

mcs-8 Microcomputer Set

8008 8 Bit Parallel Central Processor Unit

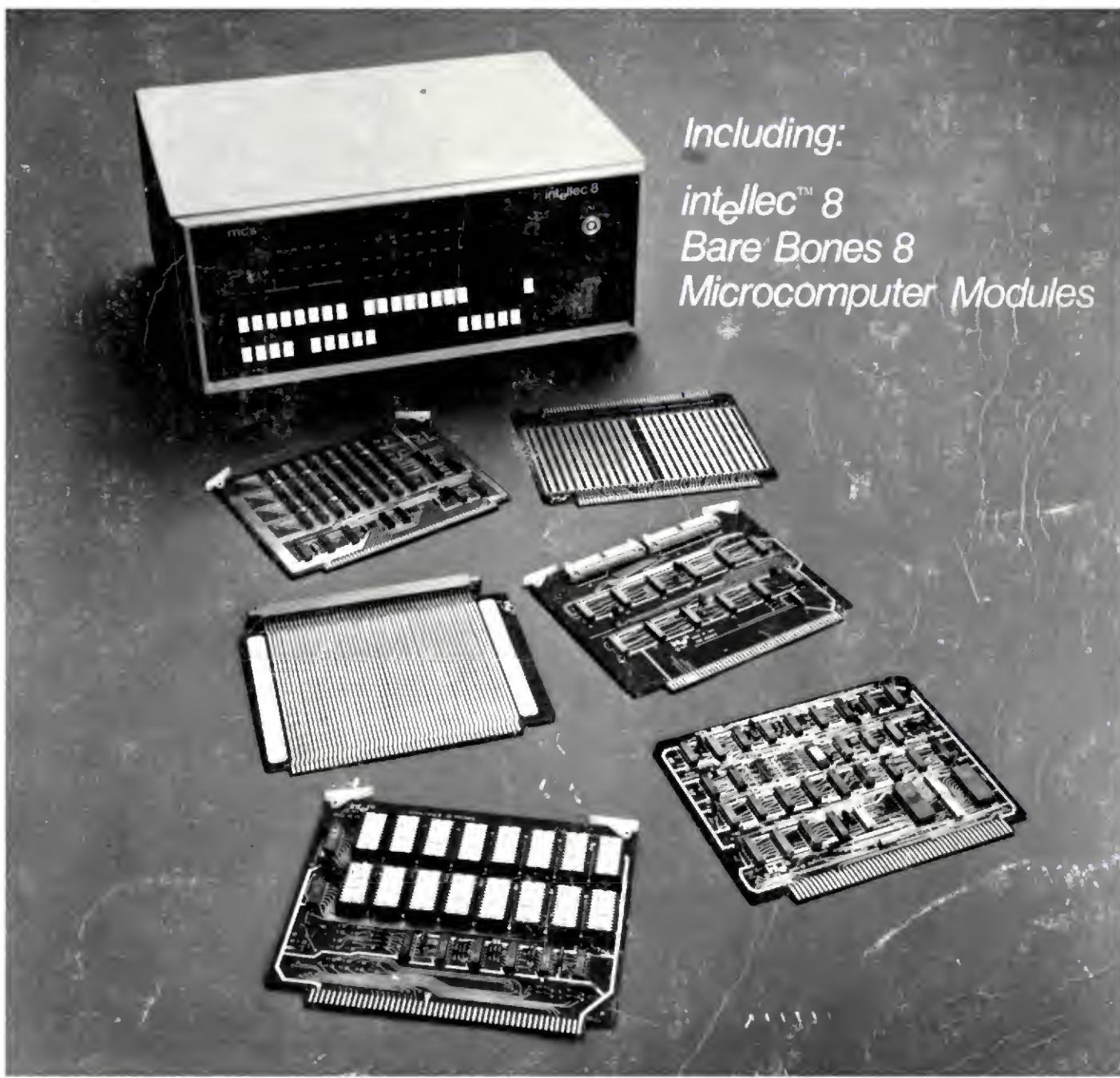
USERS MANUAL

intel[®]

NOVEMBER 1973

REV. 4

Second Printing



Including:

intellec™ 8

Bare Bones 8

Microcomputer Modules

INTEL SUPPORT MAKES SYSTEM BUILDING EASY.

The MCS-8TM 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/MTM 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.

IntellecTM8 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

intel[®]
delivers.

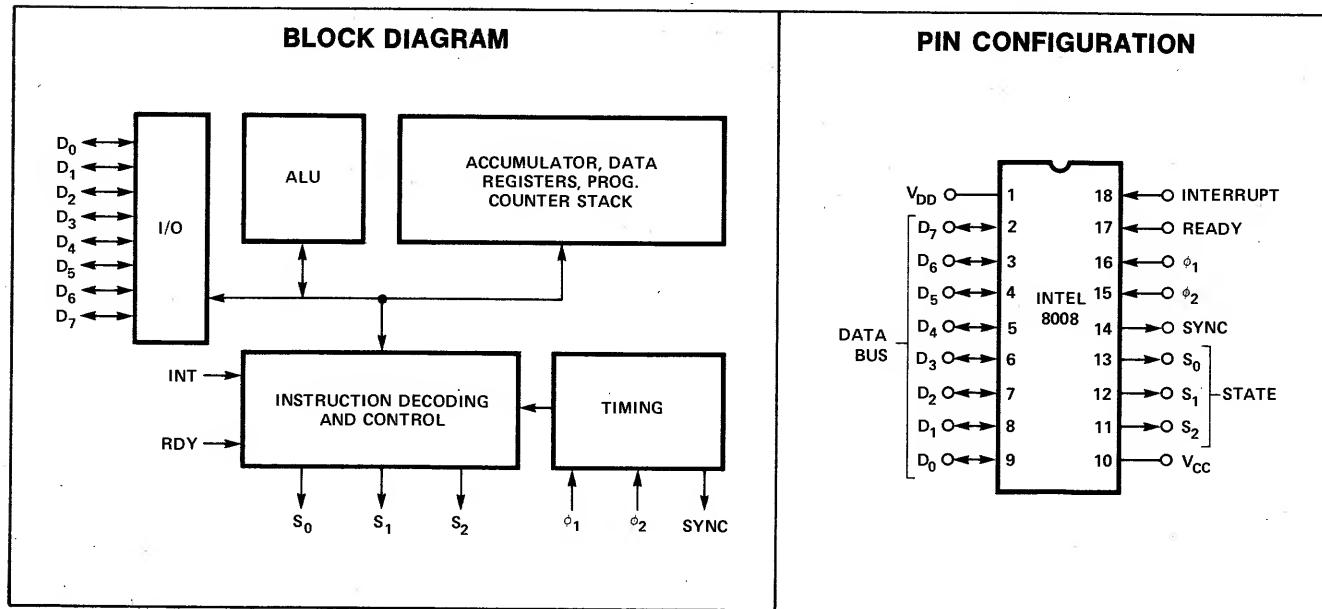
8008

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.

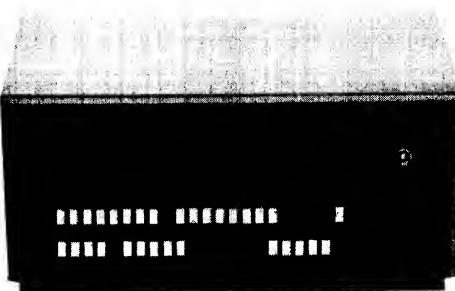
Features

- **8-Bit Parallel CPU on a Single Chip**
- **48 Instructions, Data Oriented**
- **Complete Instruction Decoding and Control Included**
- **Instruction Cycle Time — 12.5 μ s with 8008-1 or 20 μ s 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**



intellec^{T.M.}

A NEW, EASY AND INEXPENSIVE WAY TO DEVELOP MICROCOMPUTER SYSTEMS



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, ex-

pandable 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.

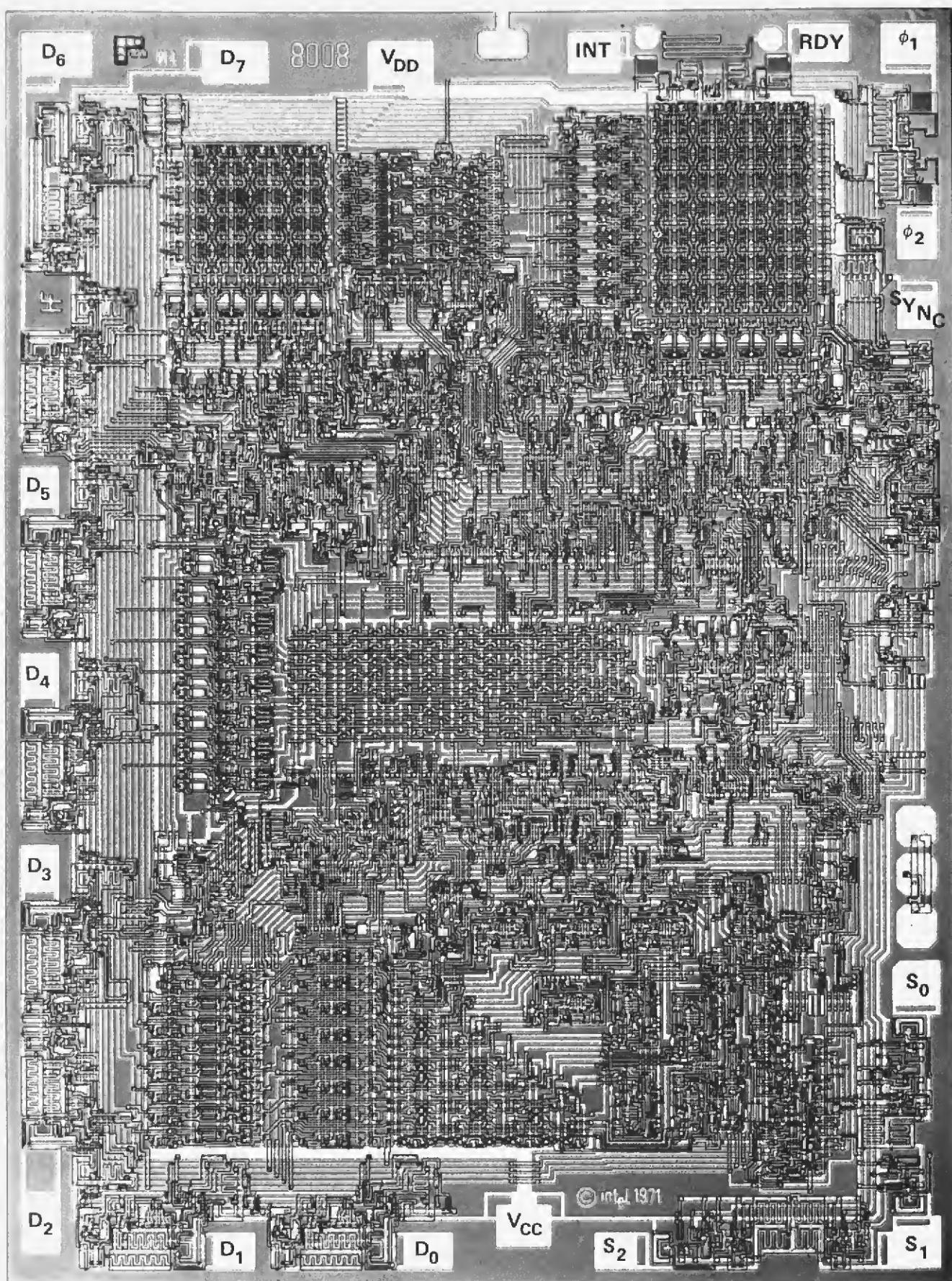
Write for complete details on the Intellec modular microcomputer development systems. They will be available in 120 days, but plan now. Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051 (408) 246-7501.

**intel[®]
delivers.**

CONTENTS

	Page No.
I. Introduction	3
II. Processor Timing	4
A. State Control Coding	4
B. Timing	4
C. Cycle Control Coding	5
III. Basic Functional Blocks	7
A. Instruction Register and Control	7
B. Memory	7
C. Arithmetic/Logic Unit	7
D. I/O Buffer	7
IV. Basic Instruction Set	8
A. Data and Instruction Formats	8
B. Summary of Processor Instructions	8
C. Complete Functional Definition	10
D. Internal Processor Operation	15
V. Processor Control Signals	18
A. Interrupt Signal	18
B. Ready Signal	20
VI. Electrical Specifications	21
A. DC and Operating Characteristics	22
B. AC Characteristics	23
C. Timing Diagram	23
D. Typical DC Characteristics	23
E. Typical AC Characteristics	23
VII. The SIM8-01 — An MCS-8 Micro Computer	24
A. SIM8-01 Specifications	25
B. SIM8-01 Schematic	26
C. System Description	28
D. Normal Operation	29
E. SIM8-01 Pin Description	31
VIII. MCS-8 PROM Programming System	33
A. General System Description and Operating Instructions	33
B. MP7-03 PROM Programmer	39
C. Programming System Interconnection	40
IX. Micro Computer Program Development	44
A. MCS-8 Software Library	44
B. Development of a Microcomputer System	46
C. Execution of Programs from RAM on SIM8-01 Using Memory Loader Control Programs	47
X. MCB8-10 Microcomputer Interconnect and Control Module	49
XI. Appendices	56
I. SIM8 Hardware Assembler	56
II. MCS-8 Software Package — Assembler	71
A. Assembler Specifications	71
B. Tymshare Users Guide for Assembly	81
C. General Electric Users Guide for Assembly	81
D. Sample Program Assembly	82
III. MCS-8 Software Package — Simulator	84
A. Introduction	84
B. Basic Elements	84
C. INTERP/8 Commands	84
D. I/O Formatting Commands	88
E. Error Messages	89
F. Examples	90
IV. Teletype Modifications for SIM8-01	95
V. Programming Examples	98
A. Sample Program to Search a String of Characters	98
B. Teletype and Tape Reader Control Program	99
C. Memory Chip Select Decodes and Output Test Program	99
D. RAM Test Program	99
E. Bootstrap Loader Program	100
VI. Intellec 8, Bare Bones 8, and Microcomputer Modules	103
XII. Ordering Information	124
A. Sales Offices	124
B. Distributors	125
C. Ordering Information/Packaging Information	126

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.



8008 Photomicrograph With Pin Designations

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 (D_0 through D_7) and uses two input leads (READY and INTERRUPT) and four output leads (S_0 , S_1 , S_2 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 S_0 , S_1 , and S_2 , 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.

S_0	S_1	S_2	STATE
0	1	0	T1
0	1	1	T1I
0	0	1	T2
0	0	0	WAIT
1	0	0	T3
1	1	0	STOPPED
1	1	1	T4
1	0	1	T5

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.

