available or has an error DONE and the punch cycle are inhibited. Data loaded into PPB will be ignored in this condition.

If the interrupt is enabled, an interrupt request bit will be set by either DONE, ERROR, or Unit Not Available going to a one. This request bit will be reset when the status register is read.

5.3.4 PROGRAMMING EXAMPLE

To transfer a word from memory to the punch without interrupt:

<table>
<thead>
<tr>
<th>Label</th>
<th>Opcode</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEM</td>
<td>TEXT</td>
<td>/EXAMPLE/</td>
<td></td>
</tr>
<tr>
<td>PPS</td>
<td>EQU</td>
<td>'FFEC</td>
<td></td>
</tr>
<tr>
<td>PPB</td>
<td>EQU</td>
<td>'FFEE</td>
<td></td>
</tr>
<tr>
<td>STRT</td>
<td>LM</td>
<td>R0, MEM</td>
<td>Get address of data</td>
</tr>
<tr>
<td></td>
<td>MOVL</td>
<td>R1, 7</td>
<td></td>
</tr>
<tr>
<td>PNCH</td>
<td>LMB</td>
<td>R2, PPS(2)</td>
<td>Get status</td>
</tr>
<tr>
<td>BNM</td>
<td>PNCH</td>
<td></td>
<td>Wait until ready</td>
</tr>
<tr>
<td>LI</td>
<td>R2, R0</td>
<td></td>
<td>Get data</td>
</tr>
<tr>
<td>SMB</td>
<td>R2, PPB(2)</td>
<td>Store data and turn off DONE</td>
<td></td>
</tr>
<tr>
<td>BCT</td>
<td>R1, PNCH</td>
<td>Test and branch if more data</td>
<td></td>
</tr>
<tr>
<td>DONE</td>
<td>BR</td>
<td>*</td>
<td>Finished</td>
</tr>
</tbody>
</table>

5.4 REAL TIME CLOCK (RTC)

5.4.1 DEVICE CHARACTERISTICS

This unit provides a way of generating interrupts as a rate set by program control. Four rates may be selected as follows:

a. 1000 Hz
b. 200 Hz
c. 100 Hz
d. 20 Hz

The unit is attached to the serial bus through a real time clock interface.
5.4.2.1 Interface Register

The interface contains a status register with the following format:

Real Time Clock Status Register (RTS)     Address FFE6

```
    7  2  1  0
    D  F  I

DONE
Read Only
Set by end of interval
Cleared by power up, reset,
or reading status register
```

Interrupt enable
Read/write
Set to zero by power up or reset

Frequency
Read/write
Set to zero by power up or reset

00  20 Hz
01  100 Hz
10  200 Hz
11  1000 Hz

5.4.3 INTERFACE OPERATION

The DONE bit will be set at a rate determined by the selected frequency. If interrupt is enabled this will generate an interrupt request. This will be cleared and done reset by reading the status register. DONE must be cleared before the end of the next interval after the one which set it in order not to lose a clock.

5.4.4 PROGRAMMING EXAMPLE

This routine generates a 65 second timer in storage with a 1 ms resolution.

<table>
<thead>
<tr>
<th>Label</th>
<th>Opcode</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME</td>
<td>DATA</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>RTS</td>
<td>EQU</td>
<td>'FFE6</td>
<td></td>
</tr>
<tr>
<td>STRT</td>
<td>MOVL</td>
<td>R0,'06</td>
<td>: Set to 1 KHz, no interrupt</td>
</tr>
<tr>
<td></td>
<td>SMB</td>
<td>R0, RTS(2)</td>
<td></td>
</tr>
<tr>
<td>WAIT</td>
<td>LMB</td>
<td>R0, RTS(2)</td>
<td>: Turn off DONE if set</td>
</tr>
<tr>
<td></td>
<td>BNM</td>
<td>WAIT</td>
<td>: Was DONE set</td>
</tr>
<tr>
<td></td>
<td>LM</td>
<td>R0, TIME(1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADDL</td>
<td>R0, 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SM</td>
<td>R0, TIME(1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BNV</td>
<td>WAIT</td>
<td>: Overflow after 65 seconds</td>
</tr>
<tr>
<td></td>
<td>BR</td>
<td>EXIT</td>
<td></td>
</tr>
</tbody>
</table>
5.5 SERIAL BUS PERIPHERAL ADDRESSES

The following are standard address assignments and poll positions for the devices on the serial bus. Latency times are the time within which the done bit must be reset after being set to ensure maximum speed operation.

