MCS-8 Microcomputer Set

8008
8 Bit Parallel
Central Processor Unit

USERS MANUAL

Including:
intellinc™ 8
Bare Bones 8
Microcomputer Modules

© Intel Corp. 1973
The MCS-8™ parallel 8-bit microcomputer set is designed for efficient handling of large volumes of data. It has interrupt capability, operates synchronously or asynchronously with external memory, and executes subroutines nested up to seven levels. The 8008 CPU, heart of the MCS-8, replaces 125 TTL packs. With it you can easily address up to 16k 8-bit words of ROM, RAM or shift registers. Using bank switching techniques, you can extend its memory indefinitely.

The PL/M™ High Level Language is an easy-to-learn, systems oriented language derived from IBM's PL/I by Intel for programming the MCS-8 and future 8-bit microcomputers. It gives the microcomputer programmer the same high level language advantages currently available in mini and large computers. By actual tests, PL/M programming and debugging requires less than 10% of the time needed for assembly language. The PL/M compiler is written in Fortran IV for time-share, and needs little or no alteration for most general purpose computers.

Intellec™8 Development Systems provide flexible, inexpensive, and simplified methods for OEM product development. They use RAM for program storage instead of ROM, making program loading and modification easier. The Intellec features are:

- Display and Control Console
- Standard DMA channel
- Standard software package
- Expandable memory and I/O
- TTY interface
- PROM programming capability

The Intellec control panel is used for system monitoring and debugging. These features and the many standard Intellec modules add up to faster turn around and reduced costs for your product development.

And, There's More . . . . .

Intel's Microcomputer Systems Group continues to develop new design aids that make microcomputer system-building easier. They will provide assistance in every phase of your program development.

For additional information:

Microcomputer Systems Group
INTEL Corporation
3065 Bowers Avenue
Santa Clara, California 95051
Phone (408) 246-7501
8008
8 Bit Parallel Central Processor Unit

The 8008 is a complete computer system central processor unit which may be interfaced with memories having capacities up to 16K bytes. The processor communicates over an 8-bit data and address bus and uses two leads for internal control and four leads for external control. The CPU contains an 8-bit parallel arithmetic unit, a dynamic RAM (seven 8-bit data registers and an 8x14 stack), and complete instruction decoding and control logic.

Features

- 8-Bit Parallel CPU on a Single Chip
- 48 Instructions, Data Oriented
- Complete Instruction Decoding and Control Included
- Instruction Cycle Time — 12.5 μs with 8008-1 or 20 μs with 8008
- TTL Compatible (Inputs, Outputs and Clocks)
- Can be used with any type or speed semiconductor memory in any combination
- Directly addresses 16K x 8 bits of memory (RAM, ROM, or S.R.)
- Memory capacity can be indefinitely expanded through bank switching using I/O instructions
- Address stack contains eight 14-bit registers (including program counter) which permit nesting of subroutines up to seven levels
- Contains seven 8-bit registers
- Interrupt Capability
- Packaged in 18-Pin DIP

BLOCK DIAGRAM

PIN CONFIGURATION
From Intel, the people who invented the microcomputer, comes a new, inexpensive and easy way to develop OEM microcomputer systems. The widespread usage of low-cost microcomputers is made possible by Intel's MCS-4 four-bit, and MCS-8 eight-bit, microcomputer sets. To make it easier to use these microcomputer sets, Intel now offers complete 4-bit and 8-bit modular microcomputer development systems called Intellec 4 and Intellec 8. The Intellec modular microcomputers are self-contained expandable systems complete with central processor, memory, I/O, crystal clock, TTY interface, power supplies, standard software, and a control and display console.

The Intellec microcomputer development systems feature:
- 4-bit and 8-bit parallel processor systems
- Program development using RAMs for easier loading and modification
- Standard DMA channel
- Standard software package
- Crystal controlled clocks
- Expandable memory and I/O
- Control panel for system monitoring and program debugging
- PROM programming capability
- Less time and cost for microcomputer systems development

The Intellec 8 is an eight-bit modular microcomputer development system with 5K bytes of memory, expandable to 16K bytes. At the heart of this system is the Intel 8008 CPU chip which has a repertoire of 48 instructions, seven working registers, an eight level address stack, interrupt capability and direct address capability to 16K bytes of memory.

The Intellec 4 is a four-bit modular microcomputer development system with 5K bytes of program memory. At the heart of this system is the Intel 4004 CPU chip with a repertoire of 45 instructions, sixteen working registers, a four level address stack, and the capability of directly addressing over 43K bits of memory.

Standard Microcomputer Modules. The individual modules used to develop the 4-bit and 8-bit microcomputer systems are also available as off-the-shelf microcomputer building blocks. These include 4-bit and 8-bit CPU modules, I/O Modules, PROM Programmer Modules, Data Storage Modules, Control Modules, a Universal OEM Module and other standard modules for expanding the Intellec systems or developing pre-production systems.

With these modules you can tailor the components to your specific microcomputer needs, buying as little or as much as you need to do the job.

Write for complete details on the Intellec modular microcomputer development systems. They will be available in 120 days, but plan now. Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051 (408) 246-7501.

intel® delivers.

Ad Reprint, June 1973

See Appendix VI
# CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. Introduction</td>
<td>3</td>
</tr>
<tr>
<td>II. Processor Timing</td>
<td>4</td>
</tr>
<tr>
<td>A. State Control Coding</td>
<td>4</td>
</tr>
<tr>
<td>B. Timing</td>
<td>4</td>
</tr>
<tr>
<td>C. Cycle Control Coding</td>
<td>5</td>
</tr>
<tr>
<td>III. Basic Functional Blocks</td>
<td>7</td>
</tr>
<tr>
<td>A. Instruction Register and Control</td>
<td>7</td>
</tr>
<tr>
<td>B. Memory</td>
<td>7</td>
</tr>
<tr>
<td>C. Arithmetic/Logic Unit</td>
<td>7</td>
</tr>
<tr>
<td>D. I/O Buffer</td>
<td>7</td>
</tr>
<tr>
<td>IV. Basic Instruction Set</td>
<td>8</td>
</tr>
<tr>
<td>A. Data and Instruction Formats</td>
<td>8</td>
</tr>
<tr>
<td>B. Summary of Processor Instructions</td>
<td>8</td>
</tr>
<tr>
<td>C. Complete Functional Definition</td>
<td>10</td>
</tr>
<tr>
<td>D. Internal Processor Operation</td>
<td>15</td>
</tr>
<tr>
<td>V. Processor Control Signals</td>
<td>18</td>
</tr>
<tr>
<td>A. Interrupt Signal</td>
<td>18</td>
</tr>
<tr>
<td>B. Ready Signal</td>
<td>20</td>
</tr>
<tr>
<td>VI. Electrical Specifications</td>
<td>21</td>
</tr>
<tr>
<td>A. DC and Operating Characteristics</td>
<td>22</td>
</tr>
<tr>
<td>B. AC Characteristics</td>
<td>23</td>
</tr>
<tr>
<td>C. Timing Diagram</td>
<td>23</td>
</tr>
<tr>
<td>D. Typical DC Characteristics</td>
<td>23</td>
</tr>
<tr>
<td>E. Typical AC Characteristics</td>
<td>23</td>
</tr>
<tr>
<td>VII. The SIM8-01 — An MCS-8 Micro Computer</td>
<td>24</td>
</tr>
<tr>
<td>A. SIM8-01 Specifications</td>
<td>25</td>
</tr>
<tr>
<td>B. SIM8-01 Schematic</td>
<td>26</td>
</tr>
<tr>
<td>C. System Description</td>
<td>28</td>
</tr>
<tr>
<td>D. Normal Operation</td>
<td>29</td>
</tr>
<tr>
<td>E. SIM8-01 Pin Description</td>
<td>31</td>
</tr>
<tr>
<td>VIII. MCS-8 PROM Programming System</td>
<td>33</td>
</tr>
<tr>
<td>A. General System Description and Operating Instructions</td>
<td>33</td>
</tr>
<tr>
<td>B. MP7-03 PROM Programmer</td>
<td>39</td>
</tr>
<tr>
<td>C. Programming System Interconnection</td>
<td>40</td>
</tr>
<tr>
<td>IX. Micro Computer Program Development</td>
<td>44</td>
</tr>
<tr>
<td>A. MCS-8 Software Library</td>
<td>44</td>
</tr>
<tr>
<td>B. Development of a Microcomputer System</td>
<td>46</td>
</tr>
<tr>
<td>C. Execution of Programs from RAM on SIM8-01 Using</td>
<td>47</td>
</tr>
<tr>
<td>Memory Loader Control Programs</td>
<td></td>
</tr>
<tr>
<td>X. MCB8-10 Microcomputer Interconnect and Control Module</td>
<td>49</td>
</tr>
<tr>
<td>XI. Appendices</td>
<td>56</td>
</tr>
<tr>
<td>I. SIM8 Hardware Assembler</td>
<td>56</td>
</tr>
<tr>
<td>II. MCS-8 Software Package — Assembler</td>
<td>71</td>
</tr>
<tr>
<td>A. Assembler Specifications</td>
<td>71</td>
</tr>
<tr>
<td>B. Tymshare Users Guide for Assembly</td>
<td>81</td>
</tr>
<tr>
<td>C. General Electric Users Guide for Assembly</td>
<td>81</td>
</tr>
<tr>
<td>D. Sample Program Assembly</td>
<td>82</td>
</tr>
<tr>
<td>III. MCS-8 Software Package — Simulator</td>
<td>84</td>
</tr>
<tr>
<td>A. Introduction</td>
<td>84</td>
</tr>
<tr>
<td>B. Basic Elements</td>
<td>84</td>
</tr>
<tr>
<td>C. INTERP/8 Commands</td>
<td>84</td>
</tr>
<tr>
<td>D. I/O Formatting Commands</td>
<td>88</td>
</tr>
<tr>
<td>E. Error Messages</td>
<td>89</td>
</tr>
<tr>
<td>F. Examples</td>
<td>90</td>
</tr>
<tr>
<td>IV. Teletype Modifications for SIM8-01</td>
<td>95</td>
</tr>
<tr>
<td>V. Programming Examples</td>
<td>95</td>
</tr>
<tr>
<td>A. Sample Program to Search a String of Characters</td>
<td>98</td>
</tr>
<tr>
<td>B. Teletype and Tape Reader Control Program</td>
<td>98</td>
</tr>
<tr>
<td>C. Memory Chip Select Decodes and Output Test Program</td>
<td>99</td>
</tr>
<tr>
<td>D. RAM Test Program</td>
<td>99</td>
</tr>
<tr>
<td>E. Bootstrap Loader Program</td>
<td>100</td>
</tr>
<tr>
<td>VI. Intellec 8, Bare Bones 8, and Microcomputer Modules</td>
<td>102</td>
</tr>
<tr>
<td>A. Sales Offices</td>
<td>124</td>
</tr>
<tr>
<td>B. Distributors</td>
<td>125</td>
</tr>
<tr>
<td>C. Ordering Information/Packaging Information</td>
<td>126</td>
</tr>
</tbody>
</table>

**NOTICE:** The circuits contained herein are suggested applications only. Intel Corporation makes no warranties whatsoever with respect to the completeness, accuracy, patent or copyright status, or applicability of the circuits to a user's requirements. The user is cautioned to check these circuits for applicability to his specific situation prior to use. The user is further cautioned that in the event a patent or copyright claim is made against him as a result of the use of these circuits, Intel shall have no liability to user with respect to any such claim.
I. INTRODUCTION

The 8008 is a single chip MOS 8-bit parallel central processor unit for the MCS-8 micro computer system. A micro computer system is formed when the 8008 is interfaced with any type or speed standard semiconductor memory up to 16K 8-bit words. Examples are INTEL’s 1101, 1103, 2102 (RAMs), 1302, 1602A, 1702A (ROMs), 1404, 2405 (Shift Registers).

The processor communicates over an 8-bit data and address bus (D₀ through D₇) and uses two input leads (READY and INTERRUPT) and four output leads (S₀, S₁, S₂ and Sync) for control. Time multiplexing of the data bus allows control information, 14 bit addresses, and data to be transmitted between the CPU and external memory.

This CPU contains six 8-bit data registers, an 8-bit accumulator, two 8-bit temporary registers, four flag bits, and an 8-bit parallel binary arithmetic unit which implements addition, subtraction, and logical operations. A memory stack containing a 14-bit program counter and seven 14-bit words is used internally to store program and subroutine addresses. The 14-bit address permits the direct addressing of 16K words of memory (any mix of RAM, ROM or S.R.).

The control portion of the chip contains logic to implement a variety of register transfer, arithmetic control, and logical instructions. Most instructions are coded in one byte (8 bits); data immediate instructions use two bytes; jump instructions utilize three bytes. Operating with a 500kHz clock, the 8008 CPU executes non-memory referencing instructions in 20 microseconds. A selected device, the 8008-1, executes non-memory referencing instructions in 12.5 microseconds when operating from an 800kHz clock.

All inputs (including clocks) are TTL compatible and all outputs are low-power TTL compatible.

The instruction set of the 8008 consists of 48 instructions including data manipulation, binary arithmetic, and jump to subroutine.

The normal program flow of the 8008 may be interrupted through the use of the "INTERRUPT" control line. This allows the servicing of slow I/O peripheral devices while also executing the main program.

The "READY" command line synchronizes the 8008 to the memory cycle allowing any type or speed of semiconductor memory to be used.

STATE and SYNC outputs indicate the state of the processor at any time in the instruction cycle.
II. PROCESSOR TIMING

The 8008 is a complete central processing unit intended for use in any arithmetic, control, or decision-making system. The internal organization is centered around an 8-bit internal data bus. All communication within the processor and with external components occurs on this bus in the form of 8-bit bytes of address, instruction or data. (Refer to the accompanying block diagram for the relationship of all of the internal elements of the processor to each other and to the data bus.) For the MCS-8 a logic "1" is defined as a high level and a logic "0" is defined as a low level.

A. State Control Coding

The processor controls the use of the data bus and determines whether it will be sending or receiving data. State signals S₀, S₁, and S₂, along with SYNC inform the peripheral circuitry of the state of the processor. A table of the binary state codes and the designated state names is shown below.

<table>
<thead>
<tr>
<th>S₀</th>
<th>S₁</th>
<th>S₂</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>T1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>T₁₁</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>T₂</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>WAIT</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>T₃</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>STOPPED</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>T₄</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>T₅</td>
</tr>
</tbody>
</table>

B. Timing

Typically, a machine cycle consists of five states, two states in which an address is sent to memory (T₁ and T₂), one for the instruction or data fetch (T₃), and two states for the execution of the instruction (T₄ and T₅). If the processor is used with slow memories, the READY line synchronizes the processor with the memories. When the memories are not available for either sending or receiving data, the processor goes into the WAIT state. The accompanying diagram illustrates the processor activity during a single cycle.

![Figure 1: Basic 8008 Instruction Cycle](image-url)
The receipt of an INTERRUPT is acknowledged by the T11. When the processor has been interrupted, this state replaces T1. A READY is acknowledged by T3. The STOPPED state acknowledges the receipt of a HALT instruction.

Many of the instructions for the 8008 are multi-cycle and do not require the two execution states, T4 and T5. As a result, these states are omitted when they are not needed and the 8008 operates asynchronously with respect to the cycle length. The external state transition is shown below. Note that the WAIT state and the STOPPED may be indefinite in length (each of these states will be 2n clock periods). The use of READY and INTERRUPT with regard to these states will be explained later.

![CPU State Transition Diagram]

**Figure 2. CPU State Transition Diagram**

**C. Cycle Control Coding**

As previously noted, instructions for the 8008 require one, two, or three machine cycles for complete execution. The first cycle is always an instruction fetch cycle (PCI). The second and third cycles are for data reading (PCR), data writing (PCW), or I/O operations (PCC).

The cycle types are coded with two bits, \( D_6 \) and \( D_7 \), and are only present on the data bus during T2.

<table>
<thead>
<tr>
<th>( D_6 )</th>
<th>( D_7 )</th>
<th>CYCLE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>PCI</td>
<td>Designates the address is for a memory read (first byte of instruction).</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>PCR</td>
<td>Designates the address is for a memory read data (additional bytes of instruction or data).</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>PCC</td>
<td>Designates the data as a command I/O operation.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>PCW</td>
<td>Designates the address is for a memory write data.</td>
</tr>
</tbody>
</table>
III. BASIC FUNCTIONAL BLOCKS

The four basic functional blocks of this Intel processor are the instruction register, memory, arithmetic-logic unit, and I/O buffers. They communicate with each other over the internal 8-bit data bus.

A. Instruction Register and Control

The instruction register is the heart of all processor control. Instructions are fetched from memory, stored in the instruction register, and decoded for control of both the memories and the ALU. Since instruction executions do not all require the same number of states, the instruction decoder also controls the state transitions.

B. Memory

Two separate dynamic memories are used in the 8088, the pushdown address stack and a scratch pad. These internal memories are automatically refreshed by each WAIT, T3, and STOPPED state. In the worst case the memories are completely refreshed every eighty clock periods.

1. Address Stack

The address stack contains eight 14-bit registers providing storage for eight lower and six higher order address bits in each register. One register is used as the program counter (storing the effective address) and the other seven permit address storage for nesting of subroutines up to seven levels. The stack automatically stores the content of the program counter upon the execution of a CALL instruction and automatically restores the program counter upon the execution of a RETURN. The CALLs may be nested and the registers of the stack are used as last in/first out pushdown stack. A three-bit address pointer is used to designate the present location of the program counter. When the capacity of the stack is exceeded the address pointer recycles and the content of the lowest level register is destroyed. The program counter is incremented immediately after the lower order address bits are sent out. The higher order address bits are sent out at T2 and then incremented if a carry resulted from T1. The 14-bit program counter provides direct addressing of 16K bytes of memory. Through the use of an I/O instruction for bank switching, memory may be indefinitely expanded.

2. Scratch Pad Memory or Index Registers

The scratch pad contains the accumulator (A register) and six additional 8-bit registers (B, C, D, E, H, L). All arithmetic operations use the accumulator as one of the operands. All registers are independent and may be used for temporary storage. In the case of instructions which require operations with a register in external memory, scratch pad registers H & L provide indirect addressing capability; register L contains the eight lower order bits of address and register H contains the six higher order bits of address (in this case bit 6 and bit 7 are “don’t cares”).

C. Arithmetic/Logic Unit (ALU)

All arithmetic and logical operations (ADD, ADD with carry, SUBTRACT, SUBTRACT with borrow, AND, EXCLUSIVE OR, OR, COMPARE, INCREMENT, DECREMENT) are carried out in the 8-bit parallel arithmetic unit which includes carry-look-ahead logic. Two temporary registers, register “a” and register “b”, are used to store the accumulator and operand for ALU operations. In addition, they are used for temporary address and data storage during intra-processor transfers. Four control bits, carry flip-flop (C), zero flip-flop (Z), sign flip-flop (S), and parity flip-flop (P), are set as the result of each arithmetic and logical operation. These bits provide conditional branching capability through CALL, JUMP, or RETURN on condition instructions. In addition, the carry bit provides the ability to do multiple precision binary arithmetic.

D. I/O Buffer

This buffer is the only link between the processor and the rest of the system. Each of the eight buffers is bi-directional and is under control of the instruction register and state timing. Each of the buffers is low power TTL compatible on the output and TTL compatible on the input.
IV. BASIC INSTRUCTION SET
The following section presents the basic instruction set of the 808.

A. Data and Instruction Formats
Data in the 808 is stored in the form of 8-bit binary integers. All data transfers to the system data bus will be in the same format.

```
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
```

DATA WORD

The program instructions may be one, two, or three bytes in length. Multiple byte instructions must be stored in successive words in program memory. The instruction formats then depend on the particular operation executed.

**One Byte Instructions**
```
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
```

**Two Byte Instructions**
```
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
```

**Three Byte Instructions**
```
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
X  D_5 D_4 D_3 D_2 D_1 D_0
```

**Typical Instructions**
- Register to register, memory reference, I/O arithmetic or logical, rotate or return instructions
- Immediate mode instructions
- Jump or call instructions

*For the third byte of this instruction, D_8 and D_7 are "don't care" bits.

For the MCS-8 a logic "1" is defined as a high level and a logic "0" is defined as a low level.

B. Summary of Processor Instructions

**Index Register Instructions**
The load instructions do not affect the flag flip-flops. The increment and decrement instructions affect all flip-flops except the carry.

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>MINIMUM STATES REQUIRED</th>
<th>INSTRUCTION CODE</th>
<th>DESCRIPTION OF OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lr_1 2</td>
<td>(5)</td>
<td>1 1 D D D S S S</td>
<td>Load index register r_1 with the content of index register r_2.</td>
</tr>
<tr>
<td>Lr M</td>
<td>(8)</td>
<td>1 1 D D D 1 1 1</td>
<td>Load index register r with the content of memory register M.</td>
</tr>
<tr>
<td>LMr</td>
<td>(7)</td>
<td>1 1 1 1 1 S S S</td>
<td>Load memory register M with the content of index register r.</td>
</tr>
<tr>
<td>Lr I</td>
<td>(8)</td>
<td>0 0 D D D 1 1 0</td>
<td>Load index register r with data B . . . B.</td>
</tr>
<tr>
<td>LMI</td>
<td>(9)</td>
<td>0 0 1 1 1 1 1 0</td>
<td>Load memory register M with data B . . . B.</td>
</tr>
<tr>
<td>Ir r</td>
<td>(5)</td>
<td>0 0 D D D 0 0 0</td>
<td>Increment the content of index register r (r ≠ A).</td>
</tr>
<tr>
<td>DCr</td>
<td>(5)</td>
<td>0 0 D D D 0 0 1</td>
<td>Decrement the content of index register r (r ≠ A).</td>
</tr>
</tbody>
</table>

**Accumulator Group Instructions**
The result of the ALU instructions affect all of the flag flip-flops. The rotate instructions affect only the carry flip-flop.

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>MINIMUM STATES REQUIRED</th>
<th>INSTRUCTION CODE</th>
<th>DESCRIPTION OF OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ad r</td>
<td>(5)</td>
<td>1 0 0 0 0 S S S</td>
<td>Add the content of index register r, memory register M, or data B . . . B to the accumulator. An overflow (carry) sets the carry flip-flop.</td>
</tr>
<tr>
<td>Ad M</td>
<td>(8)</td>
<td>1 0 0 0 0 1 1 1</td>
<td>Add the content of index register r, memory register M, or data B . . . B to the accumulator. An overflow (carry) sets the carry flip-flop.</td>
</tr>
<tr>
<td>Ad i</td>
<td>(8)</td>
<td>0 0 0 0 0 1 0 0</td>
<td>Add the content of index register r, memory register M, or data B . . . B to the accumulator with carry. An overflow (carry) sets the carry flip-flop.</td>
</tr>
<tr>
<td>Ac r</td>
<td>(5)</td>
<td>1 0 0 0 1 S S S</td>
<td>Subtract the content of index register r, memory register M, or data B . . . B from the accumulator. An underflow (borrow) sets the carry flip-flop.</td>
</tr>
<tr>
<td>Ac M</td>
<td>(8)</td>
<td>1 0 0 0 1 1 1 1</td>
<td>Subtract the content of index register r, memory register M, or data B . . . B from the accumulator. An underflow (borrow) sets the carry flip-flop.</td>
</tr>
<tr>
<td>Ac i</td>
<td>(8)</td>
<td>0 0 0 0 1 1 1 0</td>
<td>Subtract the content of index register r, memory register M, or data B . . . B from the accumulator with borrow. An underflow (borrow) sets the carry flip-flop.</td>
</tr>
<tr>
<td>MNEMONIC</td>
<td>MINIMUM STATES REQUIRED</td>
<td>INSTRUCTION CODE</td>
<td>DESCRIPTION OF OPERATION</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------</td>
<td>------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>NDn</td>
<td>(5)</td>
<td>D7 D6 1 0 1 0 0 S S S</td>
<td>Compute the logical AND of the content of index register ( r ), memory register ( M ), or data ( B \ldots B ) with the accumulator.</td>
</tr>
<tr>
<td>NDM</td>
<td>(8)</td>
<td>D7 D6 1 0 1 0 0 1 1 1</td>
<td>Compute the EXCLUSIVE OR of the content of index register ( r ), memory register ( M ), or data ( B \ldots B ) with the accumulator.</td>
</tr>
<tr>
<td>NDI</td>
<td>(8)</td>
<td>0 0 1 0 0 1 0 0 B B B B</td>
<td>Compute the INCLUSIVE OR of the content of index register ( r ), memory register ( M ), or data ( B \ldots B ) with the accumulator.</td>
</tr>
<tr>
<td>XRn</td>
<td>(5)</td>
<td>D7 D6 1 0 1 0 1 S S S</td>
<td>Compute the EXCLUSIVE OR of the content of index register ( r ), memory register ( M ), or data ( B \ldots B ) with the accumulator.</td>
</tr>
<tr>
<td>XRM</td>
<td>(8)</td>
<td>D7 D6 1 0 1 0 1 1 1 B B B</td>
<td>Compute the INCLUSIVE OR of the content of index register ( r ), memory register ( M ), or data ( B \ldots B ) with the accumulator.</td>
</tr>
<tr>
<td>XRI</td>
<td>(8)</td>
<td>0 0 1 0 1 1 0 B B</td>
<td>Compute the INCLUSIVE OR of the content of index register ( r ), memory register ( M ), or data ( B \ldots B ) with the accumulator.</td>
</tr>
<tr>
<td>ORn</td>
<td>(5)</td>
<td>D7 D6 1 0 1 1 0 S S S</td>
<td>Compare the content of index register ( r ), memory register ( M ), or data ( B \ldots B ) with the accumulator.</td>
</tr>
<tr>
<td>ORM</td>
<td>(8)</td>
<td>D7 D6 1 0 1 1 1 B B B</td>
<td>Compare the content of index register ( r ), memory register ( M ), or data ( B \ldots B ) with the accumulator.</td>
</tr>
<tr>
<td>ORI</td>
<td>(8)</td>
<td>0 0 1 1 1 1 0 B B</td>
<td>Compare the content of index register ( r ), memory register ( M ), or data ( B \ldots B ) with the accumulator.</td>
</tr>
<tr>
<td>CPI</td>
<td>(8)</td>
<td>D7 D6 0 0 1 1 1 1 1 B B B</td>
<td>Compare the content of index register ( r ), memory register ( M ), or data ( B \ldots B ) with the accumulator.</td>
</tr>
<tr>
<td>RLC</td>
<td>(5)</td>
<td>D7 D6 0 0 0 0 0 0 1 0</td>
<td>Rotate the content of the accumulator left.</td>
</tr>
<tr>
<td>RRC</td>
<td>(5)</td>
<td>D7 D6 0 0 0 1 0 1 0</td>
<td>Rotate the content of the accumulator right.</td>
</tr>
<tr>
<td>RAL</td>
<td>(5)</td>
<td>D7 D6 0 0 1 0 0 1 0</td>
<td>Rotate the content of the accumulator left through the carry.</td>
</tr>
<tr>
<td>RAR</td>
<td>(5)</td>
<td>D7 D6 0 0 0 1 1 0 1</td>
<td>Rotate the content of the accumulator right through the carry.</td>
</tr>
</tbody>
</table>

**Program Counter and Stack Control Instructions**

(4) **JMP**

<table>
<thead>
<tr>
<th>D7 D6 0 1 1 0 X X</th>
<th>1 0 1 0 0 0</th>
<th>Unconditionally jump to memory address ( B_3 \ldots B_{3B_2} \ldots B_2 ).</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 2 2 2 2 2</td>
<td>2 2 2 2 2 2</td>
<td>Unconditionally jump to memory address ( B_3 \ldots B_{3B_2} \ldots B_2 ).</td>
</tr>
<tr>
<td>2 2 2 2 2 2 2 2</td>
<td>2 2 2 2 2 2</td>
<td>Unconditionally jump to memory address ( B_3 \ldots B_{3B_2} \ldots B_2 ).</td>
</tr>
<tr>
<td>X X X X X X X X</td>
<td>-------------------------------------------------</td>
<td></td>
</tr>
</tbody>
</table>

(5) **JFc**

<table>
<thead>
<tr>
<th>D7 D6 0 1 0 0 0</th>
<th>0 0 0 0</th>
<th>Jump to memory address ( B_3 \ldots B_{3B_2} \ldots B_2 ) if the condition flip-flop ( c ) is false. Otherwise, execute the next instruction in sequence.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 2 2 2 2 2</td>
<td>2 2 2 2 2 2</td>
<td>Jump to memory address ( B_3 \ldots B_{3B_2} \ldots B_2 ) if the condition flip-flop ( c ) is false. Otherwise, execute the next instruction in sequence.</td>
</tr>
<tr>
<td>2 2 2 2 2 2 2 2</td>
<td>2 2 2 2 2 2</td>
<td>Jump to memory address ( B_3 \ldots B_{3B_2} \ldots B_2 ) if the condition flip-flop ( c ) is false. Otherwise, execute the next instruction in sequence.</td>
</tr>
<tr>
<td>X X X X X X X X</td>
<td>-------------------------------------------------</td>
<td></td>
</tr>
</tbody>
</table>

(5) **JAc**

<table>
<thead>
<tr>
<th>D7 D6 0 1 1 0 0</th>
<th>0 0 0 0</th>
<th>Jump to memory address ( B_3 \ldots B_{3B_2} \ldots B_2 ) if the condition flip-flop ( c ) is true. Otherwise, execute the next instruction in sequence.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 2 2 2 2 2</td>
<td>2 2 2 2 2 2</td>
<td>Jump to memory address ( B_3 \ldots B_{3B_2} \ldots B_2 ) if the condition flip-flop ( c ) is true. Otherwise, execute the next instruction in sequence.</td>
</tr>
<tr>
<td>2 2 2 2 2 2 2 2</td>
<td>2 2 2 2 2 2</td>
<td>Jump to memory address ( B_3 \ldots B_{3B_2} \ldots B_2 ) if the condition flip-flop ( c ) is true. Otherwise, execute the next instruction in sequence.</td>
</tr>
<tr>
<td>X X X X X X X X</td>
<td>-------------------------------------------------</td>
<td></td>
</tr>
</tbody>
</table>

**Input/Output Instructions**

**INP**

<table>
<thead>
<tr>
<th>D7 D6 0 1 0 0 0 M M</th>
<th>M M M</th>
<th>Read the content of the selected input port (M MM) into the accumulator.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>-------------------------------------------------</td>
<td></td>
</tr>
</tbody>
</table>

**OUT**

<table>
<thead>
<tr>
<th>D7 D6 0 1 0 0 0 0 0</th>
<th>0 0 0 1</th>
<th>Write the content of the accumulator into the selected output port (R R M M M).</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>-------------------------------------------------</td>
<td></td>
</tr>
</tbody>
</table>

**Machine Instruction**

**HLT**

<table>
<thead>
<tr>
<th>D7 D6 0 0 0 0 0 0 0</th>
<th>0 0 0 0 X</th>
<th>Enter the STOPPED state and remain there until interrupted.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 0 0 0 0</td>
<td>-------------------------------------------------</td>
<td></td>
</tr>
</tbody>
</table>

**HLT**

<table>
<thead>
<tr>
<th>D7 D6 0 1 1 1 1 1 1</th>
<th>Enter the STOPPED state and remain there until interrupted.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 1 1 1 1</td>
<td>-------------------------------------------------</td>
</tr>
</tbody>
</table>

**NOTES:**

(1) SSS = Source Index Register

(2) Memory registers are addressed by the contents of registers \( H \) & \( L \).

(3) Additional bytes of instruction are designated by BBBBBBBBB.

(4) X = "Don't Care".

(5) Flag flip-flops are defined by \( C_4 C_3 \): carry (00-overflow or underflow), zero (01-result is zero), sign (10-MSB of result is "1"), parity (11-parity is even).
C. Complete Functional Definition

The following pages present a detailed description of the complete 8008 Instruction Set.

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;B2&gt;)</td>
<td>Second byte of the instruction</td>
</tr>
<tr>
<td>(&lt;B3&gt;)</td>
<td>Third byte of the instruction</td>
</tr>
<tr>
<td>(r)</td>
<td>One of the scratch pad register references: A, B, C, D, E, H, L</td>
</tr>
<tr>
<td>(c)</td>
<td>One of the following flag flip-flop references: C, Z, S, P</td>
</tr>
<tr>
<td>(C_4C_3)</td>
<td>Flag flip-flop codes</td>
</tr>
<tr>
<td>00</td>
<td>carry</td>
</tr>
<tr>
<td>01</td>
<td>zero</td>
</tr>
<tr>
<td>10</td>
<td>sign</td>
</tr>
<tr>
<td>11</td>
<td>parity</td>
</tr>
<tr>
<td>(M)</td>
<td>Memory location indicated by the contents of registers H and L</td>
</tr>
<tr>
<td>(())</td>
<td>Contents of location or register</td>
</tr>
<tr>
<td>(\land)</td>
<td>Logical product</td>
</tr>
<tr>
<td>(\lor)</td>
<td>Exclusive “or”</td>
</tr>
<tr>
<td>(\lor)</td>
<td>Inclusive “or”</td>
</tr>
<tr>
<td>(A_m)</td>
<td>Bit m of the A-register</td>
</tr>
<tr>
<td>STACK</td>
<td>Instruction counter (P) pushdown register</td>
</tr>
<tr>
<td>(P)</td>
<td>Program Counter</td>
</tr>
<tr>
<td>(\rightarrow)</td>
<td>Is transferred to</td>
</tr>
<tr>
<td>XXX</td>
<td>A “don’t care”</td>
</tr>
<tr>
<td>SSS</td>
<td>Source register for data</td>
</tr>
<tr>
<td>DDD</td>
<td>Destination register for data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register #</th>
<th>Register Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>(SSS or DDD)</td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>A</td>
</tr>
<tr>
<td>001</td>
<td>B</td>
</tr>
<tr>
<td>010</td>
<td>C</td>
</tr>
<tr>
<td>011</td>
<td>D</td>
</tr>
<tr>
<td>100</td>
<td>E</td>
</tr>
<tr>
<td>101</td>
<td>H</td>
</tr>
<tr>
<td>110</td>
<td>L</td>
</tr>
</tbody>
</table>
INDEX REGISTER INSTRUCTIONS

LOAD DATA TO INDEX REGISTERS — One Byte
Data may be loaded into or moved between any of the index registers, or memory registers.

\[ \text{Lr}_1 \rightarrow \text{r}_2 \]
(one cycle — PCI)

\[ 11 \text{ DDD SSS} \]
\[ (r_1) \leftrightarrow (r_2) \] Load register \( r_1 \) with the content of \( r_2 \). The content of \( r_2 \) remains unchanged. If SSS=DDD, the instruction is a NOP (no operation).

\[ \text{LrM} \]
(two cycles — PCI/PCR)

\[ 11 \text{ DDD 111} \]
\[ (r) \rightarrow \text{(M)} \] Load register \( r \) with the content of the memory location addressed by the contents of registers H and L. (DDD\#111 — HALT instr.)

\[ \text{LMr} \]
(two cycles — PCI/PCW)

\[ 11 \text{ 111 SSS} \]
\[ (\text{M}) \leftrightarrow (r) \] Load the memory location addressed by the contents of registers H and L with the content of register \( r \). (SSS\#111 — HALT instr.)

LOAD DATA IMMEDIATE — Two Bytes
A byte of data immediately following the instruction may be loaded into the processor or into the memory

\[ \text{LrI} \]
(two cycles — PCI/PCR)

\[ 00 \text{ DDD 110} \]
\[ (r) \leftrightarrow <B_2> \] Load byte two of the instruction into register \( r \).

\[ \text{LMI} \]
(three cycles — PCI/PCR/PCW)

\[ 00 \text{ 111 110} \]
\[ (\text{M}) \leftrightarrow <B_2> \] Load byte two of the instruction into the memory location addressed by the contents of registers H and L.