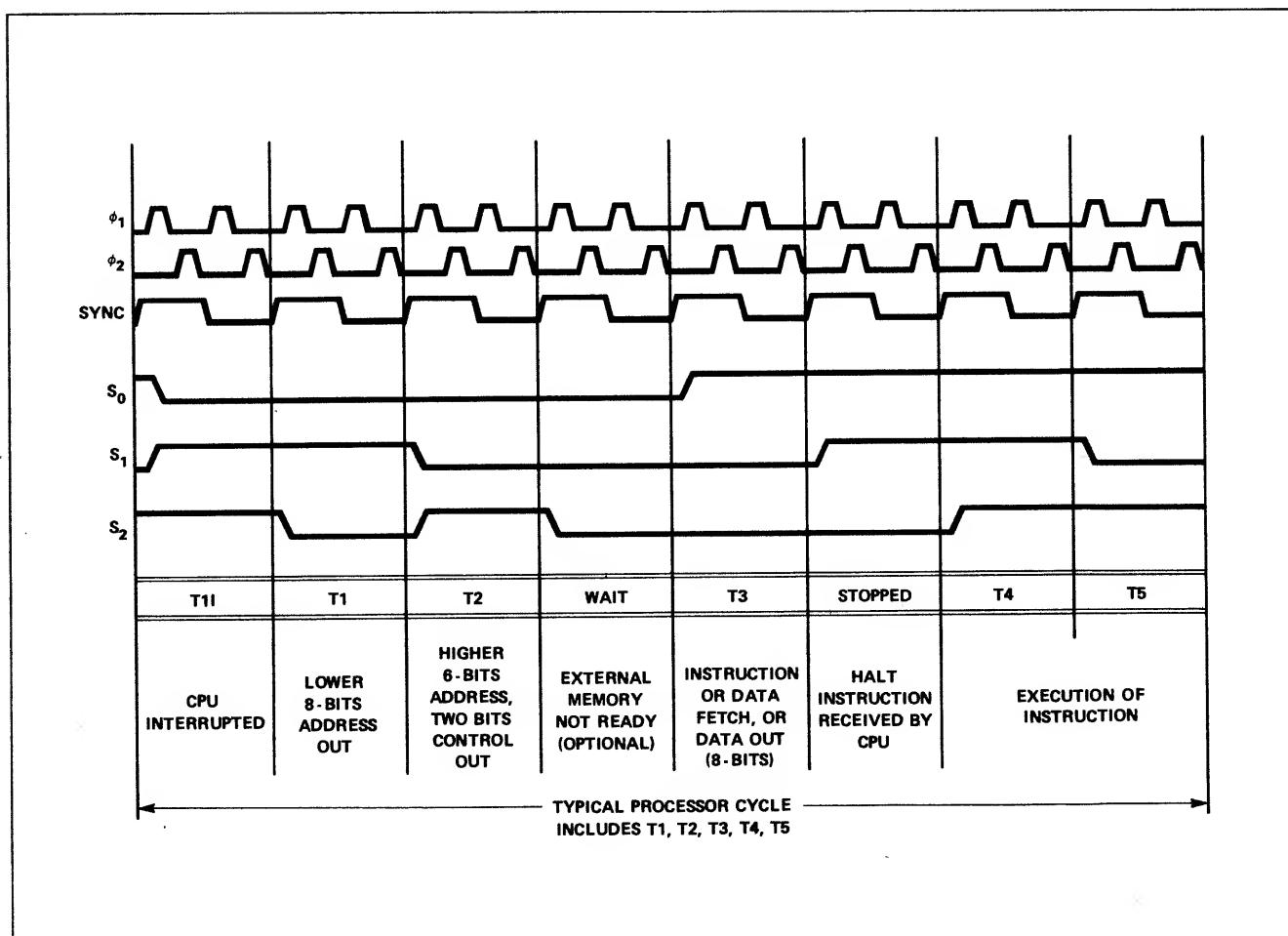


Figure 1. Basic 8008 Instruction Cycle

The receipt of an INTERRUPT is acknowledged by the T1I. 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 2^n clock periods). The use of READY and INTERRUPT with regard to these states will be explained later.

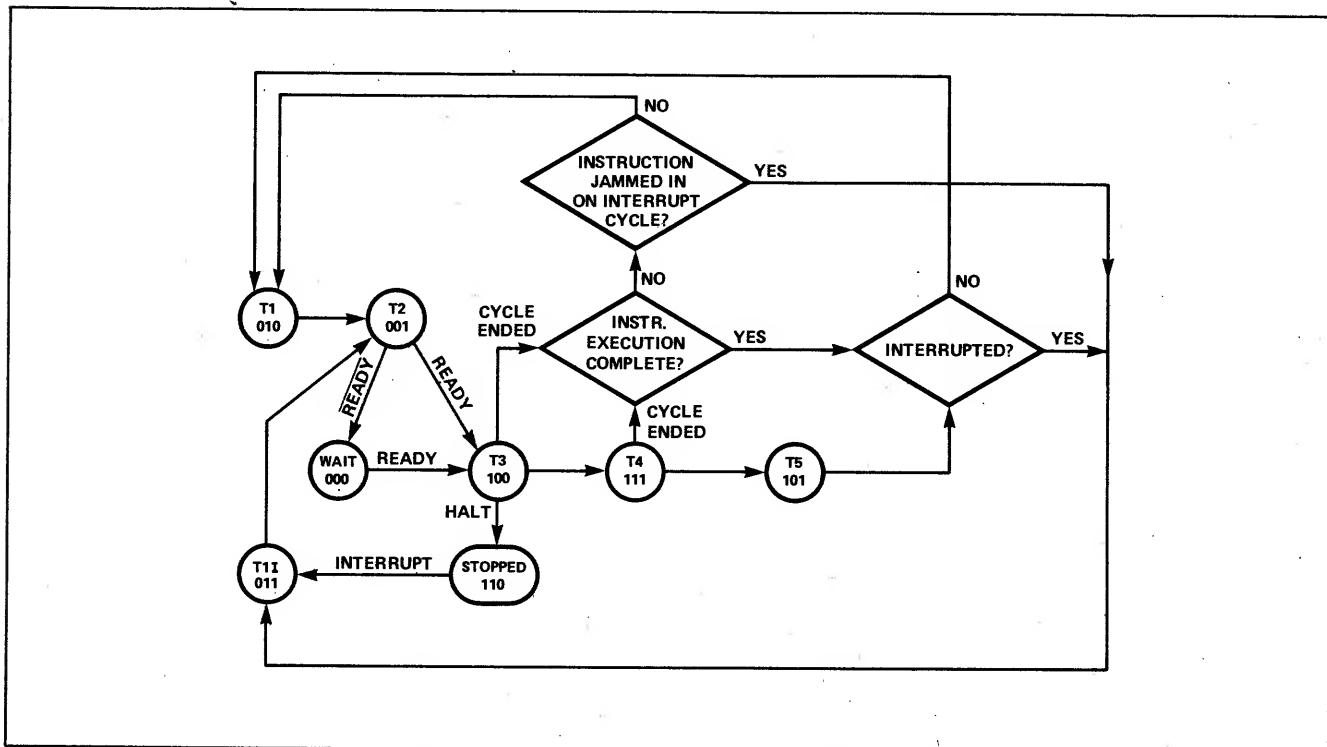


Figure 2. CPU State Transition Diagram

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.

D_6	D_7	CYCLE	FUNCTION
0	0	PCI	Designates the address is for a memory read (first byte of instruction).
0	1	PCR	Designates the address is for a memory read data (additional bytes of instruction or data).
1	0	PCC	Designates the data as a command I/O operation.
1	1	PCW	Designates the address is for a memory write data.

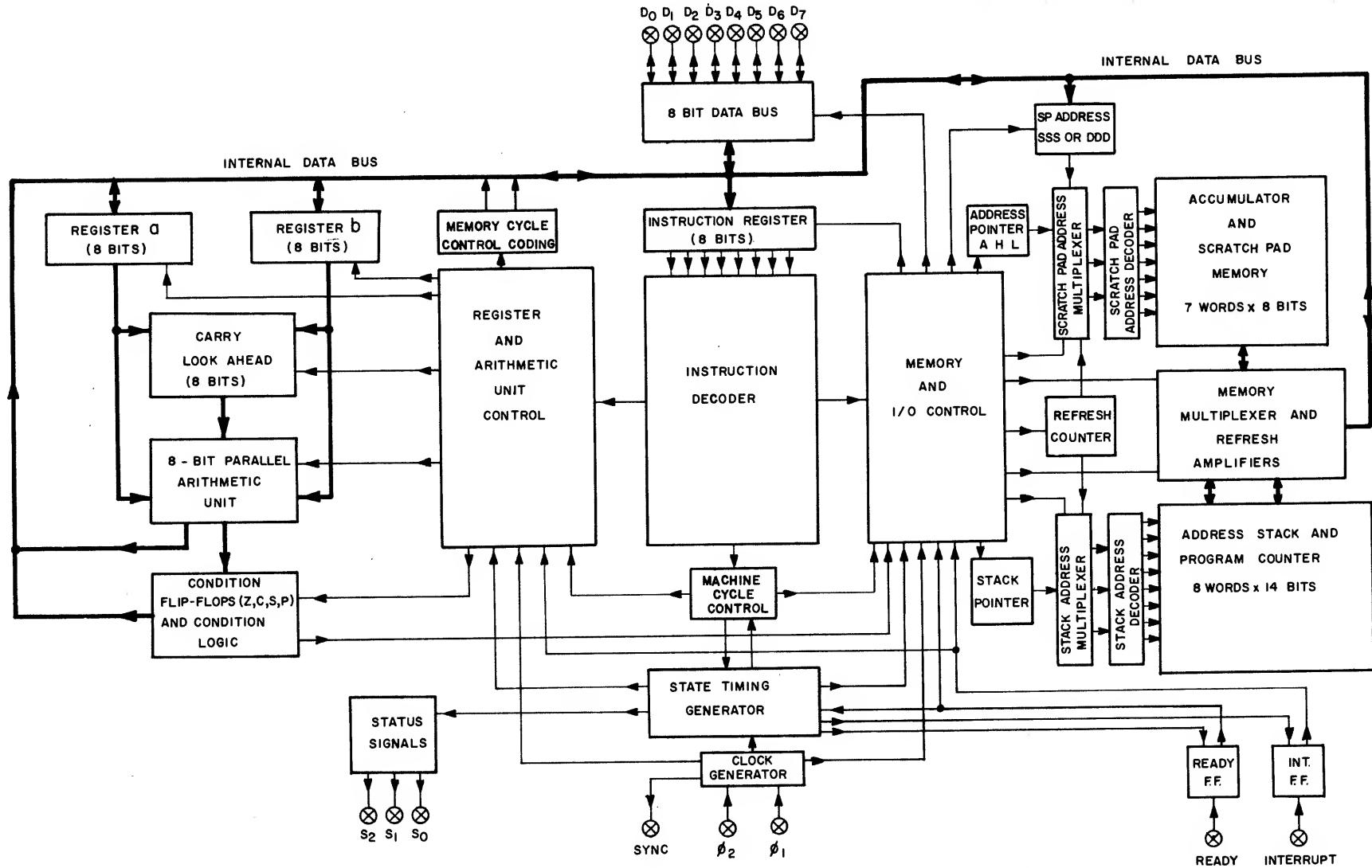


Figure 3. 8008 Block Diagram

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 8008, 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

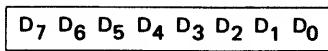
This buffer is the only link between the processor and the rest of the system. Each of the eight buffers is bi-directional and is under control of the instruction register and state timing. Each of the buffers is low power TTL compatible on the output and TTL compatible on the input.

IV. BASIC INSTRUCTION SET

The following section presents the basic instruction set of the 8008.

A. Data and Instruction Formats

Data in the 8008 is stored in the form of 8-bit binary integers. All data transfers to the system data bus will be in the same format.



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	OP CODE
---	---------

TYPICAL INSTRUCTIONS

Register to register, memory reference,
I/O arithmetic or logical, rotate or
return instructions

Two Byte Instructions

D ₇ D ₆ D ₅ D ₄ D ₃ D ₂ D ₁ D ₀	OP CODE
---	---------

OPERAND

Immediate mode instructions

Three Byte Instructions

D ₇ D ₆ D ₅ D ₄ D ₃ D ₂ D ₁ D ₀	OP CODE
---	---------

JUMP or CALL instructions

D ₇ D ₆ D ₅ D ₄ D ₃ D ₂ D ₁ D ₀	LOW ADDRESS
---	-------------

X X D ₅ D ₄ D ₃ D ₂ D ₁ D ₀	HIGH ADDRESS*
---	---------------

*For the third byte of this instruction, D₆ and D₇ 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.

MNEMONIC	MINIMUM STATES REQUIRED	INSTRUCTION CODE D ₇ D ₆ D ₅ D ₄ D ₃ D ₂ D ₁ D ₀	DESCRIPTION OF OPERATION
(1) Lr ₁ r ₂	(5)	1 1 D D D S S S	Load index register r ₁ with the content of index register r ₂ .
(2) LrM	(8)	1 1 D D D 1 1 1	Load index register r with the content of memory register M.
LMr	(7)	1 1 1 1 1 S S S	Load memory register M with the content of index register r.
(3) LrI	(8)	0 0 D D D 1 1 0 B B B B B B B B	Load index register r with data B . . . B.
LMI	(9)	0 0 1 1 1 1 1 0 B B B B B B B B	Load memory register M with data B . . . B.
INr	(5)	0 0 D D D 0 0 0	Increment the content of index register r (r ≠ A).
DCr	(5)	0 0 D D D 0 0 1	Decrement the content of index register r (r ≠ A).

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.

ADr	(5)	1 0 0 0 0 S S S	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.
ADM	(8)	1 0 0 0 0 1 1 1	
ADI	(8)	0 0 0 0 0 1 0 0 B B B B B B B B	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.
ACr	(5)	1 0 0 0 1 S S S	
ACM	(8)	1 0 0 0 1 1 1 1	
ACI	(8)	0 0 0 0 1 1 0 0 B B B B B B B B	
SUr	(5)	1 0 0 1 0 S S S	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.
SUM	(8)	1 0 0 1 0 1 1 1	
SUI	(8)	0 0 0 1 0 1 0 0 B B B B B B B B	
SBr	(5)	1 0 0 1 1 S S S	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.
SBM	(8)	1 0 0 1 1 1 1 1	
SBI	(8)	0 0 0 1 1 1 0 0 B B B B B B B B	

MNEMONIC	MINIMUM STATES REQUIRED	INSTRUCTION CODE	DESCRIPTION OF OPERATION
		D ₇ D ₆ D ₅ D ₄ D ₃ D ₂ D ₁ D ₀	
NDr	(5)	1 0 1 0 0 S S S	Compute the logical AND of the content of index register r, memory register M, or data B . . . B with the accumulator.
NDM	(8)	1 0 1 0 0 1 1 1	
NDI	(8)	0 0 1 0 0 1 0 0 B B B B B B B B	
XRr	(5)	1 0 1 0 1 S S S	Compute the EXCLUSIVE OR of the content of index register r, memory register M, or data B . . . B with the accumulator.
XRM	(8)	1 0 1 0 1 1 1 1	
XRI	(8)	0 0 1 0 1 1 0 0 B B B B B B B B	
ORr	(5)	1 0 1 1 0 S S S	Compute the INCLUSIVE OR of the content of index register r, memory register m, or data B . . . B with the accumulator.
ORM	(8)	1 0 1 1 0 1 1 1	
ORI	(8)	0 0 1 1 0 1 0 0 B B B B B B B B	
CPr	(5)	1 0 1 1 1 S S S	Compare the content of index register r, memory register M, or data B . . . B with the accumulator. The content of the accumulator is unchanged.
CPM	(8)	1 0 1 1 1 1 1 1	
CPI	(8)	0 0 1 1 1 1 0 0 B B B B B B B B	
RLC	(5)	0 0 0 0 0 0 1 0	Rotate the content of the accumulator left.
RRC	(5)	0 0 0 0 1 0 1 0	Rotate the content of the accumulator right.
RAL	(5)	0 0 0 1 0 0 1 0	Rotate the content of the accumulator left through the carry.
RAR	(5)	0 0 0 1 1 0 1 0	Rotate the content of the accumulator right through the carry.