<table>
<thead>
<tr>
<th>Device</th>
<th>Register</th>
<th>Address</th>
<th>T.V.</th>
<th>Poll Position</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teletype</td>
<td>TKB</td>
<td>FFF2</td>
<td>248</td>
<td>1</td>
<td>100 ms</td>
</tr>
<tr>
<td>Keyboard/Reader</td>
<td>TKS</td>
<td>FFF0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Teletype Printer/</td>
<td>TPB</td>
<td>FFF6</td>
<td>244</td>
<td>2</td>
<td>100 ms</td>
</tr>
<tr>
<td>Punch</td>
<td>TPS</td>
<td>FFF4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paper Tape Punch</td>
<td>PPB</td>
<td>FFEE</td>
<td>240</td>
<td>3</td>
<td>13.3 ms</td>
</tr>
<tr>
<td></td>
<td>PPS</td>
<td>FFEC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real Time Clock</td>
<td>RTS</td>
<td>FFE6</td>
<td>236</td>
<td>5</td>
<td>1 ms</td>
</tr>
<tr>
<td>Paper Tape Reader</td>
<td>PRB</td>
<td>FFEA</td>
<td>232</td>
<td>4</td>
<td>3 ms</td>
</tr>
<tr>
<td></td>
<td>PRS</td>
<td>FFE8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control Panel</td>
<td>DSR</td>
<td>FFFA</td>
<td>252</td>
<td>0</td>
<td>Indefinite</td>
</tr>
<tr>
<td></td>
<td>DDR</td>
<td>FFFD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DAR</td>
<td>FFFC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SECTION 6
OPERATOR'S CONTROL PANEL

6.0 INTRODUCTION

The MIP-116 Operator's Control Panel (OCP) permits the processor to be controlled from distances of up to fifty feet. Using the panel, data in memory or processor registers can be read or altered. The panel is shown in figure 6-1.

Using the OCP the operator may perform any of the following functions:

a. Read and modify any of the 16 general purpose registers
b. Read and modify PC
c. Read and modify PSR
d. Read and modify any I/O address
e. Inhibit power up or down interrupts
f. Control execution of instructions
g. Control execution of microinstructions
h. Reset system
i. Power ON/OFF control
j. Initiate interrupt

For debugging, these provide the capability of halting program execution, reading and modifying registers or memory, perform single instruction execution, and then resume normal execution. Program start functions include initializing any register or memory location, resetting system, loading PC and PSR and then initiating execution.

The OCP may be used as an I/O device under program control. The programmer may load the data display and read the address register or switch register. The operator may initiate interrupts using the control panel interrupt.

6.1 OCP DISPLAYS

6.1.1 DATA DISPLAY

This consists of 16 binary indicators located above the data switches. An indicator is lit to indicate a binary one.
Figure 6-1 Operators Control Panel.
6.1.2 INDICATORS

6.1.2.1 RUN/HALT

This indicator is lit when the computer is executing instructions.

6.2 OCP CONTROLS

The OCP has six control switches and a display selector switch.

6.2.1 RUN/HALT/SIE SWITCH

This three position controls of the execution of instructions. Pressing the switch up to the RUN position will latch it and cause execution to begin at the address in the program counter. In RUN position the Read/Write and Reset controls are locked out.

Moving the switch to the middle position will cause execution to halt at the end of the instruction being executed. The address of the next instruction will be displayed on the data display.

Depressing the switch into the momentary down position will cause a single instruction to be executed at the address specified by PC. The address of the next instruction will be displayed at the end of the cycle.

6.2.2 SELECTOR SWITCH

This operates in conjunction with the read/write switch to examine/alter registers or memory. It is a six position rotary switch with the following positions:

a. Address
b. Data
c. Register
d. PSR
e. PC
f. IB

The display will be driven by the OCP address register when the switch is in the address position and by the data register in any other position.

6.2.3 WRITE/READ SWITCH

This momentary up/down switch is used in conjunction with the selector switch to examine/alter registers or memory.