INCREMENT INDEX REGISTER — One Byte

\[ \text{INr} \]
(one cycle — PCI)

\[ 00 \text{ DDD 000} \]
\[ (r) \leftarrow (r)+1 \]. The content of register \( r \) is incremented by one. All of the condition flip-flops except carry are affected by the result. Note that DDD\#000 (HALT instr.) and DDD\#111 (content of memory may not be incremented).

DECREMENT INDEX REGISTER — One Byte

\[ \text{DCr} \]
(one cycle — PCI)

\[ 00 \text{ DDD 001} \]
\[ (r) \leftarrow (r)-1 \]. The content of register \( r \) is decremented by one. All of the condition flip-flops except carry are affected by the result. Note that DDD\#000 (HALT instr.) and DDD\#111 (content of memory may not be decremented).

ACCUMULATOR GROUP INSTRUCTIONS

Operations are performed and the status flip-flops, C, Z, S, P, are set based on the result of the operation. Logical operations (ANDr, XORr, ORr) set the carry flip-flop to zero. Rotate operations affect only the carry flip-flop. Two's complement subtraction is used.

ALU INDEX REGISTER INSTRUCTIONS — One Byte
(one cycle — PCI)

Index Register operations are carried out between the accumulator and the content of one of the index registers (SSS=000 thru SSS=110). The previous content of register SSS is unchanged by the operation.

\[ \text{ADr} \]

\[ 10 \text{ 000 SSS} \]
\[ (A) \leftarrow (A)+(r) \] Add the content of register \( r \) to the content of register A and place the result into register A.

\[ \text{ACr} \]

\[ 10 \text{ 001 SSS} \]
\[ (A) \leftarrow (A)+(r)+(\text{carry}) \] Add the content of register \( r \) and the contents of the carry flip-flop to the content of the A register and place the result into Register A.

\[ \text{SUr} \]

\[ 10 \text{ 010 SSS} \]
\[ (A) \leftarrow (A)-(r) \] Subtract the content of register \( r \) from the content of register A and place the result into register A. Two's complement subtraction is used.
### ACCUMULATOR GROUP INSTRUCTIONS - Cont'd.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Opcode</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBr</td>
<td>10 011</td>
<td>SSS</td>
<td>(A)←(A)−(r)−(borrow) Subtract the content of register r and the content of the carry flip-flop from the content of register A and place the result into register A.</td>
</tr>
<tr>
<td>NDr</td>
<td>10 100</td>
<td>SSS</td>
<td>(A)←(A)∧(r) Place the logical product of the register A and register r into register A.</td>
</tr>
<tr>
<td>XDr</td>
<td>10 101</td>
<td>SSS</td>
<td>(A)←(A)∨(r) Place the “exclusive - or” of the content of register A and register r into register A.</td>
</tr>
<tr>
<td>ORr</td>
<td>10 110</td>
<td>SSS</td>
<td>(A)←(A)∨(r) Place the “inclusive - or” of the content of register A and register r into register A.</td>
</tr>
<tr>
<td>CPr</td>
<td>10 111</td>
<td>SSS</td>
<td>(A)−(r) Compare the content of register A with the content of register r. The content of register A remains unchanged. The flag flip-flops are set by the result of the subtraction. Equality (A=r) is indicated by the zero flip-flop set to “1”. Less than (A&lt;r) is indicated by the carry flip-flop, set to “1”.</td>
</tr>
</tbody>
</table>

### ALU OPERATIONS WITH MEMORY — One Byte
(two cycles — PCI/PCR)
Arithmetic and logical operations are carried out between the accumulator and the byte of data addressed by the contents of registers H and L.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Opcode</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM</td>
<td>10 000</td>
<td>111</td>
<td>(A)←(A)+(M) ADD</td>
</tr>
<tr>
<td>ACM</td>
<td>10 001</td>
<td>111</td>
<td>(A)←(A)+(M)+(carry) ADD with carry</td>
</tr>
<tr>
<td>SUM</td>
<td>10 010</td>
<td>111</td>
<td>(A)←(A)−(M) SUBTRACT</td>
</tr>
<tr>
<td>SBM</td>
<td>10 011</td>
<td>111</td>
<td>(A)←(A)−(M)−(borrow) SUBTRACT with borrow</td>
</tr>
<tr>
<td>NDM</td>
<td>10 100</td>
<td>111</td>
<td>(A)←(A)∧(M) Logical AND</td>
</tr>
<tr>
<td>XRM</td>
<td>10 101</td>
<td>111</td>
<td>(A)←(A)∨(M) Exclusive OR</td>
</tr>
<tr>
<td>ORM</td>
<td>10 110</td>
<td>111</td>
<td>(A)←(A)∨(M) Inclusive OR</td>
</tr>
<tr>
<td>CPM</td>
<td>10 111</td>
<td>111</td>
<td>(A)−(M) COMPARE</td>
</tr>
</tbody>
</table>

### ALU IMMEDIATE INSTRUCTIONS — Two Bytes
(two cycles — PCI/PCR)
Arithmetic and logical operations are carried out between the accumulator and the byte of data immediately following the instruction.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Opcode</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADI</td>
<td>00 000</td>
<td>100</td>
<td>(A)←(A)+&lt;B₂&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ADD</td>
</tr>
<tr>
<td></td>
<td>00 001</td>
<td>100</td>
<td>(A)←(A)+&lt;B₂&gt;+(carry) ADD with carry</td>
</tr>
<tr>
<td></td>
<td>&lt;B₂&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACI</td>
<td>00 010</td>
<td>100</td>
<td>(A)←(A)&lt;&lt;B₂&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;B₂&gt;</td>
<td></td>
<td>SUBTRACT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUI</td>
<td>00 011</td>
<td>100</td>
<td>(A)←(A)−&lt;B₂&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;B₂&gt;</td>
<td></td>
<td>SUBTRACT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBI</td>
<td>00 100</td>
<td>100</td>
<td>(A)←(A)−&lt;B₂&gt;−(borrow)</td>
</tr>
<tr>
<td></td>
<td>&lt;B₂&gt;</td>
<td></td>
<td>SUBTRACT with borrow</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDI</td>
<td>00 101</td>
<td>100</td>
<td>(A)←(A)∧&lt;B₂&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;B₂&gt;</td>
<td></td>
<td>Logical AND</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XRI</td>
<td>00 110</td>
<td>100</td>
<td>(A)←(A)∨&lt;B₂&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;B₂&gt;</td>
<td></td>
<td>Exclusive OR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORI</td>
<td>00 111</td>
<td>100</td>
<td>(A)←(A)∨&lt;B₂&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;B₂&gt;</td>
<td></td>
<td>Inclusive OR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPI</td>
<td>00 111</td>
<td>100</td>
<td>(A)←&lt;B₂&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;B₂&gt;</td>
<td></td>
<td>COMPARE</td>
</tr>
</tbody>
</table>
ROTATE INSTRUCTIONS — One Byte
(one cycle — PCI)
The accumulator content (register A) may be rotated either right or left, around the carry bit or through the carry bit. Only the carry flip-flop is affected by these instructions; the other flags are unchanged.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Opcode</th>
<th>Shift Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLC</td>
<td>00 000 010</td>
<td>$A_{m+1} \leftarrow A_m$, $A_0 \leftarrow A_7$, (carry) $\leftarrow A_7$</td>
<td>Rotate the content of register A left one bit. Rotate $A_7$ into $A_0$ and into the carry flip-flop.</td>
</tr>
<tr>
<td>RRC</td>
<td>00 001 010</td>
<td>$A_m \leftarrow A_{m+1}$, $A_7 \leftarrow A_0$, (carry) $\leftarrow A_0$</td>
<td>Rotate the content of register A right one bit. Rotate $A_0$ into $A_7$ and into the carry flip-flop.</td>
</tr>
<tr>
<td>RAL</td>
<td>00 010 010</td>
<td>$A_{m+1} \leftarrow A_m$, $A_0 \leftarrow \text{(carry)}$, (carry) $\leftarrow A_7$</td>
<td>Rotate the content of register A left one bit. Rotate the content of the carry flip-flop into $A_0$. Rotate $A_7$ into the carry flip-flop.</td>
</tr>
<tr>
<td>RAR</td>
<td>00 011 010</td>
<td>$A_m \leftarrow A_{m+1}$, $A_7 \leftarrow \text{(carry)}$, (carry) $\leftarrow A_0$</td>
<td>Rotate the content of register A right one bit. Rotate the content of the carry flip-flop into $A_7$. Rotate $A_0$ into the carry flip-flop.</td>
</tr>
</tbody>
</table>

PROGRAM COUNTER AND STACK CONTROL INSTRUCTIONS

JUMP INSTRUCTIONS — Three Bytes
(three cycles — PCI/PCR/PCR)
Normal flow of the microprogram may be altered by jumping to an address specified by bytes two and three of an instruction.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP</td>
<td>01 XXX 000</td>
<td>(P)$\leftarrow B_2 &lt; B_3$</td>
</tr>
<tr>
<td>(Jump Unconditionally)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JFc</td>
<td>01 0C4C3 000</td>
<td>If (c) = 0, (P)$\leftarrow B_3 &lt; B_2$. Otherwise, (P) = (P)+3. If the content of flip-flop c is zero, then jump to the instruction located in memory location $B_3 &lt; B_2$; otherwise, execute the next instruction in sequence.</td>
</tr>
<tr>
<td>(Jump if Condition False)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JTc</td>
<td>01 1C4C3 000</td>
<td>If (c) = 1, (P)$\leftarrow B_2 &lt; B_3$. Otherwise, (P) = (P)+3. If the content of flip-flop c is one, then jump to the instruction located in memory location $B_3 &lt; B_2$; otherwise, execute the next instruction in sequence.</td>
</tr>
<tr>
<td>(Jump if Condition True)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CALL INSTRUCTIONS — Three Bytes
(three cycles — PCI/PCR/PCR)
Subroutines may be called and nested up to seven levels.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAL</td>
<td>01 XXX 110</td>
<td>(Stack)$\leftarrow (P)$, (P)$\leftarrow B_2 &lt; B_3$. Shift the content of P to the pushdown stack. Jump unconditionally to the instruction located in memory location addressed by byte two and byte three.</td>
</tr>
<tr>
<td>(Call subroutine Unconditionally)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CFc</td>
<td>01 0C4C3 010</td>
<td>If (c) = 0, (Stack)$\leftarrow (P)$, (P)$\leftarrow B_3 &lt; B_2$. Otherwise, (P) = (P)+3. If the content of flip-flop c is zero, then shift contents of P to the pushdown stack and jump to the instruction located in memory location $B_3 &lt; B_2$; otherwise, execute the next instruction in sequence.</td>
</tr>
<tr>
<td>(Call subroutine if Condition False)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTc</td>
<td>01 1C4C3 010</td>
<td>If (c) = 1, (Stack)$\leftarrow (P)$, (P)$\leftarrow B_2 &lt; B_3$. Otherwise, (P) = (P)+3. If the content of flip-flop c is one, then shift contents of P to the pushdown stack and jump to the instruction located in memory location $B_3 &lt; B_2$; otherwise, execute the next instruction in sequence.</td>
</tr>
<tr>
<td>(Call subroutine if Condition True)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the above JUMP and CALL instructions $B_2$ contains the least significant half of the address and $B_3$ contains the most significant half of the address. Note that $D_6$ and $D_7$ of $B_3$ are "don't care" bits since the CPU uses fourteen bits of address.
RETURN INSTRUCTIONS — One Byte
(one cycle — PCI)
A return instruction may be used to exit from a subroutine; the stack is popped-up one level at a time.

RET  00  XXX  111  (P)←(Stack). Return to the instruction in the memory location addressed by the last value shifted into the pushdown stack. The stack pops up one level.

RFc  00  0C4C3  011  If (c) = 0, (P)←(Stack); otherwise, (P) = (P)+1.
(Return Condition False)
If the content of flip-flop c is zero, then return to the instruction in the memory location addressed by the last value inserted in the pushdown stack. The stack pops up one level. Otherwise, execute the next instruction in sequence.

RTc  00  1C4C3  011  If (c) = 1, (P)←(Stack); otherwise, (P) = (P)+1.
(Return Condition True)
If the content of flip-flop c is one, then return to the instruction in the memory location addressed by the last value inserted in the pushdown stack. The stack pops up one level. Otherwise, execute the next instruction in sequence.

RESTART INSTRUCTION — One Byte
(one cycle — PCI)
The restart instruction acts as a one byte call on eight specified locations of page 0, the first 256 instruction words.

RST  00  AAA  101  (Stack)←(P),(P)←(000000 00AAA000)
Shift the contents of P to the pushdown stack. The content, AAA, of the instruction register is shifted into bits 3 through 5 of the P-counter. All other bits of the P-counter are set to zero. As a one-word “call”, eight eight-byte subroutines may be accessed in the lower 64 words of memory.

INPUT/OUTPUT INSTRUCTIONS

One Byte
(two cycles — PCI/PCC)
Eight input devices may be referenced by the input instruction

INP  01  00M  MM1  (A)←(input data lines). The content of register A is made available to external equipment at state T1 of the PCC cycle. The content of the instruction register is made available to external equipment at state T2 of the PCC cycle. New data for the accumulator is loaded at T3 of the PCC cycle. MMM denotes input device number. The content of the condition flip-flops, S,Z,P,C, is output on D0, D1, D2, D3 respectively at T4 on the PCC cycle.

Twenty-four output devices may be referenced by the output instruction.

OUT  01  RRM  MM1  (Output data lines)←(A). The content of register A is made available to external equipment at state T1 and the content of the instruction register is made available to external equipment at state T2 of the PCC cycle. RRRMM denotes output device number (RR ≠ 00).

MACHINE INSTRUCTION

HALT INSTRUCTION — One Byte
(one cycle — PCI)

HLT  00  000  00X  or  11  111  111  On receipt of the Halt Instruction, the activity of the processor is immediately suspended in the STOPPED state. The content of all registers and memory is unchanged. The P-counter has been updated and the internal dynamic memories continue to be refreshed.
D. Internal Processor Operation

Internally the processor operates through five different states:

<table>
<thead>
<tr>
<th>Internal State</th>
<th>Typical Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 NORMAL</td>
<td>Send out lower eight bits of address and increment program counter.</td>
</tr>
<tr>
<td>T1 INTERRUPT</td>
<td>Send out lower eight bits of address and suppress incrementing of program counter and acknowledge interrupt.</td>
</tr>
<tr>
<td>T2 WAIT</td>
<td>Send out six higher order bits of address and two control bits, D6 and D7. Increment program counter if there has been a carry from T1.</td>
</tr>
<tr>
<td>T3 NORMAL</td>
<td>Wait for READY signal to come true. Refresh internal dynamic memories while waiting. Fetch and decode instruction; fetch data from memory; output data to memory. Refresh internal memories.</td>
</tr>
<tr>
<td>T3 STOPPED</td>
<td>Remain stopped until INTERRUPT occurs. Refresh internal memories. Execute instruction and appropriately transfer data within processor. Content of data bus transfer is available at I/O bus for convenience in testing. Some cycles do not require these states. In those cases, the states are skipped and the processor goes directly to T1.</td>
</tr>
<tr>
<td>T4 and T5</td>
<td></td>
</tr>
</tbody>
</table>

The 8008 is driven by two non-overlapping clocks. Two clock periods are required for each state of the processor. $\phi_1$ is generally used to precharge all data lines and memories and $\phi_2$ controls all data transfers within the processor. A SYNC signal (divide by two of $\phi_2$) is sent out by the 8008. This signal distinguishes between the two clock periods of each state.

![Processor Clocks](image)

The figure below shows state transitions relative to the internal operation of the processor. As noted in the previous table, the processor skips unnecessary execution steps during any cycle. The state counter within the 8008 operates is a five bit feedback shift register with the feedback path controlled by the instruction being executed. When the processor is either waiting or stopped, it is internally cycling through the T3 state. This state is the only time in the cycle when the internal dynamic memories can be refreshed.

![Transition State Diagram (Internal)](image)

The following pages show the processor activity during each state of the execution of each instruction.
### INDEX REGISTER INSTRUCTIONS

<table>
<thead>
<tr>
<th>D7 D6</th>
<th>D5 D4 D3</th>
<th>D2 D1 D0</th>
<th>OPERATION</th>
<th># OF STATES TO EXECUTE INSTRUCTION</th>
<th>T1(2)</th>
<th>T2</th>
<th>T3</th>
<th>T4(3)</th>
<th>T5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 D D D S S</td>
<td>1 1 1 1 1 1 1</td>
<td>1 1 1 1 1 1 1 0</td>
<td>0 0 0 0 0 0 0 1</td>
<td>0 0 0 0 0 1 1 0</td>
<td>1 1 1 1 1 1 0</td>
<td>0 0 1 1 1 1 1 0</td>
<td>0 0 D D D 0 0 1</td>
<td>0 0 D D D 0 0 1</td>
<td>0 0 D D D 0 0 1</td>
</tr>
</tbody>
</table>

### ACCUMULATOR GROUP INSTRUCTIONS

| 1 0 0 P P S S S  | 1 0 0 P P 1 1 1  | 0 0 0 0 0 1 1 1 0  | 0 0 0 0 0 1 0 0 0 1 0  | 0 0 1 1 1 1 0  | 1 1 1 1 0  | 0 0 1 1 1 1 0  | 1 0 0 P P 1 0 0 0 1 | 1 0 0 P P 1 0 0 1 0 0 0 1 | 1 0 0 P P 1 0 0 1 0 0 0 1 | \( P C L \) OUT | \( P C H \) OUT | \( F E T C H \) INSTR. \( T O \) IR & \( R E G. \) b | SS TO \( R E G. \) b | \( S S S \) TO \( R E G. \) b | \( R E G. \) b TO DDD | ADD OP - FLAGS AFFECTED | SUB OP - FLAGS AFFECTED |

### PROGRAM COUNTER AND STACK CONTROL INSTRUCTIONS

| 0 1 X X X 1 0 0  | 0 1 U C C 0 0 0  | 0 1 1 C C 0 0 0  | 0 1 X X X 1 1 0  | 0 1 0 C C 0 1 0  | 0 1 1 C C 0 1 0  | 0 0 X X X 1 1 1  | 0 0 0 0 0 1 1 1 0  | 0 0 0 0 0 1 1 1 0  | 0 0 0 0 0 1 1 1 0  | \( P C L \) OUT | \( P C H \) OUT | \( F E T C H \) INSTR. \( T O \) IR & \( R E G. \) b | POP STACK | POP STACK | POP STACK | POP STACK |

### I/O INSTRUCTIONS

| 0 1 0 0 M M M 1 | 0 1 R H M M M 1  | \( P C L \) OUT | \( P C H \) OUT | \( F E T C H \) INSTR. \( T O \) IR & \( R E G. \) b | \( P C L \) OUT | \( P C H \) OUT | \( F E T C H \) INSTR. \( T O \) IR & \( R E G. \) b |

### MACHINE INSTRUCTIONS

| 0 0 0 0 0 1 1 X  | 1 1 1 1 1 1 1 1 0  | \( P C L \) OUT | \( P C H \) OUT | \( F E T C H \) INSTR. \( T O \) IR & \( R E G. \) b & HALT | \( P C L \) OUT | \( P C H \) OUT | \( F E T C H \) INSTR. \( T O \) IR & \( R E G. \) b & HALT | \( P C L \) OUT | \( P C H \) OUT |

### NOTES:
1. The first memory cycle is always a \( P C L \) (instruction) cycle.
2. Internally, states are defined as T1 through T5. In some cases, more than one memory cycle is required to execute an instruction.
3. Content of the internal data bus at T4 and T5 is available at the data bus. This is designed for testing purposes only.
4. Lower order address bits in the program counter are denoted by \( P C L \) and higher order bits are designated by \( P C H \).
5. During an instruction fetch, the instruction comes from memory to the instruction register and is decoded.

6. Temporary registers are used internally for arithmetic operations and data transfers (Register a and Register b.)
7. These states are skipped.
8. \( P C R \) cycle (Memory Read Cycle).
9. "X" denotes an idle state.
10. \( P C W \) cycle (Memory Write Cycle).
11. When the JUMP is conditional and the condition fails, states T4 and T5 are skipped and the state counter advances to the next memory cycle.
<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4(3)</th>
<th>T5</th>
</tr>
</thead>
<tbody>
<tr>
<td>REG. L OUT (8)</td>
<td>REG. H OUT</td>
<td>DATA TO REG. b</td>
<td>X (9) REG. b TO DDD</td>
<td></td>
</tr>
<tr>
<td>REG. L OUT (10)</td>
<td>REG. H OUT</td>
<td>REG. b TO OUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC_OUT (8)</td>
<td>PC_OUT</td>
<td>DATA TO REG. b</td>
<td>X REG. b TO DDD</td>
<td></td>
</tr>
<tr>
<td>PC_OUT (8)</td>
<td>PC_OUT</td>
<td>DATA TO REG. b</td>
<td>REG. b TO OUT</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4(3)</th>
<th>T5</th>
</tr>
</thead>
<tbody>
<tr>
<td>REG. L OUT (10)</td>
<td>REG. H OUT</td>
<td>REG. b TO OUT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When the CALL is conditional and the condition fails, states T4 and T5 are skipped and the state counter advances to the next memory cycle. If the condition is true, the stack is pushed at T4, and the lower and higher order address bytes are loaded into the program counter.

13. When the RETURN condition is true, pop up the stack; otherwise, advance to next memory cycle skipping T4 and T5.

14. Bits D_3 through D_0 are loaded into PC_L and all other bits are set to zero; zeros are loaded into PC_H.

15. PC cycle (I/O Cycle).

16. The content of the condition flip-flops is available at the data bus: S at D_0, Z at D_1, P at D_2, C at D_3, Q_0 at D_7 all ones.

17. A READY command must be supplied for the OUT operation to be completed. An idle T3 state is used and then the state counter advances to the next memory cycle.

18. When a HALT command occurs, the CPU internally remains in the T3 state until an INTERRUPT is recognized. Externally, the STOPPED state is indicated.
V. PROCESSOR CONTROL SIGNALS

A. Interrupt Signal (INT)

1) INTERRUPT REQUEST
If the interrupt line is enabled (Logic "1"), the CPU recognizes an interrupt request at the
next instruction fetch (PCI) cycle by outputting $S_0 S_1 S_2 = 011$ at T11 time. The lower
and higher order address bytes of the program counter are sent out, but the program
counter is not advanced. A successive instruction fetch cycle can be used to insert an
arbitrary instruction into the instruction register in the CPU. (If a multi-cycle or multi-
byte instruction is inserted, an interrupt need only be inserted for the first cycle.)

When the processor is interrupted, the system INTERRUPT signal must be synchronized with
the leading edge of the $\phi_1$ or $\phi_2$ clock. To assure proper operation of the system, the interrupt
line to the CPU must not be allowed to change within 200ns of the falling edge of $\phi_1$. An
example of a synchronizing circuit is shown on the schematic for the SIM8-01 (Section VII).

![Figure 4. Recognition of Interrupt](image-url)

If a HALT is inserted, the CPU enters a STOPPED state; if a NOP is inserted, the CPU
continues; if a "JUMP to 0" is inserted, the processor executes program from location 0,
etc. The RESTART instruction is particularly useful for handling interrupt routines since
it is a one byte call.
2) START-UP OF THE 8008

When power ($V_{DD}$) and clocks ($\phi_1, \phi_2$) are first turned on, a flip-flop internal to the 8008 is set by sensing the rise of $V_{DD}$. This internal signal forces a HALT (00000000) into the instruction register and the 8008 is then in the STOPPED state. The following sixteen clock periods after entering the STOPPED state are required to clear (logic “0”) memories (accumulator, scratch pad, program counter, and stack). During this time the interrupt line has been at logic “0”. Any time after the memories are cleared, the 8008 is ready for normal operation.

To reset the flip-flop and also escape from the stopped state, the interrupt line must go to a logic “1”; it should be returned to logic “0” by decoding the state T11 at some time later than $\phi_{11}$. Note that whenever the 8008 is in a T11 state, the program counter is not incremented. As a result, the same address is sent out on two successive cycles.

Three possible sequences for starting the 8008 are shown on the following page. The RESTART instruction is effectively a one cycle call instruction, and it is convenient to use this instruction to call an initiation subroutine. Note that it is not necessary to start the 8008 with a RESTART instruction.

The selection of initiation technique to use depends on the sophistication of the system using the 8008. If the interrupt feature is used only for the start-up of the 8008 use the ROM directly, no additional external logic associated with instructions from other than the ROM program need be considered. If the interrupt feature is used to jam instructions into the 8008, it would then be consistent to use it to jam the initial instruction.

The timing for the interrupt with the start-up timing is shown on an accompanying sheet. The jamming of an instruction and the suppression of the program counter update are handled the same for all interrupts.
EXAMPLE 1:
Shown below are two start-up alternatives where an instruction is not forced into the 8008 during the interrupt cycle. The normal program flow starts the 8008.

a. 8008 ADDRESS OUT INSTRUCTION IN ROM

| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | NOP (LAA 11 000 000) |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | NOP | Entry Directly To Main Program |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 | INSTR₁ |
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 | INSTR₂ |

b. 8008 ADDRESS OUT INSTRUCTION IN ROM

| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | RST (RST = 00 XYZ 101) |
| 0 0 0 0 0 0 0 0 X Y Z 0 0 0 | INSTR₁ |
| 0 0 0 0 0 0 0 0 X Y Z 0 0 1 | INSTR₂ |

EXAMPLE 2:
A RESTART instruction is jammed in and first instruction in ROM initially ignored.

| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | INSTR₁ (RST = 00 XYZ 101) |
| 0 0 0 0 0 0 0 0 X Y Z 0 0 0 | INSTRₐ |
| 0 0 0 0 0 0 0 0 X Y Z 0 0 1 | INSTR₏ |

Note that during the interrupt cycle the flow of the instruction to the 8008 either from ROM or another source must be controlled by hardware external to 8008.

START-UP OF THE 8008

B. Ready (RDY)
The 8008 is designed to operate with any type or speed of semiconductor memory. This flexibility is provided by the READY command line. A high-speed memory will always be ready with data (tie READY line to VCC) almost immediately after the second byte of the address has been sent out. As a result the 8008 will never be required to wait for the memory. On the other hand, with slow ROMs, RAMs or shift registers, the data will not be immediately available; the 8008 must wait until the READY command indicates that the valid memory data is available. As a result any type or any combination of memory types may be used. The READY command line synchronizes the 8008 to the memory cycle. When a program is being developed, the READY signal provides a means of stepping through the program, one cycle at a time.
VI. ELECTRICAL SPECIFICATION

The following pages provide the electrical characteristics for the 8008. All of the inputs are TTL compatible, but input pull-up resistors are recommended to insure proper $V_{IH}$ levels. All outputs are low-power TTL compatible. The transfer of data to and from the data bus is controlled by the CPU. During both the WAIT and STOPPED states the data bus output buffers are disabled and the data bus is floating.

![Figure 6. Data Bus I/O Buffer](image)

![Figure 7. I/O Circuitry](image)
ABSOLUTE MAXIMUM RATINGS*

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PARAMETER</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_{DD}$</td>
<td>AVERAGE SUPPLY CURRENT-OUTS LOADED*</td>
</tr>
<tr>
<td>$I_{IL}$</td>
<td>INPUT LEAKAGE CURRENT</td>
</tr>
<tr>
<td>$V_{IL}$</td>
<td>INPUT LOW VOLTAGE (INCLUDING CLOCKS)</td>
</tr>
<tr>
<td>$V_{IH}$</td>
<td>INPUT HIGH VOLTAGE (INCLUDING CLOCKS)</td>
</tr>
<tr>
<td>$V_{OL}$</td>
<td>OUTPUT LOW VOLTAGE</td>
</tr>
<tr>
<td>$V_{OH}$</td>
<td>OUTPUT HIGH VOLTAGE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LIMITS</th>
<th>MIN.</th>
<th>TYP.</th>
<th>MAX.</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_{DD}$</td>
<td>30</td>
<td>60</td>
<td>mA</td>
<td>$T_A = 25^\circ C$</td>
</tr>
<tr>
<td>$I_{IL}$</td>
<td>10</td>
<td>µA</td>
<td></td>
<td>$V_{IN} = 0V$</td>
</tr>
<tr>
<td>$V_{IL}$</td>
<td>$V_{DD}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$V_{IH}$</td>
<td>$V_{CC} - 4.2$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$V_{OL}$</td>
<td>0.4</td>
<td>V</td>
<td></td>
<td>$I_{OL} = 0.44mA$</td>
</tr>
<tr>
<td>$V_{OH}$</td>
<td>$V_{CC} - 1.5$</td>
<td>V</td>
<td></td>
<td>$I_{OH} = 0.2mA$</td>
</tr>
</tbody>
</table>

*COMMENT
Stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other condition above those indicated in the operational sections of this specification is not implied.

D.C. AND OPERATING CHARACTERISTICS

$T_A = 0^\circ C$ to $70^\circ C$, $V_{CC} = +5V \pm 5\%$, $V_{DD} = -9V \pm 5\%$ unless otherwise specified. Logic “1” is defined as the more positive level ($V_{IH}$, $V_{OH}$). Logic “0” is defined as the more negative level ($V_{IL}$, $V_{OL}$).

A.C. CHARACTERISTICS

$T_A = 0^\circ C$ to $70^\circ C$; $V_{CC} = +5V \pm 5\%$, $V_{DD} = -9V \pm 5\%$. All measurements are referenced to 1.5V levels.

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PARAMETER</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{CY}$</td>
<td>CLOCK PERIOD</td>
</tr>
<tr>
<td>$t_{R,F}$</td>
<td>CLOCK RISE AND FALL TIMES</td>
</tr>
<tr>
<td>$t_{\phi_1}$</td>
<td>PULSE WIDTH OF $\phi_1$</td>
</tr>
<tr>
<td>$t_{\phi_2}$</td>
<td>PULSE WIDTH OF $\phi_2$</td>
</tr>
<tr>
<td>$t_{D1}$</td>
<td>CLOCK DELAY FROM FALLING EDGE OF $\phi_1$ TO FALLING EDGE OF $\phi_2$</td>
</tr>
<tr>
<td>$t_{D2}$</td>
<td>CLOCK DELAY FROM $\phi_2$ TO $\phi_1$</td>
</tr>
<tr>
<td>$t_{D3}$</td>
<td>CLOCK DELAY FROM $\phi_1$ TO $\phi_2$</td>
</tr>
<tr>
<td>$t_{DD}$</td>
<td>DATA OUT DELAY</td>
</tr>
<tr>
<td>$t_{OH}$</td>
<td>HOLD TIME FOR DATA BUS OUT</td>
</tr>
<tr>
<td>$t_{IH}$</td>
<td>HOLD TIME FOR DATA IN</td>
</tr>
<tr>
<td>$t_{SD}$</td>
<td>SYNC OUT DELAY</td>
</tr>
<tr>
<td>$t_{S1}$</td>
<td>STATE OUT DELAY (ALL STATES EXCEPT T1 AND T11) [2]</td>
</tr>
<tr>
<td>$t_{S2}$</td>
<td>STATE OUT DELAY (STATES T1 AND T11)</td>
</tr>
<tr>
<td>$t_{RW}$</td>
<td>PULSE WIDTH OF READY DURING $\phi_{22}$ TO ENTER T3 STATE</td>
</tr>
<tr>
<td>$t_{RD}$</td>
<td>READY DELAY TO ENTER WAIT STATE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LIMITS</th>
<th>MIN.</th>
<th>MAX.</th>
<th>MIN.</th>
<th>MAX.</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{CY}$</td>
<td>2</td>
<td>3</td>
<td>1.25</td>
<td>3</td>
<td>µs</td>
</tr>
<tr>
<td>$t_{R,F}$</td>
<td>50</td>
<td>50</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>$t_{\phi_1}$</td>
<td>0.70</td>
<td></td>
<td>0.35</td>
<td></td>
<td>µs</td>
</tr>
<tr>
<td>$t_{\phi_2}$</td>
<td>0.55</td>
<td></td>
<td>0.35</td>
<td></td>
<td>µs</td>
</tr>
<tr>
<td>$t_{D1}$</td>
<td>0.90</td>
<td>1.1</td>
<td>1.1</td>
<td></td>
<td>µs</td>
</tr>
<tr>
<td>$t_{D2}$</td>
<td>0.40</td>
<td></td>
<td>0.35</td>
<td></td>
<td>µs</td>
</tr>
<tr>
<td>$t_{D3}$</td>
<td>0.20</td>
<td></td>
<td>0.20</td>
<td></td>
<td>µs</td>
</tr>
<tr>
<td>$t_{DD}$</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
<td></td>
<td>µs</td>
</tr>
<tr>
<td>$t_{OH}$</td>
<td>0.10</td>
<td>0.10</td>
<td></td>
<td></td>
<td>µs</td>
</tr>
<tr>
<td>$t_{IH}$</td>
<td>[1]</td>
<td>[1]</td>
<td></td>
<td></td>
<td>µs</td>
</tr>
<tr>
<td>$t_{SD}$</td>
<td>0.70</td>
<td>0.70</td>
<td></td>
<td></td>
<td>µs</td>
</tr>
<tr>
<td>$t_{S1}$</td>
<td>1.1</td>
<td>1.1</td>
<td></td>
<td></td>
<td>µs</td>
</tr>
<tr>
<td>$t_{S2}$</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
<td></td>
<td>µs</td>
</tr>
<tr>
<td>$t_{RW}$</td>
<td>0.35</td>
<td>0.35</td>
<td></td>
<td></td>
<td>µs</td>
</tr>
<tr>
<td>$t_{RD}$</td>
<td>0.20</td>
<td>0.20</td>
<td></td>
<td></td>
<td>µs</td>
</tr>
</tbody>
</table>

*Measurements are made while the 8008 is executing a typical sequence of instructions. The test load is selected such that at $V_{DD} = 0.4V$, $I_{OL} = 0.44mA$ on each output.

[1] $t_{IH} \text{ MIN} \geq t_{SD}$

[2] If the INTERRUPT is not used, all states have the same output delay, $t_{S1}$.
Notes: 1. READY line must be at "0" prior to \( t_{Q2} \) of \( T_2 \) to guarantee entry into the WAIT state.
2. INTERRUPT line must not change levels within 200 ns (max.) of falling edge of \( \phi_1 \).

TYPICAL D.C. CHARACTERISTICS

- **Power Supply Current vs. Temperature**
- **Output Sinking Current vs. Temperature**
- **Output Source Current vs. Output Voltage**

TYPICAL A.C. CHARACTERISTICS

- **Data Out Delay vs. Output Load Capacitance**

CAPACITANCE \( f = 1 \text{MHz}; T_A = 25^\circ \text{C}; \) Unmeasured Pins Grounded

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>TEST</th>
<th>LIMIT (pF)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TYP.</td>
<td>MAX.</td>
</tr>
<tr>
<td>( C_{\text{IN}} )</td>
<td>INPUT CAPACITANCE</td>
<td>5</td>
</tr>
<tr>
<td>( C_{\text{DB}} )</td>
<td>DATA BUS I/O CAPACITANCE</td>
<td>5</td>
</tr>
<tr>
<td>( C_{\text{OUT}} )</td>
<td>OUTPUT CAPACITANCE</td>
<td>5</td>
</tr>
</tbody>
</table>
During the development phase of systems using the 8008, Intel’s single chip 8-bit parallel central processor unit, both hardware and software must be designed. Since many systems will require similar memory and I/O interface to the 8008, Intel has developed a prototyping system, the SIM8-01. Through the use of this system and Intel's programmable and erasable ROMs (1702), MCS-8 systems can be completely developed and checked-out before committing to mask programmed ROMs (1301).