Program Counter and Stack Control Instructions

(4) JMP	(11)	0 1 X X X 1 0 0 B ₂ B ₂ B ₂ B ₂ B ₂ B ₂ X X B ₃ B ₃ B ₃ B ₃ B ₃ B ₃	Unconditionally jump to memory address B ₃ . . . B ₃ B ₂ . . . B ₂ .
(5) JFc	(9 or 11)	0 1 0 C ₄ C ₃ 0 0 0 B ₂ B ₂ B ₂ B ₂ B ₂ B ₂ X X B ₃ B ₃ B ₃ B ₃ B ₃ B ₃	Jump to memory address B ₃ . . . B ₃ B ₂ . . . B ₂ if the condition flip-flop c is false. Otherwise, execute the next instruction in sequence.
JTc	(9 or 11)	0 1 1 C ₄ C ₃ 0 0 0 B ₂ B ₂ B ₂ B ₂ B ₂ B ₂ X X B ₃ B ₃ B ₃ B ₃ B ₃ B ₃	Jump to memory address B ₃ . . . B ₃ B ₂ . . . B ₂ if the condition flip-flop c is true. Otherwise, execute the next instruction in sequence.
CAL	(11)	0 1 X X X 1 1 0 B ₂ B ₂ B ₂ B ₂ B ₂ B ₂ X X B ₃ B ₃ B ₃ B ₃ B ₃ B ₃	Unconditionally call the subroutine at memory address B ₃ . . . B ₃ B ₂ . . . B ₂ . Save the current address (up one level in the stack).
CFc	(9 or 11)	0 1 0 C ₄ C ₃ 0 1 0 B ₂ B ₂ B ₂ B ₂ B ₂ B ₂ X X B ₃ B ₃ B ₃ B ₃ B ₃ B ₃	Call the subroutine at memory address B ₃ . . . B ₃ B ₂ . . . B ₂ if the condition flip-flop c is false, and save the current address (up one level in the stack.) Otherwise, execute the next instruction in sequence.
CTc	(9 or 11)	0 1 1 C ₄ C ₃ 0 1 0 B ₂ B ₂ B ₂ B ₂ B ₂ B ₂ X X B ₃ B ₃ B ₃ B ₃ B ₃ B ₃	Call the subroutine at memory address B ₃ . . . B ₃ B ₂ . . . B ₂ if the condition flip-flop c is true, and save the current address (up one level in the stack). Otherwise, execute the next instruction in sequence.
RET	(5)	0 0 X X X 1 1 1	Unconditionally return (down one level in the stack).
RFc	(3 or 5)	0 0 0 C ₄ C ₃ 0 1 1	Return (down one level in the stack) if the condition flip-flop c is false. Otherwise, execute the next instruction in sequence.
RTc	(3 or 5)	0 0 1 C ₄ C ₃ 0 1 1	Return (down one level in the stack) if the condition flip-flop c is true. Otherwise, execute the next instruction in sequence.
RST	(5)	0 0 A A A 1 0 1	Call the subroutine at memory address AAA000 (up one level in the stack).

Input/Output Instructions

INP	(8)	0 1 0 0 M M M 1	Read the content of the selected input port (MMM) into the accumulator.
OUT	(6)	0 1 R R M M M 1	Write the content of the accumulator into the selected output port (RRMM, RR ≠ 00).

Machine Instruction

HLT	(4)	0 0 0 0 0 0 0 X	Enter the STOPPED state and remain there until interrupted.
HLT	(4)	1 1 1 1 1 1 1 1	Enter the STOPPED state and remain there until interrupted.

NOTES:

- (1) SSS = Source Index Register These registers, r_i, are designated A(accumulator-000),
 DDD = Destination Index Register B(001), C(010), D(011), E(100), H(101), L(110).
 (2) Memory registers are addressed by the contents of registers H & L.
 (3) Additional bytes of instruction are designated by BBBB BBBB.
 (4) X = "Don't Care".
 (5) Flag flip-flops are defined by C₄C₃: 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.

Symbols	Meaning		
<B2>	Second byte of the instruction		
<B3>	Third byte of the instruction		
r		One of the scratch pad register references: A, B, C, D, E, H, L	
c		One of the following flag flip-flop references: C, Z, S, P	
C_4C_3	Flag flip-flop codes	Condition for True	
	00	carry	Overflow, underflow
	01	zero	Result is zero
	10	sign	MSB of result is "1"
	11	parity	Parity of result is even
M		Memory location indicated by the contents of registers H and L	
()		Contents of location or register	
\wedge		Logical product	
\oplus		Exclusive "or"	
\vee		Inclusive "or"	
A_m		Bit m of the A-register	
STACK		Instruction counter (P) pushdown register	
P		Program Counter	
\leftarrow		Is transferred to	
XXX		A "don't care"	
SSS		Source register for data	
DDD		Destination register for data	
	Register #	Register Name	
	(SSS or DDD)		
	000	A	
	001	B	
	010	C	
	011	D	
	100	E	
	101	H	
	110	L	

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.

Lr₁r₂ (one cycle – PCI)	11 DDD SSS	$(r_1) \leftarrow (r_2)$ 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).
LrM (two cycles – PCI/PCR)	11 DDD 111	$(r) \leftarrow (M)$ Load register r with the content of the memory location addressed by the contents of registers H and L. ($DDD \neq 111$ – HALT instr.)
LMr (two cycles – PCI/PCW)	11 111 SSS	$(M) \leftarrow (r)$ Load the memory location addressed by the contents of registers H and L with the content of register r . ($SSS \neq 111$ – 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

Lrl (two cycles – PCI/PCR)	00 DDD 110 $\langle B_2 \rangle$	$(r) \leftarrow \langle B_2 \rangle$ Load byte two of the instruction into register r .
LMI (three cycles – PCI/PCR/PCW)	00 111 110 $\langle B_2 \rangle$	$(M) \leftarrow \langle B_2 \rangle$ Load byte two of the instruction into the memory location addressed by the contents of registers H and L.

INCREMENT INDEX REGISTER – One Byte

INr (one cycle – PCI)	00 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 $DDD \neq 000$ (HALT instr.) and $DDD \neq 111$ (content of memory may not be incremented).
---------------------------------	------------	--

DECREMENT INDEX REGISTER – One Byte

DCr (one cycle – PCI)	00 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 $DDD \neq 000$ (HALT instr.) and $DDD \neq 111$ (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 (NDr, XNr, 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.

ADr	10 000 SSS	$(A) \leftarrow (A)+(r)$ Add the content of register r to the content of register A and place the result into register A.
ACr	10 001 SSS	$(A) \leftarrow (A)+(r)+(carry)$ 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.
SUr	10 010 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.

SBr	10 011	SSS	$(A) \leftarrow (A) - (r) - (\text{borrow})$ 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.
NDr	10 100	SSS	$(A) \leftarrow (A) \wedge (r)$ Place the logical product of the register A and register r into register A.
XRr	10 101	SSS	$(A) \leftarrow (A) \vee (r)$ Place the "exclusive - or" of the content of register A and register r into register A.
ORr	10 110	SSS	$(A) \leftarrow (A) V (r)$ Place the "inclusive - or" of the content of register A and register r into register A.
CPr	10 111	SSS	$(A) - (r)$ 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 ($A < r$) is indicated by the carry flip-flop, set to "1".

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.

ADM	10 000	111	$(A) \leftarrow (A) + (M)$ ADD
ACM	10 001	111	$(A) \leftarrow (A) + (M) + (\text{carry})$ ADD with carry
SUM	10 010	111	$(A) \leftarrow (A) - (M)$ SUBTRACT
SBM	10 011	111	$(A) \leftarrow (A) - (M) - (\text{borrow})$ SUBTRACT with borrow
NDM	10 100	111	$(A) \leftarrow (A) \wedge (M)$ Logical AND
XRM	10 101	111	$(A) \leftarrow (A) \vee (M)$ Exclusive OR
ORM	10 110	111	$(A) \leftarrow (A) V (M)$ Inclusive OR
CPM	10 111	111	$(A) - (M)$ COMPARE

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.

ADI	00 000	100 $< B_2 >$	$(A) \leftarrow (A) + < B_2 >$ ADD
ACI	00 001	100 $< B_2 >$	$(A) \leftarrow (A) + < B_2 > + (\text{carry})$ ADD with carry
SUI	00 010	100 $< B_2 >$	$(A) \leftarrow (A) - < B_2 >$ SUBTRACT
SBI	00 011	100 $< B_2 >$	$(A) \leftarrow (A) - < B_2 > - (\text{borrow})$ SUBTRACT with borrow
NDI	00 100	100 $< B_2 >$	$(A) \leftarrow (A) \wedge < B_2 >$ Logical AND
XRI	00 101	100 $< B_2 >$	$(A) \leftarrow (A) \vee < B_2 >$ Exclusive OR
ORI	00 110	100 $< B_2 >$	$(A) \leftarrow (A) V < B_2 >$ Inclusive OR
CPI	00 111	100 $< B_2 >$	$(A) - < B_2 >$ COMPARE

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.

RLC	00 000 010	$A_{m+1} \leftarrow A_m, A_0 \leftarrow A_7, (\text{carry}) \leftarrow A_7$ Rotate the content of register A left one bit. Rotate A_7 into A_0 and into the carry flip-flop.
RRC	00 001 010	$A_m \leftarrow A_{m+1}, A_7 \leftarrow A_0, (\text{carry}) \leftarrow A_0$ Rotate the content of register A right one bit. Rotate A_0 into A_7 and into the carry flip-flop.
RAL	00 010 010	$A_{m+1} \leftarrow A_m, A_0 \leftarrow (\text{carry}), (\text{carry}) \leftarrow A_7$ 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.
RAR	00 011 010	$A_m \leftarrow A_{m+1}, A_7 \leftarrow (\text{carry}), (\text{carry}) \leftarrow A_0$ 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.

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.

JMP (Jump Unconditionally)	01 XXX 100 $\langle B_2 \rangle$ $\langle B_3 \rangle$	$(P) \leftarrow \langle B_3 \rangle \langle B_2 \rangle$ Jump unconditionally to the instruction located in memory location addressed by byte two and byte three.
JFc (Jump if Condition False)	01 $0C_4 C_3$ 000 $\langle B_2 \rangle$ $\langle B_3 \rangle$	If $(c) = 0$, $(P) \leftarrow \langle B_3 \rangle \langle B_2 \rangle$. Otherwise, $(P) = (P)+3$. If the content of flip-flop c is zero, then jump to the instruction located in memory location $\langle B_3 \rangle \langle B_2 \rangle$; otherwise, execute the next instruction in sequence.
JTc (Jump if Condition True)	01 $1C_4 C_3$ 000 $\langle B_2 \rangle$ $\langle B_3 \rangle$	If $(c) = 1$, $(P) \leftarrow \langle B_3 \rangle \langle B_2 \rangle$. Otherwise, $(P) = (P)+3$. If the content of flip-flop c is one, then jump to the instruction located in memory location $\langle B_3 \rangle \langle B_2 \rangle$; otherwise, execute the next instruction in sequence.

CALL INSTRUCTIONS – Three Bytes

(three cycles – PCI/PCR/PCR)

Subroutines may be called and nested up to seven levels.

CAL (Call subroutine Unconditionally)	01 XXX 110 $\langle B_2 \rangle$ $\langle B_3 \rangle$	$(\text{Stack}) \leftarrow (P), (P) \leftarrow \langle B_3 \rangle \langle B_2 \rangle$. 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.
CFc (Call subroutine if Condition False)	01 $0C_4 C_3$ 010 $\langle B_2 \rangle$ $\langle B_3 \rangle$	If $(c) = 0$, $(\text{Stack}) \leftarrow (P), (P) \leftarrow \langle B_3 \rangle \langle B_2 \rangle$. 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 $\langle B_3 \rangle \langle B_2 \rangle$; otherwise, execute the next instruction in sequence.
CTc (Call subroutine if Condition True)	01 $1C_4 C_3$ 010 $\langle B_2 \rangle$ $\langle B_3 \rangle$	If $(c) = 1$, $(\text{Stack}) \leftarrow (P), (P) \leftarrow \langle B_3 \rangle \langle B_2 \rangle$. 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 $\langle B_3 \rangle \langle B_2 \rangle$; otherwise, execute the next instruction in sequence.

In the above JUMP and CALL instructions $\langle B_2 \rangle$ contains the least significant half of the address and $\langle B_3 \rangle$ contains the most significant half of the address. Note that D_6 and D_7 of $\langle B_3 \rangle$ 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 (Return Condition False)	00 0C ₄ C ₃ 011	If (c) = 0, (P)←(Stack); otherwise, (P) = (P)+1. 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 (Return Condition True)	00 1C ₄ C ₃ 011	If (c) = 1, (P)←(Stack); otherwise, (P) = (P)+1. 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 D ₀ , D ₁ , D ₂ , D ₃ 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. RRMM denotes output device number (RR ≠ 00).
-----	------------	---

MACHINE INSTRUCTION

HALT INSTRUCTION – One Byte

(one cycle – PCI)

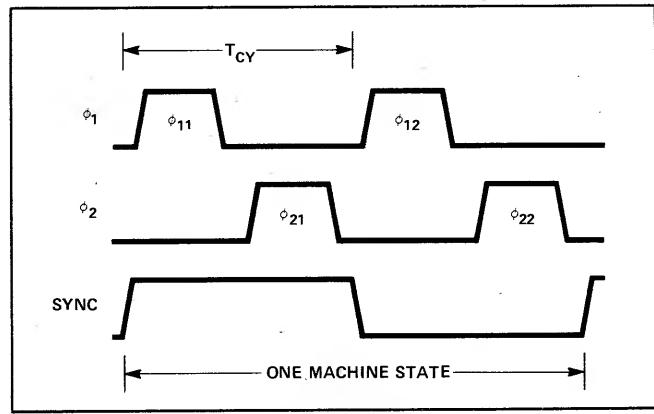
HLT	00 000 00X or 11 111 111	On receipt of the Halt Instruction, the activity of the processor is immediately suspended in the STOPPED state. The content of all registers and memory is unchanged. The P-counter has been updated and the internal dynamic memories continue to be refreshed.
-----	--------------------------------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

D. Internal Processor Operation

Internally the processor operates through five different states:

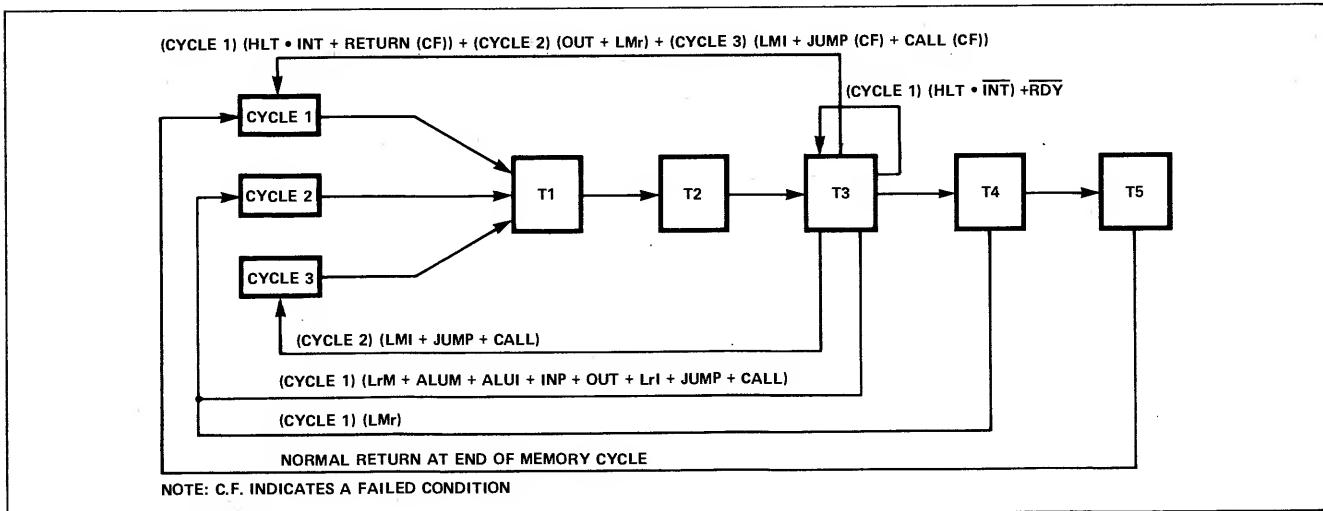
	Internal State	Typical Function
T1	NORMAL	Send out lower eight bits of address and increment program counter.
T2	INTERRUPT	Send out lower eight bits of address and suppress incrementing of program counter and acknowledge interrupt.
T3	WAIT	Send out six higher order bits of address and two control bits, D ₆ and D ₇ . Increment program counter if there has been a carry from T1.
T3	NORMAL	Wait for READY signal to come true. Refresh internal dynamic memories while waiting.
T3	STOPPED	Fetch and decode instruction; fetch data from memory; output data to memory. Refresh internal memories.
T4 and T5		Remain stopped until INTERRUPT occurs. Refresh internal memories.
T4 and T5		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.