6.2.3.1 Address Selector Switch

The OCP address register is displayed by the selector switch. A Read has no effect. A Write transfers the data on the switch register into the control panel address register.
6.2.3.2 Data Selector Switch

A Read will cause the memory location addressed by the OCP address register to be read and loaded into the data register where it will be displayed. The second read with the selector switch in this position after address register load will cause the address register to be incremented by 2 and then the data to be fetched and displayed.

A Write will cause the memory location addressed by the OCP address register to be loaded from the data switches. A second write with the selector switch in this position after a read or loading the address register will cause the address register to be incremented by 2 and the data switches to be transferred to memory.

With the selector switch in this mode the contents of memory may be examined, altered or loaded. Read sequences are used to examine successive locations. Read-Write sequences are used to read and then alter that word in successive locations. Write sequences are used to load memory locations.

Note that memory and parallel and serial I/O device registers may be read or written with this mode. Operations are always performed in word mode.

6.2.3.3 Register Selector Switch

This mode is used to examine, alter or load any of the sixteen general registers. The OCP address register is loaded with addresses 0 through 30 to read or write registers 0 through 16 respectively. Operation from that point is the same as for the Data Selector Switch mode described above.

6.2.3.4 PSR Selector Switch

In this mode, a Read transfers the PSR onto the display and a write loads the PSR from the data switches. The address register is not used and will not count.

6.2.3.5 PC Selector Switch

A Read transfers the PC onto the display and a write loads the PC from the data switches. The address register is not used and will not count.

6.2.3.6 Selector Switch

A Read transfers the instruction addressed by PC onto the display and Write loads the memory location addressed by PC. The OCP address register is not used and will not count.

6.2.4 RESET/LAMP TEST SWITCH

Raising this switch to the momentary up position causes the system to be reset. Depressing the switch to the momentary down position forces all the lights in the display to be lit. Reset is not active if in RUN mode.

6.2.5 CONTROL PANEL INTERRUPT (CPI) SWITCH

Depressing this momentary down switch causes an interrupt to be initiated which may be
serviced under program control. This switch is only active in RUN mode.

6.2.6 POWER UP/DOWN INHIBIT

Raising this switch to the latched up position inhibits the normal power up/down interrupt operation. Removing the inhibit after a power up will not cause a power up sequence.

6.2.7 MICROINSTRUCTION RUN/HALT/STEP

This switch should be normally latched up and is used only for diagnostic purposes. Raising this switch to the latched up position permits microinstructions to be executed by the processor. In the middle position no microinstructions are executed. Depressing the switch to the momentary down position causes the processor to execute one microinstruction. Normal operation of the control panel may only be obtained if this switch is in the latched up position.

6.3 OCP PROGRAMMING

The OCP may be used as an I/O device by the programmer. It is attached to the serial I/O bus at the lowest interrupt poll position. This capability is used by the microroutines to provide the function capability described above.

6.3.1 OCP I/O REGISTERS

6.3.1.1 Address Register

This is a 16 bit counter which may be loaded and incremented by the front panel controls. It is read only and is addressed by address FFEC in word mode.

6.3.1.2 Data Switch Register

This is a 16 bit register loaded from the 16 panel data switches. It is read only and is addressed by FFFA in word mode.

6.3.1.3 Data Display Register

This is a 16 bit register used to drive the display when the selector switch is not in address position. It is write only and is addressed by FFFD in word mode.

6.3.1.4 Function Switch Register

This 8 bit register is used to indicate control panel functions. It is active only when the panel is not in RUN mode and is used only by the microprogram to handle control panel function requests such as read or write. It is read only and is addressed by FFFF in byte mode.

6.3.2 OCP I/O OPERATION

The programmer may load data into the display register or read data from the address or switch register at any time. Control Panel Interrupt sets a serial interrupt request bit which is reset by performing any of the three I/O operations.
In use, the operator will set up data on the data switches and then indicate its availability by hitting CPI. Or the computer may load a control word into the display to indicate its progress in a program. CPI could be used here to indicate operator response.

The transfer vector for CPI is located in positions 252-255.