The SIM8-01 is a complete byte-oriented computing system including the processor (8008), 1K x 8 memory (1101), six I/O ports (two in and four out), and a two-phase clock generator. Sockets are provided for 2K x 8 of ROM or PROM memory for the system microprogram. The SIM8-01 may be used with either the 8008 or 8008-1. To operate at clock frequencies greater than 500kHz, former SIM8-01 boards must be modified as detailed in the schematic and the following system description. Note that all Intel-developed 8008 programs interface with TTY and require system operation at 500kHz. Currently, the SIM8-01 is supplied with the 8008-1 CPU and the system clock preset to 500kHz.

The following block diagram shows the basic configuration of the SIM8-01. All interface logic for the 8008 to operate with standard ROM and RAM memory is included on the board. The following pages present the SIM8-01 schematic and detailed system description.

Figure 8. MCS-8 Basic System
SIM8-01 SPECIFICATIONS

Card Dimensions:
- 11.5 inches high
- 9.5 inches deep

System Components Included on Board:
- 8008-1
- Complete TTL interface to memory
- 1K x 8 RAM memory
- Sockets for 2K x 8 PROM memory
- TTY interface ckt.
- Two input and four output ports (8 bits each)
- Two phase clock generator

Maximum Memory Configuration:
- 1K x 8 RAM
- 2K x 8 PROM
- All control lines are provided for memory expansion

Operating Speed
- 2 μs clock period
- 20 μs typical instruction cycle

D.C. Power Requirement:
- Voltage:
  \[ V_{CC} = 5V \pm 5\% \]
  \[ V_{GRD} = 0V \]
  \[ V_{DD} = -9V \pm 5\% \]
- Current:
  Eight ROMs
  \[ I_{CC} = \begin{array}{c}
  \text{Typical} \\
  \text{Maximum}
  \end{array} \quad \begin{array}{c}
  2.5 \text{ amps} \\
  4.0 \text{ amps}
  \end{array} \]
  \[ I_{DD} = \begin{array}{c}
  \text{Typical} \\
  \text{Maximum}
  \end{array} \quad \begin{array}{c}
  1.0 \text{ amps} \\
  1.5 \text{ amps}
  \end{array} \]

Connector:
- Wire wrap type Amphenol 86 pin connector P/N 261-10043-2

---

Figure 9. MCS-8 Memory System
SYSTEM DESCRIPTION

The 8008 processor communicates over an 8-bit data bus (D0 through D7) and uses two input lines (READY and INTERRUPT) and four output lines (S0, S1, S2, and SYNC) for control. Time multiplexing of the data bus allows control information, 14-bit addresses, and data to be transmitted between the CPU, memory, and I/O. All inputs, outputs, and control lines for the SIM8-01 are positive-logic TTL compatible.

Two Phase Clock Generator

The basic system timing for the SIM8-01 is provided by two non-overlapping clock phases generated by 9602 single shot multivibrators (A1, A2). The clocks are factory adjusted as shown in the timing diagram below. Note that this is the maximum specified operating frequency of the 8008. In addition, all Intel-developed TTY programs are synchronized to operate with the SIM8-01 at 500kHz. The clock widths and delays are set in accordance with the 8008-1 specification since an 8008-1 is provided on the board. An option is provided on the board for using external clocks. If the jumper wires in box A are removed, external clocks may be connected at pins J1-52 and J1-12. (Normally these pins are the output of the clock generators on the board.) The clock generator may be adjusted for operation up to 800kHz when using the 8008-1 at maximum speed.

![Diagram of SIM8-01 Timing Diagram]

Figure 11. SIM8-01 Timing Diagram

Memory Organization

The SIM8-01 has capacity for 2K x 8 of ROM or PROM and 1K x 8 of RAM. The memory can easily be expanded to 16K x 8 using the address and chip select control lines provided. Further memory expansion may be accomplished by dedicating an output port to the control of memory bank switching.

In an MCS-8 system, it is possible to use any combination of memory elements. The SIM8-01 is shipped from the factory with the ROM memory designated from address 0 → 2047, RAM memory from 2048→3071, and memory expansion for all addresses 3072 and above. Jumper wires provided on the board (boxes C, D, E) allow complete flexibility of the memory organization. They may be rearranged to meet any requirement. the Intel 3205 data sheet provides a complete description of the one of eight decoder used in this system. the 3205 truth table is shown below.

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>ENABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0 A1 A2</td>
<td>E1 E2 E3</td>
</tr>
<tr>
<td>L L L</td>
<td>L L H</td>
</tr>
<tr>
<td>H L L</td>
<td>L L H</td>
</tr>
<tr>
<td>L H L</td>
<td>L L H</td>
</tr>
<tr>
<td>L L H</td>
<td>L L H</td>
</tr>
<tr>
<td>H H L</td>
<td>L L H</td>
</tr>
<tr>
<td>L L H</td>
<td>L L H</td>
</tr>
<tr>
<td>H L H</td>
<td>L L H</td>
</tr>
<tr>
<td>L H H</td>
<td>L L H</td>
</tr>
<tr>
<td>H H H</td>
<td>L L H</td>
</tr>
<tr>
<td>X X X</td>
<td>L L L</td>
</tr>
<tr>
<td>X X X</td>
<td>L L L</td>
</tr>
<tr>
<td>X X L</td>
<td>L L L</td>
</tr>
<tr>
<td>X X L</td>
<td>L L L</td>
</tr>
<tr>
<td>X X X</td>
<td>L L H</td>
</tr>
<tr>
<td>X X X</td>
<td>L L H</td>
</tr>
<tr>
<td>X X X</td>
<td>L H H</td>
</tr>
<tr>
<td>X X X</td>
<td>L H H</td>
</tr>
</tbody>
</table>

Control Lines

• Interrupt

The interrupt control line is directly available as an input to the board. For manual control, a normally open push-button switch may be connected to terminals J1-50 and J1-53. The interrupt may be inserted
under system control on pin J1-1. An external flip-flop (A33) latches the interrupt and is reset by T11 when the CPU recognizes the interrupt. Instructions inserted under interrupt control may be set up automatically or by toggle switches at the interrupt input port as shown on the schematic. Use the interrupt line and interrupt input port to start up the 8008.

Note that the interrupt line has two different connections to the input to the board (box B). The path from J1-1 directly to pin 4 of package A3 is the normal interrupt path (the board is shipped from the factory with this connection). If the connection from pin 8 of package A15 to pin 4 of package A3 is made instead, the processor will recognize an interrupt only when it is in the STOPPED state. This is used to recognize the "start character" when entering data from TTY.

- Ready

The ready line on the 8008 provides the flexibility for operation with any type of semiconductor memory. On the SIM8-01 board, the ready line is buffered; and at the connector (J1-30), the READY line is active low. During program development, the READY line may be used to step the system through a program.

NORMAL OPERATION OF SYSTEM

The 8008 CPU exercises control over the entire system using its state lines (S0, S1, S2) and two control bits (CC0, CC1) which are sent onto the data bus with the address. The state lines are decoded by a 3205 (A44) and gated with appropriate clock and SYNC signals. The two control bits form part of the control for the multiplexers to the data bus (A55, A56), the memory read/write line (A33) and the I/O line (A17).

In normal operation, the lower order address is sent out of the CPU at state T1, stored in 3404 latches (A59, A72) and provided to all memories. The high order address is sent out at state T2 and stored in 3404 latches (A72, A73). These lines are decoded as the chip selects to the memory. The two highest order bits (CC0, CC1) are decoded for control.

To guarantee that instructions and data are available to the CPU at the proper time, the T3 state is anticipated by setting a D-type flip-flop (A16) at the end of each T2 state. This line controls the multiplexing of data to the 8008. This flip-flop is reset at the end of each T3 state. In addition, switched pull-up resistors are used on the data-bus to minimize data bus loading and increase bus response. The use of switched resistors on the data bus is mandatory when using the 8008-1. SIM8-01 boards built prior to October, 1972 must be modified in order to operate with the 8008-1 at clock frequencies greater than 500kHz.

Normally, the 8008 executes instructions and has no interaction with the rest of the system during states T4 and T5. In the case of the INP instruction, the content of the flag flip-flops internal to the 8008 is sent out at state T4 and stored in a 3404 latch (A43).

Instructions and data are multiplexed onto the 8008 data bus through four multiplexers (A55, A56, A69, A70). In normal operation, line J1-29 should be at +5V in order for "true" data to reach the 8008 data bus.

System I/O

The SIM8-01 communicates with other systems or peripherals through two input ports and four output ports. All control and I/O selection decoding lines are provided for expansion to the full complement of eight input ports and twenty-four output ports. To expand the number of input ports, break the trace at the output of Device A68, pin 11, and generate input port decoding external to the SIM8-01. Control the input multiplexer through pin J1-69. The output ports latch data and remain unchanged until referenced again under software control. Note that all output ports complement data. When power is first applied to the board, the output ports should be cleared under software control to guarantee a known output state. To enable the I/O device decoder, pin J2-8 should be at ground.

Teletype Interface

The 8008 is designed to operate with all types of terminal devices. A typical example of peripheral interface is the teletype (ASR-33). The SIM8-01 contains the three simple transistor TTY interface circuits shown on the following page. One transistor is used for receiving serial data from the teletype, one for transmitting data back to the teletype, and the third for tape reader control.

The teletype must be operating in the full duplex mode. Refer to your teletype operating manual for making connections within the TTY itself. Many models include a nine terminal barrier strip in the rear of
the machine. It is at this point where the connections are made for full duplex operation. The interconnections to the SIM8-01 for transmit and receive are made at this same point.

A complete description of the interconnection of the SIM8-01 and the ASR-33 is presented in Appendix IV.

![Figure 12. Teletype Terminal Strip](image)

![Figure 13. SIM8-01 Teletype Interface Circuitry](image)

To use the teletype tape reader with the SIM8-01, the machine must contain a reader power pack. The contacts of a 10V dc relay must be connected in series with the TTY automatic reader (refer to TTY manual) and the coil is connected to the SIM8-01 tape reader control as shown.

For all Intel developed TTY programs for the SIM8-01, the following I/O port assignments have been made:

1. DATA IN -- INPUT PORT 0, BIT 0 (J2-83 connected to J1-11)
2. DATA OUT -- OUTPUT PORT 2, BIT 0 (J1-84 connected to J2-36)
3. READER CONTROL -- OUTPUT PORT 3, BIT 0 (J2-27 connected to J2-44)

Note that the SIM8-01 clock generator must remain set at 500kHz. All Intel developed TTY programs are synchronized to operate with the SIM8-01 at 500kHz.

In order to sense the start character, data in is also sensed at the interrupt input (J2-83 connected to J1-1) and the interrupt jumper (box B) must be between pin 8 of A15 and pin 4 of A3. It requires approximately 110ms for the teletype to transmit or receive eight serial data bits plus three control bits. The first and last bits are idling bits, the second is the start bit, and the following eight bits are data. Each bit stays 9.09ms. While waiting for data to be transmitted, the 8008 is in the STOPPED state; when the start character is received, the processor is interrupted and forced to call the TTY processing routine. Under software control, the processor can determine the duration of each bit and strobe the character at the proper time.

A listing of a teletype control program is shown in Appendix V.
<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Connector</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 3</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>4, 6</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>7</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>10</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>13</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>16</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>19</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>22</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>25</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>28</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>31</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>34</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>37</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>40</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>43</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>46</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Connector</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>4</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>7</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>10</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>13</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>16</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>19</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>22</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>25</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>28</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>31</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>34</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>37</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>40</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>43</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>46</td>
<td>J2</td>
<td>1</td>
<td>GND</td>
</tr>
</tbody>
</table>
VIII. MCS-8 PROM PROGRAMMING SYSTEM

A. General System Description and Operating Instructions

Intel has developed a low-cost micro computer programming system for its electrically programmable ROMs. Using Intel's eight bit micro computer system and a standard ASR 33 teletype (TTY), a complete low cost and easy to use ROM programming system may be assembled. The system features the following functions:

1) Memory loading
2) Format checking
3) ROM programming
4) Error checking
5) Program listing

For specifications of the Intel PROMs, (1602A/1702A) refer to the Intel Data Catalog.

This programming system has four basic parts:

1) The micro computer (SIM8-01)
   This is the MCS-8 prototype board, a complete micro-computer which uses 1702A PROMs for the microprogram control. The total system is controlled by the 8008 CPU.

2) The control program (A0860, A0861, A0863)
   These control ROMs contain the microprograms which control the bootstrap loading, programming, format and error checking, and listing functions. For programming of Intel's 1702A PROM, use control PROM A0863.

3) The programmer (MP7-03)
   This is the programmer board which contains all of the timing and level shifting required to program the Intel ROMs. This is the successor of the MP7-02.

4) ASR 33 (Automatic Send Receive) Teletype
   This provides both the keyboard and paper tape I/O devices for the programming system.

In addition, a short-wave ultraviolet light is required if the erasable and reprogrammable 1702As are used.

This system has two modes of operation:

1) Automatic — A paper tape is used in conjunction with the tape reader on the teletype. The tape contains the program for the ROM.

2) Manual — The keyboard of the TTY is used to enter the data content of the word to be programmed.
PROGRAMMING THE 1602A/1702A

Information is introduced by selectively programming "1"s (output high) and "0"s (output low) into the proper bit locations. Note that these ROMs are defined in terms of positive logic.

Word address selection is done by the same decoding circuitry used in the READ mode. The eight output terminals are used as data inputs to determine the information pattern in the eight bits of each word. A low data input level (ground – P on tape) will leave a "1" and a high data input level (+48V – N on tape) will allow programming of "0". All eight bits of one word are programmed simultaneously by setting the desired bit information patterns on the data input terminals.

TAPE FORMAT

The tape reader used with a model 33 ASR teletype accepts 1" wide paper tape using 7 or 8 bit ASCII code. For a tape to correctly program a 1602A/1702A, it must follow exactly the format rules below:

![Diagram of tape format]

The format requirements are as follows:

1) There must be exactly 256 word fields in consecutive sequence, starting with word field 0 (all address lines low) to program an entire ROM. If a short tape is needed to program only a portion of the ROM, the same format requirements apply.

2) Each word field must consist of ten consecutive characters, the first of which must be the start character B. Following that start character, there must be exactly eight data characters (P's or N's) and ending with the stop character F. NO OTHER CHARACTERS ARE ALLOWED ANYWHERE IN A WORD FIELD. If an error is made while preparing a tape and the stop character "F" has not been typed, a typed "B" will eliminate the previous characters entered. This is a feature not available on Intel's 7600 programmer; the format shown in the Intel Data Catalog must be used when preparing tapes for other programming systems. An example of this error correcting feature is shown below:

<table>
<thead>
<tr>
<th>TYPED ON TTY</th>
<th>PROGRAMMED IN ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BNNPPNPBNPPNPNF</td>
<td>NPPPNPNPF</td>
</tr>
</tbody>
</table>

If any character other than P or N is entered, a format error is indicated. If the stop character is entered before the error is noticed, the entire word field, including the B and F, must be rubbed out. Within the word field, a P results in a high level output, and N results in a low level output. The first data character corresponds to the desired output for data bit 8 (pin 11), the second for data bit 7 (pin 10), etc.

3) Preceding the first word field and following the last word field, there must be a leader/trailer length of at least 25 characters. This should consist of rubout punches.
4) Between word fields, comments not containing B's or F's may be inserted. It is important that a carriage return and line feed characters be inserted (as a "comment") just before each word field or at least between every four word fields. When these carriage returns are inserted, the tape may be easily listed on the teletype for purposes of error checking. It may also be helpful to insert the word number (as a "comment") at least every four word fields.

## IMPORTANT

It should be noted that the PROM's are described in the data sheet with respect to positive logic (high level = p-logic 1). The MCS-8 system is also defined in terms of positive logic. Consider the instruction code for LHD (one of the 48 instructions for the MCS-8).

1 1 1 0 1 0 1 1

When entering this code to the programmer it should be typed,

B P P P N P N P P F

This is the code that will be put into the 1302, Intel's mask programmed ROM, when the final system is defined.

## OPERATING THE PROGRAMMER

The SIM8-01 is used as the micro computer controller for the programming. The control program performs the function of a bootstrap loader of data from the TTY into the RAM memory. It then presents data and addresses to the PROM to be programmed and controls the programming pulse. The following steps must be followed when programming a PROM:

1) Place control ROMs in SIM8-01
2) Turn on system power
3) Turn on TTY to "line" position
4) Reset system with an INTERRUPT (Instr. RST = 00000 101)
5) Change instruction at interrupt port to a NO OP
6) Start system with an INTERRUPT (Instr NO OP = 11 000 000)
7) Load data from TTY into micro computer memory
8) Insert PROM into MP7-03
9) Program PROM
10) Remove PROM from MP7-03. To prevent programming of unwanted bits, never turn power on or off while the PROM is in the MP7-03.

## LOADING DATA TO THE MICRO COMPUTER (THE BOOTSTRAP LOADER)

The programming system operates in an interactive mode with the user. After resetting and starting the system with an INTERRUPT [steps 4, 5, 6], a "*" will appear on the TTY. This is the signal that the system is ready for a command. To load a data tape, the following sequence must be followed:
This RAM bank may be edited by re-entering blocks of data prior to programming a PROM. More than one RAM bank may be loaded in preparation for programming several different PROMs or to permit the merging of blocks of data from different banks into a single PROM. (See the explanation of the CONTINUE command in section IX.)

FORMAT CHECKING

When the system detects the first format error (data words entered either on tape or manually), it will stop loading data and it will print out the address where the format error occurred.

At this time, an "R" may be typed and the data can be RE-ENTERED manually. This is shown below.

EXAMPLE 1:

```
Listing by TTY

020 B N N P N P N P N P F
021 B P P P P N N N N F
022 B N N N N P P P P N F E
023 B N N P N P N P N P F
024 B P N M F E

Format error indicated at address #022 (too many characters in data field).

R RE-ENTER command
B N N N P P P P F Stop tape reader and manually
RE-ENTER the data word

023 B N P N P N P N P F
024 B P N M F E
025 B P N P N P N P N F

Format error indicated at address #024 (illegal character in data field).

R RE-ENTER command
B P N P N P N P N F RE-ENTER data

continue to completion of data entry.

* Ready for new command
```
PROGRAMMING

After data has been entered, the PROM may be programmed. Data from a designated address field in a designated RAM bank is programmed into corresponding addresses in the PROM. A complete PROM or any portion of a PROM may be programmed in the following manner:

**TYPED BY SYSTEM**
- Ready for command
- Request for RAM BANK #
- Request for address of data field within RAM bank

**TYPED BY USER**
- "P"
- Bn
- A
- xxx
- YYY
- Initial address
- Final address
- Address 0 through 255
- TTY will list data address as each location in PROM is programmed.

- Ready for new command

ERROR CHECKING

After each location in ROM is programmed, the content of the location is read and compared against the programming data. In the event that the programming is not correct, the ROM location will be programmed again. The MCS-8 programming system allows each location of the ROM to be reprogrammed up to four times. A "$" will be printed for each reprogramming. If a location in ROM will not accept a data word after the fourth time, the system will stop programming and a "?" will be printed. This feature of the system guarantees that the programmed ROM will be correct, and incompletely erased or defective ROMs will be identified.

**EXAMPLE 2:**

1st programming
2nd programming
3rd programming

Listed by System
006

failure to program

If a location in the ROM will not program, a new ROM must be inserted in the programmer. The system must be reset before continuing. (If erasable ROMs are being used, the "faulty" ROM should be erased and reprogrammed).

PROGRAM LISTING

Before or after the programming is finished, the complete content of the ROM, or any portion may be listed on the teletype. A duplicated programming tape may also be made using the teletype tape punch. To list the ROM:
The listing feature may also be used to verify that a 1702A is completely erased.

**EXAMPLE 3:**

- **Ready for command** → *T → **DATA ENTRY**
  - Specification of RAM memory address
  - Loading of data listing of tape and verifying correct format

- **Ready for command** → *P → **PROGRAM**
  - Specification of PROM locations to be programmed
  - Programming of PROM and verifying correct transfer of data

- **Ready for command** → *L → **LIST**
  - Address specification
  - Listing of PROM

- **Ready for command** → *
1702A ERASING PROCEDURE

The 1702A may be erased by exposure to high intensity short-wave ultraviolet light at a wavelength of 2537 A. The recommended integrated dose (i.e., UV intensity x exposure time) is 6W-sec/cm². Example of ultraviolet sources which can erase the 1702A in 10 to 20 minutes is the Model S-52 and Model UVS-54 short-wave ultraviolet lamps manufactured by Ultra-Violet Products, Inc. (San Gabriel, California). The lamps should be used without short-wave filters, and the 1702A to be erased should be placed about one inch away from the lamp tubes.

B. MP7-03 PROM Programmer

The MP7-03 is the PROM programming board which easily interfaces with the SIM8-01. All address and data lines are completely TTL compatible. The MP7-03 requires +5VDC @ 0.8 amps, -9 VDC @ 0.1 amps, and 50 Vrms @ 1 amp. Two Stancor P8180 (or equivalent) filament transformers (25.2 Vrms @ 1 amp) with their secondaries connected in series provide the 50 Vrms.

This programmer board is the successor of the MP7-02. The MP7-03 enables programming of Intel’s 1702A, a pin-for-pin replacement for the 1702.

When the MP7-03 is used under SIM8-01 control with control ROM A0862 replaced by A0863, the 1702A may be programmed an order of magnitude faster than the 1702, less than three minutes.

**IMPORTANT:**

Only use the A0863 control PROM when programming the new 1702A. Never use it when programming the 1702. The programming duty cycle is too high for the 1702 and it may be permanently damaged.

The MP7-03 features three data control options:

1) Data-in switch (Normal-Complement). If this switch is in the complement position, data into the PROM is complemented.

2) Data-out switch (Normal-Complement). If this switch is in the complement position, data read from the PROM is complemented.

3) Data-out switch (Enable-Disable). If this switch is in the enable position, data may be read from the PROM. In the disable position, the output line may float up to a high level (logic “1”). As a result, the input ports on the prototype system may be used for other functions without removing the MP7-03 card.

**MP7-03 Programmer Board Specifications**

**Connector:**

a. Solder lug type/Amphenol
   72 pin connector
   P/N 225-23621-101

b. Wire wrap type - Amphenol
   72 pin connector
   P/N 261-15636

**Features:**

- High speed programming of Intel’s 1702A (three minutes)
- Inputs and outputs TTL compatible
- Board sold complete with transformers, capacitor and connector
- Directly interfaces with SIM8-01 Board

**Dimensions:**

- 8.4 inches high
- 9.5 inches deep

**Power Requirement:**

- $V_{CC} = +5 \, @ \, 0.8 \, \text{amps}$
- $\text{TTL GRD} = 0V$
- $V_{DD} = -9 \, \text{V} \, @ \, 0.1 \, \text{amps}$
- $V_P = 50 \text{Vrms} @ 1 \, \text{amp}$

*This board may be used with a $-10V$ supply because a pair of diodes (i.e., 1N914 or equivalent) are located on the board in series with the supply. Select the appropriate pin for either $-9V$ or $-10V$ operation.

A micro computer bulletin which describes the modification of the MP7-02 for programming the 1602A/1702A is available on request. These modifications include complete failsafe circuitry (now on MP7-03) to protect the PROMs and the 50V power supply.
C. Programming System Interconnection

NOTES:
1. SIM8-02 Connector:
   a. Solder lug type/Amphenol
   b. Wire wrap type/Amphenol (shown above)
   80 pin connector P/N 261-15636-2.
2. MP7-02 Connectors:
   a. Solder lug type/Amphenol
   b. Wire wrap type/Amphenol (shown above)
   72 pin connector P/N 261-15636-2.
3. If the use of the 24 pin socket on the MP7-03 is not desired, the pin connections for external socket are as follows:

EXTERNAL SOCKET PROGRAMMING

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>MP7-03 PIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>A₂ &quot;OUT&quot; DEVICE UNDER TEST</td>
<td>56</td>
</tr>
<tr>
<td>A₁</td>
<td>58</td>
</tr>
<tr>
<td>A₀</td>
<td>60</td>
</tr>
<tr>
<td>A₃</td>
<td>62</td>
</tr>
<tr>
<td>A₄</td>
<td>64</td>
</tr>
<tr>
<td>A₅</td>
<td>66</td>
</tr>
<tr>
<td>A₆</td>
<td>68</td>
</tr>
<tr>
<td>A₇</td>
<td>70</td>
</tr>
<tr>
<td>D₁ &quot;OUT&quot; DEVICE UNDER TEST</td>
<td>71</td>
</tr>
<tr>
<td>D₂</td>
<td>69</td>
</tr>
<tr>
<td>D₃</td>
<td>67</td>
</tr>
<tr>
<td>D₄</td>
<td>65</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>MP7-03 PIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>D₁</td>
<td>63</td>
</tr>
<tr>
<td>D₂</td>
<td>61</td>
</tr>
<tr>
<td>D₃</td>
<td>59</td>
</tr>
<tr>
<td>D₄</td>
<td>57</td>
</tr>
<tr>
<td>CHIP SELECT OUT</td>
<td>72</td>
</tr>
<tr>
<td>PROGRAM OUT</td>
<td>22</td>
</tr>
<tr>
<td>V₉ OUT</td>
<td>2,4</td>
</tr>
<tr>
<td>V₀₂ OUT</td>
<td>26</td>
</tr>
<tr>
<td>V₀₃ OUT</td>
<td>24</td>
</tr>
<tr>
<td>V₀₄ OUT</td>
<td>30</td>
</tr>
<tr>
<td>±1, ±2 OUT</td>
<td>2</td>
</tr>
</tbody>
</table>

The complete interconnection between the SIM8-01 and the MP7-03 is provided by the MC88-10 system interface and control module. See the MC88-10 description.

Figure 16. MP7-03/Sim8-01 PROM Programming System
Figure 17a. Component Side of MP7-03 Card

Figure 17b. Pin Definition — Reverse Side of MP7-03 Card
Figure 18. MP7-03 PROM Programmer Board Schematic

NOTE:
THIS SCHEMATIC IS INCLUDED FOR REFERENCE ONLY.
IX. MICROCOMPUTER PROGRAM DEVELOPMENT

A. MCS-8 Software Library

1.0 PL/M™ COMPILER — A High Level Systems Language

It's easy to program the MCS-8 Microcomputer using PL/M, a new high level language concept developed to meet the special needs of microcomputer systems programming. Programmers can now utilize a true high level language to efficiently program microcomputers. PL/M is an assembly language replacement that can fully command the 8008 CPU and future processors to produce efficient run-time object code. PL/M was designed to provide additional development software support for the MCS-8 microcomputer system, permitting the programmer to concentrate more on his problem and less on the actual task of programming than is possible with assembly language.

Programming time and costs are drastically reduced, and training, documentation and program maintenance are simplified. User application programs and standard systems programs may be transferred to future computer systems that support PL/M with little or no reprogramming. These are advantages of high-level language programming that have been proven in the large computer field and are now available to the microcomputer user.

PL/M is derived from IBM's PL/I, a very extensive and sophisticated language which promises to become the most widely known and used language in the near future. PL/M is designed with emphasis on those features that accurately reflect the nature of systems programming requirements for the MCS-8 microcomputer system.

PL/M Coding

Program Development Time: 15 minutes

PL/M vs ASSEMBLY LANGUAGE

As an example of comparative programming effort between PL/M and assembly language, this program to computer prime numbers was written twice, first in PL/M, and then in assembly language. The PL/M version was written in fifteen minutes, compiled correctly on the second try (an "end" was omitted the first time) and ran correctly the first time. The program was then coded in Intel MCS-8 assembly language. Coding took four hours, program entry and editing another two hours, debug took an hour to find incorrect register designation, the kind of problem completely eliminated by coding in PL/M. Results of this one short test shows a 28 to 1 reduction in coding time. This ratio may be somewhat high, overall ratio in a mix of programs is more on the order of 10 to 1.

PL/M Is An Efficient Language

Tests on sample programs indicate that a PL/M program can be written in less than 10% of the time it takes to write the same program in assembly language with little efficiency loss. The main reason for this savings in time is the fact that PL/M allows the programmer to define his problem in terms natural to him, not in the computer's terms. Consider the following sample program which selects the largest of two numbers. In PL/M, the programmer might write:

If A > B, then C = A; else C = B;

Meaning: "If variable A is greater than variable B, then assign A to variable C; otherwise, assign B to C."

Assembly Coding

Program Development Time: 7 hours
A corresponding program in assembly language is twelve separate machine instructions, and conveys little of original intent of the program.

Because of the ease and conciseness with which programs can be written and the error free translation into machine language achieved by the compiler, the time to program a given system is reduced substantially over assembly language.

Debug and checkout time of a PL/M program is also much less than that of an assembly language program, partly because of the inherent clarity of PL/M, but also because writing a program in PL/M encourages good programming techniques. Furthermore, the structure of the PL/M language enables the PL/M compiler to detect error conditions that would slip by an assembler. The PL/M compiler is written in ANSI FORTRAN IV and thus will execute on most large scale machines with little alteration.

2.0 MCS-8 CROSS ASSEMBLER SOFTWARE PACKAGE

The MCS-8 cross assembler translates a symbolic representation of the instructions and data into a form which can be loaded and executed by the MCS-8. By cross assembler, we mean an assembler executing on a machine other than the MCS-8, which generates code for the MCS-8. Initial development time can be significantly reduced by taking advantage of a large scale computer's processing, editing and high speed peripheral capability. Programs are written in the assembly language using mnemonic symbols both for 8008 instruction and for special assembler operations. Symbolic addresses can be used in the source program; however, the assembled program will use absolute address. (See Appendix II.)

The Assembler is designed to operate from a time shared terminal. The assembled program may be punched out at the terminal in BNPF format.

The Assembler is written in FORTRAN IV and is designed to run on a PDP-10. Modifications to the program may be required for machines other than PDP-10.

3.0 MCS-8 SIMULATOR SOFTWARE PACKAGE

The MCS-8 Simulator is a computer program written in FORTRAN IV language and called INTERP/8. This program provides a software simulation of the Intel 8008 CPU, along with execution monitoring commands to aid program development for the MCS-8.

INTERP/8 accepts machine code produced by the 8008 Assembler, along with execution commands from a time sharing terminal, card reader, or disk file. The execution commands allow manipulation of the simulated MCS-8 memory and the 8008 CPU registers. In addition, operand and instruction breakpoints may be set to stop execution at crucial points in the program. Tracing features are also available which allow the CPU operation to be monitored. INTERP/8 also accepts symbol tables from either the PL/M compiler or MCS-8 cross assembler to allow debugging, tracing and braking, and displaying of program using symbolic names.

The PL/M compiler, MCS-8 assembler, and MCS-8 simulator software packages may be procured from Intel on magnetic tape. Alternatively, designers may contact several nation-wide computer time sharing services for access to the programs.

4.0 BOOTSTRAP LOADER FOR SIM8-01

When developing MCS-8 software using the SIM8-01, programs may be loaded, stored, and executed directly from RAM memory. A set of three 1702A control PROMs (1702A/860 set) is required for this function. In addition, this same control PROM set is required when the SIM8-01 is used as the controller for PROM programming. (See Appendix V.)

5.0 SIM8 HARDWARE ASSEMBLER

The SIM8 Hardware Assembler is a program which translates a symbolic assembly language into an octal representation of the SIM8 machine language. An auxiliary program then translates the octal object code into the "BNPF" format suitable for bootstrap loading or PROM programming. Eight PROMs and three tapes (1702A/840 set) containing the assembly program plug into the SIM8-01 prototyping board permitting assembly of all MCS-8 software when used with an ASR 33 teletype.

The assembler accepts the source text from the paper tape reader on the first of two passes and constructs a name table. On a second pass the assembler translates the source using the previously determined name values, creates an octal object paper tape, and if directed, writes the object code into Read/Write memory.

The assembler's commands allow for TTY keyboard manipulation of R/W memory and execution of stored programs so that program debugging may be undertaken directly after assembly. If a "BNPF" tape is desired, an auxiliary "tape generator" program may be loaded and executed by the assembler. (See Appendix I.)
6.0 PROGRAM LIBRARY

These program listings are available to all Intel microcomputer users. We encourage all users to submit all non-proprietary programs to Intel to add to the program library so that we may make them available to other users.

- MCS-8 bootstrap loader and control program and PROM programming systems routine for the SIM8-01 and SIM8-01/MP7-03 PROM programming system (A0860, A0861, A0863)[1].
- Floating point multiply routine for the MCS-8.
- Fixed point multiply routine for the MCS-8.
- Fast Fourier transform program for the MCS-8 using the algorithm by G.D. Berglund (see IEEE Transactions on Computers, April, 1972).
- Debug Program
- Binary Search Routine
- Interrupt Service Routine
- Analog to digital controller — MCS-8.
- MCS-8 driving an incremental X-Y plotter such as those manufactured by CALCOMP.
- Three dimensional blackboard stroke generator using MCS-8.
- MCS-8 program for saving CPU status on an interrupt.
- MCS-8 program for controlling the timing for a serial input from a teletype.
- Fast Fourier transform program for the MCS-8.
- MCS-8 Assembler for use on HP 2100
- MCS-8 teletype and tape reader control program (A0800)[1].
- MCS-8 memory chip select decode and output test program for the SIM8-01 card (A0801)[1].
- MCS-8 RAM test program for the SIM8-01 card (A0802)[1].
- Single precision multiply/divide.
- Program written by Intel. • Program submitted by customers.

Note 1. These are the program numbers that should be used when ordering the programs in PROMs.

B. Development of a Microcomputer System

The flowchart shows the steps required for the development of a microcomputer system. The SIM8-01 system can be used throughout the complete cycle for program assembly, PROM programming, and prototype system hardware. Ultimately, custom systems using 1702A PROMs may be delivered. For high volume applications (100 or more identical systems) lower cost metal masked ROMs may be used.

To combine the advantages of the metal masked ROM and the PROMs, subroutines may be stored in metal masked ROMs and a customized main program may be stored in PROM.
C. Execution of Programs from RAM on SIM8-01 Using Memory Loader Control Programs

The previous section provided a description of the preparation of tapes and the programming of PROMs for permanently storing the microcomputer programs. During the system development, programs may be loaded, stored, and executed directly from RAM memory. This section explains these additional features.

![Diagram of SIM8-01 Memory Organization](image)

**Figure 19. MCS-8 Operating System**

The system has three basic parts:
1. The microcomputer (SIM8-01)
2. The bootstrap memory loader control program (A0860, A0861, A0863)
3. ASR 33 (Automatic Send Receive) Teletype

The control program provides the complete capability for executing programs from RAM. Two additional program commands are required: "C", the CONTINUE command for loading more than one bank of memory, and "E", the program EXECUTION command.

**Operating The Microcomputer System**

To use the SIM8-01 as the microcomputer controller for the bootstrap loading of a program from the TTY into RAM memory and the execution of programs stored in RAM, the following steps must be followed:

1. Place control ROMs in SIM8-01
2. Turn on system power
3. Turn on TTY to "line" position
4. Reset system with an INTERRUPT (Instr. RST = 00 000 101)
5. Change instruction at interrupt port to a NO OP
6. Start system with an INTERRUPT (Instr. NO OP = 11 000 000)
7. Load data from TTY into microcomputer RAM memory
8. Execute the program stored in RAM

**Loading of Multiple RAM Banks**

Through the use of the command "C", (CONTINUE) subsequent RAM banks may be loaded with data without entering a new data entry command and new memory bank and address designations.

Note that the CONTINUE command should only be used when the subsequent RAM will be completely loaded with 256 bytes of data. For partial loading of RAM banks, always use the DATA ENTRY command. The content of a RAM bank may be edited by using the DATA ENTRY command and revising.
and re-entering sections of the bank. When a program is being stored in memory, the first instruction of the program should be located at address \( \text{FF00} \) in a RAM bank. The entire RAM memory with the exception of the last fifteen bytes of RAM bank 3 may be used for program storage in conjunction with the bootstrap loader.