The 8008 is driven by two non-overlapping clocks. Two clock periods are required for each state of the processor. ϕ_1 is generally used to precharge all data lines and memories and ϕ_2 controls all data transfers within the processor. A SYNC signal (divide by two of ϕ_2) is sent out by the 8008. This signal distinguishes between the two clock periods of each state.



Processor Clocks

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 as 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)

The following pages show the processor activity during each state of the execution of each instruction.

INTERNAL PROCESSOR OPERATION

INDEX REGISTER INSTRUCTIONS

INSTRUCTION CODING D ₇ D ₆ D ₅ D ₄ D ₃ D ₂ D ₁ D ₀		OPERATION	#OF STATES TO EXECUTE INSTRUCTION	MEMORY CYCLE ONE (1)				
				T1(2)	T2	T3	T4(3)	T5
1 1 D D D S S S		Lr1r2	5	PC _L OUT (4)	PC _H OUT	FETCH INSTR.(5) TO IR & REG. b	SSS TO REG. b (6)	REG. b TO DDD
1 1 D D D 1 1 1		LrM	8	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b		→
1 1 1 1 1 S S S		LMr	7	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b	SSS TO REG. b	→
0 0 D D D 1 1 0		Lrl	8	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b		→
0 0 1 1 1 1 1 0		LMI	9	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b		→
0 0 D D D 0 0 0		INr	5	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b	X	ADD OP - FLAGS AFFECTED
0 0 D D D 0 0 1		DCr	5	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b	X	SUB OP - FLAGS AFFECTED

ACCUMULATOR GROUP INSTRUCTIONS

1 0 P P P S S S	ALU OP r	5	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b	SSS TO REG. b	ALU OP - FLAGS AFFECTED
1 0 P P P 1 1 1	ALU OP M	8	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b		→
0 0 P P P 1 0 0	ALU OP I	8	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b		→
0 0 0 0 0 0 1 0	RLC	5	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b	X	ROTATE REG. A CARRY AFFECTED
0 0 0 0 1 0 1 0	RRC	5	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b	X	ROTATE REG. A CARRY AFFECTED
0 0 0 1 0 0 1 0	RAL	5	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b	X	ROTATE REG. A CARRY AFFECTED
0 0 0 1 1 0 1 0	RAR	5	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b	X	ROTATE REG. A CARRY AFFECTED

PROGRAM COUNTER AND STACK CONTROL INSTRUCTIONS

0 1 X X X 1 0 0	JMP	11	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b		→
0 1 U C C 0 0 0	JFc	9 or 11	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b		→
0 1 1 C C 0 0 0	JTc	9 or 11	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b		→
0 1 X X X 1 1 0	CAL	11	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b		→
0 1 0 C C 0 1 0	CFc	9 or 11	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b		→
0 1 1 C C 0 1 0	CTc	9 or 11	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b		→
0 0 X X X 1 1 1	RET	5	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b	POP STACK	X
0 0 0 C C 0 1 1	RFc	3 or 5	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b	POP STACK (13)	X
0 0 1 C C 0 1 1	RTc	3 or 5	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b	POP STACK (13)	X
0 0 A A A 1 0 1	RST	5	PC _L OUT	PC _H OUT	FETCH INSTR. TO REG. b AND PUSH STACK (0→REG. a)	REG. a TO PC _L	REG. b TO PC _H (14)

I/O INSTRUCTIONS

0 1 0 0 M M M 1	INP	8	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b		→
0 1 R R M M M 1	OUT	6	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b		→

MACHINE INSTRUCTIONS

0 0 0 0 0 0 0 X	HLT	4	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b & HALT (18)		
1 1 1 1 1 1 1 1	HLT	4	PC _L OUT	PC _H OUT	FETCH INSTR. TO IR & REG. b & HALT (18)		

NOTES:

1. The first memory cycle is always a PC_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 PC_L and higher order bits are designated by PC_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. PCR cycle (Memory Read Cycle).
9. "X" denotes an idle state.
10. PCW 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.

MEMORY CYCLE TWO					MEMORY CYCLE THREE				
T1	T2	T3	T4(3)	T5	T1	T2	T3	T4(3)	T5
REG. L OUT (8)	REG. H OUT	DATA TO REG. b	X (9)	REG. b TO DDD					
REG. L OUT (10)	REG. H OUT	REG. b TO OUT							
PCL OUT (8)	PC _H OUT	DATA TO REG. b	X	REG. b TO DDD					
PCL OUT (8)	PC _H OUT	DATA TO REG. b		→	REG. L OUT (10)	REG. H OUT	REG. b TO OUT		

REG. L OUT (8)	REG. H OUT	DATA TO REG. b	X	ALU OP - FLAGS AFFECTED					
PCL OUT (8)	PC _H OUT	DATA TO REG. b	X	ARITH OP - FLAGS AFFECTED					

PCL OUT (8)	PC _H OUT	LOWER ADD. TO REG. b	→	PCL OUT (8)	PC _H OUT	HIGHER ADD. REG. a	REG. a TO PC _H	REG. b TO PCL
PCL OUT (8)	PC _H OUT	LOWER ADD. TO REG. b	→	PCL OUT (8)	PC _H OUT	HIGHER ADD. REG. a (11)	REG. a TO PC _H	REG. b TO PCL
PCL OUT (8)	PC _H OUT	LOWER ADD. TO REG. b	→	PCL OUT (8)	PC _H OUT	HIGHER ADD. REG. a (11)	REG. a TO PC _H	REG. b TO PCL
PCL OUT (8)	PC _H OUT	LOWER ADD. TO REG. b	→	PCL OUT (8)	PC _H OUT	HIGHER ADD. REG. a	REG. a TO PC _H	REG. b TO PCL
PCL OUT (8)	PC _H OUT	LOWER ADD. TO REG. b	→	PCL OUT (8)	PC _H OUT	HIGHER ADD. REG. a (12)	REG. a TO PC _H	REG. b TO PCL
PCL OUT (8)	PC _H OUT	LOWER ADD. TO REG. b	→	PCL OUT (8)	PC _H OUT	HIGHER ADD. REG. a (12)	REG. a TO PC _H	REG. b TO PCL

REG. A TO OUT (15)	REG. b TO OUT	DATA TO REG. b	COND ff OUT (16)	REG. b TO REG. A				
REG. A (15) TO OUT	REG. b TO OUT	X (17)						

12. 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 D₃ through D₅ are loaded into PCL and all other bits are set to zero; zeros are loaded into PC_H.
15. PCC cycle (I/O Cycle).
16. The content of the condition flip-flops is available at the data bus: S at D₀, Z at D₁, P at D₂, C at D₃. (D₄ – D₇ 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 ϕ_1 or ϕ_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 ϕ_1 . An example of a synchronizing circuit is shown on the schematic for the SIM8-01 (Section VII).

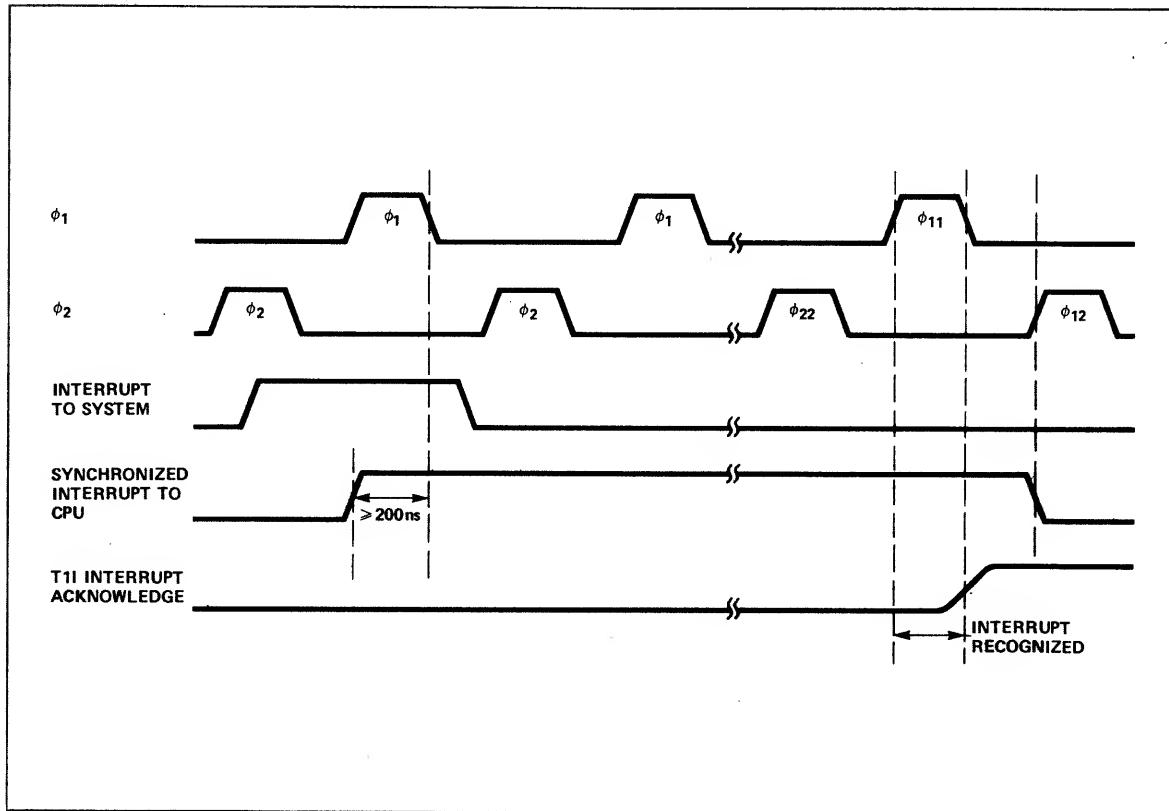


Figure 4. Recognition of Interrupt

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.

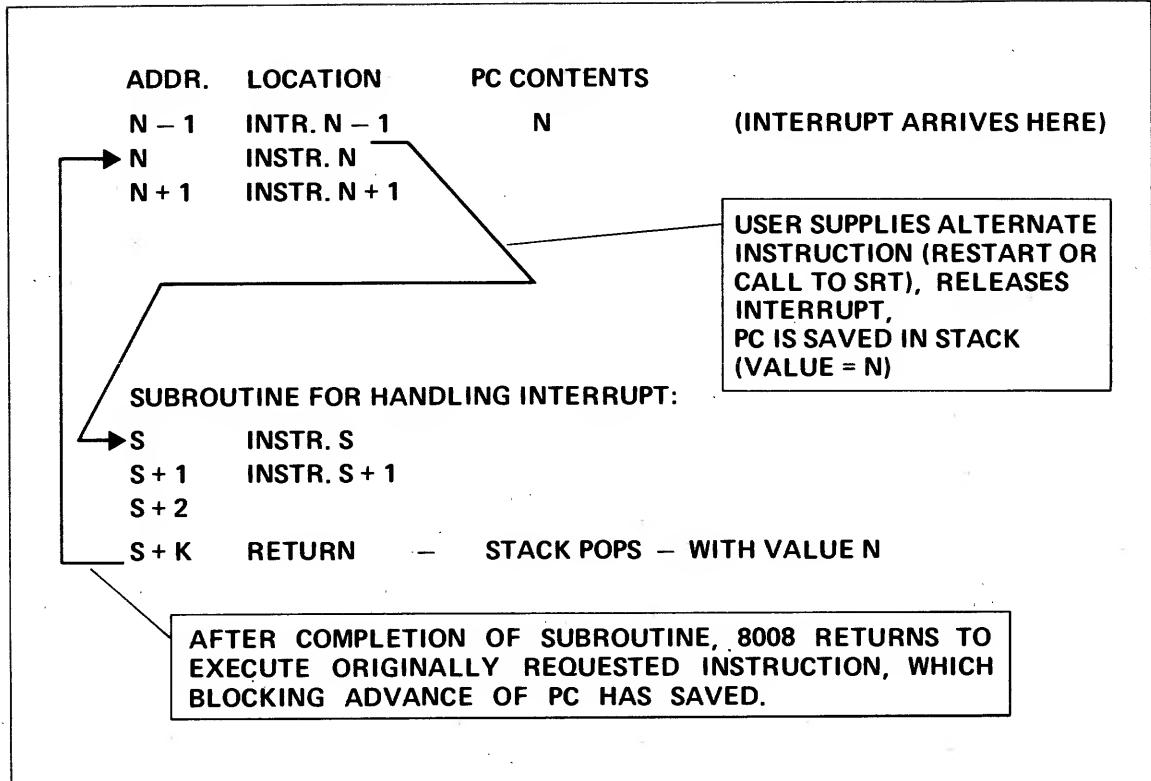


Figure 5. 8008 Interrupt

2) START-UP OF THE 8008

When power (V_{DD}) and clocks (ϕ_1, ϕ_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 T1I at some time later than ϕ_{11} . Note that whenever the 8008 is in a T1I 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 source 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
0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 1	INSTR ₁
0 0 0 0 0 0 0	0 0 0 0 0 0 0 1 0	INSTR ₂

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 0 X Y Z 0 0 0	INSTR ₁
0 0 0 0 0 0 0	0 0 X Y Z 0 0 1	INSTR ₂

EXAMPLE 2:

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

8008 ADDRESS OUT	INSTRUCTION IN ROM	
0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	INSTR ₁ (RST = 00 XYZ 101)
0 0 0 0 0 0 0	0 0 X Y Z 0 0 0	INSTR _a
0 0 0 0 0 0 0	0 0 X Y Z 0 0 1	INSTR _b

0 0 0 0 0 0 0	0 0 n n n n n n	RETURN	
0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	INSTR ₁	(INSTR ₁ executed now) Main Program
0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 1	INSTR ₂	

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 V_{CC}) 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.