6.4 OCP INTERCONNECTION

When attached as part of computer enclosure the OCP is wired directly to the backpanel. When operated remotely the OCP is connected to the serial I/O bus (5 pairs) but requires 3 additional twisted pair connections as well. These include:

a. Function Request
b. Microinstruction Control
c. Power Up/Down Inhibit

A total of only 16 wires are required to connect the OCP to the remote processor.
SECTION 7
INTERFACING

7.0 INTRODUCTION

The CPU communicates with all peripheral devices and memories via two busses.

The Parallel Asynchronous I/O Bus is used for instruction fetch and high speed data transfer. This bus utilizes parallel address and data busses to achieve transfer rates of 5 MHz.

The Serial I/O Bus transmits addresses and data serially, two bits in parallel, over bi-directional lines to achieve 250 KHz transfer rates.

Both busses operate in a polling mode to affect priority assignment for interrupts.

One interrupt level is associated with the Serial I/O bus and sixteen devices may be assigned to that level. The Serial I/O interrupt has lowest priority.

The next four higher levels of interrupt are assigned to the Parallel I/O bus. Each of these levels may have eight devices assigned.

Operation of the data transfer and interrupt polling for the Parallel and Serial I/O busses is described in Sections 7.1 and 7.2 respectively.

7.1 PARALLEL ASYNCHRONOUS I/O BUS

7.1.1 GENERAL BUS DESCRIPTION

The Parallel Asynchronous I/O Bus is used for instruction fetch, data fetch, data store, and priority interrupts. Devices and memories of different types, sizes and speeds may be attached to the bus.

Maximum word transfer rate over the bus is limited by the speed of the device and memory, but can be as high as 8 MHz. Words as well as bytes may be transferred. The total addressing capability is 64 K bytes or 32 K words.
External devices may share the bus to effect Direct Memory Access (DMA). This takes place through "cycle stealing" while the processor is operating.

Devices requiring priority interrupts also utilize the Bus for interrupt requests and interrupt device poll.

A block diagram of the Parallel Asynchronous I/O Bus is shown in Figure 7-1.

Devices and memories are connected to the bus on a command-response basis. Each device connected to the bus contains an I/O mode decoder and an address decoder for function and device selection.

Devices which do not require bus control for DMA connect to the I/O Address, I/O Data, I/O Mode, I/O Initiate, and I/O Response lines. Devices which require bus control for DMA also connect to the Bus Request and Bus Grant lines. Devices requiring priority interrupt connect to one of four interrupt lines, the I/O Initiate, I/O Mode and I/O Data lines.

Each of the 43 I/O bus signals is transmitted over a twisted pair, giving a total of 86 connections for the bus including signals and return. External connections are made via connectors at the rear of the enclosure. When external devices are not connected to the bus, termination plugs must be in place at the bus connectors.

The maximum bus length must not exceed five feet total, but to minimize line propagation times it is desirable to place high speed devices as close to the CPU as possible. The Parallel Asynchronous Bus is not designed to be connected between units in a high noise environment.

7.1.2 BUS SIGNAL DESCRIPTIONS

Table 7-1 shows the signals which comprise the Parallel Asynchronous I/O Bus. All signals are active-low logic with the exception of the MODE lines which are a three-bit encoded function.

Table 7-2 describes the code used for the I/O MODE functions.

All signals are transmitted by open-collector gates over twisted pairs having a characteristic impedance of approximately 120 ohms. Signal returns are run through the connectors on each interface card and should be grounded at the ground pin of the circuit which drives or receives the associated signal. Drive capability of each signal and the number of loads presented to the bus by the processor is shown in Table 7-1
Figure 7-1 Parallel Asynchronous I/O Bus.
TABLE 7-1
ASYNCHRONOUS PARALLEL I/O BUS SIGNALS