**Program Execution**

The program which has been loaded into RAM may be executed directly from RAM.

**TYPED BY SYSTEM**

- Ready for command
- Request for RAM BANK

**TYPED BY USER**

- *E
- Bn

Program EXECUTION command

RAM BANK in which the program has been stored.

*The first instruction in a program must be at address \( \text{FF00} \) in a RAM bank.*

Program beginning at address \( \text{FF00} \) of RAM BANK # n will be executed by the MCS-8 system.

*To return to the bootstrap control program, the ending statement of the program being executed should be "JMP 462"*

CAUTION: When executing a program from a single RAM bank or multiple RAM banks, care must be taken to insure that all JUMP addresses and subroutine CALL addresses are appropriately assigned within the memory storage being used.

**Summary of System Commands**

Using Intel's special control ROMs (A0860, A0861, A0863) the following control commands are available:

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>DATA ENTRY — Enter data from TTY into a RAM bank</td>
</tr>
<tr>
<td>C</td>
<td>CONTINUE — Continue entering 256 byte blocks of data into subsequent RAM banks</td>
</tr>
<tr>
<td>R</td>
<td>RE-ENTER — Re-enter a data word where a format error has occurred and continue entering data</td>
</tr>
<tr>
<td>E</td>
<td>EXECUTE — Execute the program stored in RAM memory</td>
</tr>
<tr>
<td>P</td>
<td>PROGRAM — Program a PROM using data stored in RAM memory</td>
</tr>
<tr>
<td>L</td>
<td>LIST — List the content of the PROM on the TTY</td>
</tr>
</tbody>
</table>

The complete Bootstrap Loader Program is presented in Appendix V.
X. MCB8-10 MICRO COMPUTER INTERCONNECT AND CONTROL MODULE

The MCB8-10 is a completely assembled interconnect, display and control switch assembly which eliminates all hand wiring associated with an MP7-03/SIM8-01 setup. With the additions noted below, it becomes a self-contained system featuring the following:

1. **General Purpose Micro Processor** with I/O and Display (with SIM8-01, power supplies)
2. **Automatic PROM Programming** (with SIM8-01, PROM set A0860, A0861, A0863, MP7-03, power supplies, TTY)
3. **Test System** for checkout of programs, features single-step capability (with SIM8-01, power supplies)

The MCB8-10 shown in Figure 20 includes the following:

1. All interconnect circuitry necessary to implement the programming system described in Section VIII of the MCS-8 Users Manual.
2. Connectors for the SIM8-01 and MP7-03 boards.
3. A zero insertion force 24-pin socket for PROMs to be programmed. Appropriate connections to the MP7-03 connector are provided.
4. Teletype, keyboard, printer, tape punch and reader control connections to SIM8-01. Access to these signals is provided by a 16-pin socket (TTY-J8). A flat cable is provided for the connection.
5. Control switches (2) and logic necessary for true-complement of programmer input or output data.
6. Breakout of all computer signals to open sockets for easy access. This includes output ports, flags (carry, sign, parity, zero), I/O decode (select I/O port 0, 1, 2, 3), I/O selection, cycle control, two decoded states (stop and wait), lower and higher order address.
7. 60 bits of LED display from SIM8-01.
8. All control lines are “OR-tied” to MCB8-10 or its connectors for external control.
9. Two toggle switches are provided for the following operations:

![Diagram of MCB8-10](image)

- For A0860 program (Bootstrap Loader and PROM programmer control ROMs), set the switches as shown in the figure above.
- For A0840 program (SIM8 Hardware Assembler) set S16* to “INTERRUPT” and S15* to “TTY”.
- For operation not using teletype as an I/O device, set S16 to “INTERRUPT” and S15 to “IN-A0”.

10. Two memory pushbutton switches are used for interrupt and single step function.
11. 8 toggle switches are provided for interrupt instruction input.
12. A toggle switch is provided for “WAIT” control.
13. Two transformers, 115V AC/220V AC, capacitor, fuse holder and AC input jack wired to develop the unregulated 80V DC which in turn is regulated on MP7-03 to 47V DC programming voltage.
14. A control switch for disabling the programming voltage.
15. Input jacks for applying externally supplied +5V DC and -9V DC to the assembly. (Note: internal supplies are not included).

*See figure 24.

The setup for the PROM programming application is shown in Figure 21. The MP7-03 (rear) and the SIM8-01 boards are installed in the MCB8-10.
A. Micro Processor System

When the MCB8-10 is used as a microprocessor, its features, such as the display (for the output ports, I/O decode, flag flip flops, cycle control, step and wait state, and in and out control and input ports), may be utilized at the discretion of the user. As an example, consider the testing of the SIM8-01 boards loaded with a PROM containing the following program: Read Port A and Port B, add the two values and output the results at Port A. The test could be implemented by connecting 8 switches to the A and B input sockets. The actual switch circuit would consist of a single pole double throw switch wired with one pole to ground and the wiper wired to the appropriate socket connector pin in accordance with the MCB8-10 schematic. The SIM8-01 is then inserted into the “SIM8-01” connector and a bench supply connected to the +5V DC and the −9V DC input jacks. The actual test may now be performed. The system is started according to the user’s instructions and the program is executed. The result appears at the LED display and may be verified for correctness. The display lights of interest are identified on the system’s printed circuit board (Figure 22) as “OUTPUT PORTS” 0, 1, 2, 3 (Bits 0-7).
B. Programming System

Consider the actual programming (in the hardware sense) of the 1702A PROM in the example above. The system can perform this function with the addition of an MP7-03 board inserted into the MP7-03 connector. An automatic programming system which allows data entry from a keyboard or paper tape, automatic verification, listing of ROM contents, and hands-off programming is provided by the further addition of three preprogrammed PROMs (A0860, A0861, A0863) and a modified teletype. The teletype modification consists of the addition of simple relay network described by the MCS-8 Users Manual. The procedure for programming a PROM, then, is as follows:
1. Insert MP7-03 and SIM8-01 boards (SIM8-01 loaded with PROMs A0860, A0861, A0863).
2. Connect teletype to “TTY” socket.
3. Connect +5V DC, —9V DC and 115/220V AC. Verify 115/220 switch is in proper position.
4. Insert instruction “00000101” with the 8 toggle switches provided for interrupt instruction input (i.e., RESTART to location 0).
   Depress “INTERRUPT”
   Insert instruction “11000000” (i.e., NOP) with the same 8 toggle switches
   Depress “INTERRUPT”
5. Set PROG.AC” to “ON”
6. Set data enable switch to “ENABLE”.
7. Set the data “IN/OUT” switches to “TRUE” or “COMPLEMENT”
8. Place teletype in “ON-LINE” mode
9. Insert PROM
10. Use A0860 program directives as described in Section IX of this Users Manual.

C. Program Debugging

Program debugging may be performed by using the “SINGLE-STEP” switch and LED display provided. The procedure is as follows:
1. For executing program in ROM (or ROMs):
   a. Turn off system power.
   b. Set toggle switch to “WAIT”.
   c. Insert programmed ROM (or ROMs).
   d. Turn on system power.
   e. Set interrupt instruction input (using the 8 toggle switches provided) with an RST 0 (00000101)
      instruction.
   f. Depress “INTERRUPT” switch.
   g. Depress “SINGLE-STEP” switch. This causes the CPU to execute the RST 0 instruction.
   h. Continue to depress “SINGLE-STEP” switch to advance the program one location at a time (a
      three-byte instruction requires three depressions of the “SINGLE-STEP” switch).
2. For executing program in RAM:
   a. Load program in RAM using A0860, A0861, A0863 program.
   b. Set toggle switch to “WAIT”.
   c. Set interrupt instruction input (using the 8 toggle switches provided) with a JMP instruction to
      select the desired RAM bank where the program has been loaded in step a. Enter the three byte
      JMP instruction as follows:
      Load 1st byte (01000100).
      Depress “INTERRUPT” switch.
      Depress “SINGLE STEP” switch.
      Load 2nd byte.
      Depress “SINGLE-STEP” switch.
      Load 3rd byte.
      Depress “SINGLE-STEP” switch.
      Set the 2nd and 3rd bytes according to the following examples:
      For BANK 0 —
      00000000 (2nd byte)
      00001000 (3rd byte)
      For BANK 1 —
      00000000 (2nd byte)
      00001001 (3rd byte)
      For BANK 2 —
      00000000 (2nd byte)
      00001010 (3rd byte)
For BANK 3 —
00000000 (2nd byte)
00001011 (3rd byte)
The above procedure causes the CPU to execute the JMP instruction that has been jammed in.
d. Continue to depress "SINGLE-STEP" switch to advance the program one location at a time.

D. Procedural Precautions

1. **CAUTION:** Do not remove DC power while programming AC power is on. Permanent damage to MP7-03 and PROM may result.
2. The MP7-03 board should be removed when SIM8-01 is not programmed to drive it.
3. Power up and power down for the programming system should be performed as follows:
   a. +5V DC and −9V DC on
   b. Restart procedure:
      - Restart instruction 00 000 101
      - Interrupt
      - Restart instruction 11 000 000
      - Interrupt
   c. TTY on
d. Programming AC on
e. Insert PROM
f. Execute
g. Remove PROM
h. Programming AC off
i. TTY off
j. +5V DC and −9V DC off

---

Figure 24. MCB8-10 Assembly Drawing
### SIMC-01 Connector Symbol

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Connector Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2, 4</td>
<td>J1 +5V</td>
<td>+5VDC POWER SUPPLY</td>
</tr>
<tr>
<td>4, 6</td>
<td>J2 -5V</td>
<td>-5VDC POWER SUPPLY</td>
</tr>
<tr>
<td>1, 3</td>
<td>J3 GND</td>
<td>GROUND</td>
</tr>
<tr>
<td>60</td>
<td>J1 M/D0</td>
<td>DATA FROM MEMORY # BIT 0</td>
</tr>
<tr>
<td>63</td>
<td>J1 M/D1</td>
<td>DATA FROM MEMORY 1 BIT 1</td>
</tr>
<tr>
<td>17</td>
<td>J1 M/D2</td>
<td>DATA FROM MEMORY 2 BIT 2</td>
</tr>
<tr>
<td>77</td>
<td>J1 M/D3</td>
<td>DATA FROM MEMORY 3 BIT 3</td>
</tr>
<tr>
<td>38</td>
<td>J2 M/D4</td>
<td>DATA FROM MEMORY 4 BIT 4</td>
</tr>
<tr>
<td>41</td>
<td>J2 M/D5</td>
<td>DATA FROM MEMORY 5 BIT 5</td>
</tr>
<tr>
<td>45</td>
<td>J2 M/D6</td>
<td>DATA FROM MEMORY 6 BIT 6</td>
</tr>
<tr>
<td>74</td>
<td>J2 M/D7</td>
<td>DATA FROM MEMORY 7 BIT 7</td>
</tr>
<tr>
<td>11</td>
<td>J1 TP3</td>
<td>DATA INPUT PORT # BIT 3</td>
</tr>
<tr>
<td>10</td>
<td>J1 TP1</td>
<td>DATA INPUT PORT # BIT 1</td>
</tr>
<tr>
<td>14</td>
<td>J1 TP5</td>
<td>DATA INPUT PORT # BIT 5</td>
</tr>
<tr>
<td>19</td>
<td>J1 TP6</td>
<td>DATA INPUT PORT # BIT 6</td>
</tr>
<tr>
<td>28</td>
<td>J1 TP2</td>
<td>DATA INPUT PORT # BIT 2</td>
</tr>
<tr>
<td>29</td>
<td>J1 TP3</td>
<td>DATA INPUT PORT # BIT 3</td>
</tr>
<tr>
<td>31</td>
<td>J1 TP4</td>
<td>DATA INPUT PORT # BIT 4</td>
</tr>
<tr>
<td>37</td>
<td>J1 TP5</td>
<td>DATA INPUT PORT # BIT 5</td>
</tr>
<tr>
<td>36</td>
<td>J1 TP6</td>
<td>DATA INPUT PORT # BIT 6</td>
</tr>
<tr>
<td>6</td>
<td>J1 TP1</td>
<td>DATA INPUT PORT # BIT 1</td>
</tr>
<tr>
<td>13</td>
<td>J1 TP2</td>
<td>DATA INPUT PORT # BIT 2</td>
</tr>
<tr>
<td>16</td>
<td>J1 TP3</td>
<td>DATA INPUT PORT # BIT 3</td>
</tr>
<tr>
<td>21</td>
<td>J1 TP4</td>
<td>DATA INPUT PORT # BIT 4</td>
</tr>
<tr>
<td>26</td>
<td>J1 TP5</td>
<td>DATA INPUT PORT # BIT 5</td>
</tr>
<tr>
<td>34</td>
<td>J1 TP6</td>
<td>DATA INPUT PORT # BIT 6</td>
</tr>
<tr>
<td>39</td>
<td>J1 TP7</td>
<td>DATA INPUT PORT # BIT 7</td>
</tr>
<tr>
<td>43</td>
<td>J2 TP1</td>
<td>OUTPUT PORT # BIT 1</td>
</tr>
<tr>
<td>47</td>
<td>J2 TP2</td>
<td>OUTPUT PORT # BIT 2</td>
</tr>
<tr>
<td>48</td>
<td>J2 TP3</td>
<td>OUTPUT PORT # BIT 3</td>
</tr>
<tr>
<td>49</td>
<td>J2 TP4</td>
<td>OUTPUT PORT # BIT 4</td>
</tr>
<tr>
<td>50</td>
<td>J2 TP5</td>
<td>OUTPUT PORT # BIT 5</td>
</tr>
<tr>
<td>51</td>
<td>J2 TP6</td>
<td>OUTPUT PORT # BIT 6</td>
</tr>
<tr>
<td>56</td>
<td>J2 TP1</td>
<td>OUTPUT PORT # BIT 1</td>
</tr>
<tr>
<td>57</td>
<td>J2 TP2</td>
<td>OUTPUT PORT # BIT 2</td>
</tr>
<tr>
<td>58</td>
<td>J2 TP3</td>
<td>OUTPUT PORT # BIT 3</td>
</tr>
<tr>
<td>59</td>
<td>J2 TP4</td>
<td>OUTPUT PORT # BIT 4</td>
</tr>
<tr>
<td>60</td>
<td>J2 TP5</td>
<td>OUTPUT PORT # BIT 5</td>
</tr>
<tr>
<td>61</td>
<td>J2 TP6</td>
<td>OUTPUT PORT # BIT 6</td>
</tr>
</tbody>
</table>

### SIMC-01 Connector Symbol

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Connector Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>57</td>
<td>J1 D1</td>
<td>RAM DATA IN D1</td>
</tr>
<tr>
<td>55</td>
<td>J1 D0</td>
<td>RAM DATA IN D0</td>
</tr>
<tr>
<td>53</td>
<td>J1 D2</td>
<td>RAM DATA IN D2</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D0</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D1</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D2</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D3</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D4</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D5</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D6</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D7</td>
</tr>
<tr>
<td>47</td>
<td>J1 TP1</td>
<td>OUTPUT D0</td>
</tr>
<tr>
<td>46</td>
<td>J1 TP2</td>
<td>OUTPUT D1</td>
</tr>
<tr>
<td>45</td>
<td>J1 TP3</td>
<td>OUTPUT D2</td>
</tr>
<tr>
<td>44</td>
<td>J1 TP4</td>
<td>OUTPUT D3</td>
</tr>
<tr>
<td>43</td>
<td>J1 TP5</td>
<td>OUTPUT D4</td>
</tr>
<tr>
<td>42</td>
<td>J1 TP6</td>
<td>OUTPUT D5</td>
</tr>
<tr>
<td>41</td>
<td>J1 TP7</td>
<td>OUTPUT D6</td>
</tr>
<tr>
<td>51</td>
<td>J1 TP1</td>
<td>OUTPUT D0</td>
</tr>
<tr>
<td>50</td>
<td>J1 TP2</td>
<td>OUTPUT D1</td>
</tr>
<tr>
<td>49</td>
<td>J1 TP3</td>
<td>OUTPUT D2</td>
</tr>
<tr>
<td>48</td>
<td>J1 TP4</td>
<td>OUTPUT D3</td>
</tr>
<tr>
<td>47</td>
<td>J1 TP5</td>
<td>OUTPUT D4</td>
</tr>
<tr>
<td>46</td>
<td>J1 TP6</td>
<td>OUTPUT D5</td>
</tr>
<tr>
<td>45</td>
<td>J1 TP7</td>
<td>OUTPUT D6</td>
</tr>
<tr>
<td>35</td>
<td>J1 TP1</td>
<td>OUTPUT D0</td>
</tr>
<tr>
<td>34</td>
<td>J1 TP2</td>
<td>OUTPUT D1</td>
</tr>
<tr>
<td>33</td>
<td>J1 TP3</td>
<td>OUTPUT D2</td>
</tr>
<tr>
<td>32</td>
<td>J1 TP4</td>
<td>OUTPUT D3</td>
</tr>
</tbody>
</table>

### SIMC-01 Connector Symbol

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Connector Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>62</td>
<td>J1 TP1</td>
<td>OUTPUT D0</td>
</tr>
<tr>
<td>61</td>
<td>J1 TP2</td>
<td>OUTPUT D1</td>
</tr>
<tr>
<td>60</td>
<td>J1 TP3</td>
<td>OUTPUT D2</td>
</tr>
<tr>
<td>59</td>
<td>J1 TP4</td>
<td>OUTPUT D3</td>
</tr>
<tr>
<td>58</td>
<td>J1 TP5</td>
<td>OUTPUT D4</td>
</tr>
</tbody>
</table>

### SIMC-01 Connector Symbol

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Connector Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D0</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D1</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D2</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D3</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D4</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D5</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D6</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D7</td>
</tr>
<tr>
<td>47</td>
<td>J1 TP1</td>
<td>OUTPUT D0</td>
</tr>
<tr>
<td>46</td>
<td>J1 TP2</td>
<td>OUTPUT D1</td>
</tr>
<tr>
<td>45</td>
<td>J1 TP3</td>
<td>OUTPUT D2</td>
</tr>
<tr>
<td>44</td>
<td>J1 TP4</td>
<td>OUTPUT D3</td>
</tr>
<tr>
<td>43</td>
<td>J1 TP5</td>
<td>OUTPUT D4</td>
</tr>
<tr>
<td>42</td>
<td>J1 TP6</td>
<td>OUTPUT D5</td>
</tr>
<tr>
<td>41</td>
<td>J1 TP7</td>
<td>OUTPUT D6</td>
</tr>
<tr>
<td>61</td>
<td>J1 TP1</td>
<td>OUTPUT D0</td>
</tr>
<tr>
<td>59</td>
<td>J1 TP2</td>
<td>OUTPUT D1</td>
</tr>
<tr>
<td>57</td>
<td>J1 TP3</td>
<td>OUTPUT D2</td>
</tr>
<tr>
<td>55</td>
<td>J1 TP4</td>
<td>OUTPUT D3</td>
</tr>
<tr>
<td>53</td>
<td>J1 TP5</td>
<td>OUTPUT D4</td>
</tr>
<tr>
<td>51</td>
<td>J1 TP6</td>
<td>OUTPUT D5</td>
</tr>
<tr>
<td>49</td>
<td>J1 TP7</td>
<td>OUTPUT D6</td>
</tr>
</tbody>
</table>

### SIMC-01 Connector Symbol

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Connector Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D0</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D1</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D2</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D3</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D4</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D5</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D6</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D7</td>
</tr>
<tr>
<td>47</td>
<td>J1 TP1</td>
<td>OUTPUT D0</td>
</tr>
<tr>
<td>46</td>
<td>J1 TP2</td>
<td>OUTPUT D1</td>
</tr>
<tr>
<td>45</td>
<td>J1 TP3</td>
<td>OUTPUT D2</td>
</tr>
<tr>
<td>44</td>
<td>J1 TP4</td>
<td>OUTPUT D3</td>
</tr>
<tr>
<td>43</td>
<td>J1 TP5</td>
<td>OUTPUT D4</td>
</tr>
<tr>
<td>42</td>
<td>J1 TP6</td>
<td>OUTPUT D5</td>
</tr>
<tr>
<td>41</td>
<td>J1 TP7</td>
<td>OUTPUT D6</td>
</tr>
<tr>
<td>61</td>
<td>J1 TP1</td>
<td>OUTPUT D0</td>
</tr>
<tr>
<td>59</td>
<td>J1 TP2</td>
<td>OUTPUT D1</td>
</tr>
<tr>
<td>57</td>
<td>J1 TP3</td>
<td>OUTPUT D2</td>
</tr>
<tr>
<td>55</td>
<td>J1 TP4</td>
<td>OUTPUT D3</td>
</tr>
<tr>
<td>53</td>
<td>J1 TP5</td>
<td>OUTPUT D4</td>
</tr>
<tr>
<td>51</td>
<td>J1 TP6</td>
<td>OUTPUT D5</td>
</tr>
<tr>
<td>49</td>
<td>J1 TP7</td>
<td>OUTPUT D6</td>
</tr>
</tbody>
</table>

### SIMC-01 Connector Symbol

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Connector Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D0</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D1</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D2</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D3</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D4</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D5</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D6</td>
</tr>
<tr>
<td>49</td>
<td>J1 M/D</td>
<td>DATA IN D7</td>
</tr>
<tr>
<td>47</td>
<td>J1 TP1</td>
<td>OUTPUT D0</td>
</tr>
<tr>
<td>46</td>
<td>J1 TP2</td>
<td>OUTPUT D1</td>
</tr>
<tr>
<td>45</td>
<td>J1 TP3</td>
<td>OUTPUT D2</td>
</tr>
<tr>
<td>44</td>
<td>J1 TP4</td>
<td>OUTPUT D3</td>
</tr>
<tr>
<td>43</td>
<td>J1 TP5</td>
<td>OUTPUT D4</td>
</tr>
<tr>
<td>42</td>
<td>J1 TP6</td>
<td>OUTPUT D5</td>
</tr>
<tr>
<td>41</td>
<td>J1 TP7</td>
<td>OUTPUT D6</td>
</tr>
<tr>
<td>61</td>
<td>J1 TP1</td>
<td>OUTPUT D0</td>
</tr>
<tr>
<td>59</td>
<td>J1 TP2</td>
<td>OUTPUT D1</td>
</tr>
<tr>
<td>57</td>
<td>J1 TP3</td>
<td>OUTPUT D2</td>
</tr>
<tr>
<td>55</td>
<td>J1 TP4</td>
<td>OUTPUT D3</td>
</tr>
<tr>
<td>53</td>
<td>J1 TP5</td>
<td>OUTPUT D4</td>
</tr>
<tr>
<td>51</td>
<td>J1 TP6</td>
<td>OUTPUT D5</td>
</tr>
<tr>
<td>49</td>
<td>J1 TP7</td>
<td>OUTPUT D6</td>
</tr>
</tbody>
</table>
APPENDIX I. SIM8 HARDWARE ASSEMBLER

1.0 INTRODUCTION

The SIM8 Hardware Assembler is a program which translates a symbolic assembly language into an octal representation of the SIM8 machine language. An auxiliary program then translates the octal object code into the “BNPF” format suitable for bootstrap loading or PROM programming. The program operates on the SIM8-01 micro computer system with an ASR 33 teletype and utilizes all memory of that system. The components included are the following:

8 PROMs (1702): A0840, A0841, ..., A0847
8 RAMs 1101): Last 256 bytes of assembler
24 RAMs (1101): Name table or object code

Upon purchase of the assembler the customer will receive the following:

8 PROMs (A0840-A0847) or 8 paper tapes
1 “SIM8 Hardware Assembler - page 8” paper tape (A0848)
1 “BNPF Tape Generator” (OCTAL) paper tape (A0849)
1 “BNPF Tape Generator” (SOURCE) paper tape (A0850)
1 “BNPF Tape Generator” Listing
1 SIM8 Hardware Assembler Listing
1 8008 Users Manual

A system block diagram is given in Figure 1.1.

![Sim8 Hardware Assembler System Configuration](image)

**Figure 1.1. SIM8 Hardware Assembler System Configuration**

The assembler accepts the source text from the paper tape reader on the first of two passes and constructs a name table. On a second pass the assembler translates the source text using the previously determined name values, creates an octal object paper tape, and if directed, writes the object code into Read/Write memory.

The assembler’s commands allow for TTY keyboard manipulation of R/W memory and execution of stored programs so that program debugging may be undertaken directly after assembly. If a “BNPF” tape is desired, an auxiliary “tape generator” program may be loaded and executed by the assembler.

2.0 DESCRIPTION

2.1 Assembly Passes

During Pass 1 the assembler reads the paper tape, constructs a name table and generates a listing. The listing consists of a line by line copy of the source text with each line prompted by an assembly address. When the assembler detects a source termination the process is stopped and a symbol table listing all labeled lines is generated. At this point no diagnostics have been acted upon.
During pass 2 the assembler generates an object code by reading the source tape and interrogating the name table for all labeled addresses. The object code is written into pre-assigned R/W memory or onto paper tape at the operator's option. Diagnostics performed during pass 2 result in omission of the erroneous line and a printout signaling the error. Errors detected are given below:

Detectable Errors
1. Unrecognized mnemonics
2. Unidentified labels
3. Illegal restart instruction
4. Non numeric literals
5. Illegal I/O instruction formats

2.2 Operating Procedures

In addition to being an assembler, this program offers some of the features of a teletype operating system. Its commands offer the operator a useful interactive mode. The commands "LOAD", "DUMP", and "BEGIN" allow the operator to read, write, and execute small programs directly from the keyboard.

The assembler requires a source text presented via a teletype reader. The first step of the assembly procedure is therefore the preparation of a punched paper tape version of the source text. (See Section 9 for details.) This is accomplished in an "off line" mode.

Before proceeding with the "on line" operations the hardware configuration must be correct. This requires a system equivalent with one exception to the SIMB-01 portion of the MP7-02/SIMB-01 PROM programming system described in the 8008 Users manual. The exception is the teletype connection. On the programming system the teletype transmit line drives both the interrupt line and the TTY buffer. The hardware assembler, however, must receive TTY data from the buffer only, so the interrupt must not be connected. A detailed description of the required connections for the Hardware Assembler is given in Section 10.

The assembler is a program which resides in nine 256 byte blocks or "pages" of memory. On the SIMB-01 eight pages are permanently stored in the "read only" section of its memory. The ninth page must be reloaded into R/W memory at each "power on" and becomes the second step in the operating procedure. To accomplish this, the paper tape containing the octal version of "SIMB Hardware Assembler - Page 8" is placed in the reader. If the "interrupt" input is stimulated, the assembler will bootstrap its 9th page into the R/W memory.

The assembler is now ready to execute commands.

The third step of the procedure is pass 1 of the assembly. To accomplish this the source tape is placed into the reader and the command below is typed.

ASSEMBLE: 032: 000:

The numeric values select the memory origin point for the assembly. When the reader is placed in the "start" mode the assembler will read the tape, generate a listing, and assemble a name table.

The fourth step is pass 2 during which the assembler rereads the source tape and compiles the object code. Line addresses and an octal representation of the object code is printed on the TTY and, if desired, simultaneously loaded into memory. Pass 2 may be initiated by typing "LOAD:" or "LIST:". "LOAD" will result in loading of memory and "LIST" will not. If the paper tape punch is enabled, an octal tape of the object code is created. Diagnostics are performed by the assembler during this pass and errors are flagged by a "?".

At this point the errors have been flagged and an edit of the source tape may proceed. If the program has been loaded into memory interactive editing is possible. This procedure is continued until the assembly is correct.

If the "BNPF" formatted object tape is required, an auxiliary program must be loaded into memory and executed. The "LOAD:" command is used to load the program "BNPF Tape Generator" into memory. The octal tape (256 character maximum) is then loaded into another area of the memory with a second "LOAD:" command. The tape generator program is executed by asserting the command "BEGIN:". The tape generator program accepts a three digit octal value terminated by a colon as a start address and begins to translate the memory contents into the "BNPF" format. A printout and a paper tape will be generated. Sample listings generated during each step described above are given in Figures 2.1, 2.2, 2.3, 2.4, and 2.5. Another example with a step-by-step procedure is given in Section 9.
2.3 Assembly Language

The assembler operates with the 64 character subset of ASCII generated by the ASR-33 teletype with the commercial at sign, @, given special significance and control characters, carriage return, and linefeed. Instruction source fields utilize a subset of the above including numerics, upper case alphabets, the colon, quote sign, commercial at, and the control characters.

The MCS-8 instruction mnemonics as described in the MCS-8 manual and pocket guide are recognized by the assembler. The instructions set is augmented by three pseudo operators, “PAM”, “ADR” and “LOC” which simplify the assembly process.

Symbolic addressing and selection of constants are provided by the definition of labels and use of the pseudo operators. A comment field is also provided.
3.0 ASSEMBLER COMMANDS

Five commands are used to direct the assembler which provide for teletype/memory interaction, assembly, and execution of loaded programs. They are defined as follows:

LOAD: The LOAD command is used to store keyboard or paper tape entries into consecutive locations beginning with an address specified by an address modifier. The modifier consists of 2 three digit octal numbers each terminated by a colon. The first defines a page address (see memory organization - section 5.0) and the second defines the character address. The format, described below, requires that leading zeroes be typed. Note that the character address has the range 000 to 3778 = 25610. LOAD: 011; 008; 3778 = 25610

Characters of the input tape must be 3 digit octal with leading zeroes, terminated with a colon. During an assembly the LOAD command may be used without a modifier to initiate pass 2. The source tape is then loaded and the object code is printed on the teletype printer and stored into memory as well.

DUMP: The DUMP command is used to display memory contents on the teletype printer. The command requires two address modifier pairs similar to that described for the LOAD command. The first pair is the address of the last content to be printed and the second pair is the first. The format is as follows:

<table>
<thead>
<tr>
<th>Last Address</th>
<th>First Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>011; 008;</td>
<td>011; 000;</td>
</tr>
<tr>
<td>Page Char.</td>
<td>Page Char.</td>
</tr>
</tbody>
</table>

The printout is 3 digit octal with 8 characters per line. Each line is prompted by a 6 digit octal memory address.

ASSEMBLE: The assemble command initiates pass 1 of the assembly. It is associated with an address modifier which establishes the origin of the program to be assembled. This address need not be related to the usable memory of the SIM8-01 card performing the assembly. The format of the command is described below:

<table>
<thead>
<tr>
<th>Origin</th>
</tr>
</thead>
<tbody>
<tr>
<td>032; 000;</td>
</tr>
<tr>
<td>Page Char.</td>
</tr>
</tbody>
</table>

LIST: The LIST command is recognized only during an assembly. It will initiate pass 2 in such a way that the source tape is loaded and the object code printed but not stored in memory. The LIST command does not require an address modifier. Its format is simply:

LIST:

BEGIN: The BEGIN command will initiate execution of a program located at the address specified by its address modifier. If an RST0 instruction is hardwired into the interrupt input port, assembler control may be recovered by generating an external interrupt. It should be noted that the ninth page of memory is not protected, hence care in execution of a secondary program is warranted. The format of the instruction is as follows:

<table>
<thead>
<tr>
<th>Address Modifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>032; 000;</td>
</tr>
<tr>
<td>Page Char.</td>
</tr>
</tbody>
</table>

4.0 NUMBER SYSTEM

All numbers used by the assembler are in three digit octal form and require leading zeroes to be typed.

5.0 MEMORY ORGANIZATION

Interaction with memory requires an understanding of its utilization by the assembler. The memory consists of 3000 8 bit bytes each directly addressable by the CPU. It is organized in blocks of 256 bytes called pages as shown in Figure 5.1. Addresses are specified by 2 three digit octal numbers each terminated by colon. The first number presented to the assembler is interpreted as a page designator and the second as a character designator.
The assembler resides in the first 9 pages of memory. Two bytes of the 10th page are also dedicated. The first 8 pages, number 0 through 7, are preprogrammed read only memories and the 9th resides in read write memory, page 8. The last page is volatile and must be reloaded if power is removed. The memory is unprotected so care must be exercised in selection of the assembly origin if the object code is to be stored in memory.

The name table created during pass 1 begins at location 011:020: and displaces 8 contiguous locations for each entry. The usable R/W memory for loading of object code in pass 2 diminishes as the table develops. The maximum number of names allowed is 94.

6.0 FORMAT

The assembler is a line-statement, fixed format assembler. Each field of the source statement is defined by its position in the line. If the positional format is violated the assembler will reject the statement. The format, depicted in Figure 6.1, provides fields for a 6 character label, a 3 character instruction, a 6 character operand, and variable length comment. The line is terminated by a carriage return followed by a linefeed but may be entirely cancelled by a commercial at sign, @.

Detailed descriptions of the fields are provided in the following sections.
6.1 Labels

Any line of the assembly may be assigned a label by placing a one to six character name into the label field. The label field is the first six positions of each line. If no label is to be assigned to the line, the field must be filled with spaces. Each entry into a label field must satisfy the following requirements.

1. The name must be left justified in the field.
2. The name can contain any character except the commercial at sign, @.
3. All unused positions in the field must be filled with spaces.
4. The name must appear exactly once in a label field of the source text.
5. The total number of names for a single assembly cannot exceed 94.

6.2 Instruction Mnemonics

All mnemonics defined in the MCS-8 Users Manual and pocket guide are recognized by the assembler. A concise description of each is provided in Appendix A. The reader is referred to the Users Manual for detailed information.

Further explanation and qualifications related to some of the instructions is given below.

**JUMP and CALL:** The operand field of a JUMP or CALL instruction can contain either a name or an address. If a name is used, it must be defined at some point in the source input or an error message will result. If an address is used, the assembler expects the first three digits to be the octal value of the page address and the second three to be the value of the character address. Examples of the two forms are given below:

```
000000 JMP □ START □ COMMENT

000000 JMP □ 004006 □ COMMENT
```

**RESTART:** The assembler operates on the operand field of a RESTART instruction in the same manner as on the operand field of a JUMP or CALL instruction. Its assembled value, however, must be consistent with the 6 bit "AAA 000" format utilized by the processor. If not, an error indication will result.

**IMMEDIATE:** All immediate instructions such as LAI can have an operand field occupied by a three digit octal number (left justified within field) or a character surrounded by double quote marks. (See section 6.3). If an octal number is found, it will be assembled directly as the immediate value. If a quote mark is found in the first position of the field, the ASCII equivalent of the character in the second position will be used as the operand value. If the first character of the operand field is neither a number or double quote mark, an error message will result. Examples of the formats are given below:

```
LEFT JUSTIFIED NUMERIC

000000 LAI □ 567 □ COMMENT

000000 LAI □ "A" □ COMMENT

QUOTE MARK IN FIRST POSITION IMMEDIATE VALUE IS AN ASCII A = 11000001
```
INPUT: The INPUT instruction may have either a name or an octal digit with two leading zeroes. The three digit numeric value is of the form "00X" where X can vary from zero to seven. The formats are as follows:

```
INP  NAME  COMMENT
```

```
INP  007  COMMENT
```

CONSTANTS

The name must assemble to a value between 0 and 7, and numerics must be within the specified range or an error flag will result.

OUTPUT: The OUTPUT instruction format is similar to the INPUT instruction but range of operand values is larger. Numeric operands may assume values from octal 010 to octal 037. The leading zero is required. Names must assemble to values within the specified range or an error flag will result. Examples of the formats are given below:

```
OUT  NAME  COMMENT
```

```
OUT  037  COMMENT
```

CONSTANT  MAXIMUM VALUE

HALT: The HALT instruction may be used as a pseudo operator. If the operand field is blank, it will assemble to its normal value of 000. If a non-zero value is placed into the first three digits of the operand field, that value will be assigned. If a quote mark is found in the first position of the operand field, the ASCII value of the digit in the second position will be assigned.