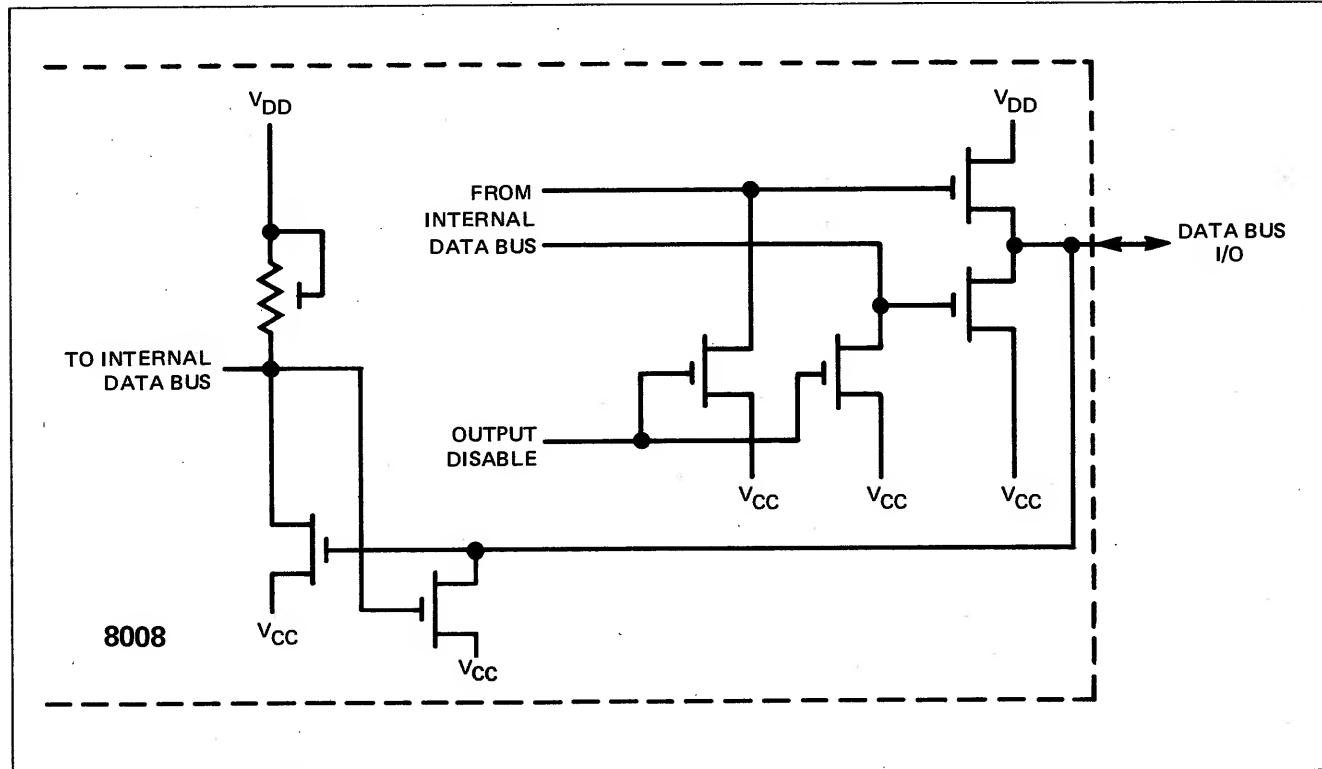


Figure 6. Data Bus I/O Buffer

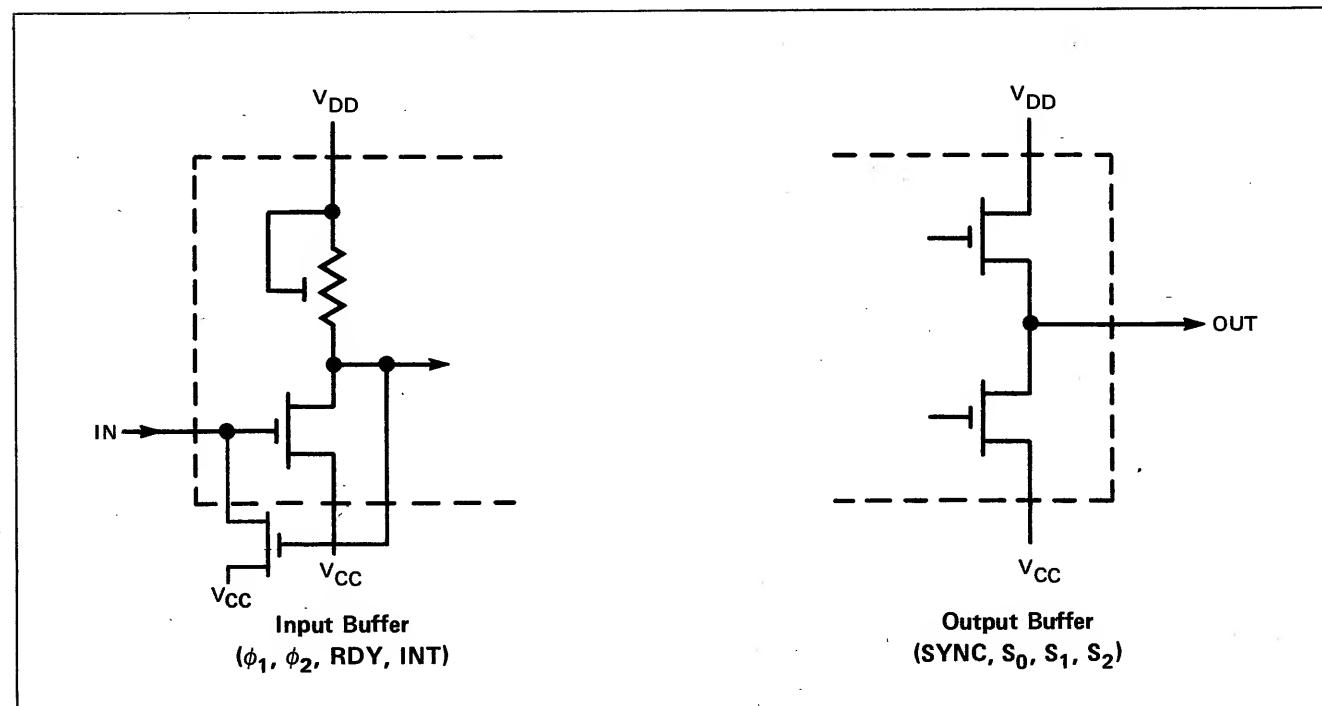


Figure 7. I/O Circuitry

ABSOLUTE MAXIMUM RATINGS*

	*COMMENT			
Ambient Temperature Under Bias	0°C to $+70^{\circ}\text{C}$			
Storage Temperature	-55°C to $+150^{\circ}\text{C}$			
Input Voltages and Supply Voltage With Respect to V_{CC}	$+0.5$ to -20V			
Power Dissipation	1.0 W @ 25°C			

D.C. AND OPERATING CHARACTERISTICS

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

SYMBOL	PARAMETER	LIMITS			UNIT	TEST CONDITIONS
		MIN.	TYP.	MAX.		
I_{DD}	AVERAGE SUPPLY CURRENT-OUTPUTS LOADED*		30	60	mA	$T_A = 25^{\circ}\text{C}$
I_{LI}	INPUT LEAKAGE CURRENT			10	μA	$V_{\text{IN}} = 0\text{V}$
V_{IL}	INPUT LOW VOLTAGE (INCLUDING CLOCKS)	V_{DD}		$V_{\text{CC}} - 4.2$	V	
V_{IH}	INPUT HIGH VOLTAGE (INCLUDING CLOCKS)	$V_{\text{CC}} - 1.5$		$V_{\text{CC}} + 0.3$	V	
V_{OL}	OUTPUT LOW VOLTAGE			0.4	V	$I_{\text{OL}} = 0.44\text{mA}$ $C_L = 200\text{ pF}$
V_{OH}	OUTPUT HIGH VOLTAGE	$V_{\text{CC}} - 1.5$			V	$I_{\text{OH}} = 0.2\text{mA}$

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

A.C. CHARACTERISTICS

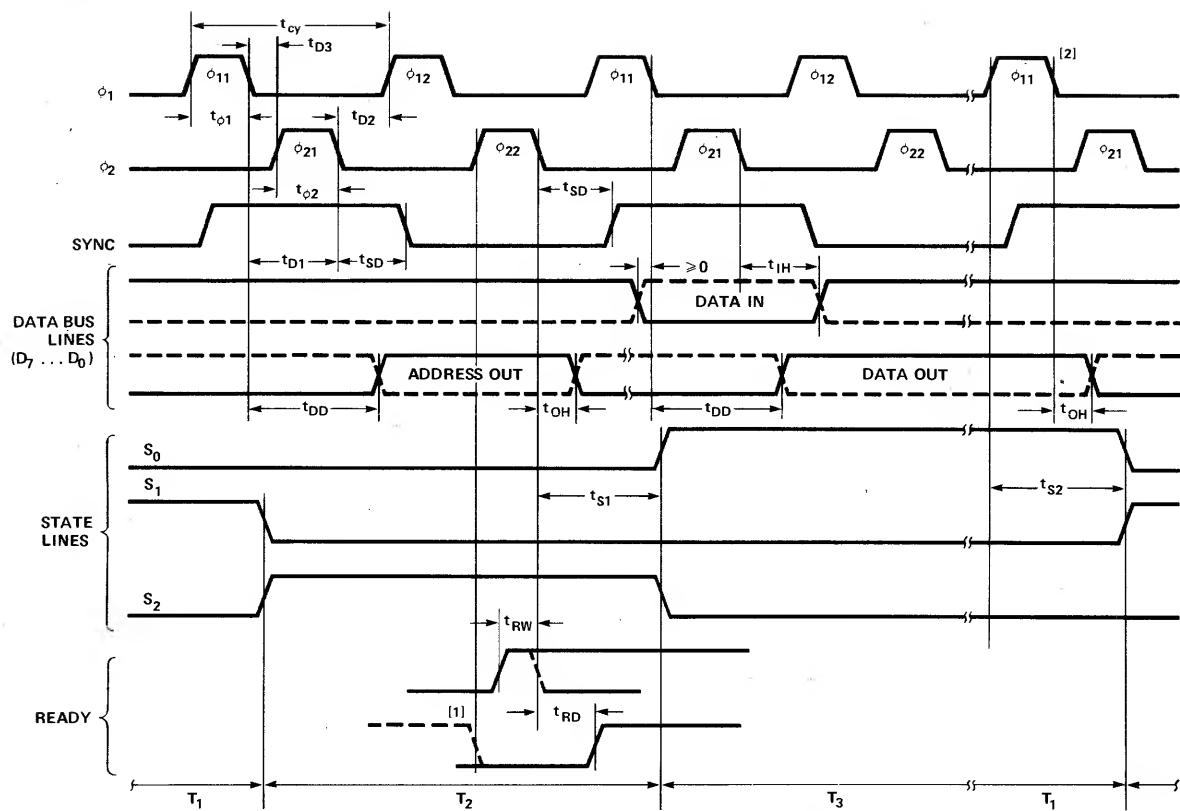
$T_A = 0^{\circ}\text{C}$ to 70°C ; $V_{\text{CC}} = +5\text{V} \pm 5\%$, $V_{\text{DD}} = -9\text{V} \pm 5\%$. All measurements are referenced to 1.5V levels.

SYMBOL	PARAMETER	8008		8008-1		UNIT	TEST CONDITIONS		
		LIMITS		LIMITS					
		MIN.	MAX.	MIN.	MAX.				
t_{CY}	CLOCK PERIOD	2	3	1.25	3	μs	$t_R, t_F = 50\text{ns}$		
t_R, t_F	CLOCK RISE AND FALL TIMES		50		50	ns			
$t_{\phi 1}$	PULSE WIDTH OF ϕ_1	.70		.35		μs			
$t_{\phi 2}$	PULSE WIDTH OF ϕ_2	.55		.35		μs			
t_{D1}	CLOCK DELAY FROM FALLING EDGE OF ϕ_1 TO FALLING EDGE OF ϕ_2	.90	1.1		1.1	μs			
t_{D2}	CLOCK DELAY FROM ϕ_2 TO ϕ_1	.40		.35		μs			
t_{D3}	CLOCK DELAY FROM ϕ_1 TO ϕ_2	.20		.20		μs			
t_{DD}	DATA OUT DELAY		1.0		1.0	μs	$C_L = 100\text{pF}$		
t_{OH}	HOLD TIME FOR DATA BUS OUT	.10		.10		μs			
t_{IH}	HOLD TIME FOR DATA IN	[1]		[1]		μs			
t_{SD}	SYNC OUT DELAY		.70		.70	μs	$C_L = 100\text{pF}$		
t_{S1}	STATE OUT DELAY (ALL STATES EXCEPT T1 AND T1I) ^[2]		1.1		1.1	μs	$C_L = 100\text{pF}$		
t_{S2}	STATE OUT DELAY (STATES T1 AND T1I)		1.0		1.0	μs	$C_L = 100\text{pF}$		
t_{RW}	PULSE WIDTH OF READY DURING ϕ_{22} TO ENTER T3 STATE	.35		.35		μs			
t_{RD}	READY DELAY TO ENTER WAIT STATE	.20		.20		μs			

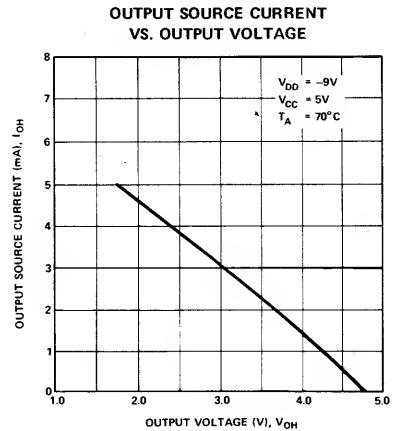
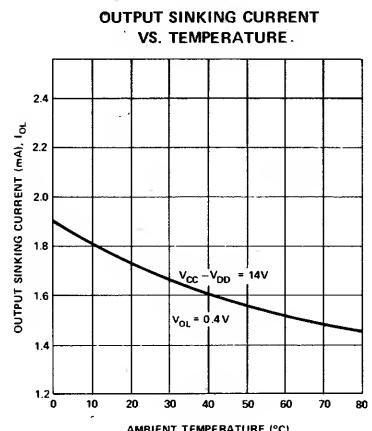
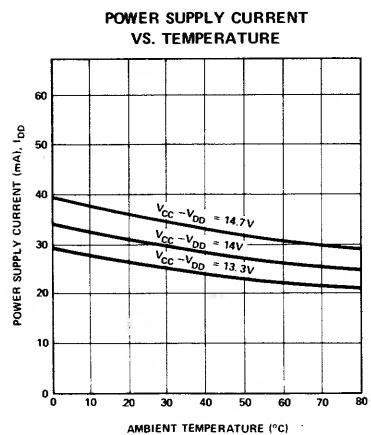
[1] $t_{IH} \text{ MIN} \geq t_{SD}$

[2] If the INTERRUPT is not used, all states have the same output delay, t_{S1} .

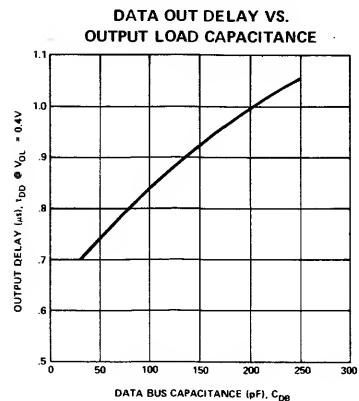
TIMING DIAGRAM



TYPICAL D.C. CHARACTERISTICS



TYPICAL A.C. CHARACTERISTICS



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

SYMBOL	TEST	LIMIT (pF)	
		TYP.	MAX.
C_{IN}	INPUT CAPACITANCE	5	10
C_{DB}	DATA BUS I/O CAPACITANCE	5	10
C_{OUT}	OUTPUT CAPACITANCE	5	10

VII THE SIM8-01 — AN MCS-8 T.M. MICRO COMPUTER

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.