<table>
<thead>
<tr>
<th>SIGNAL</th>
<th>MNEMONIC</th>
<th>NUMBER OF PAIRS</th>
<th>ACTIVE LOGIC LEVEL</th>
<th>DRIVE STRUCTURE</th>
<th>D-C CHARACTERISTICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O DATA</td>
<td>IOD</td>
<td>16</td>
<td>LOW</td>
<td>OPEN COLLECTOR</td>
<td>0.9  100 µA  16.4  3.6</td>
</tr>
<tr>
<td>I/O ADDRESS</td>
<td>IOA</td>
<td>16</td>
<td>LOW</td>
<td>TOTEM POLE</td>
<td>1.0 - 20.0</td>
</tr>
<tr>
<td>I/O MODE</td>
<td>MODE</td>
<td>3</td>
<td>ENCODED</td>
<td>OPEN COLLECTOR</td>
<td>1.0 - 20.0</td>
</tr>
<tr>
<td>I/O INITIATE</td>
<td>INIT</td>
<td>1</td>
<td>LOW</td>
<td>TOTEM POLE</td>
<td>1.0 - 20.0</td>
</tr>
<tr>
<td>I/O Response</td>
<td>RSPNS</td>
<td>1</td>
<td>LOW</td>
<td>OPEN COLLECTOR</td>
<td>- 50 µA - 2.0</td>
</tr>
<tr>
<td>DMA Request</td>
<td>RQST</td>
<td>1</td>
<td>LOW</td>
<td>OPEN COLLECTOR</td>
<td>- 40 µA - 1.6</td>
</tr>
<tr>
<td>DMA Grant</td>
<td>GRNT</td>
<td>1</td>
<td>LOW</td>
<td>TOTEM POLE</td>
<td>1.0 - 20.0</td>
</tr>
<tr>
<td>INTERRUPT REQUEST</td>
<td>PIOI</td>
<td>4</td>
<td>LOW</td>
<td>OPEN COLLECTOR</td>
<td>- 40 µA - 1.6</td>
</tr>
</tbody>
</table>

*Loading includes effect of external terminations shoes.*
TABLE 7-2
MODE FUNCTIONS

<table>
<thead>
<tr>
<th>MODE 2</th>
<th>MODE 1</th>
<th>MODE φ</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>H</td>
</tr>
</tbody>
</table>

L = Low Voltage Level
H = High Voltage Level

7.1.3 DESCRIPTION OF BUS OPERATIONS

Figures 7-2 through 7-6 are typical timing diagrams for the various bus operations. Table 7-3 lists the minimum, typical and maximum intervals shown in the timing diagrams. Note that all timing diagrams show active-high logic signals, whereas the actual logic sense is as shown in Table 7-1.

7.1.3.1 Data or Instruction Fetch (Figure 7-2)

Data or instruction fetches are begun at the processor cycle-execute time. The processor gates the IOA and MODE lines and then initiates the Fetch by activating the INIT line.

The device may immediately access data since the address is valid. During the access time of the device or memory the higher order address bits are decoded, and after the device data is valid the data will be gated to the IOD lines. The device also activates the RSPNS line, indicating that valid data is on the IOD lines.

The processor receives the response, accepts the data, and releases the INIT line, indicating that the data has been accepted. The device immediately removes data from the IOD lines, resets, and waits for the next INIT command.

Alternatively the device address decoding may be done prior to accessing the device, but the device address decoder will add directly to the effective access time of the device.

For word data fetches the 16 data bits from the IOD bus are loaded into the appropriate register. For byte data fetches the data from the appropriate byte address is loaded into the least significant byte of the designated register.
Figure 7-2 Data or Instruction Fetch.

Figure 7-3 Data Store Timing.

\[ \text{\textbullet T} \text{ DENOTES SIGNAL TRANSMITTED BY PROCESSOR} \]
\[ \text{\textbullet R} \text{ DENOTES SIGNAL RECEIVED BY PROCESSOR} \]
7.1.3.2 Data Store (Figure 7-3)

Data stores may be either byte or word operations depending upon the state of the MODE lines. For word stores, the entire 16-bit word on the IOD bus is stored at the appropriate word address. For byte stores, the least significant eight bits of the IOD bus are stored at the appropriate byte address. The upper eight bits of the IOD bus are ignored.

As with fetches, data stores are begun at processor execute time. The processor gates the IOA, MODE, and IOD lines and then initiates the store operation by activating the INIT line.

The device decodes the most significant bits of the IOA bus, determining whether it is being addressed. The addressed device then loads the data from the IOD bus into a buffer, and activates the RSPNS line, indicating to the processor that data has been accepted. The processor releases the INIT line and continues other operations while the device completes data storage.