6.3 Pseudo Operators

Four additional instructions are provided to simplify the assembly process. These instructions are “pseudo operators” because they are not included in the MCS-8 instruction set. These instructions provide for name address assignment, memory block address assignment, a double register load for the H and L registers (see 8008 Manual), and termination of each pass of the assembly.

Detailed descriptions of these instructions are provided below:

PAM: The instruction “PAM” will assemble as two instructions, “LHI” followed by an “LLI”. Its operand field will be interpreted as two 3 digit octal values. The first and second values specify the LHI and LLI operand fields, respectively. The values may be numeric or named, but must meet the format requirements of the JMP or CALL instructions. The realizable range of the first is octal 000 to 077 and 000 to 377 for the second. An example is given below:

```
SOURCE STATEMENT
```

```
PAM  010377 COMMENT
```

```
EQUIVALENT SOURCE STATEMENT
```

```
LHI  010 COMMENT
```

```
LLI  377 COMMENT
```

62
ADR: The instruction “ADR” is non-executable and may appear anywhere in a program except the first instruction. The address specified in the operand field will be assigned to the name specified in the instruction. With this instruction, names may be assigned to external subroutines and I/O units. An example is given below:

```
SOURCE STATEMENT
START □ ADR □ 001377 COMMENT
```

RESULT OF ASSEMBLY
```
START ←—— 001377
```

LOC: The instruction “LOC” is nonexecutable and must only appear after the last executable instruction. It is used to reserve blocks of memory locations directly after the assembled programs and to assign a name to the first location. The name field should contain the desired name and the operand field should contain two three-digit octal numbers to indicate the length of the array. The form of the number is the same as that used to indicate an address. For example, the number 001000 would reserve 256 locations and the number 000377 would reserve 255 locations.

END: If the instruction END is encountered by the assembler it will terminate the current pass in process.

HALT: If the operand value of a HLT instruction is non-zero it is treated as a pseudo operator. Section 6.2 provides a detailed description.

7.0 ERROR FLAGS

Diagnostics performed in pass 1 and pass 2 may result in error flags during pass 2. If an error is detected, the invalid source entry followed by a question mark is printed. If the error exists in the operand field but not in the instruction field, the object code for the instruction will be printed and punched. The assembly must therefore be repeated after source text corrections are made.

The conditions that result in error flags are described below:

INVALID MNEMONICS
Every mnemonic field must contain three letters which can be exactly identified as an instruction; otherwise, it will be rejected as an error.

UNDEFINED NAMES
If a referenced name is not found an error message will result.

INVALID RESTART ADDRESS
The RESTART instruction operates on the operand in the same manner as the JUMP and CALL instruction, except that it requires that the resulting address be one of the valid restart locations. If this is not true, an error message will result.

INVALID OPERAND FIELD FOR IMMEDIATES
For immediate instructions, the first character of the operand field must be a number or a quote mark.

INVALID OPERAND FIELD FOR JUMP AND CALL INSTRUCTIONS
Operand fields for JUMP and CALL instructions must be a valid name or an octal number.

INVALID OPERAND FIELDS FOR INPUT/OUTPUT INSTRUCTIONS
Section 6.2 defines valid operands fields for the input and output instructions. If those definitions are violated in the source text, error flags will result.

8.0 OUTPUT TAPE

The assembler generates an octal output tape representation of the object code. Each byte is represented by three digits terminated with a colon (see Section 9). Lines of 8 bytes are prefixed by the address of the first byte. The address is not terminated by a colon and will therefore not be accepted by the assembler “LOAD” instruction.

The octal listing is compact and intended for editing operations. To perform standard Intel programming functions, a “BNPF” formatted tape version of the object tape must be prepared. To accomplish this, a “BNPF Tape Generator” program supplied by Intel, and a page of the octal object code is loaded into memory. The BEGIN instruction is then used to execute the “Tape Generator” program which reads 256 bytes of memory, translates them to a “BNPF” format, and transmits them to the teletype for printing and punching.

As an option a “BNPF Tape Generator” source tape is provided so that the customer may assemble the auxilliary program with an origin of his choosing. Section 11 provides a detailed, step-by-step description.

A detailed description of the procedure and tape outputs is provided in Section 9.
9.0 SAMPLE ASSEMBLY WITH A STEP-BY-STEP PROCEDURE

The sample program used in this description is not executable, but includes every instruction, several register pair selections, erroneous instructions, and the pseudo operators.

STEP 1. PREPARE SOURCE TEXT

The first step, after handwriting of the program, in symbolic language, is to create a punched paper tape and print out on an ASR 33 teletype. The result of this transcription applied to the sample program is shown in Figure 9.1.

The procedure for creating the source tape is given below:

1. The TTY was placed in the “off-line” mode.
2. The paper tape punch control was placed in an “on” condition.
3. Handwritten data was keyed into the teletype keyboard.

Some typographical errors were edited by using the TTY’s backspace punch control and rubout character. The rubout is an all “1”s character which effectively deletes any character over which it is superimposed. The procedure is as follows:

1. Determine the number of backspaces required to return the punch to the erroneous character.
2. Depress the paper tape punch backspace control until the erroneous character is reached.
3. Enter a “rubout” from the keyboard. If a new character must be inserted, the previous character and the remaining line or lines must be deleted with rubouts.
4. Enter the desired character and remaining lines.

The assembler’s recognition of a commercial at sign, @, may be used as an editing feature since it will effectively delete the line from the assembly process.

Some comments regarding the format are given below.

1. The first line of the source listing must be named.
2. Strict adherence to the positional nature of the format is essential.
3. The source listing is terminated by the pseudo operator END.

STEP 2. PREPARE SIM8-01

Step 2 of the procedure is the preparation of the SIM8-01. This requires loading of the assembler ROMs, presetting the interrupt instruction, and bootstrap loading of the last page of the assembler into R/W memory. The procedure is as follows:

1. Wire SIM8-01 connections in accordance with 8008 Users Manual description of MP7-03/SIM8-01 PROM Programming Systems with exceptions cited in Appendix C of this note.
2. Hardwire or select by switch a RESTART instruction (00000101) at the interrupt port (see 8008 Users Manual).
3. Install 8 1702 PROMs, A0840 to A0847, into the SIM8-01.
4. Connect a teletype and power supplies to the SIM8-01 as described in the section VII of the 8008 Users Manual.
5. Place the teletype in the “ON-LINE” mode and set the reader to “FREE”.
6. Place the paper tape “SIM8 Hardware Assembler - page 8 for 1101 RAM” (A0848) in the reader.
7. Depress the interrupt switch.
8. Place the reader in the start mode.

Approximately 256 locations will be loaded into RAM starting at location 010: 000: At completion of load the assembler is ready to receive commands. Note that its readiness to accept a command is not prompted by a special character such as carriage return.

STEP 3. COMPLETE PASS 1

With the reader placed in a “free” or “off” mode the source paper tape is placed into the reader. The assembler command and an origin for the program is then input from the keyboard. The command is shown below:

ASSEMBLE: __032: __000:

SIGNIFICIES SPACE ORIGIN
Figure 9.2 Pass 1 Listing
The origin may assume any octal value from 000: 000: to 777: 777: without consequence if a load command is not used to enter pass 2. If a load command is used to start pass 2, the object code will be loaded into memory beginning at the specified origin. If this is done the operator must be sure that page 9 and the name table created during pass 1 are not affected. (See Figure 1.) As an example, if 30 names are used, only 512 object code locations remain available (012: 000: to 013: 377:). An example of the listing generated during pass 1 is given in Figure 9.2. The example is a test program which includes all instructions, pseudo ops, and some erroneous instructions. The assembler reads the source tape, prompts all assembly lines, ignores comments, and generates a symbol table. The completion of pass 1 is evidenced by the completion of the symbol table.

STEP 4. COMPLETE PASS 2

Pass 2 requires a reread of the source paper tape so it must be repositioned with the reader in a "STOP" or "FREE" mode. A "LOAD" or a "LIST" command is used to initiate pass 2 of the assembly. The load command will cause the object code to be loaded into memory during pass 2. A list command will not affect memory. When the load instruction is used the object code must not overlap dedicated memory. (See Figure 5.1.) The commands are entered from the keyboard as follows:

LOAD: or LIST:

A listing generated during pass 2 is shown in Figure 9.3. If the paper tape punch is turned on when the "LOAD:" or "LIST:" command is typed, an octal version of the object code is generated.

---

**Figure 9.3 Pass 2 Listing**

---

STEP 5. EDIT AND REASSEMBLE

If errors occur during the assembly, the source text should be edited and the assembly process repeated. If no assembly errors occur, the user may elect to load the program into memory, assert the "BEGIN" command, and execute the program. Caution is warranted in this case because the load of the program or its execution may alter the name table or the 9th page of the assembler. An example of the load and execute is provided in the next section ("BNPF" tape generation).
STEP 6. CREATE A "BNPF" PROGRAMMING TAPE

The octal object tape of the assembler is not suitable for PROM programming or bootstrap loading so the next step is the conversion of the octal tape into a "BNPF" formatted tape.

In summary, this requires the following:
1. Loading of a "BNPF Tape Generator" program (Tape A0849) into R/W memory.
2. Loading a block of 256 bytes of memory with octal object code.
3. Executing the "BNPF Tape Generator" program which creates the desired output tape.

A detailed description is provided below:

The "BNPF Tape Generator" program reads 256 memory locations, translates them, and sends them to the TTY. If the punch is on, a "BNPF" tape will be generated. The RAM must therefore be loaded with the octal data that must be translated. The load command; LOAD: 012: 000: was used to load the test tape into locations 012: 000: to 012: 157: as shown in Figure 9.4. Note that the load instruction does not prefix the data. Also, RAM overlap onto "BNPF" at 013: 000: and page 8 at 010: 000: must be avoided by proper addressing. With object code loaded a translation may now be accomplished. The begin instruction is used to jump to the "BNPF" program loaded at 013: 000:. The punch is turned on and 256 lines of "BNPF" tape are generated. The command; BEGIN: 013: 000: was used as shown in Figure 9.5. Long tapes must be processed in blocks of 256 eight bit codes.

Figure 9.4. Loading of "BNPF Tape Generator" and Object Code
Figure 9.5. Output of "BNPF Tape Generator"
10.0 HARDWARE CONFIGURATION DETAILS

The basic wiring required for the assembler is shown in Figure 10-1. This is compatible with the PROM programming system with two exceptions:

1. The auxiliary interrupt input (J1-1) is not used by the assembler and must be grounded. The PROM Programming System software utilizes this input to initiate a teletype receive sequence. A switched selection is recommended.

2. The interrupt instruction port can be permanently wired as an RST instruction for the assembler but must be selectable for the Bootstrap Loader program. To satisfy both, it is recommended that switches be used to drive inputs J1-7, 9, 18, 20, 24, 27, 38 and 40 between ground and +5V.

![Figure 10.1. SIM8-01 Minimum Configuration Requirement](image)

11.0 ASSEMBLY OF "BNPF TAPE GENERATOR"

The tape "BNPF Tape Generator" (source), tape A0850, may be used to relocate the "BNPF Tape Generator" object code. The object code, A0849, provided has origin 013: 000: and may be changed if desired.

The assembly process described in Section 9 is applied to the source tape A0850. At Step 3 (Section 9) of the assembly, the origin is changed to the value desired. When Steps 4 and 5 are completed, an object code for the relocated tape generator is created. The object tape may then be loaded at the new location using the "LOAD" command and executed using the "BEGIN" command. (See Step 6 of Section 9).
APPENDIX II. MCS-8 SOFTWARE PACKAGE — ASSEMBLER

A. Assembler Specification

1.0 GENERAL DESCRIPTION

The 8008 Assembler generates object programs from symbolic assembly language instructions. Programs are written in the assembly language using mnemonic symbols both for 8008 instruction and for special assembler operations. Symbolic addresses can be used in the program code; however, the assembled program will use absolute addresses.

The Assembler is designed to operate from a time shared terminal with input by paper tape or directly from the terminal keyboard. The assembled program is punched out at the terminal in BNPF format paper tape.

This routine is written in FORTRAN IV. It may be procured from Intel on magnetic tape. Alternatively, designers may contact several nationwide timesharing services for access to the programs.

The program specifications are presented first and are followed by a user’s guide for some of the timesharing services.

1.1 Assembler Use and Operation

Source programs are written in assembly language and edited prior to assembling, using the time sharing EDITOR program. Edited programs can then be assembled. The Assembler processes the source program in two passes.

The Assembler generates a symbol table from the source statement names in the first pass and checks for errors.

In the second pass the Assembler uses the symbol table and the source program to generate both a program listing and an absolute binary program. Error conditions are indicated in the program listing.

1.2 Symbol Usage

Symbols can represent specific addresses in memory for data and program words, or can be defined as constants. Symbols are used as labels for locations in the program or as data storage area labels or as constants.

Expressions can be formed from a symbol combined by plus or minus operators with other symbols or numbers to indicate a location other than that named by the symbol. Every symbol appearing as part of an operand must also appear as a statement label or else it is not defined and will be treated as an error. Symbols that are used as labels for two or more statements are also in error.

1.3 Absolute Addressing

Object programs use all absolute addresses. The starting address is specified by a pseudo instruction at the beginning of the source program. All subroutines referenced by symbol in the main program must be assembled as part of the main program. Subroutines not assembled with the main program must be referenced by their starting addresses.

1.4 Program Addresses

Consecutive memory addresses are generated by the Assembler program counter and assigned to each source statement. Two byte source statements are assigned two consecutive addresses and three byte source statements are assigned three consecutive addresses.

The starting address is set by an ORG pseudo instruction at the beginning of the source program.

1.5 Output Options

The Assembler output is stored in files and can be read out in several forms under control of the time sharing EXECUTIVE. Some of the options available are:

a. binary paper tape at the terminal;
b. card output at computer center;
c. program listing at the terminal;
d. program listing at the computer center;
e. symbol table listing at the terminal;
f. symbol table listing at the computer center.

2.0 INSTRUCTION FORMAT

The Intel Assembly program consists of a sequence of symbolic statements. Each source language statement contains a maximum of four fields in the following order:

location field;
operation field;
operand field;
comment field.

The format is essentially free field. Fields are delimited by one or more blanks. Blanks are interpreted as field separators in all cases, except in the comments field or in a literal character string.
Each statement is terminated by an end of statement mark. On punched paper tape a carriage return and a line feed punch terminates a statement.

The maximum length of any statement is 80 characters, not including the end of statement mark. The instruction must end prior to character 48 but the comments may extend to column 80.

2.1 Symbols
Symbols are used in the location field and in the operand field. A symbol is a sequence of one to six characters representing a value. The first character of any symbol must be an alphabetic symbol. Symbols are comprised of the characters A through Z, and zero through nine.

The value of a symbol is determined by its use. In the location field of a machine instruction or a data definition, the value assigned to the symbol is the current value of the program counter. In the location field of an EQU pseudo instruction, the value of the operand field is assigned to the symbol.

An asterisk is a special purpose symbol. It represents the location of the first byte of the current instruction. Thus if an operand contains *-1, then the value calculated by the Assembler is one less than the location of the first byte of the current instruction.

Examples of legal symbols:

MAT START2
MIKE Z148
TED24 RONA3Z
*

2.2 Numeric Constants
Two types of numeric constants are recognized by the Assembler: decimal and octal. A decimal number is represented by one to five digits (0-9) within the range of 0 to 16383. An octal number contains from one to five digits (0-7) followed by the letter B. The range of octal numbers is 0 to 37777B.

Numeric constants can be positive or negative. Positive constants are preceded by a plus sign or no sign. Negative constants are preceded by a minus sign. There can be no blanks between the sign and the digits. If a minus sign precedes the number, then the complement of the binary equivalent is used.

2.3 Expressions
Expressions may occur in the operand field. The Assembler evaluates the expression from left to right and produces an absolute value for the object code. There can be symbols and numbers in expressions separated by arithmetic operators + and —. Octal decimal numbers are acceptable. No embedded blanks are allowed within expressions.

Parentheses are not permitted in an expression. Thus terms cannot be grouped as in the expression Z-(4+T). That expression must be written as Z-4-T to be acceptable to the Assembler.

2.4 Location Field
The location field of a statement contains a symbol when needed as a reference by other statements. If a statement is not referenced explicitly, then the location field may be blank.

The symbol must start in column 1 of the statement. That is, if a symbol is required it must be punched immediately following the end of statement mark of the preceding statement. The Assembler therefore assumes that if column 1 is blank, the location field of that statement does not contain a symbol.

Column 1 of the location field can also indicate that the entire line is a comment. If an asterisk occurs in column 1, then positions 2 through 80 contain remarks about the program. These remarks have no effect on the assembled program but do appear in the output listing.

2.5 Operation Field
The operation field must be present and is represented by a mnemonic code. The code describes a machine operation or an Assembler operation.

The operation code follows the location field and is separated by one or more blanks from the location field. The operation field is terminated by a blank or an end of statement mark when there is no operand field and no comment field.

Examples of machine operations:

LAB Load Register A with the contents of Register B
CPM Compare contents of A register with contents of memory location m.

Example of Assembler operation:

ORG Set program counter to specified origin

72
2.6 Operand Field
The contents and significance of the operand field are dictated by the operation code. The operand field can contain the following:
  - blank
  - symbol
  - numeric
  - expression
  - data list

The operand field follows the operation code and is separated from that code by one or more blanks. The operand is terminated by a blank or an end of statement mark if no comments follow the operand.

Examples of operands:
- \texttt{DANI MIKE2-MIKE4 + 1}
- \texttt{143B 773B + X2}
- \texttt{1869 "-1}
- \texttt{RON+33B AA44-22B}
- (blank)

2.7 Comment Field
The comment field is optional. It follows the operand field and is separated from that field by at least one blank. If there is no operand field for a given operation code, then the comment field follows the operation field. Once again at least one blank separates the operation code and the comments. Comments must terminate on or before the 80th character position. If the comment extends beyond that position, it will be truncated on the output listing. Comments up to the 48th character position are printed along with the source code. If comments are in positions 49 through 80, then they are printed on the next line.

3.0 MACHINE OPERATION
Each instruction in the 8008 repertoire can be represented by a three letter mnemonic in the 8008 assembly language. For each source statement in the assembly language (except for some pseudo instructions), the Assembler will generate one or more bytes of object code. Source language statements use the following notation:
- \texttt{Label} — Optional statement label;
- \texttt{Operand} — One of the following:
  - data — A number, symbol or expression used to generate the second byte of an immediate instruction.
  - address — A number, symbol or expression used to generate the second and third bytes of a call or jump instruction.
  - device — A number, symbol or expression used to define input/output instructions to select specific devices.
  - start — A number, symbol or expression used to define a starting address after a restart instruction.
- \texttt{Comment} — Optional comment.
- ( ) — Information enclosed in brackets is optional.

3.1 Move Statements — 1 byte, or 2 bytes when operand is used.
Move instructions replace the contents of memory or of the A, B, C, D, E, H and L Registers with the contents of one of the Registers A, B, C, D, E, H or L or with the contents of the memory location specified by H and L or with an operand from the second byte of the instruction. In what follows, \( r_1 \) can represent A, B, C, D, E, H, L, or M. \( r_2 \) can represent A, B, C, D, E, H, L, M or I. If \( r_1 = M \), the contents of memory are replaced by the contents of \( r_2 \). If \( r_2 = M \), the contents of \( r_1 \) are replaced by the contents of memory. If \( r_2 = I \), the contents of \( r_1 \) are replaced by the operand from the second byte of the instruction.

\[
\begin{array}{lcl}
\text{Label} & | & \text{LEH} & | & \text{Comment} \\
\text{Move H to E.}
\end{array}
\]

\[
\begin{array}{lcl}
\text{Label} & | & \text{LAM} & | & \text{Comment} \\
\text{Load A from memory.}
\end{array}
\]

\[
\begin{array}{lcl}
\text{Label} & | & \text{LMB} & | & \text{Comment} \\
\text{Move B to memory.}
\end{array}
\]

73
Label | LCI | 062B | Comment
Load octal 062 into C.

Label | LMI | 135B | Comment
Load octal 135 into memory.

The contents of the sending location are unchanged after each move. An operand is required if and only if $r_2 = 1$.

3.2 Arithmetic and Logical Operation Statements - 1 byte, or 2 bytes when operand is used.

These instructions perform arithmetic or logical operations between the contents of the A Register and the contents of one of the Registers B, C, D, E, H or L or the contents of a memory location specified by H and L or an operand. The result is placed in the A Register. In what follows, $r$ may be B, C, D, E, H or L, M or I. If $r = M$, memory location is specified. If $r = I$, the operand from the second byte of the instruction is specified.

3.2.1 (Label) | ADr | data | (Comment)
Add $r$ to A.

3.2.2 (Label) | ACr | data | (Comment)
Add $r$ to A with carry.

3.2.3 (Label) | SUr | data | (Comment)
Subtract $r$ from A.

3.2.4 (Label) | SBr | data | (Comment)
Subtract $r$ from A with borrow.

3.2.5 (Label) | NDr | data | (Comment)
Logical AND $r$ with A.

3.2.6 (Label) | Xr | data | (Comment)
Exclusive OR $r$ with A.

3.2.7 (Label) | Or | data | (Comment)
Inclusive OR $r$ with A.

3.2.8 (Label) | CPr | data | (Comment)
Compare $r$ with A.

Examples:

Label | ADB | Comment
Add B to A.

Label | SUM | Comment
Subtract the contents of the memory location specified by H and L from A.

Label | CPI | 024B | Comment
Compare octal 024 with A.

An operand is required if and only if $r = I$.

3.3 Rotate Statements -- 1 byte

3.3.1 (Label) | RLC | (Comment)
Rotate A one bit left.
3.3.2 (Label) | RRC | (Comment)
  Rotate A one bit right.

3.3.3 (Label) | RAL | (Comment)
  Rotate A through the carry one bit left.

3.3.4 (Label) | RAR | (Comment)
  Rotate A through the carry one bit right.

3.4 Call Statements - - 3 bytes
Call instructions are used to enter subroutines. The second and third bytes of call instructions are generated from source program operands and are used to address the starting locations for the called subroutines. An operand is always required.

3.4.1 (Label) | CAL | address | (Comment)
  Call subroutine unconditionally.

3.4.2 (Label) | CTC | address | (Comment)
  Call subroutine if carry = 1.

3.4.3 (Label) | CFC | address | (Comment)
  Call subroutine if carry = 0

3.4.4 (Label) | CTZ | address | (Comment)
  Call subroutine if accumulator = 0.

3.4.5 (Label) | CFZ | address | (Comment)
  Call subroutine if accumulator ≠ 0.

3.4.6 (Label) | CTP | address | (Comment)
  Call subroutine if accumulator parity is even.

3.4.7 (Label) | CFP | address | (Comment)
  Call subroutine if accumulator parity is odd.

3.4.8 (Label) | CTS | address | (Comment)
  Call subroutine if accumulator sign is minus.

3.4.9 (Label) | CFS | address | (Comment)
  Call subroutine if accumulator sign is plus.

At the conclusion of each subroutine, control returns to the address “Label + 3”.

3.5 Jump Statements - - 3 bytes
Jump instructions are used to alter the normal program sequence. The second and third bytes of jump instructions are generated from source program operands and are used as the address of the next instruction. An operand is always required.

3.5.1 (Label) | JMP | address | (Comment)
  Jump to address unconditionally.

3.5.2 (Label) | JTC | address | (Comment)
  Jump to address if carry = 1.

3.5.3 (Label) | JFC | address | (Comment)
  Jump to address if carry = 0.
3.5.4  | (Label)   | JTZ    | address | (Comment)
       |          | Jump to address if accumulator = 0.

3.5.5  | (Label)   | JFZ    | address | (Comment)
       |          | Jump to address if accumulator ≠ 0.

3.5.6  | (Label)   | JTP    | address | (Comment)
       |          | Jump to address if accumulator parity is even.

3.5.7  | (Label)   | JFP    | address | (Comment)
       |          | Jump to address if accumulator parity is odd.

3.5.8  | (Label)   | JTS    | address | (Comment)
       |          | Jump to address if accumulator sign is minus.

3.5.9  | (Label)   | JFS    | address | (Comment)
       |          | Jump to address if accumulator sign is plus.

3.6    | Return Statements - - 1 byte

Return instructions are used at the end of subroutines to return control to the address following the call instruction that entered the subroutine. In what follows, assume a subroutine was called as shown:

```
MAIN  | CAL     | SUBRTN | Comment

3.6.1  | (Label)   | RET    | (Comment)
       |          | Return unconditionally to “MAIN + 3”

3.6.2  | (Label)   | RTC    | (Comment)
       |          | Return to “MAIN + 3” if carry = 1.

3.6.3  | (Label)   | RFC    | (Comment)
       |          | Return to “MAIN + 3” if carry = 0.

3.6.4  | (Label)   | RTZ    | (Comment)
       |          | Return to “MAIN + 3” if accumulator = 0.

3.6.5  | (Label)   | RFZ    | (Comment)
       |          | Return to “MAIN + 3” if accumulator ≠ 0.

3.6.6  | (Label)   | RTP    | (Comment)
       |          | Return to “MAIN + 3” if accumulator parity is even.

3.6.7  | (Label)   | RFP    | (Comment)
       |          | Return to “MAIN + 3” if accumulator parity is odd.

3.6.8  | (Label)   | RTS    | (Comment)
       |          | Return to “MAIN + 3” if accumulator sign is minus.

3.6.9  | (Label)   | RFS    | (Comment)
       |          | Return to “MAIN + 3” if accumulator sign is plus.
3.7 Input/Output Statements - - 1 byte
These instructions are used to input or output data, one byte at a time, between the A Register and the external device selected by the operand. An operand is always required.

3.7.1

(Label) | INP | device | (Comment)

Inputs one byte of data from device to the A Register.

3.7.2

(Label) | OUT | device | (Comment)

Outputs one byte of data from the A Register to device.

The device operand must have a value between 0 and 7 for input instructions and between 10 and 37 octal for output instructions.

3.8 Increment/Decrement Statements - - 1 byte
These instructions are used to increment by one or decrement by one any of the registers r. In what follows, r can represent B, C, D, E, H or L. Increment and decrement operations affect the accumulator conditions zero, parity and sign, but not carry.

3.8.1

(Label) | INr | (Comment)

Add 1 to r.

3.8.2

(Label) | DCr | (Comment)

Subtract 1 from r

Example:

LABEL | INB | (Comment)

Add 1 to B.

3.9 Halt Statement - - 1 byte
The halt instruction is used to stop the 8008 processor.

3.10 Restart Statement - - 1 byte
The restart instruction is used in conjunction with an interrupt signal to start the 8008 after a halt. The program counter is set to a starting address equal to the operand multiplied by octal 10. A start operand is required which may have a value from 0 to 7.

3.11 Load Address Statement - - 4 bytes
This instruction is used to load H and L with a memory address and is simply an assembly language convention equivalent to the two separate instructions LHI and LLI. An operand is required.

4.0 PSEUDO INSTRUCTIONS
The purpose of pseudo instructions is to direct the Assembler, to define constants used by the object code, and define values required by the Assembler. The following is a list of pseudo operations.

ASB Define paper tape output
ORG Define origin of program
EQU Define symbol value for Assembler
DEF Define constants for object code
DAD Define two byte address
4.1 Program Origin

The program origin can be defined by the user by an ORG pseudo operation. If no ORG statement is defined, the origin is assumed to be zero. The origin can be redefined whenever necessary by including an ORG statement prior to the section of code which starts at a specific program location.

The format of the ORG statement is:

```
|   ORG   | n   | (Comment) |
```

The operand n can be a number symbol, or an expression. If a symbol is used it must be predefined in the code.

Example of the ORG statement:

```
LAB
LCD
.
.
.
ORG  1000B
SAM  LCD
.
.
.
ORG  5000B
SALLY DEF 1, 4, 777B, 7000B
END
```

Instruction starts in LOC 0000

Instruction stored in LOC 1000

Data starts in LOC 5000

4.2 Equate Symbol

A symbol can be given a value other than the one normally assigned by the program location counter by using the EQU pseudo operation. The symbol contained in the location field is given the value defined by the operand field.

The EQU statement does not produce a machine instruction or data word in the object code. It merely assigns a value to a symbol used in the source code.

Format of the EQU statement:

```
Symbol  | EQU   | operand | (Comment)
```

The operand may contain a numeric, a symbol, or an expression. Symbols which appear in the operand must be previously defined in the source code.

All fields are required except for the comment field, which is always optional.

Example of EQU statements:

```
TELET  EQU  4
MAGT2  EQU  2
MAGT6  EQU  6
SAM    EQU  1000B
INP    TELET
LAB    CALL  SAM
OUT    MAGT2
```

4.3 Define Constant

Constant data values can be defined using the DEF pseudo statement. The data values are placed in sequential words in the object code. If a symbol appears in the location field, it is associated with the first data word. That symbol can be then used to reference the defined data.

Format of the DEF statement:

```
(Symbol) | DEF  | data list | (Comment)
```

The data list consists of one or more terms separated by commas. There can be no embedded blanks in the data list (except in a literal character string). The terms can be octal or decimal numerics, literal character strings, symbols or expressions.
A literal character string is enclosed in single quote marks (''). It can contain any ASCII characters, including blanks. The internal BCD 8 bit codes corresponding to the given characters are stored in sequential bytes, one character per byte.

Octal and decimal numbers are stored one per byte in binary.
Octal numbers must be in the range 0 to 377B.
Decimal numbers must be in the range 0 to 255.
Two's complements are stored for minus numbers.

The program counter is incremented by one for each numeric term in the data string and by n for each literal string of n characters.

Examples of data strings:

MESS1   DEF   "SYMBOL TABLE OVERFLOWED", Y-2, SUB2
MESS2   DEF   "LITERAL STRING 1", "LITERAL STRING 2"
MASKS   DEF   77B, 177B, 130B, LABEL 3, X + 3 Required masks
DEF       24, 133, 37B, 99, 232, "ERROR" Required constants

4.4 Define Address

Program addresses, defined by alphabetic symbols, are stored as data by the DAD pseudo operation. The 16 bit address is stored in sequential bytes; the first byte contains the 8 least significant bits and the second byte contains the 8 most significant bit of the address.

Format of the DAD statement:

(Symbol) | DAD     | data list | (Comment)

The data list consists of one or more symbols separated by commas. There can be no embedded blanks in the data list.
The program counter is incremented by one for each symbol in the data list.

Examples of DAD statements:

LINK     DAD     SUB1, SUB2, SUB3
ERRSUB   DAD     ERRORX   Print Errors
DAD       SOCTAL, SPECM, SYMBOL, SEXP, SLIT

4.5 End of Source

The end of the source code statements is defined with the END pseudo statement. The END operation code generates no object code; it merely signals to the Assembler that there is no more source code.

Format of the END statement:

| END | (Comment)

Note that no symbol is allowed in the location field of the END statement.

4.6 Assembler Paper Tape Output

The format of the paper tape output is defined by the ASB pseudo output. The operand specifies the format with the following mnemonic codes.

F1601— 1601 format described in Intel Data Catalog.
F8008— F8008 Format (This logic is not included in the Assembler but the position of the code is described in the PAPER Subroutine.)

The entire 80 character statement is written on the paper tape file as the first record. It is used to describe the contents of the paper tape. If no ASB pseudo operation appears, then format F1601 is assumed and a string of asterisks appear on the paper tape file as the first record.

Examples of ASB statements:

ASB F1601 Keyboard Code
ASB F1601 Data Transmission Code
5.0 ERRORS

Various types of errors can be detected by the Assembler. Message is emitted following the statement which contains the error. The error messages and their meanings follow.

$ERRORS ILLEGAL CHARACTER X
The special character X (such as $, /, .) appears in the statement (not in the comment) or perhaps a required operand field is missing.

$ERRORS MULTIPLY DEFINED SYMBOL XXXXXX
The symbol XXXXXX has been defined more than one time.

$ERRORS UNDEFINED SYMBOL XXXXXX
The symbol XXXXXX has been used but never defined.

$ERRORS ILLEGAL NUMERIC CONTAINS CHARACTER X
An octal number includes an illegal digit (such as 8 or 9) or the numeric contains non numeric characters.

$ERRORS ILLEGAL OPCODE XXX
The operation code XXX is not one of the acceptable mnemonics.

$ERRORS MISSING OPERAND FIELD
No operand found for an operation code which requires one.

$ERRORS ILLEGAL VALUE = YYYYY, MAXIMUM = XXXXXX
The numeric value of an octal or decimal number of an expression has overflowed its limit.

| XXXXXX=  | 377B  | for 1 byte operands or data word |
| XXXXXX=  | 3777B | for 2 byte operands |
| YYYYYY=  | 7     | for input device numbers |
| YYYYYY=  | given operand value |

$ERRORS ILLEGAL SYMBOL
A location field contains a symbol that has more than six characters or that does not start with an alphabetic.

$ERRORS MISSING LABEL
The label, which is required by the EQU pseudo operation, is missing.

$ERRORS SYMBOL TABLE OVERFLOW, MAXIMUM = XXXXXX
Too many symbols in source program to fit into allocated symbol table.

$ERRORS LINE OVERFLOW, MAXIMUM = XXXX
Input line exceeds 48 characters; or missing carriage return.

$ERRORS ERRONEOUS LABEL
Opcodes END and ORG may not have a label.

$ERRORS ILLEGAL ORIGIN XXXXXX is less than XXXXXX
Value of new origin is less than current program count.

$ERRORS ILLEGAL OPERAND
DAD opcode requires symbolic operand

6.0 SYSTEM OPERATION

Source programs may be entered directly from the terminal keyboard or through a paper tape reader into a file. The user can then edit the source program by calling the EDITOR routine. After editing, the user calls and runs the ASSEMBLER routine.

6.1 Output Control

At the conclusion of the Assembly process, the user can request the following output:

- Local binary object tape
- Remote binary object tape
- Local program listing
- Remote program listing
- Local source statement listing
- Remote source statement listing
- Local symbol table listing
- Remote symbol table listing
- Remote card object deck
6.2 Binary Output
The formatted object code is punched out on request in sequence on 8 level paper tape.

6.3 Program Listing
The printout of the program listing will have the following format:

Columns

1-5 Location (octal) of first byte of object code
6-7 Blank
8-10 First byte object code word in octal
11 Blank
12-14 Second byte object code word in octal
15 Blank
16-18 Third byte object code word in octal
19 Blank
20-22 Fourth byte object code word in octal
23-24 Blank
25-72 First 48 characters of source statement

B. Tymshare User's Guide for Assembly
This section contains the operating procedure for the Tymshare PDP-10 version of the assembler. Information on manipulation and editing of files is contained in the TYMEX and EDITOR reference manuals distributed by Tymshare.

The assembly language is described in Section A of this appendix. In addition to the standard features, the Tymshare PDP-10 version of the assembler permits the use of tabs in place of blanks (outside ASCII string constants), simplifying formatting of the assembly listings. ("Tabs" are set in every eighth column in the PDP-10 system.)

To use the assembler, the user must create an assembly language source file on the disk. This file may not contain line numbers. The file name consists of one to five characters with the file name extension ".DAT".

To start the assembly, type:

```
RUN (UPL) ASM8
```

in either the TYMEX or PDP-10 mode. The assembler will request the input (source) file name. The user replies by typing the file name exclusive of the .DAT file name extension. For example, if the source file is named SRC,DAT, the reply is SRC.

When the assembly is complete, the assembler will type a stop message and return to the monitor. Output files from the assembler may then be listed or punched on the user's terminal.

Three output files are produced by the assembler:

- LOGOU,DAT contains the assembly listing
- LOGBI,DAT contains the 1601/1701 object tape
- LOGMI,DAT contains intermediate pass code (this file may be deleted to reduce storage charges)

The output from the assembler is described in Section A of this appendix. Section F contains an example of the assembly language listing.

C. General Electric User's Guide for Assembly
This section contains the operating procedure for the General Electric version of the assembler. Information on manipulation and editing of files is contained in the COMMAND SYSTEM and EDITING COMMANDS reference manuals distributed by General Electric. The assembly language is described in Section A of this appendix.

To use the assembler, the user must create an assembly language source file on the disk. This file may not contain line numbers. The file name consists of one to eight characters. Output files for the assembler must already exist or be created before starting the assembler. The files referenced are LOGOUT, LOGMID, and LOGBIN. All of these files are sequential ASCII. No password is permitted for any assembler file.
To start the assembler, type:

OLD ASM8

When the program prints "READY", type:

RUN

The assembler will request the input file name. The user replies by typing the source file name of the file to be assembled.
When the assembly is complete, the assembler will type a stop message and return to the monitor. Output files from the assembler may then be listed or punched on the user's terminal.

Three output files are produced by the assembler:

LOGOUT contains the assembly listing
LOGBIN contains the object tape
LOGMID contains intermediate pass code (this file may be deleted to reduce storage charges)

The output from the assembler is described in Section A of this appendix. Section D contains an example of the assembly language listing (leading zeroes are suppressed by the General Electric version of the assembler).

D. Sample Program Assembly

```
*************** SYMBOLO VALUE
***************
 1: MUL  0000
 2: MUL  0000 0013
 3: MULSP 0025
 4: UMUL 0030
 5: UMUL 0040
 6: UMUL  0042
 7: UMULP 0054
 8: DIV  0061
 9: DIVSP 0076
10: DIVSP 0018
11: DIVSP 0040
12: UMUL 0044
13: UMUL 0046
14: UMULP 0051
15: UMULP 0073
16: ONEG 0080

***************
LOC OBJECT CODE SOURCE STATEMENTS
***************
00000 MUL - SIGNED INTEGER MULTIPLY
00000 CALL I ARGUMENTS IN C & D
00000 EXIT HI ORDER PRODUCT IN D
00000 LO ORDER PRODUCT IN C
00000 REGS A,B,C,D,E, AND FLAGS ALTERED
00000 TIME 874 TO 1498 MICROSECONDS (AMPS)
00000 258 MUL XRA 13 COUNT AND NEGATE
00001 349 LEA 14 NEGATIVE ARGUMENTS
00002 222 SUC
00003 168 013 000 JSR MUL008
00004 150 013 000 JSR MUL008
00011 320 LDA
00012 240 INC
00013 258 MUL008 XRA
00014 223 SUC
00015 168 025 000 JSR MUL019
00016 150 025 000 JSR MUL019
00023 338 LDA
00024 338 INC
00025 384 MUL019 LAC 2) MOVE COUNT MOD 4
00026 032 RAN TO CARRY
00027 168 036 000 CALL UMUL 31 CALL UNSIGNED
00032 142 204 000 CTC ONEG MULTIPLY, IF CARRY
00035 087 NEGATE RESULT, EXIT
00036 UMUL - UNSIGNED INTEGER MULTIPLY
00036 CALL I ARGUMENTS IN C & D
00036 EXIT HI ORDER PRODUCT IN D
00036 LO ORDER PRODUCT IN C
00036 REGS A,B,C,D,E, AND FLAGS EXCEPT CARRY ALTERED
00036 TIME 890 TO 1390 MICROSECONDS (AMPS)
00036 UMUL - MULTI-PRECISION MULTIPLY ENTRY
00036 (BEC = C * D + B)
00038 076 016 000 UMUL LBL 8
00038 046 011 UMULS LEL 9
00042 382 UMULS LAC
00043 232 UMULS RAN 1) ROTATE CARRY INTO
```

82
#0044 120  LCA  Shared register.
#0045 041  DCE  Forcing next LSB.
#0046 093  RTS  To carry.
#0047 191  LAB  2) Exit if 8th iteration.
#0058 108 05A 88H  JFC  UNMUL$ 3) If step (6) set carry.
#0059 230  ADD  Add multiplicand to.
#0054 032  RAR  Product.
#0056 11A 000 51H  JMP  UNMUL$ 4) Rotate most significant.
#0056 104 042 080  JMP  UNMUL$  Product and go to (1).
#0061 00H 001  DIV  5) Signed integer divide.
#0061 00H 001  CALL  HI Order Dividend in B.
#0061 00H 001  LG Order Dividend in C.
#0061 00H 001  DIV  Division in D.
#0061 00H 001  EXIT  Quotient in C.
#0061 00H 001  REMAINDER  in B.
#0061 00H 001  OVERFLOW  Flag in carry (CY).#0061 00H 001  REGS A.B.C.D.E. and flags are altered.
#0061 00H 001  TIME  724 to 1456 microseconds (#S).
#0061 00H 001  DIV  XRA  XRA  1) Count and negate.
#0062 340  LEA  Negative arguments.
#0063 31A  SUB  0.
#0064 16C 076 00H  JTS  DIVMOD.
#0065 270 076 00H  JTE  DIVMOD.
#0067 444  INC  0.
#0073 180 204 00H  CAL  DNEG.
#0076 20B  DNEG  XRA  0.
#0077 223  SUB  0.
#0079 103 110 00H  JTS  DIVMOD.
#0083 150 110 00H  JTE  DIVMOD.
#0087 07A  LOA  0.
#0087 049  DIVER  2) Move count MOD 2.
#0087 093  RAR  3) Carry.
#0087 146 048 00H  DIVER  4) Call 'UDIV'.
#0087 120 030  RAR  #0 if overflow.
#0087 146 04C 00H  LOA  Overflow.
#0087 116 259  XRA  0.
#0087 26C  DHE  4) If carry was.
#0087 238 381  LAB  0.
#0087 223  SUB  0.
#0087 248 003  RFC  0.
#0087 259  XRA  0.
#0087 250  XRA  0.
#0087 22C  SUB  0.
#0087 25A  XRA  0.
#0087 276 140 00H  JWE  DIVMOD  Negate quotient.
#0087 250  XRA  0.
#0087 222  SUB  0.
#0087 258  XRA  0.
#0087 221  SUB  0.
#0087 258  XRA  0.
#0087 221  SUB  0.
#0087 258  XRA  0.
#0087 221  SUB  0.
#0087 258  XRA  0.
#0087 221  SUB  0.
#0087 258  XRA  0.
#0087 221  SUB  0.
#0087 258  XRA  0.
#0087 221  SUB  0.
#0087 258  XRA  0.
#0087 221  SUB  0.
#0087 258  XRA  0.
#0087 221  SUB  0.
#0087 258  XRA  0.
#0087 221  SUB  0.
#0087 258  XRA  0.
#0087 221  SUB  0.
#0087 258  XRA  0.
#0087 221  SUB  0.
#0087 258  XRA  0.
#0087 221  SUB  0.
#0087 258  XRA  0.
#0087 221  SUB  0.
#0087 258  XRA  0.
#0087 221  SUB  0.
#0087 258  XRA  0.
#0087 221  SUB  0.
#0087 258  XRA  0.
#0087 221  SUB  0.
#0087 258  XRA  0.
#0087 221  SUB  0.
#0087 258  XRA  0.
#0087 221  SUB  0.
#0087 258  XRA  0.
#0087 221  SUB  0.
#0087 258  XRA  0.
#0087 221  SUB  0.
#0087 258  XRA  0.
#0087 221  SUB  0.
APPENDIX III. MCS-8 SOFTWARE PACKAGE – SIMULATOR

A. Introduction
This Appendix describes the use of a FORTRAN IV program called INTERP/8. This program provides a software simulation of the INTEL 8008 CPU, along with execution monitoring commands to aid program development for the MCS-8. INTERP/8 accepts machine code produced by the INTEL 8008 Assembler, along with execution commands from a time-sharing terminal, card reader, or disk file. The execution commands allow manipulation of the simulated MCS-8 memory and the 8008 CPU registers. In addition, operand and instruction breakpoints may be set to stop execution at crucial points in the program. Tracing features are also available which allow the CPU operation to be monitored. INTERP/8 provides symbolic reference to storage locations as well as numeric reference in various number bases. The command language is described in the paragraphs which follow.

B. Basic Elements
All input to INTERP/8 is "free form". Numbers, symbolic names, and special characters may be placed anywhere within the input line (see margin commands in Section D). Comments may be interspersed in the input, but must be enclosed within the bracketing symbols /* and */.

1. Numbers. Numeric input to INTERP/8 can be expressed in binary, octal, decimal or hexadecimal. The letters B, O, Q, D, and H following the integer number indicates the base, as shown below:

<table>
<thead>
<tr>
<th>Number</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>11011B</td>
<td>11011₂</td>
</tr>
<tr>
<td>28D</td>
<td>28₁₀</td>
</tr>
<tr>
<td>33O</td>
<td>33₈</td>
</tr>
<tr>
<td>33Q</td>
<td>33₈</td>
</tr>
<tr>
<td>1CH</td>
<td>1C₁₆</td>
</tr>
<tr>
<td>28</td>
<td>28₁₀</td>
</tr>
</tbody>
</table>

A decimal number is assumed if the base is omitted. Note that although O is allowed to indicate octal integers, Q is also permitted to avoid confusion with the integer 0. Note that the leading digit of a hexadecimal number must be one of the digits 0, 1, ..., 9. Thus, EF₂₁₆ must be expressed as 0EF₂₁₆.

On output, INTERP/8 indicates octal integers with Q and omits the D on decimal values. The base used on output defaults to decimal, but may be changed by the user. (See the BASE command in Section C.)

2. Symbolic Names. Symbolic names are strings of contiguous alphabetic and numeric characters not exceeding 32 characters in length. The first character must be alphabetic. Valid symbolic names are:

- SYMBOLICNAME
- X3
- G1G2G3
- LONGSTRINGOFCHARACTERS

3. Special Characters. The special characters recognized by INTERP/8 are: $ = / ( ) + - * . . All other special characters are replaced by a blank.

C. INTERP/8 Commands
The commands available in INTERP/8 are summarized briefly below. Full details of each command are given in following paragraphs.

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD</td>
<td>Causes symbol tables and code to be loaded into the simulated MCS-8 memory.</td>
</tr>
<tr>
<td>GO</td>
<td>Starts execution of the loaded 8008 code.</td>
</tr>
<tr>
<td>INTER</td>
<td>Simulates an 8008 interrupt.</td>
</tr>
<tr>
<td>TIME</td>
<td>Displays time used in the 8008 simulation.</td>
</tr>
<tr>
<td>CYCLE</td>
<td>Allows the simulated CPU to be stopped after a given number of cycles.</td>
</tr>
<tr>
<td>TRACE</td>
<td>Enables tracing feature when particular portions of the program are executed.</td>
</tr>
<tr>
<td>REFER</td>
<td>Causes the CPU simulation to stop when a particular storage location is referenced.</td>
</tr>
<tr>
<td>ALTER</td>
<td>Causes the CPU simulation to stop when the contents of a particular memory location is altered.</td>
</tr>
<tr>
<td>CONV</td>
<td>Displays the values of numbers converted to the various number bases.</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>Displays memory locations, CPU registers, symbolic locations, and IO ports.</td>
</tr>
<tr>
<td>SET</td>
<td>Allows the values of memory locations, CPU registers, and IO ports to be altered.</td>
</tr>
<tr>
<td>BASE</td>
<td>Allows the default number base used for output to be changed.</td>
</tr>
<tr>
<td>PUNCH</td>
<td>Causes output of machine code in BPNF format.</td>
</tr>
<tr>
<td>END</td>
<td>Terminates execution of an 8008 program.</td>
</tr>
</tbody>
</table>
The commands NOTRACE, NOREFER, and NOALTER are also defined. These commands negate the effects of TRACE, REFER, and ALTER, respectively. In all cases, the commands may be abbreviated (but not misspelled!). These abbreviations are indicated with the command description.

Commands are typed anywhere on the input line, with as many commands on a line as desired. The symbol "." must follow each command.

The end of data for the execution of INTERP/8 is indicated by a "$EOF" starting in column 1 of the last card.

1. Range-Lists. Many of the INTERP/8 commands accept a "range-list" as an operand. Tracing, for example, can be enabled for a specific range of addresses in the program. The range-list specifies a sequence of contiguous addresses in memory, or a range of numeric values to which the command is applied.

In its simplest form, a range-list is a number (binary, octal, decimal, or hexadecimal), or it may be a pair of numbers separated by the symbol "TO." Thus, valid range-lists are:

- \(10\)
- \(63Q\)
- \(50\ TO\ 63Q\)
- \(0FH\ TO\ 11001111B\).

A range-list, however, can also reference a symbolic location, with or without a numeric displacement from the location. Suppose, for example, the symbols START and INCR appear at locations 10 and 32 in the source program. Valid range-lists involving these symbols are:

- \(\text{START}\)
- \(\text{START} + 6\)
- \(\text{START} - 101B\)
- \(10\ TO\ \text{INCR}\)
- \(\text{INCR} - 2\)

The range-list may also contain a reference to the current value of the program counter of the simulated 8088 CPU. The symbol "*" represents this value. If the value of the program counter is 16, for example, the following is a valid range-list:

- \(\text{START\ TO\ *}\)

The exact use of the range-list is illustrated with the individual commands.

2. Notation. The following notation is used to describe the INTERP/8 command structure. Elements enclosed within braces \{ and \} are optional, while elements enclosed within the brackets [and ] are alternatives, where at least one alternative must be present.

A range-list, for example, can be specified as:

- \(\text{range-element}\ \{\ \text{TO}\ \text{range-element}\}\)

where a range-element is defined as:

\[
\begin{align*}
\text{number} & \rightarrow \{ + \text{\ number}\} \\
\text{symbolic-name} & \rightarrow \{ - \text{\ number}\} \\
* & \end{align*}
\]

As mentioned previously, command names can always be abbreviated. The required portion of the command is underlined in the command description. The symbol "TO" in the range list can be abbreviated as "T." Thus, the range list above can be redefined as:

- \(\text{range-element}\ \{\ \text{TO}\ \text{range-element}\}\).

Finally, the ellipses "..." indicate a list of indefinite length.

The commands are given alphabetically in the following paragraphs starting with a prototype statement using the above notation. A brief description is then given, followed by examples.

3. \([\text{ALTER}\ NOALTER}\] range list \{, range-list, range-list, ..., range-list\}.

The ALTER command is an operand breakpoint command which causes the execution of the 8088 CPU to stop whenever an attempt is made by the CPU to store values into a memory location specified in the range-list. When the breakpoint is encountered, INTERP/8 prints ALTER \(x\), where \(x\) is the value of the program counter. Execution can be started again with the GO, RUN, or INTER commands. Examples of the command are:

- \(\text{ALTER 0}\)
- \(\text{ALTER 0 TO 10}\)
- \(\text{ALTER 10 T INCR}\).
- \(\text{ALTER START + 2 TO INCR - 0AH}\)
- \(\text{AL 5, START, X2, 7 T 10, INCR-3}\)
4. BASE

This command causes the INTERP/8 system to use the number base specified by the second argument when printing results. This command has no effect on the number bases which are acceptable in the input.

5. CONV range-list { , range-list, range-list, ... , range-list }

The conversion command prints the values of the numbers specified in the range-list in binary, octal, decimal, and hexa-decimal forms. Examples are:

CONV 23
CONV*.
CON 10 TO START + 3
CO 10, 30, 280, 1101B T 33H

6. CYCLE Number

The cycle command causes a breakpoint to occur when the CPU cycle count reaches its current value plus the number specified in the cycle command (see the GO command, also).

7. DISPLAY display-element { , display-element, ... , display-element }

The display command causes the values of memory locations, symbolic names, CPU registers, and I/O ports to be printed. The output form of these values is determined by the current default base (see the BASE command). The width of the output line determines the output formatting (see the SWIDTH command of Section D).

In its simplest form, a display-element can be one of the 8008 CPU registers:

- CY (carry)
- D (entire program stack)
- Z (zero)
- E PS 0
- S (sign)
- H PS 1 (program stack elements)
- P (parity)
- L ...
- A HL (H&L) PS 7
- B SP (program stack pointer)
- C PC (program counter)

In this case, valid DISPLAY commands are:

DISPLAY CY
DISPLAY CY, Z, H, HL.
D P, A, PS 0.

A display-element can also be the symbol CPU, in which case all registers are displayed.

The values latched into the I/O ports can be displayed by using a display element of the form:

PORT range-list

The ports specified in the range-list (between 0 and 31) are printed. Examples are:

DISPLAY PORT 0
DI PO 3, PO 5, PORT 5 TO 8, PO 1001B

The contents of the symbol table can be examined by using a display-element of the form:

SYMBOLS { symbolic-name [number] }

The form

DISPLAY SYMBOLS.

prints the entire symbol table, while the form

DISPLAY SYMBOLS number.

responds with the symbolic name (± a numeric displacement) which is closest to the address specified by the number.

Examples are:

DISP SY.
DI SY OFFH, SY 32

If the symbol "" is used in the command, the symbolic location closest to the current program counter is printed.

The values contained in memory locations can also be displayed. In this case, the display-element takes the form

MEMORY range-list

[CODE]
[CODE]
[CODE]

86
The range of elements printed is specified in the range-list, while the form of the elements in the display is controlled by the command CODE (decoded instructions) or one of the number bases. If the form is omitted, the default number base is used in the display (see the BASE command). Valid DISPLAY commands are:

- DISPLAY MEMORY 20.
- DISP MEM 20 TO 30H.
- DI M START T START+5.
- DI MEM 0 TO 30 CODE.
- D M 0 T 30 D, M 40 TO INCR+10 OCT.

The various display-elements may be mixed in a single DISPLAY command.

8. END.
The END command reinitializes the INTERP/8 system. If another program is subsequently loaded into memory, all break and trace points are reset. Otherwise, the currently loaded program may be rerun with all break and trace points remaining.

9. GO \{\text{\text{\textit{* number}}}}

The GO command causes the execution of the loaded program to begin. In the case that a break point was previously encountered, the execution continues through the breakpoint. If the GO is followed by a \text{\textit{*}}, the breakpoint addresses are printed as they are encountered, but the 8008 CPU does not halt until completion. If the GO is followed by a number, the effect is exactly the same as CYCLE number. GO.

10. INTER \{\text{number \{ number \{ number \}}}}

The INTER command simulates the 8008 interrupt system. The numbers which follow the INTER command correspond to an instruction and its operands which will be “jammed” into the instruction register. If no instructions follow the INTER command, the instructions from the last interrupt are used. If no previous command has been specified, a LAA (NOP) instruction is used. The INTER command causes the simulated execution to continue. Examples are:

- INTER.
- INT.
- INTER 00010101B (this is an RST 20).

11. LOAD number \{ number \}

The LOAD command reads the symbol table and 8008 machine code into the simulated memory. The form LOAD number, reads only the machine code from the file specified by number (see file numbering in Section D). The form LOAD number number, reads the symbol table from the file specified by the first number and the machine code from the second file. The symbol table is in the form produced by the 8008 assembler (i.e., the first part of the listing file), and the machine code is in “BNPFF” format (see PROM programming specifications in the INTEL Data Catalog). This format is also produced by the INTEL 8008 assembler. The end of the code file is indicated by a “$” appearing in the input. INTERP/8 responds to this command by printing the number of locations used by the program. Examples are:

- LOAD 1.
- LOAD 6 7.

12. \text{\text{\textbf{REFER NOREFER}}} range-list \{ \text{range-list}, \ldots, \text{range-list} \}

This command is similar to the ALTER command except that a breakpoint occurs whenever any reference to the memory location takes place. Thus, an instruction fetch, an operand fetch, or an operand store all cause a breakpoint when this command is used. Examples are:

- REFER 10.
- RE 10 TO 30Q.
- REF 5, 7, START TO START + 5, 71Q.
- NOREF 0 TO 10.

13. RUN.
The RUN command has exactly the same effect as the command GO \text{\textit{*}}.

14. SET. set-element \{ set-element, \ldots, set-element \}

The SET command allows memory locations, CPU registers, and IO ports to be set to specific values. The register names described under the DISPLAY command can be used in the set-element:

\text{\textbf{register = \{ number * \}}}

87
The value of the specified register is set to the number following the "=" or to the value of the program counter if "#" is specified. Thus, valid commands are:

```
SET Z = 0
SE A = 3, B = 77Q, PS 0 = 0EEH,
S HL = 28.
```

A set-element can also be the symbol "CPU" in which case all registers are set to zero, including the simulated 8008 timer.

Examples are:

```
SET CPU.
S CP, PC = 25.
```

The values of IO ports can also be set by using a set-element of the form

```
PORT range-list = number { number number ... number }
```

In this case, the IO ports specified in the range-list are set to the list of numbers following the "=". If more ports are specified than there are numbers in the list, the numbers are reused starting at the beginning. Examples are:

```
SET PORT 5 = 10.
SET PO 6 TO 8 = 1 2 3
S PO 10 TO 13 = 77Q 2,
S PO 8 = 10B, PO 12 = 13H, PO 30Q = 16.
```

The values contained in memory locations can be altered directly by using a set element of the form

```
MEMORY range-list = number { number ... number }
```

As in the case of IO ports, the memory locations are filled from the list to the right of the equal sign, with numbers being reused if the list is exhausted. Examples of this command are:

```
SET MEMORY 0 = 0,
S MEM 0 TO 50 = 0.
```

The SET command does not change break or trace points which are in effect.

```
S M START TO START +5 = 111110008 22Q 33H.
```

As in the DISPLAY command, set-elements of each type may be intermixed:

```
SET CP, CY=0, M 5 = 10, PO 6=12, PC = 30.
```

15. **TIME.**

The TIME command causes INTERP/8 to print the number of states used by the simulated 8008 CPU since the last LOAD, END, or SET CPU command.

16. **TRACE**

```
TRACE range-list { , range-list, . . . , range-list }
```

The TRACE command causes the INTERP/8 system to print the CPU register contents and the decoded instruction whenever an instruction is fetched from the memory region specified in the range-list. The form of the elements in the trace is defined by the current default base (see BASE command). The trace shows the register contents and operation code before the instruction is executed. The result of the operation is found in the next line of the trace, or through the DISPLAY CPU command.

A heading showing the various columns in the trace is printed after each tenth line of the trace. Examples of the TRACE command are:

```
TRACE 0 TO 100.
TR START TO START + 11B.
NOTRACE START, INCR, FOUND TO FOUND+3, 7Q.
```

17. **PUNCH**

```
punch-list { number }
```

The PUNCH command causes the specified region of the simulated memory to be output in the BPNF format. If the number is present, the code is written into the corresponding INTERP/8 output file; otherwise the currently defined file is used. Examples are:

```
PUNCH 0 TO 0FFH.
PU START TO FINISH.
```

D. **I/O Formatting Commands**

INTERP/8 has a generalized I/O formatting interface which is somewhat dependent upon the installation. In general, a number of files are defined by file numbers (not necessarily corresponding externally to FORTRAN unit numbers). These file numbers correspond to devices as follows:
### INPUT

<table>
<thead>
<tr>
<th>INTERP/8 No.</th>
<th>Device</th>
<th>PDP-10 Device</th>
<th>File Name</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>User's Console</td>
<td>TTY 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Card Reader</td>
<td>CDR 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Paper Tape</td>
<td>PAP 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Magnetic Tape</td>
<td>MAG 16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Magnetic Tape</td>
<td>DEC 9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Disk</td>
<td>DISK 20</td>
<td>FOR20.DAT</td>
<td>LOGOUT</td>
</tr>
<tr>
<td>7</td>
<td>Disk</td>
<td>DISK 21</td>
<td>FOR21.DAT</td>
<td>LOGBIN</td>
</tr>
</tbody>
</table>

### OUTPUT

<table>
<thead>
<tr>
<th>INTERP/8 No.</th>
<th>Device</th>
<th>PDP-10 Device</th>
<th>File Name</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>User's Console</td>
<td>TTY5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Printer</td>
<td>PTr 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Paper Tape</td>
<td>PAP 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Magnetic Tape</td>
<td>MAG 17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Magnetic Tape</td>
<td>DEC 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Disk</td>
<td>DISK 22</td>
<td>FOR22.DAT</td>
<td>Disk φ1</td>
</tr>
<tr>
<td>7</td>
<td>Disk</td>
<td>DISK 23</td>
<td>FOR23.DAT</td>
<td>Disk φ2</td>
</tr>
</tbody>
</table>

I/O functions are controlled through "$" commands which may be interspersed throughout the input.

Any input line with a "$" in column one, followed by a non-blank character is considered an I/O command. The card is then scanned for an "$=n$" followed by a decimal integer. The character following the "$" and the integer value affect the I/O formatting functions as follows:

<table>
<thead>
<tr>
<th>Control</th>
<th>Meaning</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$COUNT=n$</td>
<td>Start the output line count at the value n.</td>
<td>1</td>
</tr>
<tr>
<td>$DELETE=n$</td>
<td>Delete all characters after column n of the output</td>
<td></td>
</tr>
<tr>
<td>$EOF=1$</td>
<td>End-of-file on this device</td>
<td>0</td>
</tr>
<tr>
<td>$INPUT=n$</td>
<td>Read subsequent input from file number n</td>
<td>1</td>
</tr>
<tr>
<td>$LEFT=n$</td>
<td>Ignore character positions 1 through n-1 of the input.</td>
<td>1</td>
</tr>
<tr>
<td>$OUTPUT=n$</td>
<td>Write subsequent output to file number n.</td>
<td>1</td>
</tr>
<tr>
<td>$PRINT=n$</td>
<td>Controls listing of the output. If n = 0, input lines are not printed; otherwise input is echoed.</td>
<td>0</td>
</tr>
<tr>
<td>$RIGHT=n$</td>
<td>Ignore all character positions beyond column n of the input.</td>
<td>80</td>
</tr>
<tr>
<td>$TERMINAL=n$</td>
<td>INTERP/8 assumes conversational usage if n = 1; otherwise batch processing is assumed.</td>
<td>1</td>
</tr>
<tr>
<td>$WIDTH=n$</td>
<td>This command sets the width of the output line. Note that this affects the format of the DISPLAY MEMORY command.</td>
<td>72</td>
</tr>
</tbody>
</table>

The default values shown above assume conversational use with a teletype or similar device. The defaults can easily be changed by recompiling the INTERP/8 program.

In the case of controls which take on only 0 or 1 values (e.g., $PRINT$, $TERMINAL$, and $EOF$), the equal sign and decimal number may be omitted. The value of the control is complemented in this case.

### E. Error Messages

#### ERROR MESSAGES

1. PROGRAM COUNTER STACK OVERFLOW
2. PROGRAM COUNTER STACK UNDERFLOW
3. PROGRAM COUNTER OUTSIDE SIMULATED MCS-8 MEMORY
4. MEMORY REFERENCE

#### COMMAND MODE ERRORS

1. REFERENCE OUTSIDE SIMULATED MCS-8 MEMORY
2. INSUFFICIENT SPACE REMAINING IN SIMULATED MCS-8 MEMORY
3. END-OF-FILE ENCOUNTERED BEFORE EXPECTED
4. INPUT FILE NUMBER STACK OVERFLOW (MAX 7 INDIRECT REFERENCES)
5. UNUSED
F. Examples

Two sample INTERP/8 executions are given in this section which illustrate the commands available with the INTERP/8 system. The first example illustrates the basic commands. A simple program is constructed in the simulated MCS-8 memory. This program is then executed, showing the use of break and trace points. The second execution shows the use of symbol tables and 8008 code which is produced by the INTEL 8008 assembler. In each case, the actual commands which initiate the INTERP/8 system may vary from installation to installation.

```plaintext
.SET OK
.CY2SP A B C D E H L HL SP PSB
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
.SMW #255
.SMW #255
.END

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
.END

1101 1000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
.END

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
.END

.END

.END
```
THIS EXAMPLE SHOWS A COMPLETE ASSEMBLY AND INTERP/E EXECUTION

TYPE ASML.DAT
SAMPLE MCS-8 PROGRAM (PAGE 47 OF 8888 MANUAL)
START LLI 286
LNI 0
LOOP
LAM
CP1 46
JZT FOUND
CAL INCR
LAL
CP1 286
JZT LOOP
FOUND
RET
INCR
INH
IN
RET
END

JR ASML

PLEASE TYPE INPUT FILE NAME ASML

***************
8086 INTEL ASSEMBLER
***************

CPU TIME: 3.72 ELAPSED TIME: 9.73
NO EXECUTION ERRORS DETECTED

EXIT

**

.RENAME FOR28.DAT = LOGOU.DAT, FOR21.DAT = LOGBI.DAT
FILES RENAMED:
LOGOU.DAT
LOGBI.DAT

**

.TYPE FOR28.DAT
***************
SYMBOL VALUE
***************
1: START #8888
2: LOOP #8884
3: FOUND #8883
4: INCR #8884

***************
**

.TYPE FOR21.DAT

***************
**

AND THEN PUNCH THE CODE BETWEEN LOCATIONS 10 AND 20 (WE WILL USE
THE CONSOLE AS THE OUTPUT DEVICE ) */
PUNCH 10 TO 20 1.

END.

CPU TIME: 12.93 ELAPSED TIME: 46:12.73
NO EXECUTION ERRORS DETECTED
SET OK

08020 843 046 048 032 120 043 046 048 032 120 043
/* SET A COMPLETE TRACE OF THE PROGRAM /* TR 0 TO 0100.*/

TRACE OK

GO.

CYSEP A B C D E H L HL SP PS8

LII 200
0000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
LRI 0
0000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
LAM
0000 043 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
CPI 46
+0101 043 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
JZT 19
+0101 043 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
CAL 28
+0100 043 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
JNL
+0101 043 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
RFZ
+0101 043 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
LAL
+0101 201 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
CPI 2188

CYSEP A B C D E H L HL SP PS8

ŁII 201
0000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
JZT 4
+0101 201 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
LAM
+0101 046 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
CPI 46
+0101 046 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
JZT 19
+0101 046 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
RET
EXECUTION ERROR 2 AT 22
/* THE ERROR OCCURS BECAUSE THE PROGRAM TERMINATES WITH A RET */

RATHER THAN A HLT. FIX THE INSTRUCTION IN MEMORY /*

DI MEM 19.

00019 007
DI IN M'S 19 C.D.

00019 RET

SET M 19 = 0. DI MEM 19 C.D.

00019 HLT

SET OK
00019 HLT
NOTE 0 TO 100. SET CPU. GO.

TRACE OK
SET OK
HLT CYCLE 117
DI CPU

CYSEP A B C D E H L HL SP PS8

0101 046 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
/* THE PROGRAM TERMINATES CORRECTLY AFTER 117 MACHINE STATES */

TIME.

TIME=117
/* SET SELECTIVE BREAK POINTS */
REF START, INCR+1, LOOP, SET CPU, GO.

REFER OK
SET OK
REFER AT 8
DI SY +*

START
G.

REFER AT 4
DI SY +*, GO.

LOOP
REFER AT 21
DI SY +*, GO.

INCR+1
REFER AT 4
D SY +*

000000 0000000000H START
000004 0000000004H LOOP
000026 000000190013H FOUND
000028 00000014H INCR
NOREF START TO INCR+5.

REFER OK
/* SET SELECTIVE TRACE POINTS (TRACE AND REFER POINTS CAN BE IN EFFECT

AT THE SAME TIME, IF DESIRED) */

TR START, LOOP, FOUND, REFER FOUND, GO.

TRACE OK
/* SET SELECTIVE TRACE POINTS (TRACE AND REFER POINTS CAN BE IN EFFECT

AT THE SAME TIME, IF DESIRED) */

TR START, LOOP, FOUND, REFER FOUND, GO.

CYZSP A H C D E H L HL SP PS0
0191 046 000 000 000 000 000 000 201 000000019
SET CP, GO.

SET OK
000000 0000000000H START
000004 0000000004H LOOP
000026 000000190013H FOUND
000028 00000014H INCR
NOREF START TO INCR+5.

PORTS COMMANDS */

DI PORT 4.
PA=0
DI PORT 4, PO 3, PO 7 TO 164.
PA=0
P3=0
P7=0 P8=0
DI PO 28 TO 25.

P20=0 P2I=0 P22=0 P23=0 P24=0 P25=0
SET PORT 5 = 11001100B. PO 18H = 55H.

SET OK
DI POR 5 TO 17.

P5=0 P6=0 P7=0 P8=0 P9=0 P10=0 P11=0 P12=0 P13=0 P14=0 P15=0
P16=45 P
17=0
END.

$EOF
APPENDIX IV
TELETYPETE MODIFICATIONS

The SIM8-01 microcomputer systems and associated software have been designed for interface to a model ASR 33 teletype wired in accordance with the following description.

The ASR 33 teletype must receive the following internal modifications and external connections:

Internal Modifications

1. The current source resistor value must be changed to 1450 ohms. This is accomplished by moving a single wire. (See Figures 5 and 6.)

2. A full duplex hook-up must be created externally. This is accomplished by moving two wires on a terminal strip. (See Figures 4 and 6.)

3. The receiver current level must be changed from 60 mA to 20 mA. This is accomplished by moving a single wire. (See Figures 4 and 6.)

4. A relay circuit must be introduced into the paper tape reader drive circuit. The recommended circuit consists of a relay, a resistor, a capacitor and suitable mounting fixture. An alternate circuit utilizes a thyractor for suppression of inductive spikes. This change requires the assembly of a small “vector” board with the relay circuit on it. It may be mounted in the teletype by using two tapped holes in the mounting plate shown in Figure 1. The relay circuit may then be added without alteration of the existing circuit. (See Figures 2, 3, and 6.) That is, wire “A”, to be connected to the brown wire in Figure 2, may be spliced into the brown wire near its connector plug. The “line” and “local” wires must then be connected to the mode switch as shown. Existing reader control circuitry within the teletype need not be altered.

External Connections

1. A two-wire receive loop must be created. This is accomplished by the connection of two wires between the teletype and the “SIM” board in accordance with Figure 6.

2. A two-wire send loop similar to the receive loop must be created. (See Figure 6.)

3. A two-wire tape reader loop connecting the reader control relay to the “SIM” board must be created. (See Figure 6.)

Figure 1. Relay Circuit (Alternate)
**Figure 6. Schematic**

**Figure 7. Block Diagram**
APPENDIX V. PROGRAMMING EXAMPLES

A. Sample Program to Search A String Of Characters In Memory Locations 200-219 For A Period (.)

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>OPERAND</th>
<th>EXPLANATION</th>
<th>BYTES</th>
<th>LOCATION</th>
<th>ROM CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start:</td>
<td>LLI</td>
<td>Load L with 200</td>
<td>2</td>
<td>100</td>
<td>101</td>
<td>00110110</td>
</tr>
<tr>
<td></td>
<td>LHI</td>
<td>Load H with 0</td>
<td>2</td>
<td>102</td>
<td>103</td>
<td>00101110</td>
</tr>
<tr>
<td>Loop:</td>
<td>LAM</td>
<td>Fetch Character from Memory</td>
<td>1</td>
<td>104</td>
<td></td>
<td>11000111</td>
</tr>
<tr>
<td></td>
<td>CPI</td>
<td>&quot;&quot; Compare it with period</td>
<td>2</td>
<td>105</td>
<td>106</td>
<td>00111100</td>
</tr>
<tr>
<td></td>
<td>JTZ</td>
<td>Found If equal go to return</td>
<td>3</td>
<td>107</td>
<td>108</td>
<td>11010000</td>
</tr>
<tr>
<td></td>
<td>CAL</td>
<td>Call increment H&amp;L subroutine</td>
<td>3</td>
<td>110</td>
<td>111</td>
<td>01000110</td>
</tr>
<tr>
<td></td>
<td>LAL</td>
<td>Load L to A</td>
<td>1</td>
<td>113</td>
<td></td>
<td>11000110</td>
</tr>
<tr>
<td></td>
<td>CPI</td>
<td>Compare it with 220</td>
<td>2</td>
<td>114</td>
<td>115</td>
<td>00111100</td>
</tr>
<tr>
<td></td>
<td>JFZ</td>
<td>Loop If unequal go to loop</td>
<td>3</td>
<td>116</td>
<td>117</td>
<td>01001000</td>
</tr>
<tr>
<td>Found:</td>
<td>RET</td>
<td>Return</td>
<td>1</td>
<td>119</td>
<td></td>
<td>00000111</td>
</tr>
<tr>
<td>INCR:</td>
<td>INL</td>
<td>Increment L</td>
<td>1</td>
<td>60</td>
<td></td>
<td>00110000</td>
</tr>
<tr>
<td></td>
<td>RFZ</td>
<td>Return if not zero</td>
<td>1</td>
<td>61</td>
<td></td>
<td>00001011</td>
</tr>
<tr>
<td></td>
<td>INH</td>
<td>Increment H</td>
<td>1</td>
<td>62</td>
<td></td>
<td>00101000</td>
</tr>
<tr>
<td></td>
<td>RET</td>
<td>Return</td>
<td>1</td>
<td>63</td>
<td></td>
<td>00000111</td>
</tr>
</tbody>
</table>

Subroutine to Search for Period.

98
B. Teletype and Tape Reader Control Program (A0800)