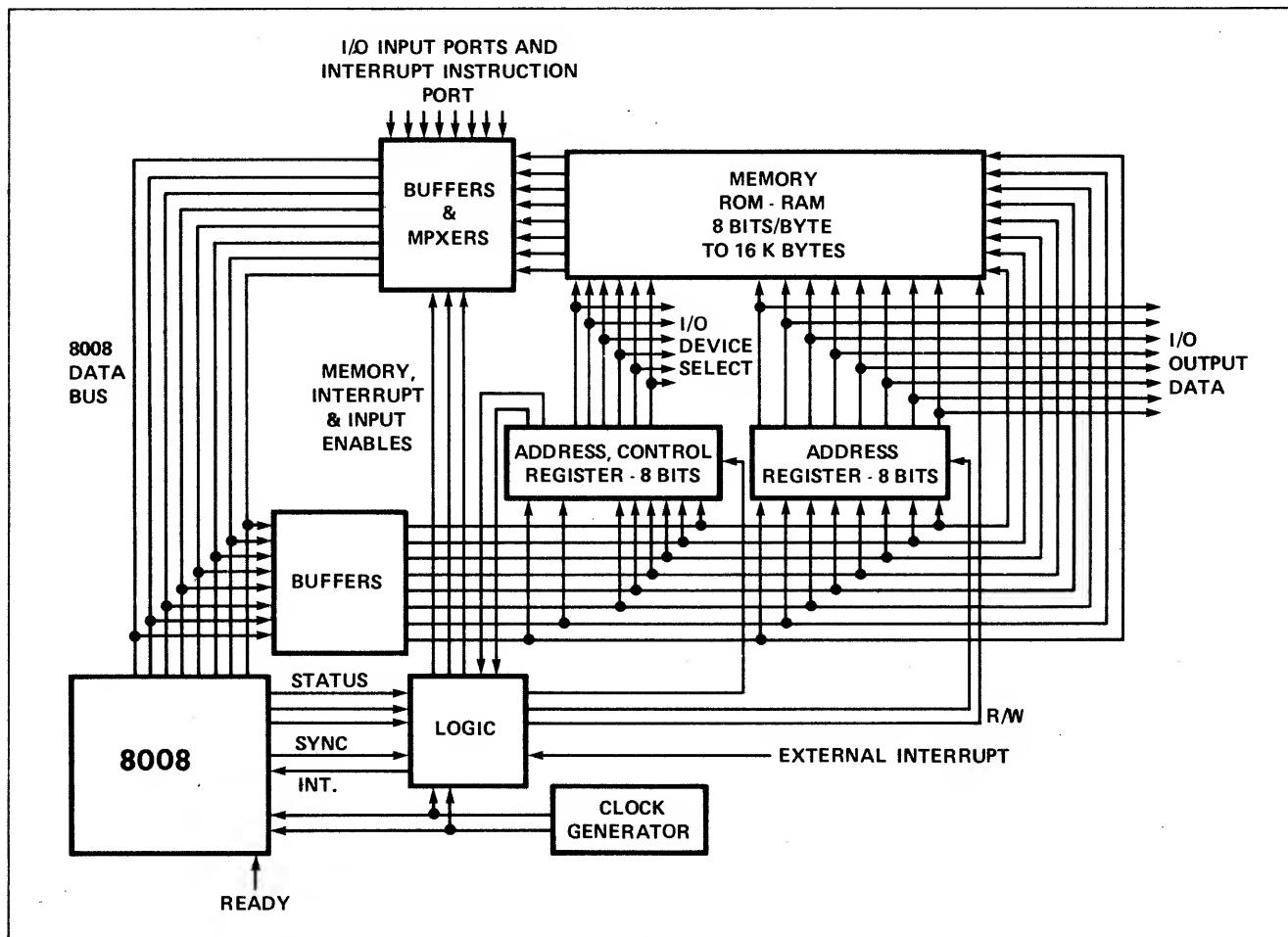


Figure 8. MCS-8 Basic System

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 cks.
- 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 μ s clock period
- 20 μ s typical instruction cycle

D.C. Power Requirement:

- Voltage:

$$V_{CC} = 5V \pm 5\%$$

$$TTL GRD = 0V$$

$$V_{DD} = -9V \pm 5\%$$

- Current:

Eight ROMs

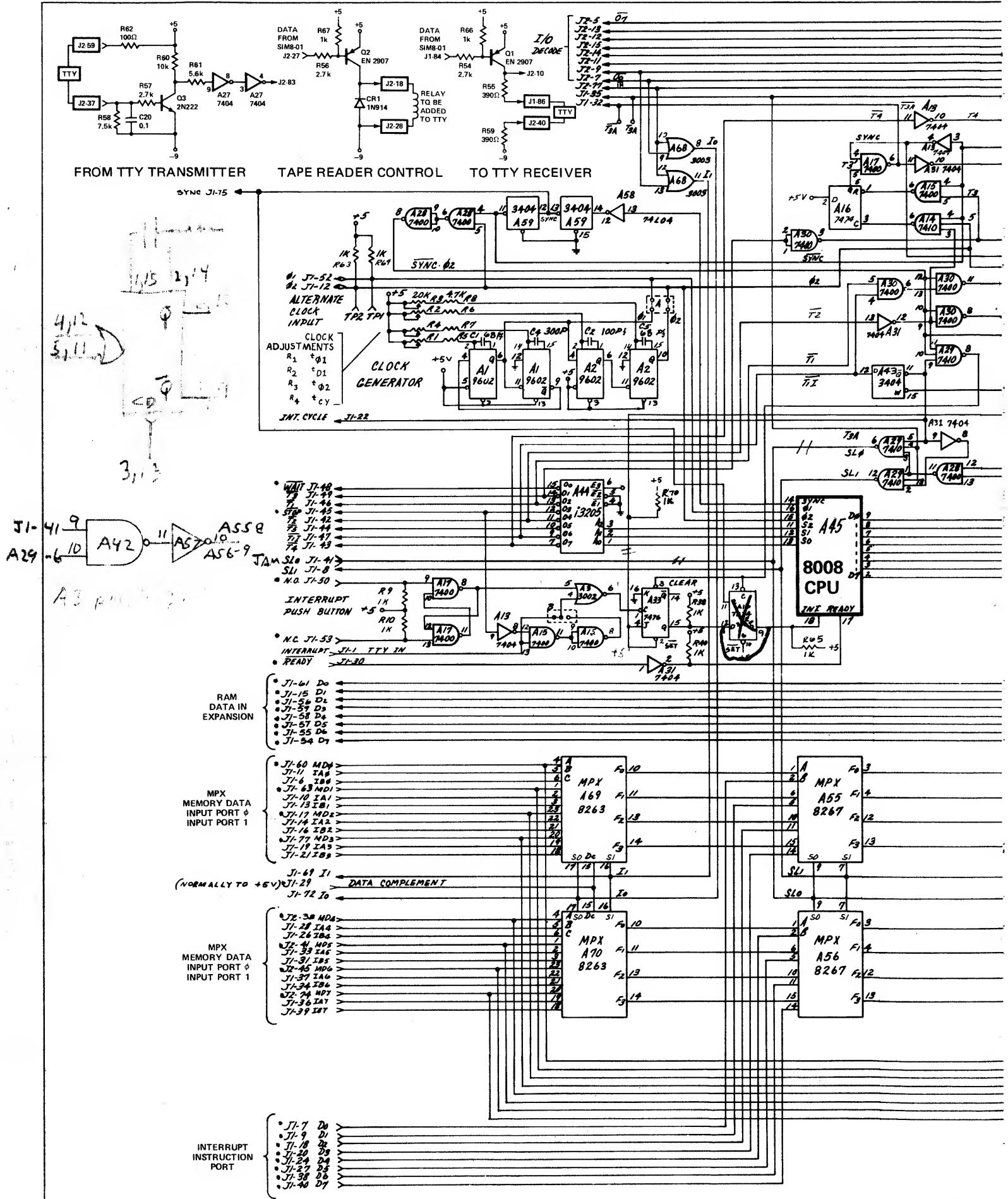
	<u>Typical</u>	<u>Maximum</u>
I_{CC}	2.5 amps	4.0 amps.
I_{DD}	1.0 amps	1.5 amps.

Connector:

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



Figure 9. MCS-8 Memory System



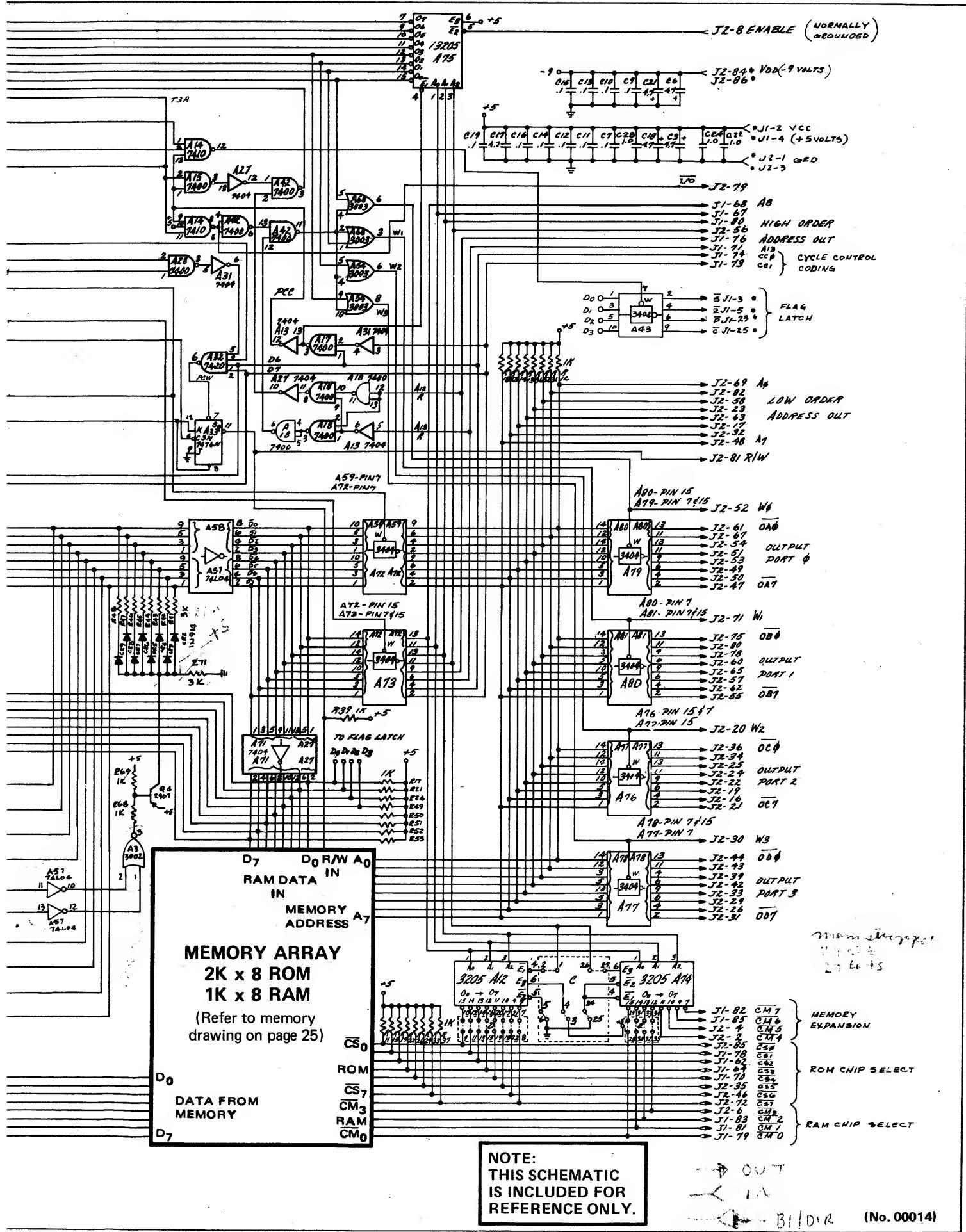


Figure 10. Complete SIM8-01 Schematic

SYSTEM DESCRIPTION

The 8008 processor communicates over an 8-bit data bus (D_0 through D_7) and uses two input lines (READY and INTERRUPT) and four output lines (S_0 , S_1 , S_2 , 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 (A_1 , A_2). 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.

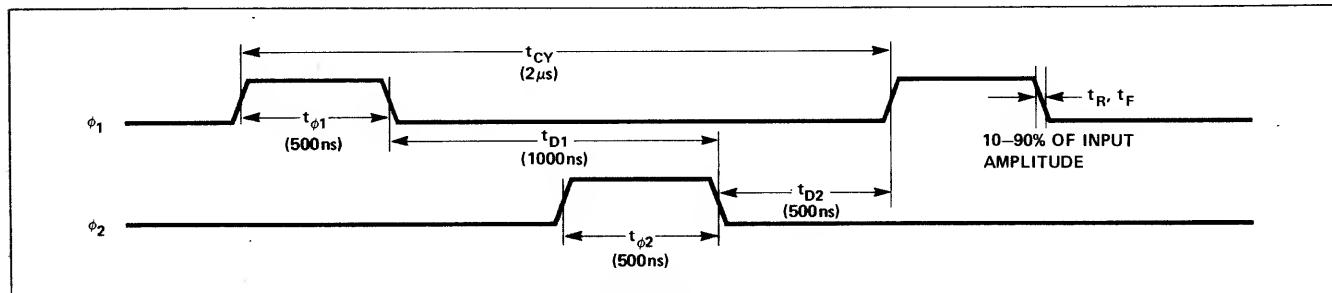


Figure 11. SIM8-01 Timing Diagram

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.

ADDRESS			ENABLE			OUTPUTS							
A_0	A_1	A_2	E_1	E_2	E_3	0	1	2	3	4	5	6	7
L	L	L	L	L	H	L	H	H	H	H	H	H	H
H	L	L	L	L	H	H	L	H	H	H	H	H	H
L	H	L	L	L	H	H	H	L	H	H	H	H	H
H	H	L	L	L	H	H	H	H	L	H	H	H	H
L	L	H	L	L	H	H	H	H	H	L	H	H	H
H	L	H	L	L	H	H	H	H	H	L	H	H	H
L	H	H	L	L	H	H	H	H	H	H	L	H	H
H	H	H	L	L	H	H	H	H	H	H	H	H	L
X	X	X	L	L	L	H	H	H	H	H	H	H	H
X	X	X	H	L	L	H	H	H	H	H	H	H	H
X	X	X	L	H	L	H	H	H	H	H	H	H	H
X	X	X	H	H	L	H	H	H	H	H	H	H	H
X	X	X	H	L	H	H	H	H	H	H	H	H	H
X	X	X	L	H	H	H	H	H	H	H	H	H	H
X	X	X	H	H	H	H	H	H	H	H	H	H	H

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 (S_0 , S_1 , S_2) 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 a 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.