Alternatively the device may capture the processor for a maximum of 10 μsec in applications where a data buffer in the device is not desirable. During this period, the processor will wait for the RSPNS line to be activated and will hold the I/O bus stable. At the end of the 10 μsec period the processor will be interrupted through the internal I/O ERROR interrupt.

7.1.3.3 DMA Trap (Figure 7-4)

Devices requiring bus control for DMA are connected in hard-wired priority configuration shown in Figure 7-9 below.

Devices requesting the bus will activate the RQST line, causing the processor to relinquish bus control at the end of its present cycle.

The processor responds by activating the GRNT line and the INIT line which propagates to the requesting device with the highest priority. The MODE lines are in the "Poll level zero" state and the DMA address to be supplied by the requesting device is gated onto the IOD bus. The RSPNS line is activated by the last device in the priority chain, indicating that the DMA address is present on the IOD bus. The INIT signal is removed by the CPU and the DMA address is gated onto the IOA bus by the CPU. The requesting DMA device activates the MODE and IOD lines. The CPU activates the INIT signal causing DMA transfer. The RSPNS signal causes GRNT to be deactivated, ending DMA.

7.1.3.4 Parallel I/O Interrupt (Figure 7-5)

Any device connected to the Parallel I/O bus may cause an interrupt at any of the four parallel I/O interrupt levels. A maximum of eight devices per level are allowed, for a total of 32 parallel I/O interrupts.

The requesting devices are assigned a level of interrupt (0 to 3, with level 3 having highest priority) and a bit on the IOD bus (IOD0 to IOD7 with bit 7 having highest priority).
Figure 7-4  DMA Transfer Timing.
Figure 7-5  Reset Timing.

Figure 7-6  Interrupt Poll Timing.
To cause an interrupt, the requesting device activates the appropriate PIOI line, causing the processor to sense a pending interrupt at the end of the present instruction execution.

The processor will toll the highest requesting level by issuing a POLL LEVEL N (N = 0, 1, 2, 3) command on the MODE lines.

Each requesting device assigned to level N will activate one of the eight least significant bits of the IOD bus, indicating that an interrupt has been requested. Priority selection is made and control is transferred to the interrupting routine.

### TABLE 7-3
**MIP-116 BUS TIMING**

<table>
<thead>
<tr>
<th>Interval</th>
<th>Time (nanoseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
</tr>
<tr>
<td>T&lt;sub&gt;AS&lt;/sub&gt;</td>
<td>Address, Mode, Data Out Setup Time</td>
</tr>
<tr>
<td>T&lt;sub&gt;RD&lt;/sub&gt;</td>
<td>Device Response Delay</td>
</tr>
<tr>
<td>T&lt;sub&gt;DS&lt;/sub&gt;</td>
<td>Processor Data Setup Time</td>
</tr>
<tr>
<td>T&lt;sub&gt;IR&lt;/sub&gt;</td>
<td>Initiate Release Time</td>
</tr>
<tr>
<td>T&lt;sub&gt;MH&lt;/sub&gt;</td>
<td>Mode Hold Time</td>
</tr>
<tr>
<td>T&lt;sub&gt;DR&lt;/sub&gt;</td>
<td>Data Release Time</td>
</tr>
<tr>
<td>T&lt;sub&gt;RR&lt;/sub&gt;</td>
<td>Response Release Time</td>
</tr>
<tr>
<td>T&lt;sub&gt;ID&lt;/sub&gt;</td>
<td>Initiate Dead Time</td>
</tr>
<tr>
<td>T&lt;sub&gt;GD&lt;/sub&gt;</td>
<td>Grant Delay</td>
</tr>
<tr>
<td>T&lt;sub&gt;II&lt;/sub&gt;</td>
<td>Initiate Width</td>
</tr>
<tr>
<td>T&lt;sub&gt;IA&lt;/sub&gt;</td>
<td>Interrupt Acknowledge Time</td>
</tr>
</tbody>
</table>

*Length of interval depends upon device characteristics
**Depends upon instruction being executed

7.1.4 **TYPICAL INTERFACE CONNECTIONS**

Figures 7-7 through 7-13 show typical circuits which may be used to interface to the Parallel Asynchronous I/O Bus.
Figure 7-7 Typical Control Decoder
Figure 7-8  Typical Memory Interface
Figure 7-9   Typical DMA Connections
Figure 7-10  Standard Line Drivers