```
BEGIN LAI 1
SUPPRESS TTY
OUT 12B
OUTPUT 2
XRA
CLEAR AC
OUT 13B
OUTPUT 3 - TAPE READER CONTROL
CAL TAPE
CALL FOR TAPE READER CONT RT*
XMP BEGIN

TAPE
LA1 I
TAPE READER ENABLE CODE
OUT 13B
OUTPUT 3 - ENABLE TAPE READER
CAL TTYD1
TAPE READER CONTROL DELAY

TTY
HLT
WAIT FOR TTY START PULSE
CAL TTYD2
TTY DELAY - 4.468 MSEC*
XRA
TAPE READER DISABLE CODE
OUT 13B
OUTPUT 3 - DISABLE TAPE READER
INP 08
INPUT 0* READ START PULSE
LCI 255
COMPLEMENT TTY START PULSE
XRC
EXCLUSIVE-OR REG C
OUT 12B
OUTPUT 2* OUTPUT START PULSE
LEI 246
TTY DATA SAMPLING COUNTER
INP 08
READ TTY DATA INPUT
LCI 255
COMPLEMENT TTY DATA
XRC
OUT 12B
OUTPUT 2* TTY DATA OUT
RAR
STORE TTY DATA
LAB
LOAD TTY DATA TO REC B
RAR
LAB
LOAD AC TO REG B
LBE
E = E + 1
JFZ TTYIN
JUMP IF ZERO F/F IS NOT SET
LAB
LOAD REG B TO AC
OUT 11B
OUTPUT 1* TTY CHARACTER
SUI 12B
REMOVE PARITY BIT
LEA
STORE TTY INPUT DATA
CAL TTYD1
LA1 I
OUT 12B
SUPPRESS TTY
RET

TTYD1
LDI 115
9.012 MSEC* DELAY
ST
D = D + 1
JFZ ST
RET

TTYD2
LDI 186
4.468 MSEC* DELAY
ST
D = D + 1
JFZ ST
RET
END
```

C. Memory Chip Select Decodes and Output Test Program (A0801)

```
BEGIN LAI 15
LOAD 15 TO AC
OUT 10B
WRITE TO OUTPUT 0
OUT 10B
OUT 11B
OUT 12B
OUT 12B
OUT 13B
OUT 14B
OUT 15B
OUT 16B
OUT 17B
CAL DELAY
CAL DELAY
CAL DELAY
CAL DELAY
XRA
CLEAR AC
OUT 10B
OUT 11B
OUT 12B
OUT 13B
OUT 14B
OUT 15B
OUT 16B
OUT 17B
LCI 240
LOAD 240 TO REC C
LLL 252B
LOAD 252B(COTCAL) TO REC C
LHI 0
LOAD 0 TO REC H
CSTEST
LAH
OUT 10B
LAL
OUT 11B
XRA
CLEAR AC
LMA
WRITE AC TO MEMORY
CAL DELAY
CAL DELAY
INH
H = H + 1
INC
C = C + 1
JFZ CSTEST
JMP BEGIN

DELAY
LDI 0
LOAD O TO REC D
DI
IND
D = D + 1
JFZ DI
RET
END
```

D. RAM Test Program (A0802)

```
BEGIN LAI 0
LOAD 0 TO AC
OUT 10B
WRITE TO OUTPUT 0
OUT 11B
WRITE TO OUTPUT 1
OUT 12B
WRITE TO OUTPUT 2
OUT 13B
WRITE TO OUTPUT 3
LBI 8
LOAD 8 TO REC B
LCI 0
LOAD 0 TO REC C
LHI 8
LOAD 8 TO REC H
LLI 0
LOAD 0 TO REC L
XRA
CLEAR AC
LMA
LOAD AC TO MEMORY
INL
L = L + 1
CPL
AC = L
JFZ LM2
JUMP IF AC IS NOT ZERO
INH
H = H + 1
LA1 12
LOAD 12 TO AC
CPI
AC = H
JFZ LM1
JUMP IF AC IS NOT ZERO
LHI 8
LOAD REC B TO AC
OUT 10B
OUT 10B
RESET
LLC
LOAD REC C TO L
LAC
LOAD REC C TO AC
OUT 13B
LA1 255
LOAD 255 TO AC
LMA
LOAD AC TO MEMORY
CPI
AC = M
JFZ LP3
JUMP IF AC IS NOT ZERO
LAH
LOAD REC H TO AC
OUT 10B
UF
LOAD REC C TO H
XRA
CLEAR AC
INL
L = L + 1
CPL
AC = L
JFZ REPT
JUMP IF AC = 0
LAL
LOAD REC L TO AC
XRA
CLEAR AC
CPI
AC = M
JFZ REPT
JUMP IF AC IS NOT ZERO
JMP REPT
LA1 12
CPH
JFZ CONT
XRA
CPI
JFZ ERROR
JUMP IF AC IS NOT ZERO
CONT
LHI
XRA
INC
INH
C = C + 1
CPC
AC = C
JFZ REPT
JUMP IF B = 0
LHI
LOAD REC P TO H
LHI
LOAD REC L TO AC
XRA
CLEAR AC
CPI
AC = B
JFZ REPT
JUMP IF AC IS NOT ZERO
JMP REPT
LHI
LOAD REC C TO AC
ADH
OUT 10B
LOAD REC L TO AC
LAL
OUT 11B
LOAD MEMORY TO AC
OUT 12B
LOAD REC C TO AC
LAC
OUT 13B
MTH
END
```
APPENDIX VI

The widespread usage of low-cost microcomputer systems is made possible by Intel’s development and volume production of MCS-8 microcomputer sets. To make it easier to use these sets, Intel now offers complete 8-bit modular microcomputer development systems called Intellec 8.

The Intellec modular microcomputers provide a flexible, inexpensive, and simplified method for developing OEM systems. They are self-contained, expandable systems complete with central processor, memory, I/O, crystal clock, power supplies, standard software, and a control and display console.

The major benefit of the Intellec modular microcomputers is that random access memories (RAMs) may be used instead of read-only-memories (ROMs) for program storage. By using RAMs, program loading and modification is made much easier. In addition, the Intellec front panel control and display console makes it easier to monitor and debug programs. What this means is faster turn-around time during development, enabling you to arrive at that finished system sooner.

The Intellec 8 Eight-Bit Microcomputer Development System. The Intellec 8 is a microcomputer development system designed for applications which require 8-bit bytes of data to perform either binary arithmetic manipulations or logical operations. The Intellec 8 comes complete with power supplies, display and control panel, and finished cabinet. It can directly address up to 16k 8-bit bytes of memory which can be any mix of ROMs, PROMs, or RAMs. The Intellec 8 is designed around the Intel 8008 central processor chip. There are 48 instructions including conditional branching, binary arithmetic, logical, register-to-register, and memory reference operations. I/O channels provide eight 8-bit input ports and twenty-four 8-bit output ports—all completely TTL compatible. The unit has interrupt capability and a two-phase crystal clock that operates at 800kHz providing an instruction cycle time of about 12.5μs.

Bare Bones 8. The Bare Bones 8 has the same capability as the Intellec 8 only it does not include the power supplies, front panel, or finished cabinet. It is designed as a rack-mountable version.

The Intellec 8 system comes with a standard software package which includes a system monitor, resident assembler, and text editor. The programmer can prepare his program in mnemonic form, load it into the Intellec 8, edit and modify it, then assemble it and use the monitor to load the assembled program.

Other development tools for the Intellec 8 include a PL/M compiler, cross assembler, and simulator designed to operate on large scale general purpose computers. PL/M, a new high-level language, has been developed as an assembly language replacement. A PL/M program can be written in less than 10% of the time it takes to write that same program in assembly language without loss of machine efficiency.

Standard Microcomputer Modules. Microcomputer Modules, standard cards that can be purchased individually so that the designer can develop his system with as little or as much as he needs, are also available. Additional CPU, Memory, Input/Output, PROM Programmer, Universal Prototype, and other standard modules provide developmental support and systems expansion capability.
MCS-8 MICROCOMPUTER DEVELOPMENT SYSTEMS

- Intellec 8 (imm8-80A): Complete Microcomputer Development System
  - Central Processor Module
  - RAM Memory Modules (8192 x 8)
  - Input/Output Module (TTL compatible)
  - PROM Memory Module (4k x 8 capacity; 1k Resident System Monitor included)
  - PROM Programmer Module
  - Control Console and Display
  - Power Supplies and Cabinet

- Bare Bones 8: MCS-8 System without power supplies, cabinet, or control console

- Standard Software
  - Resident System Monitor
  - Assembler
  - Text Editor
  - Requires 8k of RAM

The Intellec 8 is a complete microcomputer development system for MCS-8 microcomputer systems. Its modular design allows the development of any size MCS-8 system, and it has built-in features to make this task easier than it has ever been before.

The basic Intellec 8 (imm8-80A) consists of six microcomputer modules (CPU, 2-RAM, PROM, I/O and PROM programmer), power supplies, and console and displays in a small compact package. The heart of the system is the imm8-82 Central Processor Module. It is built around Intel's 8008-1, an 8-bit CPU on a chip. It contains all necessary interface to control up to 16k of memory, eight 8-bit input ports, twenty-four 8-bit output ports, and to respond to real time interrupts.

The Intellec 8 has 9k bytes of memory in its basic configuration and may be expanded up to a maximum of 16,384 bytes of memory. Of the basic 9k bytes of memory, 8192 bytes are random access read/write memory located on the imm8-26 RAM Memory Modules and are addressed as the lower 8k of memory. This memory may be used for both data storage and program storage. The remaining 1024 bytes of memory are located on the imm8-26 PROM Memory Module and addressed as the upper 1280 bytes of the 16k memory. This portion of memory is a system monitor in five 1702A PROMs. Eleven additional sockets are available on the imm8-26 for monitor or program expansion. Control for the PROM Programmer Module (imm6-76) is included with the monitor for system control.

PROM memory modules and RAM memory modules may be used in any combination to make up the 16k of directly addressable memory. Facilities are built into these modules so that any combination of RAM and ROM or PROM may be mixed in 256 byte increments.

Input and output in the Intellec 8 is provided by the imm8-60 I/O module. It contains four 8-bit input ports, and four 8-bit output ports. In addition it contains a universal asynchronous transmitter/receiver chip as well as a teletype driver, receiver, and reader control. Bit serial communication using only the teletype drivers, receivers, and the I/O port, is also possible with this module.

The universal asynchronous transmitter receiver chip may operate at either 110 baud for standard teletype interface or 1200 baud for communication with a high speed CRT terminal. Additional I/O modules, imm8-60, and output modules, imm8-62, can expand the I/O capability of the Intellec 8 to eight input ports and twenty-four output ports, all TTL compatible.

An interrupt line and an 8-bit interrupt instruction port is built into the imm8-82 Central Processor Module. When an interrupt occurs, the processor executes the instruction which is present at the interrupt instruction port. In the Intellec 8, both the interrupt line and the interrupt instruction port are connected to the console. The processor may be interrupted by depressing the switch labeled INT, and the interrupt instruction is entered in the ADDRESS/INSTRUCTION/DATA switches.

Additional module locations are available in the Intellec 8 so the user may develop his own custom interface using the imm6-70 Universal Prototype Module. All necessary control signals, data, and address buses are present at the connectors of the unused module locations for this expansion. When memory, I/O, and custom interfaces are added to the Intellec 8, care should be taken not to exceed the built-in power supply capability.

Every Intellec 8 comes with three basic pieces of software, the systems monitor, a resident program located in the upper 1280 bytes of memory, a symbolic assembler and a text editor. The resident system monitor allows the operator to punch and load tapes, display and alter memory, and execute programs.

With the PROM Programmer Module, 1702A PROMs may be programmed and verified under control of the system monitor.

The text editor is a paper tape editor to allow the operator to edit his source code before assembly. The assembler takes this source tape and translates it into object code to run on the Intellec 8 or any MCS-8 system.

The Intellec 8 microcomputer development system is also available in a Bare Bones 8 version. In this version the power supply, chassis, console, and display are removed leaving the user a compact rack mountable chassis to imbed in his own system.
INTELLEC 8 CONTROL CONSOLE AND DISPLAY

The Control Console directs and monitors all activities of the Intellec 8. Complete processor status, machine cycle conditions and operational control of all processor activity are provided, and additional controls facilitating program debugging and hardware checkout are included on the control console.

- **STATUS** is a display of the operating mode of the processor.
  1. **RUN** indicates the processor is running.
  2. **WAIT** indicates the processor is waiting for memory or I/O to be available.
  3. **HALT** indicates the processor is in a stopped state.
  4. **HOLD** indicates an I/O or memory access is in progress from the Control Console (occurs with WAIT or HALT).
  5. **SEARCH/WAIT** indicates the processor has executed instructions until the search address and pass counter settings have been reached. (See LOAD PASS 26, and SEARCH/WAIT 33)
  6. **ACCESSReq** indicates an I/O or memory access is pending from the Control Console.
  7. **INT_REQ** indicates an interrupt is pending from the Control Console (see INT 38).
  8. **INT_DISABLE** not applicable.

- **CYCLE** provides continuous display of the processor's machine cycle status.
  9. **FETCH** indicates the current machine cycle is fetching an instruction from memory.
  10. **MEM** indicates the processor is executing a memory read (PCR) or memory write (PCW) cycle, or, under manual control, a direct access to memory is in progress.
  11. **I/O** indicates the processor is executing an I/O read or write cycle (PCC) or, under manual control, a direct access to I/O is in progress.
  12. **DA** indicates a direct access to memory or I/O is in progress.
  13. **READ/INPUT** indicates a memory or input read operation is in progress.
  14. **WRITE/OUTPUT** indicates a memory or output write operation is in progress.
  15. **INT** indicates an interrupt cycle is in progress.
  16. **STACK** not applicable.

- **ADDRESS** is a display of memory and I/O address.
  17. **INDICATORS** 14-15 not applicable.
  18. **INDICATORS** 0-13 are a display of the address of memory being accessed during a Fetch, Read, Write, or during manual MEM ACCESS.
  19. **INDICATORS** 9-13 are a display of the I/O address during an input, an output, or during a manual I/O ACCESS.
**INSTRUCTION/DATA** is a display of the instruction or data.

20. **INDICATORS 0-7** are a display of the instruction or data between the processor and memory or I/O.

**REGISTER/FLAG DATA** is the display of the processor data bus during executions of an instruction (display is dependent upon instruction being executed).

21. **INDICATORS 0-7** are a display of the contents of the CPU data bus when the instruction is executed. In the case of move instructions, the contents of the source register is displayed. Flags C, P, Z, and S are a special case. The flag status appears in the lower four bits, only when an input instruction is executed.

**ADDRESS/DATA** These eight switches provide entry of address or data for manual or SENSE operation of the processor (see SENSE 30).

22. **MEM ADDRESS HIGH** The upper six bits of memory address for direct access or search operations are entered here.

23. **I/O ADDRESS** The five bit I/O address for manual I/O ACCESS is entered here.

24. **SENSE DATA** Data to be input during a SENSE mode operation is entered here (see SENSE 30).

**ADDRESS/INSTRUCTION DATA** These eight switches provide entry of data, address, and instructions during manual or interrupt operation of the processor.

25. **MEM ADDRESS LOW** The lower eight bits of memory address for direct access or search mode operation are entered here. **INT INST** During an interrupt cycle the interrupt instruction is fetched from here (see INT 38).

**DATA** Data to be deposited to memory or an output port during manual operation is entered here (see DEP 36, and DEP AT HLT 37).

**PASS COUNT** Data to be loaded into the pass count register is entered here (see LOAD PASS 26).

**ADDRESS CONTROL** These four switches control addressing of memory and I/O and loading of the search address during manual operation of the processor.

26. **LOAD PASS** Loads pass count into pass count register (PASS COUNT is the number of times the processor will iterate through the search address during a search operation before indicating SEARCH COMPLETE (see SEARCH-WAIT 33 and SEARCH COMPL 5)

27. **DECR** decrements the loaded address by one (see LOAD 29).

28. **INCR** increments the loaded address by one (see LOAD 29).

29. **LOAD** loads contents of address high and low into memory access register for manual direct access to memory or search mode operation (see MEM ACCESS 32, and SEARCH-WAIT 33).

**MODE** These five switches select the processor’s mode of operation.

30. **SENSE** causes the processor to input data from the SENSE DATA switches during execution of an input instruction instead of the addressed input port (see SENSE DATA 24).

31. **I/O ACCESS** provides access to any input port and control of any output port when the processor is in a WAIT mode.

32. **MEM ACCESS** allows access to and control of any location in memory when the processor is in the WAIT mode.

33. **SEARCH-WAIT** provides for execution of a program to a specific location, where the processor enters a wait mode and displays current system conditions.

34. **WAIT** causes the processor to go into a manual WAIT mode.

**CONTROL** These five switches provide operator control of the processor.

35. **STEP/CONT** provides single step execution of a program while the processor is in a WAIT mode or continuation of a program from the SEARCH COMPLETE condition.

36. **DEP** deposits an 8-bit word to memory or output during a memory or I/O access operation (see DATA 25).

37. **DEP AT HLT** deposits an 8-bit word to a selected memory location or output automatically during a programmed HLT (see DATA 25).

38. **INT** causes the processor to execute an interrupt cycle, fetching the interrupt instruction from the INT INST switches (see INT INST 25).

39. **RESET** causes processor to begin execution of program at memory location zero by resetting program counter to zero. All other registers remain unchanged.

**POWER and PROM PROGRAMMING**

40. **PRGM PROM PWR** Power switch for high voltage used with PROM programmer.

41. **POWER** Key operated main power switch.

42. **PRGM PROM** Zero insertion force socket for 1602A or 1702A PROM to be programmed.
SYSTEMS SOFTWARE

The Intellec 8 and Bare Bones 8 Microcomputer Development Systems come with three pieces of software: Resident System Monitor, Text Editor and Symbolic Assembler. The Text Editor and Assembler are supplied on paper tape and are loaded with the System Monitor.

SYSTEM MONITOR

- Loads and punches paper tape
- Displays and alters contents of memory
- Fills memory with constants
- Executes programs in memory
- Moves blocks of data in memory
- Programs 1602A or 1702A PROMs

The System Monitor is contained in five 1702A PROMs and is assigned to the upper 1280 words of memory, leaving the lower 15k of memory for program and data storage. This executive software allows the operator to load and punch BNPF or hexadecimal format tapes, display and alter memory, load constants to memory, move blocks of RAM memory, and execute user programs.

The System Monitor is extended by the control software for the imm6-76 programmer module, which gives the monitor the ability to program 1602A to 1702A PROMs as well as being able to load memory from already programmed PROMs for duplication and verify the contents of PROMs against master tapes.

ASSEMBLER

- Standard symbolic assembler
- Input via prepunched paper tape
- Output in 8008 object code

The Symbolic Assembler is a multiple pass type. During Pass 1 the assembler reads the source code from the paper tape and generates a symbol table for later use. During Pass 2 the assembler generates the assembly listing. Also at this time, any detectable errors such as undefined jumps or missing symbols are indicated by a diagnostic printout on the teletype. Pass 3 may now be run. It generates object code, and punches it on paper tape. [Requires a minimum of 8k x 8 of RAM.]

DEVELOPMENT SUPPORT:
PL/M COMPILER, ASSEMBLER and SIMULATOR

In addition to the standard software available with the Intellec 8, Intel offers a PL/M compiler, cross assembler, and simulator written in FORTRAN IV and designed to run on any large scale computer. These routines may be procured directly from Intel, or alternatively, designers may contact a number of nation-wide computer timesharing services for access to the programs. The output from both PL/M and the MCS-8 Assembler may be run directly on the Intellec 8 Microcomputer Development System.

PL/M Compiler: PL/M is a high level procedure-oriented systems language for programming the Intel MCS-8 microcomputer. The language retains many of the features of a high-level language, without sacrificing the efficiencies of assembly language. A significant advantage of this language is that PL/M programs can be compiled for either the Intel 8008 or future Intel 8-bit processors without altering the original program.

Assembler: The MCS-8 Assembler generates object codes from symbolic assembly language instructions. It is designed to operate from a timeshared terminal.

Simulator: The MCS-8 Simulator, called INTERP/8, provides a software simulation of the Intel 8008 CPU, along with execution monitoring commands to aid program development for the MCS-8.
SYSTEMS SPECIFICATIONS

Word Size: Data: 8 bits
Instruction: 8, 16, or 24 bits

Memory Size: 9k bytes Intellece 8/5k bytes Bare Bones expandable to 16k bytes

Instruction Set: 48, including: conditional branching, binary arithmetic, logical, register-to-register and memory reference operations

Machine Cycle Time: 12.5μs
System Clock: Crystal controlled at 800kHz ±0.01%

I/O Channels: 4 expandable to 8 input ports
24 output ports

Interrupt: Single Level
Direct Access to Memory: Standard via control console

Memory Cycle Time: 1μs
Operating Temperature: 0° to 55°C

DC Power Supplies: VCC = 5V, ICC = 12A*
VDD = -9V, ID = 1.8A*
VGG = -12V, IGG = 0.06A

DC Power Requirement: VCC = 5V±5%, ICC = 11A max., 6A typ.
VDD = -9±5%, ID = 1A max., 0.5A typ.
VGG = -12±5%, IGG = 0.03A max., 0.016A typ.

AC Power Requirement: 60Hz, 115 VAC, 200 Watts

Physical Size: Intellece 8: 7" x 17 1/8" x 12 1/4"
Bare Bones 8: 6 3/4" x 17" x 12"

Weight: 30 lb.

Standard Software: System Monitor
Resident Assembler
Text Editor

Support Software: PL/M Compiler
Cross Assembler
FORTRAN IV
Simulator

STANDARD SYSTEMS and OPTIONAL MODULES

Intellece 8 (imm8-80A) Standard System includes the following Modules and Accessories:
- Central Processor Module
- Control and Display Panel
- Input/Output Module
- Finished Cabinet
- PROM Memory Module
- Standard Software:
- PROM Memory Module (Two)
- System Monitor
- Chassis with Mother Board
- Resident Assembler
- Power Supplies
- Text Editor
- PROM Programming Module
- Chassis (rack mountable with Mother Board)

Bare Bones 8 (imm8-81) Standard System includes the following Modules:
- Central Processor Module
- Standard Software:
- Input/Output Module
- PROM Memory Module
- RAM Memory Module
- Resident Assembler *
- Chassis (rack mountable with Mother Board)

Optional Modules available for the Intellece 8 and Bare Bones 8:
- Additional I/O or Output Modules
- Additional RAM Memory Modules
- Universal Prototype Module
- Module Extender
- Rack mounting kit for Intellece 8

The standard Intellece 8 comes with the modules shown. Expansion capability of both I/O and Memory to a full MCS-8 system is provided by using open locations on the motherboard.

Intellece 8 and Bare Bones 8 Module Assignments
imm 8-82 CENTRAL PROCESSOR MODULE

- Complete Central Processor Module with system clocks, interface and control for memory, I/O ports, and real time interrupt
- The heart of this module is Intel's 8008-1 processor on a chip — p-channel silicon gate MOS
- 48 instructions, data oriented
- Accumulator and six working registers
- Direct addressing of up to 16,384 bytes of memory. (PROM, ROM, or RAM)
- Directly addresses eight input ports and twenty-four output ports
- Subroutine nesting to seven levels
- Real time interrupt capability
- Direct memory access capability
- Interface to memory, I/O and interrupt ports through separate TTL buses
- Two phase crystal clock — 800kHz
- 12.5μs instruction cycle

The imm8-82 Central Processor Module is a complete 8-bit parallel central processor unit. It contains complete control for interface to memory and I/O. This is the main module in Intel's Intellec™ 8 systems.

The imm8-82 is built around Intel's 8008-1 CPU on a chip. It executes 48 instructions including conditional branching, register to register transfers, arithmetic, logical and I/O instructions. Six 8-bit registers and an 8-bit accumulator are provided. Subroutines may be nested to seven levels. Real time interrupt capability is provided and the processor may directly address up to 16,384 bytes of memory.

The imm8-82 has a fourteen bit TTL compatible memory address bus, an 8-bit data output bus and an 8-bit memory data input bus. Memory read and write signals and the wait request signal provide interface at TTL levels to any type of memory (including PROM, ROM, and RAM). Asynchronous interface to slower speed memories (access > 1μs) is provided by the wait request signal. This causes the processor to wait for memory response to a read or write command.

The Central Processor Module directly addresses up to eight 8-bit input ports and twenty-four 8-bit output ports. The 5-bit I/O address is contained in the upper byte of the memory address bus. Addresses 0 through 7 are defined as input ports, and 8 through 31 as output ports. Control signals, I/O cycle, I/O in and I/O out, define the I/O cycle and its function. An 8-bit data output bus and an 8-bit data input bus, both TTL compatible, provide data channels in and out of the processor module.

Real time interrupt capability and direct memory access capability complete the list of functional features for the imm8-82. During an interrupt, the Central Processor Module responds to the instruction presented at the 8-bit interrupt instruction port. Unless the main program flow is altered by the interrupt instruction, the execution will continue where it left off before processing the interrupt. Eight bits of data including sign, carry, zero and parity flags are latched on a separate bus during the execution portion of most instructions.

The direct memory access capability allows an alternate source to access memory or I/O while temporarily suspending processor operation. At the end of this alternative access to memory, the processor may return to normal program execution.

All system timing is derived from a two phase crystal clock running at 800kHz. This gives a machine cycle time of 12.5μs ± 0.01% and provides an accurate timing source for software delay loops and other timing requirements.
Central Processor Module Specifications

Word Size: Instruction: 8, 16, or 24 bits
Data: 8 bits
Central Processor: 8008-1 CPU, 8 bit accumulator, six 8-bit registers, subroutine nesting to seven levels, interrupt capability, asynchronous operation with memory
Instruction Set: 48 including conditional branching, binary arithmetic, logical operations, register-to-register transfers, and I/O
Memory Addressing: Any combination of PROM, ROM and RAM up to 16,384 bytes
Memory Interface: Address: 14-bits TTL latching bus
Data: 8-bit TTL bus to and from memory
I/O Addressing: Input: Eight 8-bit input ports
Output: twenty-four 8-bit latching output ports
I/O Interface: 8-bit TTL compatible buses to and from CPU. 8-bit TTL latched bus with execution data including flags (sign, parity, zero, and carry information)

System Clock: Crystal controlled, 800kHz ± 0.01%
Processor cycle time: 12.5μs
Connector: Dual 50-pin on 0.125 in. centers.
Connectors in rack must be positioned on 0.5 in. centers minimum.
Wirewrap: P/N C800100 from SAE
P/N VP801C50E000A1 from CDC
Board Dimensions: 6.18 in. x 8.0 in. x 0.062 in. Board to be on 0.5 in. centers minimum
Operating Temp: 0°C to +55°C
DC Power
Requirements: \( V_{CC} = +5V ± 5\% \), \( I_{CC} = 2.2A \) max., 1.0A typical
\( V_{DD} = -9V ± 5\% \), \( I_{DD} = 0.06A \) max., 0.03A typical
Support Software: PL/M Compiler
Cross Assembler
Simulator
Written in FORTRAN IV

imm8-82 Block Diagram
imm6-28 RAM MEMORY MODULE

- 4096 8-bit bytes per module
- Static memory, no clocks required
- Interfaces with the imm8-82 8-bit Central Processor Module
- Single +5V power supply
- Low power requirements
- For use in expansion of Intellec 8 systems to 16k bytes of memory
- Built-in decoding of module select for expansion to 65k bytes of memory

The imm6-28 RAM Memory Module is a standard 4k x 8 memory module designed for use with the Intellec 8 Microcomputer Development System. This module contains address and data buffers, read/write timing circuits and is implemented with Intel's 2102 1k x 1 static RAM. Although the basic memory module is 4096 x 8, configurations as small as 1024 x 8 are also available.

The imm6-28 RAM Memory Module is used with the MCS-8 Micro Processor in configurations of up to 16k bytes of memory (4 modules). The imm8-82 Central Processor Module directly interfaces with the imm6-28 RAM Memory Module with all module select decoding done directly on the connector. This allows an imm6-28 to be moved to any location within the 16k of memory without making any changes in the module. This built-in decoding allows additional expansion of memory by bank switching.

RAM Memory Module
### RAM Memory Module Specifications

- **Memory Size:** 4k bytes
- **Word Size:** 8 bits
- **Memory Expansion:** To 65k bytes (16 modules)
- **Cycle Time:** 1μs
- **Interface:** TTL compatible inputs; open collector outputs (positive true logic)
- **Capacity:** 4096 bytes
- **Connector:** Dual 50-pin on 0.125 in. centers. Connectors in rack must be positioned on 0.5 in. centers min.
  - Wirewrap P/N C800100 from SAE
  - P/N VP01C50E00A1 from CDC
- **Board Dimensions:** 6.18 in. x 8.0 in. x 0.062 in. Board to be on 0.5 in. centers minimum.
- **Operating Temperature:** 0°C to 55°C
- **DC Power Requirement:** $V_{CC} = +5V \pm 5\%$, $I_{CC} = 2.5A$ max., 1.25A typical

### imm6-28 Block Diagram