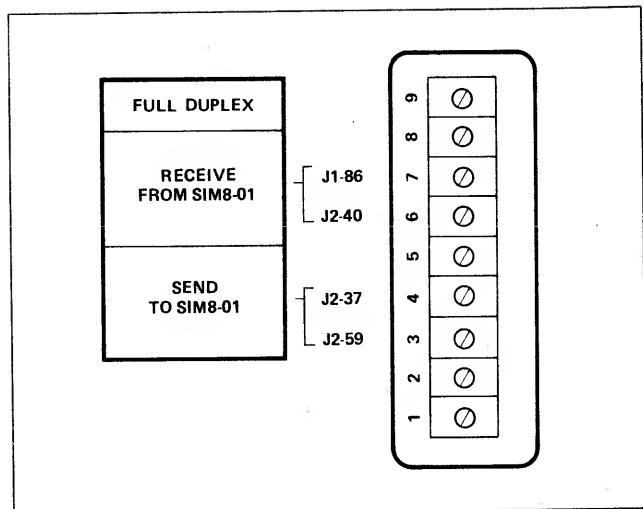


Figure 12. Teletype Terminal Strip

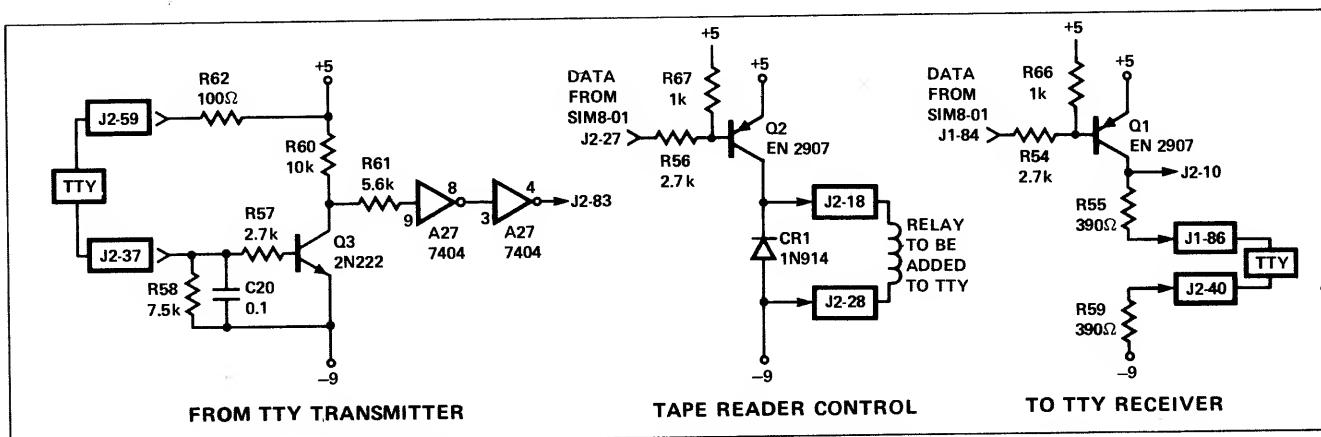


Figure 13. SIM8-01 Teletype Interface Circuitry

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.

SIM8-01 MICRO COMPUTER BOARD PIN DESCRIPTION

Pin No.	Connector	Symbol	Description	Pin No.	Connector	Symbol	Description
2,4	J1	+5V	+5VDC POWER SUPPLY	57	J1	D5	RAM DATA IN D ₅
84 & 86	J2	-9V	-9VDC POWER SUPPLY	55	J1	D6	RAM DATA IN D ₆
1,3	J2	GND	GROUND	54	J1	D7	RAM DATA IN D ₇
60	J1	MD ₀	DATA FROM MEMORY # BIT #	48	J1	WAIT	STATE COUNTER
63	J1	MD ₁	DATA FROM MEMORY 1 BIT 1	49	J1	T ₃	STATE COUNTER
17	J1	MD ₂	DATA FROM MEMORY 2 BIT 2	46	J1	T ₁	STATE COUNTER
77	J1	MD ₃	DATA FROM MEMORY 3 BIT 3	45	J1	STOP	STATE COUNTER
38	J2	MD ₄	DATA FROM MEMORY 4 BIT 4	42	J1	T ₂	STATE COUNTER
41	J2	MD ₅	DATA FROM MEMORY 5 BIT 5	44	J1	T ₅	STATE COUNTER
45	J2	MD ₆	DATA FROM MEMORY 6 BIT 6	47	J1	T ₁ X	STATE COUNTER
74	J2	MD ₇	DATA FROM MEMORY 7 BIT 7	43	J1	T ₄	STATE COUNTER
11	J1	IA ₀	DATA INPUT PORT # BIT #	79	J1	CH ₀	RAM CHIP SELECT #
10	J1	IA ₁	DATA INPUT PORT # BIT 1	81	J1	CH ₁	RAM CHIP SELECT 1
14	J1	IA ₂	DATA INPUT PORT # BIT 2	83	J1	CH ₂	RAM CHIP SELECT 2
19	J1	IA ₃	DATA INPUT PORT # BIT 3	6	J2	CH ₃	RAM CHIP SELECT 3
28	J1	IA ₄	DATA INPUT PORT # BIT 4	2	J2	CH ₄	RAM CHIP SELECT 4
33	J1	IA ₅	DATA INPUT PORT # BIT 5	4	J2	CH ₅	RAM CHIP SELECT 5
37	J1	IA ₆	DATA INPUT PORT # BIT 6	85	J1	CH ₆	RAM CHIP SELECT 6
36	J1	IA ₇	DATA INPUT PORT # BIT 7	82	J1	CH ₇	RAM CHIP SELECT 7
6	J1	IB ₀	DATA INPUT PORT 1 BIT #	85	J2	CS#	ROM CHIP SELECT #
13	J1	IB ₁	DATA INPUT PORT 1 BIT 1	78	J1	CS1	ROM CHIP SELECT 1
16	J1	IB ₂	DATA INPUT PORT 1 BIT 2	62	J1	CS2	ROM CHIP SELECT 2
21	J1	IB ₃	DATA INPUT PORT 1 BIT 3	64	J1	CS3	ROM CHIP SELECT 3
26	J1	IB ₄	DATA INPUT PORT 1 BIT 4	70	J1	CS4	ROM CHIP SELECT 4
31	J1	IB ₅	DATA INPUT PORT 1 BIT 5	35	J2	CS5	ROM CHIP SELECT 5
34	J1	IB ₆	DATA INPUT PORT 1 BIT 6	46	J2	CS6	ROM CHIP SELECT 6
39	J1	IB ₇	DATA INPUT PORT 1 BIT 7	72	J2	CS7	ROM CHIP SELECT 7
61	J2	OA ₀	OUTPUT PORT # BIT #	5	J2	O ₇	I/O DECODE OUT O ₇
67	J2	OA ₁	OUTPUT PORT # BIT 1	13	J2	O ₆	I/O DECODE OUT O ₆
54	J2	OA ₂	OUTPUT PORT # BIT 2	12	J2	O ₅	I/O DECODE OUT O ₅
51	J2	OA ₃	OUTPUT PORT # BIT 3	15	J2	O ₄	I/O DECODE OUT O ₄
53	J2	OA ₄	OUTPUT PORT # BIT 4	14	J2	O ₃	I/O DECODE OUT O ₃
49	J2	OA ₅	OUTPUT PORT # BIT 5	11	J2	O ₂	I/O DECODE OUT O ₂
50	J2	OA ₆	OUTPUT PORT # BIT 6	9	J2	O ₁	I/O DECODE OUT O ₁
47	J2	OA ₇	OUTPUT PORT # BIT 7	7	J2	O ₀	I/O DECODE OUT O ₀
75	J2	OB#	OUTPUT PORT 1 BIT #	3	J1	S	FLAG FLIP FLOP-Sign
80	J2	OB ₁	OUTPUT PORT 1 BIT 1	5	J1	Z	FLAG FLIP FLOP-Zero
78	J2	OB ₂	OUTPUT PORT 1 BIT 2	23	J1	P	FLAG FLIP FLOP-Parity
60	J2	OB ₃	OUTPUT PORT 1 BIT 3	25	J1	C	FLAG FLIP FLOP-Carry
65	J2	OB ₄	OUTPUT PORT 1 BIT 4	7	J1	D ₀	INTERRUPT INSTRUCTION INPUT #
57	J2	OB ₅	OUTPUT PORT 1 BIT 5	9	J1	D ₁	INTERRUPT INSTRUCTION INPUT 1
62	J2	OB ₆	OUTPUT PORT 1 BIT 6	18	J1	D ₂	INTERRUPT INSTRUCTION INPUT 2
55	J2	OB ₇	OUTPUT PORT 1 BIT 7	20	J1	D ₃	INTERRUPT INSTRUCTION INPUT 3
36	J2	OC#	OUTPUT PORT 2 BIT #	24	J1	D ₄	INTERRUPT INSTRUCTION INPUT 4
34	J2	OC ₁	OUTPUT PORT 2 BIT 1	27	J1	D ₅	INTERRUPT INSTRUCTION INPUT 5
25	J2	OC ₂	OUTPUT PORT 2 BIT 2	38	J1	D ₆	INTERRUPT INSTRUCTION INPUT 6
24	J2	OC ₃	OUTPUT PORT 2 BIT 3	40	J1	D ₇	INTERRUPT INSTRUCTION INPUT 7
22	J2	OC ₄	OUTPUT PORT 2 BIT 4	59	J2		FROM TTY TRANSMITTER IN }
19	J2	OC ₅	OUTPUT PORT 2 BIT 5	37	J2		FROM TTY TRANSMITTER OUT } TTY BUFFER
16	J2	OC ₆	OUTPUT PORT 2 BIT 6	83	J2		DATA FROM TTY TRANSMITTER BUFFER
21	J2	OC ₇	OUTPUT PORT 2 BIT 7	27	J2		TAPE READER CONTROL IN
44	J2	OD#	OUTPUT PORT 3 BIT #	18	J2		TAPE READER CONTROL OUT
43	J2	OD ₁	OUTPUT PORT 3 BIT 1	28	J2		TAPE READER CONTROL (-9VDC)
39	J2	OD ₂	OUTPUT PORT 3 BIT 2	84	J1		DATA TO TTY RECEIVER BUFFER
42	J2	OD ₃	OUTPUT PORT 3 BIT 3	10	J2		TO TTY RECEIVER OUT }
33	J2	OD ₄	OUTPUT PORT 3 BIT 4	86	J1		TO TTY RECEIVER OUT } TTY BUFFER
29	J2	OD ₅	OUTPUT PORT 3 BIT 5	40	J2		TO TTY RECEIVER OUT }
26	J2	OD ₆	OUTPUT PORT 3 BIT 6	81	J2		READ/WRITE
31	J2	OD ₇	OUTPUT PORT 3 BIT 7	72	J1	I#	MULTIPLEXER CONTROL LINES N8263
69	J2	A ₀	LOW ORDER ADDRESS OUT	41	J1	SL#	MULTIPLEXER CONTROL LINES N8267
82	J2	A ₁	LOW ORDER ADDRESS OUT	69	J1	IL	MULTIPLEXER CONTROL LINES N8263
58	J2	A ₂	LOW ORDER ADDRESS OUT	8	J1	SLI	MULTIPLEXER CONTROL LINES N8267
23	J2	A ₃	LOW ORDER ADDRESS OUT	29	J1		DATA COMPLEMENT
63	J2	A ₄	LOW ORDER ADDRESS OUT	52	J1	# ₁	# ₁ CLOCK (alternate clock)
17	J2	A ₅	LOW ORDER ADDRESS OUT	12	J1	# ₂	# ₂ CLOCK (alternate clock)
32	J2	A ₆	LOW ORDER ADDRESS OUT	75	J1	SYNC	SYNC OUT
48	J2	A ₇	LOW ORDER ADDRESS OUT	30	J1	READY	READY IN
68	J1	A ₈	HIGH ORDER ADDRESS OUT	1	J1		INTERRUPT INTERRUPT IN
67	J1	A ₉	HIGH ORDER ADDRESS OUT	8	J2	I/O	I/O ENABLE ENABLE OF I/O DEVICE DECODER
80	J1	A ₁₀	HIGH ORDER ADDRESS OUT	79	J2	I/O	SYSTEM I/O CONTROL
56	J2	A ₁₁	HIGH ORDER ADDRESS OUT	77	J2	IN	SYSTEM INPUT CONTROL
76	J1	A ₁₂	HIGH ORDER ADDRESS OUT	50	J1	N.O.	PUSH BUTTON SWITCH } INTERRUPT
71	J1	A ₁₃	HIGH ORDER ADDRESS OUT	53	J1	N.C.	PUSH BUTTON SWITCH } INTERRUPT
74	J1	CC#	CYCLE CONTROL CODING	52	J2	W ₀	OUTPUT LATCH STROBE PORT #
73	J1	CC ₁	CYCLE CONTROL CODING	71	J2	W ₁	OUTPUT LATCH STROBE PORT 1
61	J1	D#	RAM DATA IN D#	20	J2	W ₂	OUTPUT LATCH STROBE PORT 2
15	J1	D ₁	RAM DATA IN D ₁	30	J2	W ₃	OUTPUT LATCH STROBE PORT 3
56	J1	D ₂	RAM DATA IN D ₂	22	J1		INT CYCLE INTERRUPT CYCLE INDICATOR
59	J1	D ₃	RAM DATA IN D ₃	32	J1	T ₃ A	ANTICIPATED T ₃ OUTPUT
58	J1	D ₄	RAM DATA IN D ₄	35	J1	T ₃ A	ANTICIPATED T ₃ OUTPUT

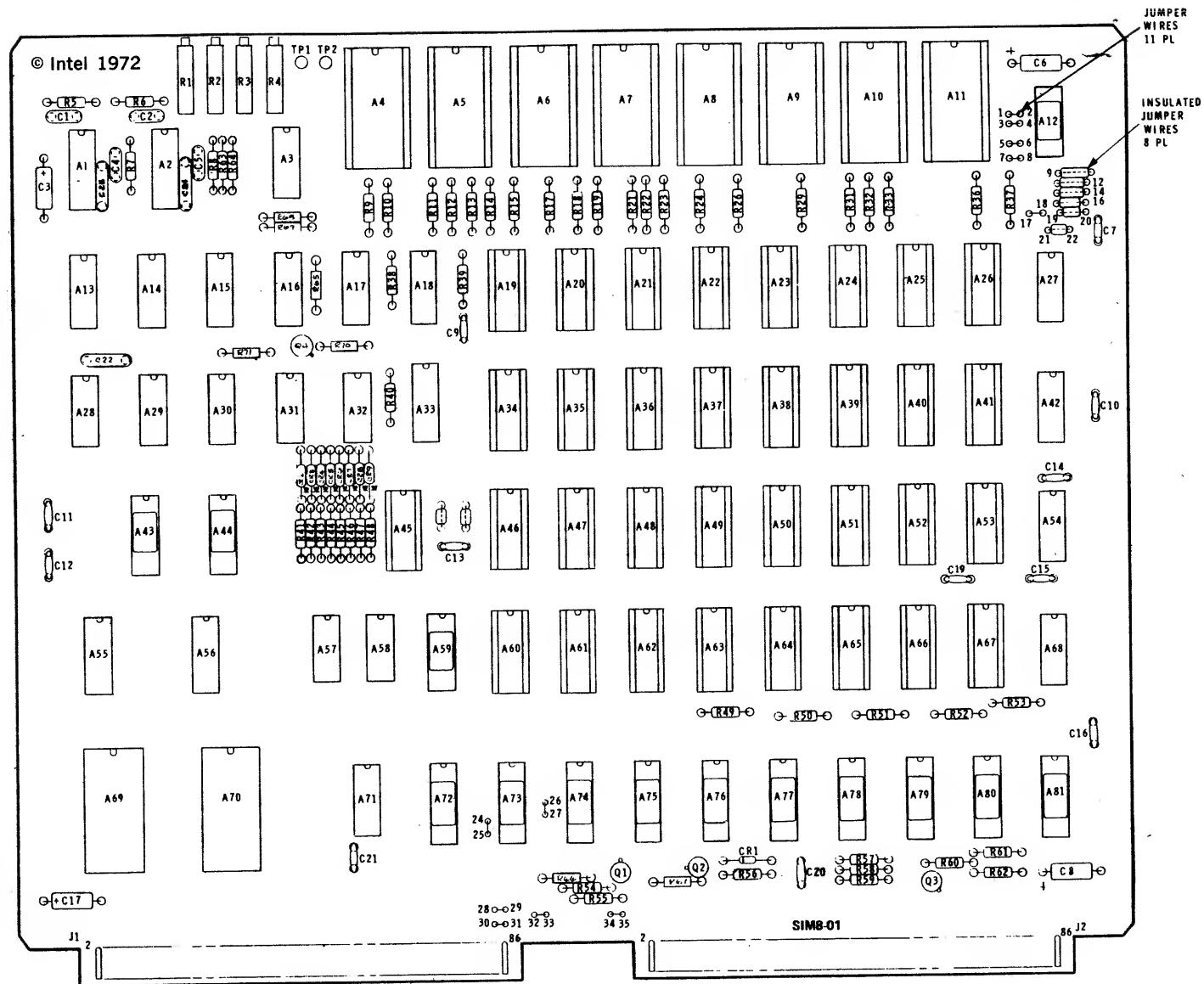


Figure 14. SIM8-01 Assembly Diagram

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.