Figure 7-11  High Speed Line Drivers
I/O BUS SIGNAL

+5V

R1

R2

BUS TERMINATION

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
</tr>
</thead>
<tbody>
<tr>
<td>STANDARD INTERFACE</td>
<td>220</td>
<td>270</td>
</tr>
<tr>
<td>(ONE SN5438 DRIVER)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HIGH SPEED INTERFACE</td>
<td>220</td>
<td>270</td>
</tr>
<tr>
<td>(ONE SN54S40 DRIVER)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7-12  I/O Bus Terminations
Figure 7-13  Typical Interrupt Connections

IOD(N) = I/O DATA LINE ASSIGNED TO DEVICE PRIORITY "N"
PIOI(L) = INTERRUPT REQUEST FOR PRIORITY LEVEL "L"
7.2 SERIAL I/O BUS

7.2.1 GENERAL SERIAL BUS DESCRIPTION

The Serial I/O Bus is designed to permit the processor to communicate with remotely located peripheral devices. Distances of up to 50 feet in extremely high noise environments are possible. High transfer rates are achieved while permitting the use of low power logic devices in the serial interfaces.

A block diagram of the transmission system is shown in figure 7-14. The Serial I/O Bus is composed of 5 signal lines which may be up to 50 feet in length. There are two bi-directional serial data lines designated SIODA and SIODB in the following discussion. Other bus signals are: a uni-directional serial clock line, a uni-directional reset line, and a uni-directional interrupt line. As many as 25 interfaces may be connected to the Serial I/O Bus.

The Serial I/O Bus connections are available at the signal connector of the enclosure. All bus signals must be terminated at the extreme ends of the line, and if the serial bus does not leave the enclosure, a termination plug must be in place at the connector.

All Serial I/O Bus signals are differential balanced signals transmitted over twisted/pair lines. Because the bus is fully synchronous, reliable high-speed operation requires a tradeoff of devices attached to the bus to ensure that the timing constraints of Figures 7-16 and 7-16 and Table 7-5 are satisfied.

7.2.2 BUS SIGNAL DESCRIPTIONS

A list of Serial I/O Bus signals is shown in Table 7-4 indicating whether the CPU drives or receives the signal.

Table 7-5 lists the order of bit transfers over the SIODA and SIODB lines while Figure 7-15 identifies the position of these bits in a CPU register.

<table>
<thead>
<tr>
<th>TABLE 7-4 SERIAL I/O BUS SIGNALS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Signal</strong></td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>Serial Data Line A</td>
</tr>
<tr>
<td>Serial Data Line B</td>
</tr>
<tr>
<td>Serial Clock</td>
</tr>
<tr>
<td>Serial I/O Reset</td>
</tr>
<tr>
<td>Serial I/O Interrupt</td>
</tr>
</tbody>
</table>

7-17
Figure 7-14 Typical Serial I/O Connections.
7.2.3 DESCRIPTION OF BUS OPERATIONS

The two basic types of data transfers which may occur on the serial I/O bus are STORE and LOAD. These operations may be either full word or byte. The first bit-pair transmitted by the CPU on the two serial data lines (SIODA and SIODB) designates the operation:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Operation</th>
<th>Transfer Time MIP-116</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>STORE WORD</td>
<td>2.6 μs</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>STORE BYTE</td>
<td>1.8 μs</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>LOAD WORD</td>
<td>4.4 μs</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>LOAD BYTE</td>
<td>2.8 μs</td>
</tr>
</tbody>
</table>
Figures 7-16 and 7-17 show timing waveforms for load and store operations on the serial I/O bus. Table 7-6 lists the timing specifications for these operations.

All timing specifications are listed as measured at the CPU bus connections.