```
R/W
BYTE 1
BYTE 2

READ/WRITE CONTROL

INPUT BUFFER

MEMORY ARRAY
4096 x 8

OUTPUT BUFFER

DATA TO MEMORY

MD0 MD1 MD2 MD3 MD4 MD5 MD6 MD7

DATA FROM MEMORY

MODULE SELECT

ADDRESS BUFFER

MAD 0 MAD 1 MAD 2 MAD 3 MAD 4 MAD 5 MAD 6 MAD 7 MAD 8 MAD 9 MAD 10 MAD 11

MAD 12 MAD 13 MAD 14 MAD 15

MS 12 MS 13 MS 14 MS 15

RAM MOD ENBL ADR STB

113
```
imm6-26 PROM MEMORY MODULE

- Provides sockets for up to sixteen PROMs (4096 x 8)
- Static memory, no clocks required
- Interfaces with imm8-82 8-bit Central Processor Module
- Accepts Intel 1602A or 1702A PROMs or 1302 ROMs
- Logic to allow any mix of PROM in 256 byte (8-bits) increments with RAM to 16k when used with the imm8-82 8-bit Central Processor Module
- Built in decoding of module select for expansion to 65k of memory

The imm6-26 PROM Memory Module may be used with the imm8-82 8-bit Central Processor Module for non-volatile program storage. Each PROM Memory Module has sockets for from one to sixteen of Intel’s 1602A or 1702A PROMs. In addition, the 1302 mask programmed ROM may be used in place of the PROMs in OEM applications.

The PROM Memory Module is used for program storage and look-up-tables with the MCS-8 8-bit Micro Processor. It interfaces directly with the imm8-82 Central Processor Module and may be used with the imm6-28 RAM Memory Module in any combination to 16k bytes. Special control logic on the imm6-28 module allows any mix of PROM and RAM in a system in 256 byte increments.

For memories larger than 4k bytes, decoding on the module allows addressing of up to sixteen imm6-28 modules for a total of 65k bytes of memory. The decoding is accomplished on the module connector. Any imm6-26 may be plugged in to any memory module connector.
PROM Memory Module Specifications

Memory Size: 4k bytes
Word Length: 8 bits
Memory Expansion: To 65k bytes (16 modules)
Interface: TTL compatible inputs; open collector outputs (positive true logic)
Capacity: 256 to 4096 bytes in 256 byte increments
Connector: Dual 50-pin on 0.125 in. centers. Connectors in rack must be positioned on 0.5 in. centers min.
        Wirewrap P/N C800100 from SAE
        P/N VP801C50E00A1 from CDC
Board Dimensions: 6.18 in. x 8.0 in. x 0.062 in. Board to be on 0.5 in. centers minimum.
Operating Temperature: 0°C to 55°C
DC Power Requirement: 
\[ V_{CC} = +5V \pm 5\% \]
\[ V_{DD} = -9V \pm 5\% \]
\[ I_{CC} = 1.6A \text{ max., } 1.1A \text{ typical}\]**
\[ I_{DD} = 1.6A \text{ max., } 1.0A \text{ typical}\]**

** Board loaded with all 16 PROMs.

imm6-26 Block Diagram
imm8-60 INPUT/OUTPUT MODULE

- Four 8-bit input ports and four 8-bit latching output ports
- TTL compatible
- Interfaces directly with imm8-82 Central Processor Module
- Teletype asynchronous transmitter/receiver and controls on board
- Transmission rates of 110 or 1200 baud
- Crystal clock for asynchronous transmitter/receiver
- Capable of high speed serial communications to 9600 baud

The imm8-60 I/O Module provides four 8-bit TTL compatible input ports and four 8-bit TTL compatible latching output ports. It interfaces directly with the imm8-82 Central Processor Module. Built-in decoding on the board provides for expansion of I/O to the maximum with the addition of one imm8-60 and two imm8-62 Output Modules (eight input ports and twenty four output ports).

For more efficient use of the imm8-82 Central Processor, an asynchronous transmitter receiver is included in the module. This frees the processor of time-consuming bit manipulation during bit serial data transmission. The transmitter receiver operates at either 110 or 1200 baud and by alteration of the basic clock frequency, data rates to 9600 baud may be obtained. The module contains drivers and receivers for connection to a teletype. These may be used with the asynchronous transmitter receiver or directly with I/O ports for bit serial transmission and reception of teletype data.

The module is configured with all common control signals bused to the module on the PC connector, while all I/O signals are available at the ribbon connectors on the top of the module.
**I/O Module Specifications**

Word Size: 8 bits

Capacity: Four 8-bit input ports, four 8-bit output ports

I/O Interface:
Input ports: TTL compatible (complement Data In)
Output ports: TTL compatible (complement Data Out)

Communications Interface:
- Direct: TTL compatible input and output
- TTY: 20mA TTY interface with discrete transmitter and receiver
- TTY RDR Control: Discrete relay interface

Serial Communication Rate: Crystal controlled to 110 or 1200 baud

Connector:
Dual 50-pin on 0.125 in. centers. Connectors in rack must be positioned on 0.5 in. centers min.
Wirewrap P/N C800100 from SAE
P/N VP801C50E00A1 from CDC
.Ribbon Type P/N 3417 from 3M

Board Dimensions: 6.18 in. x 8.0 in. x 0.062 in. Board to be on 0.5 in. centers minimum.

Operating Temperature: 0°C to 55°C

DC Power Requirement:
- \( V_{CC} = +5V \pm 5\% , \ I_{CC} = 0.820A \text{ max., } 0.478A \text{ Typical} \)
- \( V_{DD} = -9V \pm 5\% , \ I_{DD} = 0.080A \text{ max., } 0.050 \text{ Typical} \)
- \( V_{GG} = -12V \pm 5\% , \ I_{GG} = 0.030A \text{ max., } 0.016A \text{ Typical} \)

---

**imm8-60 Block Diagram**

[Diagram of the imm8-60 block diagram with ports and interfaces labeled.]
imm8-62 OUTPUT MODULE

- Eight 8-bit Latching Output Ports
- Interfaces Directly with imm8-82 CPU Module
- Decoding for Expansion to Full Output Complement
- TTL Compatible

The imm8-62 Output Module provides eight 8-bit latching output ports for direct interface with the imm8-82 CPU Module. Each port is individually addressable, and all outputs are TTL compatible. The module address includes decoding for expansion to a full complement of 24 output ports. This may be accomplished by using two imm8-60 I/O Modules and two imm8-62 Output Modules. All output signals are available through a ribbon connector at the top of the module.
Output Module Specifications

Word Size: 8-bits
Capacity: Eight 8-bit latching output ports
Interface: TTL compatible (complement Data Out)
Connector: Dual 50-pin on 0.125 in. centers. Connectors in rack must be positioned on 0.5 in. centers min.
   Wirewrap P/N C800100 from SAE
   P/N VP801C50E00A1 from CDC
   Ribbon Type P/N 3417 from 3M
Board Dimensions: 6.18 in. x 8.0 in. x 0.062 in. Board to be on 0.5 in. centers minimum,
Operating Temperature: 0°C to 55°C
DC Power Requirement: $V_{CC} = \pm 5V \pm 5\%, I_{CC} = 0.840A$ max., 0.420A typical

imm8-62 Block Diagram
imm6-76 PROM PROGRAMMER MODULE

- High speed programming of Intel's 1702A or 1602A PROM
- All necessary timing and level shifting included
- Direct interface with Intel's Intellec 8 Microcomputer Development System
- Complete software necessary for use included with Intellec 8 system monitor

The imm6-76 PROM Programmer Module provides all necessary hardware and software to add PROM programming capability to the Intellec 8 microcomputer development system. The module has been designed to slip into the Intellec 8 and provides all connections to the zero insertion force socket on the front panel. All required timing and level shifting is accomplished on the module utilizing the high voltage power supply already located in the Intellec 8.

Software to control programmer operation is included as part of the Intellec 8 system monitor. This software is specifically written for the Intellec 8 and allows both programming and verification of 1602A and 1702A PROMs. In addition, the contents of any PROM may be listed or unloaded into memory for duplication. The imm6-76 may also be used as a stand alone PROM programmer with toggle switches or with another computer providing data address and control signals.

imm6-76 Block Diagram

PROM Programmer Module Specifications

System Interface: All inputs and outputs are TTL compatible and available at the ribbon connector at the top of the module. Control for either "True" or "False" data is provided. Direct interface to Intellec 8.

Control Software: Included in the Intellec 8 executive monitor.

Connector: Dual 50-pin on 0.125 in. centers. Connectors in rack must be positioned on 0.5 in. centers min. Wirewrap P/N C800100 from SAE
P/N V801C50E00A1 from CDC
Ribbon Type P/N 3417 from 3M

Board Dimensions: 6.18 in. x 8.0 in. x 0.062 in. Board to be on 0.5 in. centers min.

Operating Temperature: 0°C to +55°C

DC Power Requirements: $V_{CC} = +5V \pm 5\%$, $I_{CC} = 0.8A$ max., 0.5A typical $V_{DD} = -9V \pm 5\%$, $I_{DD} = 0.1A$ max., 0.08A typical $V_p = +50V$, $I_p = 1.0A$ max.
imm6-70 UNIVERSAL PROTOTYPE MODULE

- Provides breadboard capability for developing custom interfaces
- Standard size of all microcomputer modules
- 3M 40 pin ribbon connector on top of module provides direct I/O connections
- Will accept standard wirewrap sockets with 0.1 in. x 0.3 in. or 0.1 in. x 0.6 in lead spacing
- Capacity for 60 16-pin or 14-pin sockets or 24 24-pin sockets
- All power is bused on board. Pins on PC connector and pins to individual sockets are uncommitted for maximum flexibility

The imm6-70 Universal Prototype Module is a standard size microcomputer module with power buses which interface with the Intellec 8. It provides a standard format for prototyping both customer interface and system control. I/O interface is provided through ribbon-type connectors on top of the module.

The module will accept dual in-line packaged components having pin center-to-center dimensions of 0.100 inch by 0.300 inch or 0.100 inch by 0.600 inch. These parts should be mounted in standard wirewrap sockets.

Universal Prototype Module Specifications

Capacity: 60 16-pin or 14-pin sockets or 24 24-pin sockets. Standard wirewrap sockets with pins on 0.100 in. by 0.300 in. centers or 0.100 in. by 0.600 in. centers. Board spacing dependent on components and sockets used.

Connector: Dual 50-pin on 0.125 in. centers.
- Wirewrap P/N C800100 from SAE
- P/N VP801C50E00A1 from CDC
- Ribbon Type P/N 3417 from 3M

Board Dimensions: 6.18 in. x 8.0 in. x 0.062 in. Board to be on 0.5 in. centers minimum,
imm6-72 MODULE EXTENDER

- Allows any module to be extended for ease of debugging, testing, and maintenance
- Standard dual 50-pin configuration for use with all microcomputer modules

The imm6-72 Module Extender is designed to be used with the Intellec 8 system. It allows the operator to extend any module out of the cage for servicing while maintaining all electrical connections.

Module Extender Specifications

Connector: Dual 50-pin on 0.125 in. centers. Connectors in rack must be positioned on 0.5 in. centers min.
- Wirewrap P/N C800100 from SAE
- P/N VPB01C50E00A1 from CDC
Extending connector is mounted on board.

Board Dimensions: 6.18 in. x 8.0 in. x 0.062 in. Board to be on 0.5 in. centers minimum.
# U.S. SALES AND MARKETING OFFICES

## U.S. MARKETING HEADQUARTERS
- **ARIZONA**
  - Sales Engineering, Inc.
  - 7155 E. Thomas Road, No. 6
  - 602/945-5781, TX: 910-950-1288
  - Scottsdale 85252

- **CALIFORNIA**
  - Intel Corp.
  - 3055 Bowers Avenue
  - 408/246-7501, TX: 910-338-0026
  - *Santa Clara, California 95051*
  - *Direct Intel Office*

- **COLORADO**
  - Intel Corp.
  - 1341 South Lama St.
  - 303/755-1335
  - *Denver 80211*

- **FLORIDA**
  - Semetronic Associates, Inc.
  - P.O. Box 1449
  - 305/771-0010
  - Pompano Beach 33061
  - Semetronic Associates, Inc.
  - 685 Chelsea Road
  - 305/621-2023
  - Longwood 32750

- **ILLINOIS**
  - Mar-Con Associates, Inc.
  - 4316 Main Street
  - 312/675-6540
  - Oak Brook 60521

- **MARYLAND**
  - Barnhill and Associates
  - 1913 Greenspring Drive
  - 301/252-8616
  - Timonium 21093
  - Barnhill and Associates
  - P.O. Box 251
  - 301/252-5610
  - Glen Arm 21057

- **MASSACHUSETTS**
  - Intel Corp.
  - 2 Militia Drive, Suite 6
  - 617/801-1336, Telex: 92-3493
  - *Lexington 02173*

- **MICHIGAN**
  - Sheridan Associates, Inc.
  - 33708 Grand River Avenue
  - 313/477-3800
  - Farmington Hills 48024

- **MINNESOTA**
  - Intel Corp.
  - 100 Southgate Office Plaza
  - 6501 West 78th Street
  - 612/835-6722
  - *Minneapolis 55437*
  - E.I. Du Pont de Nemours & Co.
  - 5280 W. 74th Street
  - 612/831-6457, TX: 910-576-3153
  - *Minneapolis 55435*

- **MISSOURI**
  - Sheridan Associates, Inc.
  - 110 S. Highway 140, Suite 10
  - 314/837-5200
  - Florissant 63033

- **NEW JERSEY**
  - Adem
  - Post Office Box 231
  - 516/867-5900
  - Kearny 07032

- **NEW YORK**
  - Osmann Components Sales Corp.
  - 395 Cleveland Avenue
  - 716/332-4211
  - Buffalo 14215
  - Adden
  - 37 Purdy Blvd.
  - 516/367-5900
  - Huntington Station, L.I. 11746

- **NEW YORK (Continued)**
  - Osmann Components Sales Corp.
  - 280 Metro Park
  - 716/442-9296
  - Rochester 14623
  - Osmann Components Sales Corp.
  - 1911 Vestal Parkway E.
  - 607/785-9449
  - Vestal 13850
  - Osmann Components Sales Corp.
  - 132 Pickard Building
  - 315/454-4477
  - Syracuse 13211
  - Osmann Components Sales Corp.
  - 411 Washington Avenue
  - 914/393-5200
  - Kingston 12401

- **NORTH CAROLINA**
  - Barnhill and Associates
  - 6030 Bellow Street
  - 919/877-5774
  - Raleigh 27602

- **OHIO**
  - Sheridan Associates, Inc.
  - 10 Keefhust Drive
  - 513/761-5432, TX: 801-461-2670
  - Cincinnati 45237
  - Sheridan Associates, Inc.
  - 7800 Wall Street
  - 216/924-8120
  - Cleveland 44125
  - Sheridan Associates, Inc.
  - Shiloh Blvd., Suite 250
  - 5045 North Main Street
  - 513/277-8911
  - Dayton 4540

- **OSSA**
  - 3065 Bowers Avenue
  - 408/246-7501, TX: 910-338-0026
  - Tel: 34-6372

- **PENNSYLVANIA**
  - Vantage Sales Company
  - 21 Bala Avenue
  - 215/667-6990
  - Bala Cynwyd 19004

- **SOUTH CAROLINA**
  - 3065 Bowers Avenue
  - 408/246-7501, TX: 910-338-0026
  - Tel: 34-6372

- **TEXAS**
  - Evans and McDowell Associates
  - 1333 N. Central Expressway
  - Room 180
  - 214/238-7157, TX: 910-867-4763
  - Dallas 75222

- **VIRGINIA**
  - Barnhill and Associates
  - P.O. Box 1104
  - 703/846-4624
  - Lynchburg 24505

- **WASHINGTON**
  - 50.50 Products and Sales
  - 1400 N.E. 24th Street
  - 206/747-7424, TX: 910-443-2305
  - Bellevue 98007

## EUROPEAN MARKETING OFFICES

- **DENMARK**
  - John Johnsen
  - Intel Office
  - Vester Farringvej 7
  - 45-1-15 5644
  - Telex: 1568
  - DK 1605 Copenhagen V

- **FRANCE**
  - Bernard Giraud
  - Intel Office
  - 617/67-60-75, Telex: 27475
  - 94-534 Rungis

- **ENGLAND**
  - Keith Clappie
  - Intel Office
  - Broadfield House
  - 4 Between Towns Road
  - 771431, Telex: 837203
  - Goswell, Oxford

- **GERMANY**
  - Erling Holst
  - Intel Office
  - Wolfsbruchhaustrasse 169
  - 79892, Telex: 5-221870
  - Dk Munchen 71

- **SWEDEN**
  - Nordisk Elektronik AB
  - Fack
  - 08-24-33-40, Telex: 10547
  - S-103 Stockholm 7

- **SWITZERLAND**
  - Industrie AG
  - Genenstrasse 29
  - 3400 Aarau 29
  - Telex: 56788
  - 8021 Zurich

- **UNITED KINGDOM**
  - Walsome Electronics Ltd.
  - 11-15 Batterston Street
  - 01-835-2001, Telex: 28752
  - London WC2H 9BS

## INTERNATIONAL DISTRIBUTORS

### AUSLANDER

<table>
<thead>
<tr>
<th>Country</th>
<th>Distributor Name</th>
<th>Contact Information</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Australia</td>
<td>A. J. Ferguson (Adelaide) Pty. Ltd.</td>
<td>125 Wright Street</td>
<td>51 6895 Adelaide 5000</td>
</tr>
<tr>
<td>Austria</td>
<td>Bachr Elektronische Gerate GmbH</td>
<td>Medizinger Hauptstrasse 78</td>
<td>0222-593-41, Telex: 0115321 A120 Vianna</td>
</tr>
<tr>
<td>Belgium</td>
<td>Inlogic Belgium S.A.</td>
<td>Avenue Val Duchesse, 3</td>
<td>(02) 62 00 12, Telex: 25442 8-1160 Bruxelles</td>
</tr>
</tbody>
</table>

### ORIENT MARKETING HEADQUARTERS

<table>
<thead>
<tr>
<th>Country</th>
<th>Distributor Company</th>
<th>Contact Information</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAPAN</td>
<td>Y. Magami</td>
<td>Intel Japan Corp.</td>
<td>Kastura Building 1-5-10 Uchikanda, Chiyoda-ku</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Country</th>
<th>Distributor Company</th>
<th>Contact Information</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAPAN</td>
<td>Pan Elektron Inc.</td>
<td>No. 1 Higashihata-Machi</td>
<td>045-471-9271, Telex: 2781-4773 Midori-Ku, Yokohama 226</td>
</tr>
<tr>
<td>WEST</td>
<td>MID-AMERICA</td>
<td>NORTHEAST</td>
<td>SOUTHEAST</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>ARIZONA</td>
<td>Hamilton/Amet Electronics 2615 South 21st Street 602/275-7851 Phoenix 85034</td>
<td>ILLINOIS</td>
<td>OHIO</td>
</tr>
<tr>
<td></td>
<td>Cramer/Aztec 238 W. 16th Street 602/263-1112 Phoenix 85034</td>
<td>Cramer/Tri States, Inc. 666 Redline Terrace 513/771-6441</td>
<td>101 South State Road 203/762-0361</td>
</tr>
<tr>
<td></td>
<td>Cramer/Aztec 238 W. 16th Street 602/263-1112 Phoenix 85034</td>
<td>Cramer/Chicago 1911 South Busse Road 312/933-8230</td>
<td>Georgetown 06829</td>
</tr>
<tr>
<td>CALIFORNIA</td>
<td>Cramer/Seattle 509 10th Avenue 206/763-2300</td>
<td>Mt. Prospect 60056</td>
<td>Cramer/Connecticut 35 Dodge Avenue 203/239-5644</td>
</tr>
<tr>
<td></td>
<td>Cramer/San Francisco 720 Polk Avenue 415/779-7631 Sunnyvale 94086</td>
<td>Hamilton/Amet Electronics 2903 North 25th Avenue 312/787-6119</td>
<td>North Haven 06473</td>
</tr>
<tr>
<td></td>
<td>Mountain View 94041</td>
<td>Schiller Park 60176</td>
<td>MARYLAND</td>
</tr>
<tr>
<td></td>
<td>Cramer/San Francisco 720 Polk Avenue 415/779-7631 Sunnyvale 94086</td>
<td>KANSAS</td>
<td>Cramer/EW Baltimore 927-24 Patapsco Avenue 301/354-0100</td>
</tr>
<tr>
<td></td>
<td>Hamilton/Amet Electronics 340 E. Middlefield Road 415/961-7600</td>
<td>Howard/Amet Electronics 118 West Park Road 513/433-8610</td>
<td>Baltimore 21203</td>
</tr>
<tr>
<td></td>
<td>Cray/San Diego 8971 Complis Drive 714/477-2421 San Diego 92123</td>
<td>Dayton 45459</td>
<td>Cramer/EW Washington 16021 Industrial Drive 301/948-0110</td>
</tr>
<tr>
<td></td>
<td>Cramer/San Diego 8971 Complis Drive 714/477-2421 San Diego 92123</td>
<td>Sheridan Sales Co. 19 Kinnickinnic Drive 513/761-5432</td>
<td>Gaithersburg 20870</td>
</tr>
<tr>
<td></td>
<td>Hamilton/Amet Electronics 8817 Complis Drive 714/279-2421</td>
<td>Cincinnati 45237</td>
<td>Hamilton/Amet Electronics 7220 Standard Drive 301/796-5000</td>
</tr>
<tr>
<td></td>
<td>San Diego 92123</td>
<td>Cramer/Cleveland 5855 Harper Road 216/248-7740</td>
<td>Massachusetts</td>
</tr>
<tr>
<td></td>
<td>Cramer/San Diego 8971 Complis Drive 714/477-2421 San Diego 92123</td>
<td>Cleveland 44119</td>
<td>Cramer/Electromatics, Inc. 85 Wells Avenue 617/969-7700</td>
</tr>
<tr>
<td></td>
<td>Hamilton/Amet Electronics 9591 North Broadway 303/314-1212</td>
<td>Sheridan Sales Co. 7800 Wall Street 216/524-0120</td>
<td>Newton 02159</td>
</tr>
<tr>
<td></td>
<td>Denver 80216</td>
<td>Cleveland 44126</td>
<td>Hamilton/Amet Electronics 189 Cambridge Street 617/273-2320</td>
</tr>
<tr>
<td></td>
<td>Hamilton/Amet Electronics 5921 South Broadway 303/314-1212</td>
<td>Sheridan Sales Co. 7800 Wall Street 216/524-0120</td>
<td>Burlington 01803</td>
</tr>
<tr>
<td></td>
<td>Denver 80216</td>
<td>Linnworth 48150</td>
<td>NEW JERSEY</td>
</tr>
<tr>
<td></td>
<td>NEW MEXICO</td>
<td>MINNESOTA</td>
<td>Hamilton/Amet Electronics 2218 little Falls Road 201/239-0880</td>
</tr>
<tr>
<td></td>
<td>Cramer/New Mexico 137 Vermont, N.E. 505/265-5767</td>
<td>Cramer/Bone 2727 North Lake Road 612/461-4860</td>
<td>Cedar Grove 07009</td>
</tr>
<tr>
<td></td>
<td>Albuquerque 87108</td>
<td>Edina 55435</td>
<td>Cramer/New Jersey 301/930-5600</td>
</tr>
<tr>
<td></td>
<td>Hamilton/Amet Electronics 2450 Baylor Drive S.E. 500/765-1500</td>
<td>Hamilton/Amet Electronics 2850 Metro Drive 612/661-4900</td>
<td>Monroe 07074</td>
</tr>
<tr>
<td></td>
<td>Albuquerque 87117</td>
<td>Minneapolis 55420</td>
<td>Hamilton/Amet Electronics 113 Gaithers Drive</td>
</tr>
<tr>
<td></td>
<td>OREGON</td>
<td>Industrial Components, Inc. 5280 West 74th Street 612/831-2666</td>
<td>East Gate Industrial Park 609/234-2333</td>
</tr>
<tr>
<td></td>
<td>Almarki/Strum Electronics 8688 S.W. Ct. Canary Road 503/292-3534</td>
<td>Minneapolis 55435</td>
<td>Mt. Laurel 08057</td>
</tr>
<tr>
<td></td>
<td>Portland 97226</td>
<td>Sheridan Sales Co. 110 South Highway 140, Suite 10</td>
<td>Cramer/Pennsylvania, Inc. 1330 Route 130 North</td>
</tr>
<tr>
<td></td>
<td>UTAH</td>
<td>314/937-5200</td>
<td>509/692-5090</td>
</tr>
<tr>
<td></td>
<td>Cramer/Utah 391 W. 2500 South 801/475-3682</td>
<td>Florissant 63033</td>
<td>Pennsauken 08110</td>
</tr>
<tr>
<td></td>
<td>Salt Lake City 84115</td>
<td>Hamilton/Amet Electronics 392 First Street 314/731-1144</td>
<td>NEW YORK</td>
</tr>
<tr>
<td></td>
<td>Hamilton/Amet Electronics 647 W. Billings Road 901/262-8451</td>
<td>Hazelwood 63042</td>
<td>Cramer/Binghamton 1720 Watson Boulevard 607/765-6661</td>
</tr>
<tr>
<td></td>
<td>Salt Lake City 84115</td>
<td>もなく</td>
<td>Endwell 13769</td>
</tr>
<tr>
<td></td>
<td>WASHINGTON</td>
<td>WASHINGTON</td>
<td>Cramer/Rochester 3000 Winton Road South 716/275-0300</td>
</tr>
<tr>
<td></td>
<td>Hamilton/Amet Electronics 13407 Northrop Way 206/765-2300</td>
<td>Bellingham 98225</td>
<td>Rochester 14623</td>
</tr>
<tr>
<td></td>
<td>Bellvue 98245</td>
<td>Almarki/Strum Electronics 5811 Sixth Avenue South 206/763-2300</td>
<td>Cramer/Syracuse 315/437-6671</td>
</tr>
<tr>
<td></td>
<td>Seattle 98108</td>
<td>Seattle 98108</td>
<td>East Syracuse 13057</td>
</tr>
<tr>
<td></td>
<td>Cramer/Seattle 5602 Sixth Avenue South 206/763-5755</td>
<td>Cramer/Amet Electronics 6400 Joy Road 315/437-2642</td>
<td>Hamilton/Amet Electronics 13057</td>
</tr>
<tr>
<td></td>
<td>Seattle 98108</td>
<td>Cramer/Seattle 5602 Sixth Avenue South 206/763-5755</td>
<td>Whitestown, T.N. 46075</td>
</tr>
<tr>
<td></td>
<td>Cramer/Amet Electronics 13407 Northrop Way 206/765-2300</td>
<td>Everett 20465</td>
<td>Cramer/Long Island 516/231-5600</td>
</tr>
<tr>
<td></td>
<td>936 Monte De Llano 614/336-6397</td>
<td>Newington 530</td>
<td>Houria, T.L. 11787</td>
</tr>
<tr>
<td></td>
<td>Haverhill, MA 01830</td>
<td>Westbury, L.I. 11590</td>
<td>Hamilton/Amet Electronics 70 State Street 516/335-5800</td>
</tr>
<tr>
<td></td>
<td>PENNSYLVANIA</td>
<td>Sheriden Sales Co. 4282 North Pike, North Pike, Pennsylvania 412/638-1070</td>
<td>Monroeville 15146</td>
</tr>
</tbody>
</table>

125
Ordering Information

1. The 8008 (CPU) is available in ceramic only and should be ordered as C8008 or C8008-1.
2. SIM8-01 Prototyping System
   This MCS-8 system for program development provides complete
   interface between the CPU and ROMs and RAMs. 1702A electri-
   cally programmable and erasable ROMs may be used for the
   program development. Each board contains one 8008 CPU,
   1k x 8 RAM, and sockets for up to eight 1702As (2k x 8 PROM).
   This system should be ordered as SIM8-01 (the number of
   PROMs should also be specified).
3. Memory Expansion
   Additional memory for the 8008 may be developed from indivi-
   dual memory components. Specify RAM 1101, 1103, 2102;
   ROM 1702, 1302.
4. MP7-03 ROM Programmer
   This is the programmer board for the 1702A. The 1702A control
   ROMs used with the SIM8-01 for an automatic programming
   system are specified by pattern numbers A0860, A0861, A0863.
5. MCB8-10 System Interface and Control Module
   The MCB8-10 is a complete chassis which provides the intercon-
   nection between the SIM8-01 and MP7-03. In addition, the
   MCB8-10 provides the 50Vrms power supply for PROM program-
   ming, complete output display, and single step control capability
   for program development.
6. Bootstrap Loader
   The same control ROM set used with the PROM programming
   system is used for the bootstrap loading of programs into RAM
   and execution of programs from RAM. Specify 1702A PROMs
   programmed to tapes A0860, A0861, and A0863.

7. SIM8 Hardware Assembler
   Eight PROMs containing the assembly program plug into the
   SIM8-01 prototyping board permitting assembly of all MCS-8
   software. To order, specify C1702A/840 set.
8. PL/M Compiler Software Package
   Programs for the MCS-8 may now be developed in a high level
   language and compiled to 8008 machine code. This program is
   written in FORTRAN IV and is available via time sharing service
   or directly from Intel.
9. MCS-8 Cross Assembler and Simulator Software Package
   This software program converts a list of instruction mnemonics
   into machine instructions and simulates the execution of instruc-
   tions by the 8008. This program is written in FORTRAN IV
   and is available via time sharing service or directly from Intel.
10. Intellec 8
    The Intellec 8, Bare Bones 8, and microcomputer modules must
    be specified individually by product code.

    imm8-80A  Intellec 8 (complete table top system)  
    imm8-81  Bare Bones 8 (complete rack mountable system)  
    imm8-82  Central Processor — includes 8008-1 CPU crystal
             clock and interface logic
    imm6-26  PROM Memory — includes sockets for sixteen
             1702A PROMs
    imm6-28  RAM Memory — 4k x 8 static memory
    imm6-60  Input/Output — 4 input and 4 output ports
    imm6-76  1702A PROM programmer and control software
    imm6-70  Universal prototype module
    imm6-72  Module extender

Packaging Information
## MCS-8™ Instruction Set

### INDEX REGISTER INSTRUCTIONS

The load instructions do not affect the flag flip-flops. The increment and decrement instructions affect all flip-flops except the carry.

<table>
<thead>
<tr>
<th>MNEONIC</th>
<th>MINIMUM STATES REQUIRED</th>
<th>INSTRUCTION CODE</th>
<th>DESCRIPTION OF OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDI R1</td>
<td>11</td>
<td>D D D D D S S S S</td>
<td>Load index register r1 with the content of index register r2.</td>
</tr>
<tr>
<td>LDI R1 M</td>
<td>11</td>
<td>D D D D D 1 1 1 1</td>
<td>Load index register r1 with the content of memory register M.</td>
</tr>
<tr>
<td>INX R1</td>
<td>11</td>
<td>1 1 1 1 1 1 S S S S</td>
<td>Load memory register M with the content of index register r1.</td>
</tr>
<tr>
<td>INX R1</td>
<td>11</td>
<td>D D D D D 1 1 1 1</td>
<td>Load index register r1 with data B...B.</td>
</tr>
<tr>
<td>LDX R1</td>
<td>11</td>
<td>D D D D D 1 1 1 1</td>
<td>Load memory register M with the content of index register r1.</td>
</tr>
<tr>
<td>LDX R1</td>
<td>11</td>
<td>B B B B B B B B</td>
<td>Load index register r1 with data B...B.</td>
</tr>
<tr>
<td>INC</td>
<td>11</td>
<td>0 0 D D D 0 0 0 0</td>
<td>Increment the content of index register r1 by 1.</td>
</tr>
<tr>
<td>DEC</td>
<td>11</td>
<td>0 0 D D D 0 0 0 0</td>
<td>Decrement the content of index register r1 by 1.</td>
</tr>
</tbody>
</table>

### ACCUMULATOR GROUP INSTRUCTIONS

The result of the ALU instructions affects all of the flag flip-flops. The rotate instructions affect only the carry flip-flop.

<table>
<thead>
<tr>
<th>MNEONIC</th>
<th>MINIMUM STATES REQUIRED</th>
<th>INSTRUCTION CODE</th>
<th>DESCRIPTION OF OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>11</td>
<td>0 0 0 0 0 S S S S</td>
<td>Add the content of index register r1, memory register M or data B...B to the accumulator. An overflow (carry) sets the carry flip-flop.</td>
</tr>
<tr>
<td>ADI</td>
<td>11</td>
<td>0 0 0 0 0 1 1 1 1</td>
<td>Add the content of index register r1, memory register M or data B...B to the accumulator and carry. An overflow (carry) sets the carry flip-flop.</td>
</tr>
<tr>
<td>ACH</td>
<td>11</td>
<td>0 0 0 0 0 1 1 1 1</td>
<td>Add the content of index register r1, memory register M or data B...B to the accumulator and carry. An overflow (carry) sets the carry flip-flop.</td>
</tr>
<tr>
<td>SUB</td>
<td>11</td>
<td>0 0 0 1 1 1 1 1 1</td>
<td>Subtract the content of index register r1, memory register M or data B...B from the accumulator with borrow. An underflow (borrow) sets the carry flip-flop.</td>
</tr>
<tr>
<td>SUBH</td>
<td>11</td>
<td>0 0 0 1 1 1 1 1 1</td>
<td>Subtract the content of index register r1, memory register M or data B...B from the accumulator. An underflow (borrow) sets the carry flip-flop.</td>
</tr>
<tr>
<td>SBB</td>
<td>11</td>
<td>0 0 0 1 1 1 1 1 1</td>
<td>Subtract the content of index register r1, memory register M or data B...B from the accumulator. An underflow (borrow) sets the carry flip-flop.</td>
</tr>
<tr>
<td>SBI</td>
<td>11</td>
<td>0 0 0 1 1 1 1 1 1</td>
<td>Subtract the content of index register r1, memory register M or data B...B from the accumulator. An underflow (borrow) sets the carry flip-flop.</td>
</tr>
<tr>
<td>ND</td>
<td>11</td>
<td>0 0 1 0 0 0 0 0 0</td>
<td>Compute the logical AND of the content of index register r1, memory register M or data B...B with the accumulator.</td>
</tr>
<tr>
<td>NDH</td>
<td>11</td>
<td>0 0 1 0 0 1 0 0 0</td>
<td>Compute the logical AND of the content of index register r1, memory register M or data B...B with the accumulator.</td>
</tr>
<tr>
<td>NDM</td>
<td>11</td>
<td>0 0 1 0 0 1 0 0 0</td>
<td>Compute the logical AND of the content of index register r1, memory register M or data B...B with the accumulator.</td>
</tr>
<tr>
<td>XOR</td>
<td>11</td>
<td>0 0 1 0 1 1 1 1 1</td>
<td>Compute the EXCLUSIVE OR of the content of index register r1, memory register M or data B...B with the accumulator.</td>
</tr>
<tr>
<td>XORH</td>
<td>11</td>
<td>0 0 1 0 1 1 1 1 1</td>
<td>Compute the EXCLUSIVE OR of the content of index register r1, memory register M or data B...B with the accumulator.</td>
</tr>
<tr>
<td>ORH</td>
<td>11</td>
<td>0 0 1 1 1 1 1 1 1</td>
<td>Compute the INCLUSIVE OR of the content of index register r1, memory register M or data B...B with the accumulator.</td>
</tr>
<tr>
<td>ORH</td>
<td>11</td>
<td>0 0 1 1 1 1 1 1 1</td>
<td>Compute the INCLUSIVE OR of the content of index register r1, memory register M or data B...B with the accumulator.</td>
</tr>
<tr>
<td>CP</td>
<td>11</td>
<td>0 0 1 1 1 1 1 1 1</td>
<td>Compare the content of index register r1, memory register M or data B...B with the accumulator. The content of the accumulator is unchanged.</td>
</tr>
<tr>
<td>CFM</td>
<td>11</td>
<td>0 0 1 1 1 1 1 1 1</td>
<td>Compare the content of index register r1, memory register M or data B...B with the accumulator. The content of the accumulator is unchanged.</td>
</tr>
<tr>
<td>CPI</td>
<td>11</td>
<td>0 0 1 1 1 1 1 1 1</td>
<td>Compare the content of index register r1, memory register M or data B...B with the accumulator. The content of the accumulator is unchanged.</td>
</tr>
<tr>
<td>RLC</td>
<td>11</td>
<td>0 0 0 0 0 0 0 0 1</td>
<td>Rotate the content of the accumulator left.</td>
</tr>
<tr>
<td>RRC</td>
<td>11</td>
<td>0 0 0 0 0 0 0 0 1</td>
<td>Rotate the content of the accumulator right.</td>
</tr>
<tr>
<td>RL</td>
<td>11</td>
<td>0 0 0 0 0 0 0 0 1</td>
<td>Rotate the content of the accumulator right through the carry.</td>
</tr>
<tr>
<td>RR</td>
<td>11</td>
<td>0 0 0 0 0 0 0 0 1</td>
<td>Rotate the content of the accumulator right through the carry.</td>
</tr>
</tbody>
</table>

### PROGRAM COUNTER AND STACK CONTROL INSTRUCTIONS

(4) JMP | (11) | 0 1 X X X 1 0 0 | Unconditionally jump to memory address B3...B5, B6, B7, B8, B9, B10, B11, B12. |

(7) JF | 0 1 0 C4 C3 0 0 0 0 | Jump to memory address B3...B5, B6, B7, B8, B9, B10, B11, B12 if the condition flip-flop c is false. Otherwise, execute the next instruction in sequence. |

(7) JF | 0 1 0 C4 C3 0 0 0 0 | Jump to memory address B3...B5, B6, B7, B8, B9, B10, B11, B12 if the condition flip-flop c is false. Otherwise, execute the next instruction in sequence. |

(7) JF | 0 1 0 C4 C3 0 0 0 0 | Jump to memory address B3...B5, B6, B7, B8, B9, B10, B11, B12 if the condition flip-flop c is false. Otherwise, execute the next instruction in sequence. |

(7) JF | 0 1 0 C4 C3 0 0 0 0 | Jump to memory address B3...B5, B6, B7, B8, B9, B10, B11, B12 if the condition flip-flop c is false. Otherwise, execute the next instruction in sequence. |

(5) CAL | 0 1 X X X 1 1 0 | Unconditionally call the subroutine at memory address B3...B5, B6, B7, B8, B9, B10, B11, B12. Save the current address (up one level in the stack). |

(5) CAL | 0 1 X X X 1 1 0 | Unconditionally call the subroutine at memory address B3...B5, B6, B7, B8, B9, B10, B11, B12. Save the current address (up one level in the stack). |

(5) CAL | 0 1 X X X 1 1 0 | Unconditionally call the subroutine at memory address B3...B5, B6, B7, B8, B9, B10, B11, B12. Save the current address (up one level in the stack). |

(5) CAL | 0 1 X X X 1 1 0 | Unconditionally call the subroutine at memory address B3...B5, B6, B7, B8, B9, B10, B11, B12. Save the current address (up one level in the stack). |

(5) CAL | 0 1 X X X 1 1 0 | Unconditionally call the subroutine at memory address B3...B5, B6, B7, B8, B9, B10, B11, B12. Save the current address (up one level in the stack). |

(5) RET | 0 X X X X X 1 1 1 | Unconditionally return (down one level in the stack). |

(5) RET | 0 X X X X X 1 1 1 | Unconditionally return (down one level in the stack). |

(5) RET | 0 X X X X X 1 1 1 | Unconditionally return (down one level in the stack). |

### INPUT/OUTPUT INSTRUCTIONS

<table>
<thead>
<tr>
<th>MNEONIC</th>
<th>MINIMUM STATES REQUIRED</th>
<th>INSTRUCTION CODE</th>
<th>DESCRIPTION OF OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>INP</td>
<td>11</td>
<td>1 0 0 0 0 M M M</td>
<td>Read the content of the selected input port (SW/RM) into the accumulator.</td>
</tr>
<tr>
<td>OUT</td>
<td>11</td>
<td>1 0 0 R R M</td>
<td>Write the content of the accumulator into the selected output port (SW/RM, RR # 00).</td>
</tr>
</tbody>
</table>

### MACHINE INSTRUCTION

- **HLT**
  - **00**
  - **0 0 0 0 0 0 0 0**
  - Enter the STOPPED state and remain there until interrupted.

- **HLT**
  - **00**
  - **1 1 1 1 1 1 1 1**
  - Enter the STOPPED state and remain there until interrupted.

**NOTES:**

1. **SS = Source Index Register**
   - These registers, r, are designated A accumulator—0000.
2. **CDX = Destination Index Register**
   - These registers, r, are designated A accumulator—0000.
3. **XI = Don't Care**
   - These registers, r, are designated A accumulator—0000.
4. **R = Memory Address**
   - These registers, r, are designated A accumulator—0000.
5. **C4 C3 Carry**
   - These registers, r, are designated A accumulator—0000.
intel Microcomputers. First from the beginning.