<table>
<thead>
<tr>
<th>TABLE 7-6 MIP-116 SERIAL I/O BUS TIMING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>T_AS Data - Clock Skew</td>
</tr>
<tr>
<td>T_AH Data Out Hold Time</td>
</tr>
<tr>
<td>T_CW Clock Pulse Width</td>
</tr>
<tr>
<td>T_AP Data Out Clock Period</td>
</tr>
<tr>
<td>T_DS Data Input Setup Time</td>
</tr>
<tr>
<td>T_DH Data Input Hold Time</td>
</tr>
<tr>
<td>T_DP Data Input Clock Period</td>
</tr>
</tbody>
</table>

7.2.3.1 Data Stores

Figure 7-16 shows typical timing for a data store operation. The first bit pair which appears on the SIODA and SIODB lines specifies the mode of operation of the bus, as shown in Table 7-5. The next four bit pairs comprise an 8-bit serial I/O device address. All devices receive these first five bit pairs, clocking the bits into an address buffer using the trailing edge of the SIOCLK signal. An unselected device will then count the data transfer clock, return to an idle state and wait for the next address bits to be transmitted. The four bit-pairs following the address (eight bit pairs in the case of full word transfers) are loaded into the selected device data buffer using the trailing edge of the SIOCLK signal.

7.2.3.2 Data Fetches

Figure 7-17 shows typical timing for a data fetch operation. The mode code and address bits are transmitted exactly as for data store operations. Unlike data stores, the CPU transmits no data on the clock pulse following address transmission, but prepares to receive data. Upon receiving this sixth clock pulse, the selected device puts data on the bus and on the trailing edge of the seventh transmitted clock pulse, the CPU accepts the data present on the bus. The selected device also changes the data on the bus on the trailing edge of the seventh pulse. Three additional clock pulses are transmitted by the processor (seven additional bits for full word transfers), the last bit resetting the selected device.
Figure 7-16 Serial I/O Store Timing.
Figure 7-17  Serial I/O Load Timing.
7.2.3.3 Serial I/O Poll

The interrupt poll operation is similar to a LoadWord operation. A special address signifies that all devices with interrupts pending will respond. Each device requesting an interrupt will transmit a data bit corresponding to one bit of the 16-bit data word. Upon receipt of the first (highest priority data bit) the CPU will transfer control to the appropriate interrupting subroutine.

7.2.3.4 Reset

All devices on the Serial I/O Bus are reset by activation of the SIORST line. This signal is activated by the power up routine, by the RESET instruction or by resetting from the OCP.

7.2.4 TYPICAL INTERFACE CONNECTIONS

Figures 7-18 and 7-19 show typical interfacing circuits for use with the serial I/O bus.
Figure 7-18  Typical Serial I/O Bus Drivers/Receivers/Terminations.
Figure 7-19 Typical Serial I/O Interface.
SECTION 8
SOFTWARE SYSTEMS

8.0 INTRODUCTION

Support software is provided for users of MIP-16 systems on two levels. The Basic Software System (BSS) is for those users primarily interested in development of dedicated application programs. The Basic Operating System (BOS) is provided for those users who require a more general capability. BSS requires a very minimal system while BOS requires a disk and 8K words of storage. These systems are described in detail in their reference manuals.

8.1 BASIC SOFTWARE SYSTEM

The Basic Software System consists of four elements:

a. Time shared Assembly Program (TSAP)
b. Diagnostic Exerciser
c. Bootstrap Loader
d. Debug Program

TSAP operates on a PDP-10 and permits the user to generate absolute programs in time sharing mode using the full facilities of the PDP-10. For use, the programs are loaded via the bootstrap loader.

The exerciser program tests each operational instruction of the processor, halting at a specified location if an error is encountered.

The bootstrap is used to enter programs or data into the processor's memory. It will accept binary objective output from TSAP on either the teletype or high speed reader.

The debug program will store into or type out an address or general purpose register. It will also execute between two specified addresses of the user’s program. Control is always returned to the debug program after each function is executed.

8.2 BASIC OPERATING SYSTEM

The Basic Operating System provides facilities for the user to create programs using a
MIP-16 system alone. The following is included in the BOS:

a. Supervisor
b. Assembler
c. Editor
d. Linking Loader
e. Debugging Editor
f. Subroutine Library

The Supervisor provides facilities for job, task, and data management. The user may initiate execution of programs stored on disk or he can generate programs using the editor, assembler, and loader and store them on disk. Data management provides access methods and peripheral handlers for access of data through peripherals. The subroutine library provides reentrant and relocatable common floating and fixed point routines useful for writing user programs. Included are:

a. Input/Output Format Conversion
b. SIN/COS
c. ATAN
d. LOG
e. EXP
f. SQRT