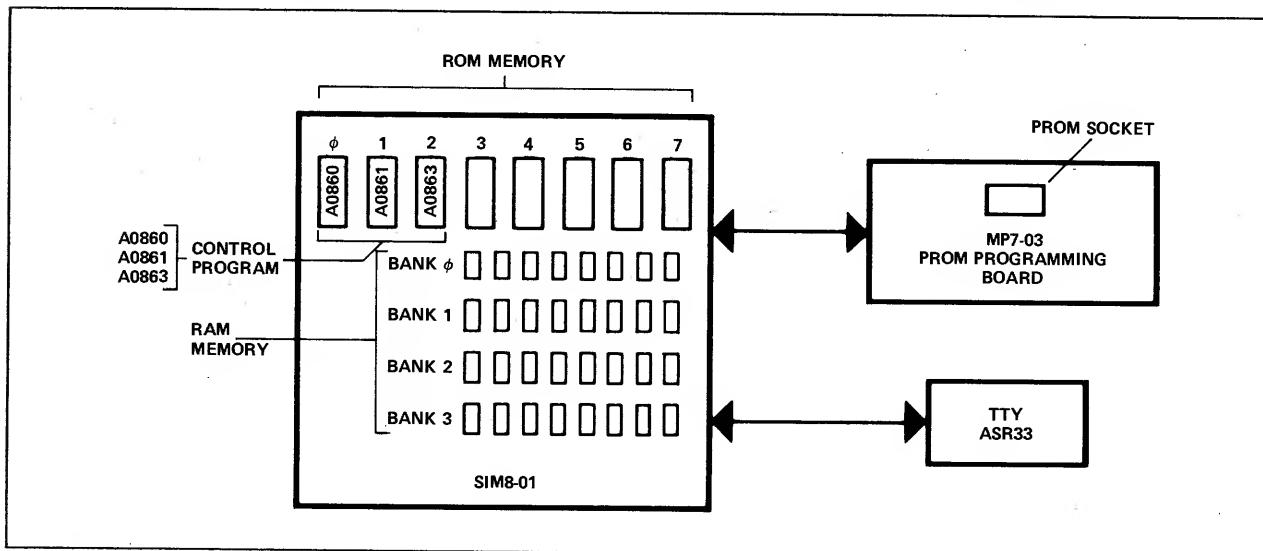


Figure 15. MCS-8 PROM Programming System

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.

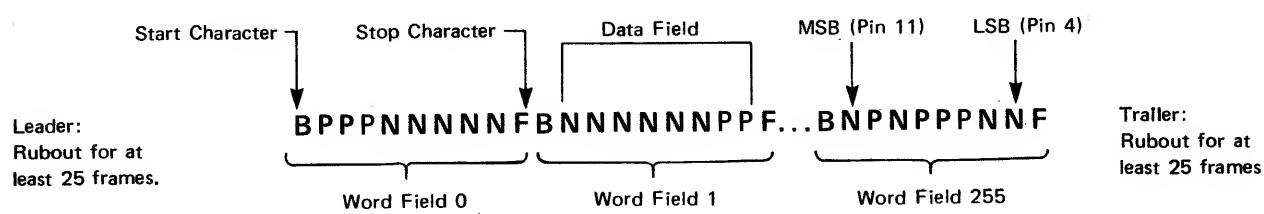
PROGRAMMING THE 1602A/1702A

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.

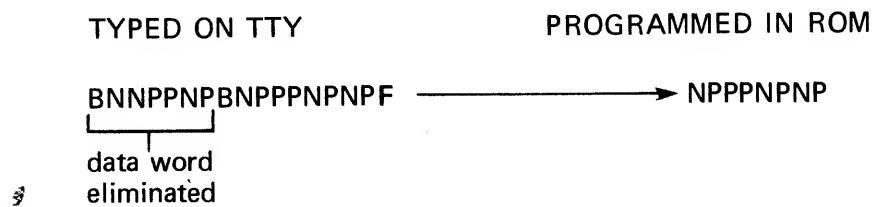
TAPE FORMAT

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:



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:

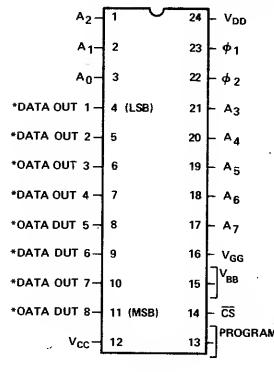


If any character other than P or N is entered, a format error is indicated. If the stop character is entered before the error is noticed, the entire word field, including the B and F, must be rubbed out. Within the word field, a P results in a high level output, and N results in a low level output. The 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.

PROM PIN CONFIGURATION



1602A/1702A

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.

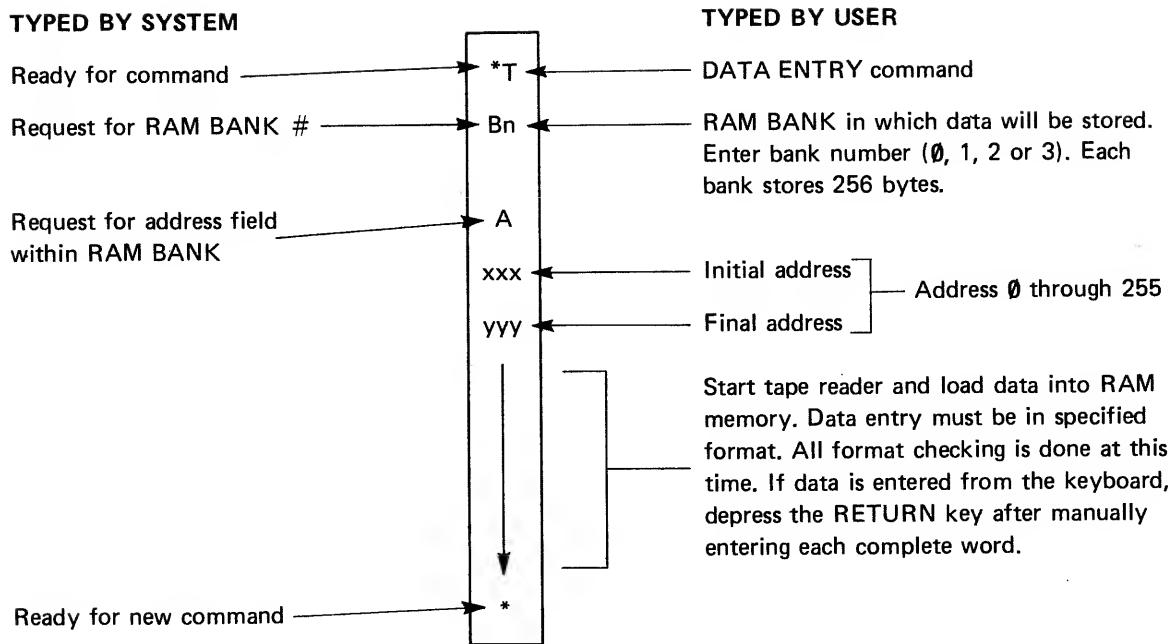
OPERATING THE PROGRAMMER

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 = 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 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.

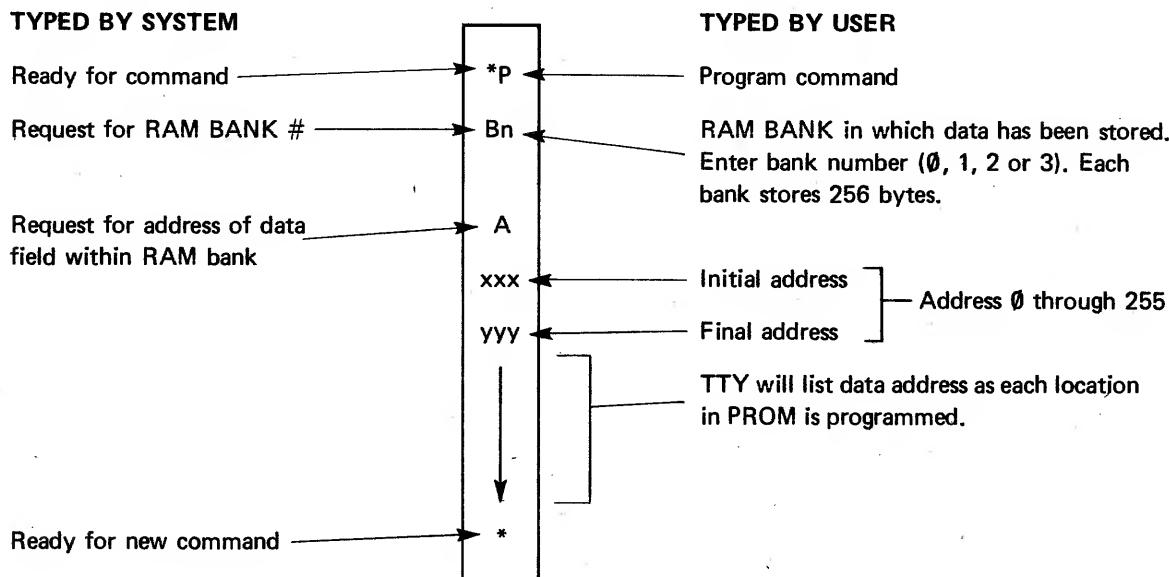
EXAMPLE 1:

Listing by TTY

020	B N 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
	0 2 2 ← format error indicated at address #022 (too many characters in data field).
	R ← RE-ENTER command
	B N 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 ← Start the tape reader and continue
024	B P N M F E
	0 2 4 ← 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

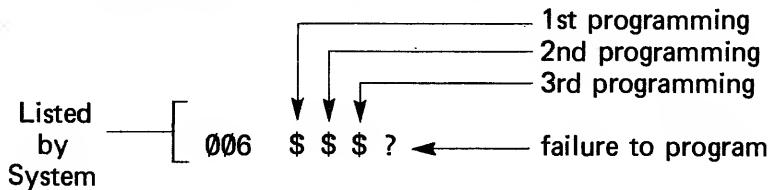
After data has been entered, the PROM may be programmed. Data from a designated address field in a designated RAM bank is programmed into corresponding addresses in the PROM. A complete PROM or any portion of a PROM may be programmed in the following manner:



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.

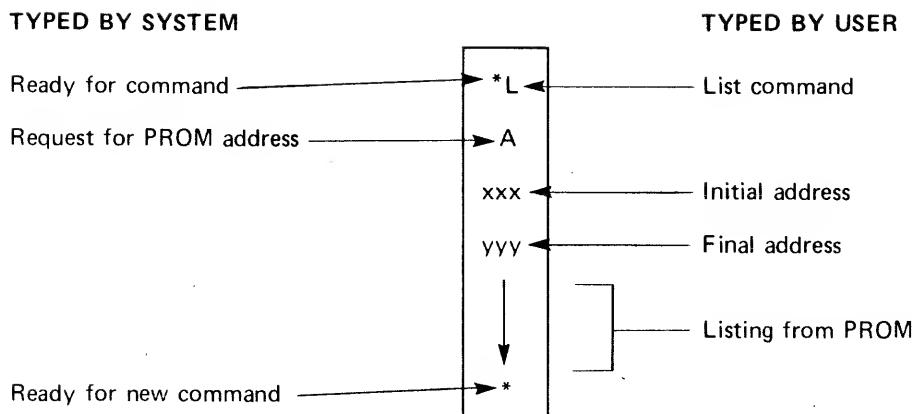
EXAMPLE 2:



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).

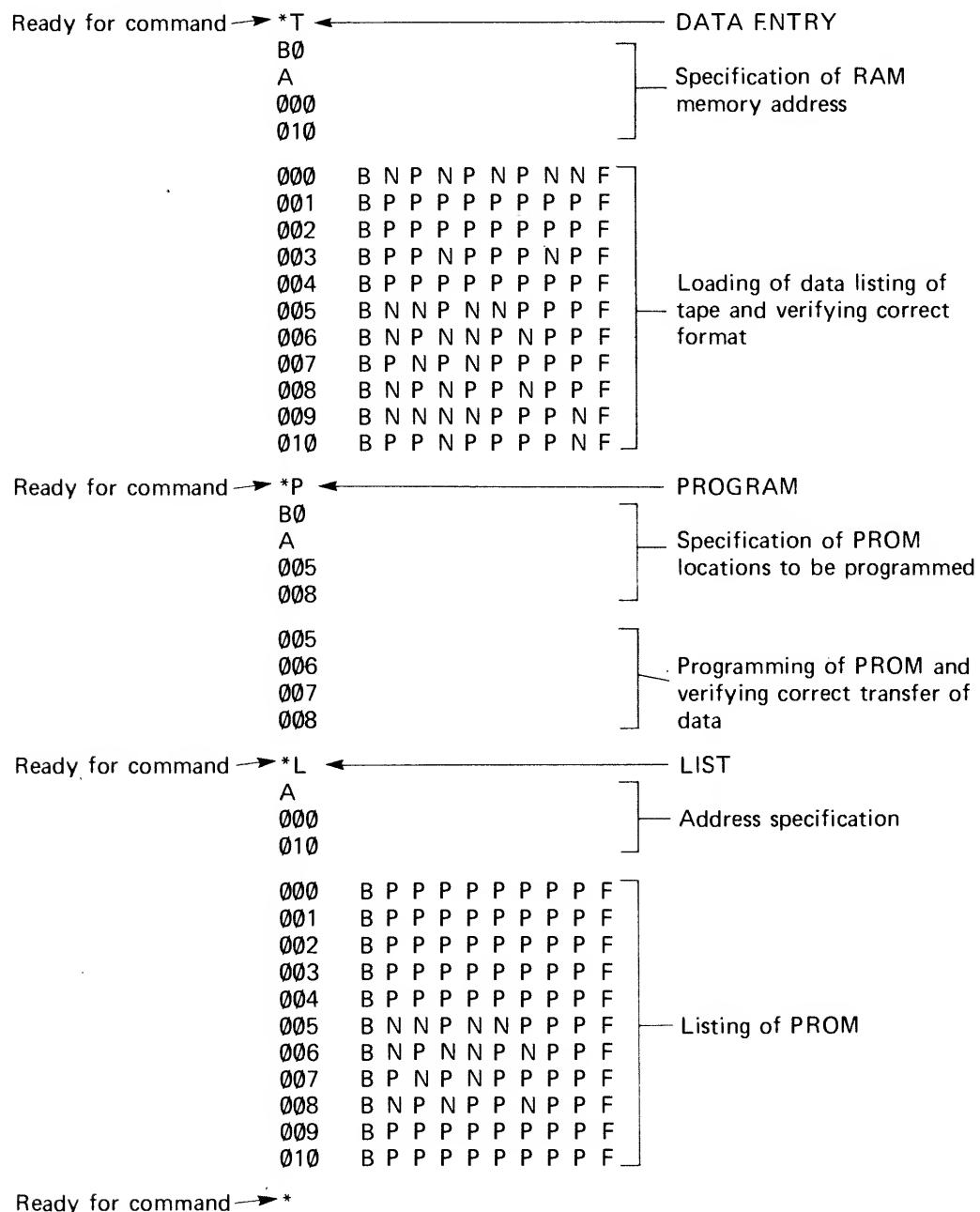
PROGRAM LISTING

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:



1702A ERASING PROCEDURE

The 1702A may be erased by exposure to high intensity short-wave ultraviolet light at a wavelength of 2537 Å. 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

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

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

Dimensions:

8.4 inches high
9.5 inches deep

Power Requirement:

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

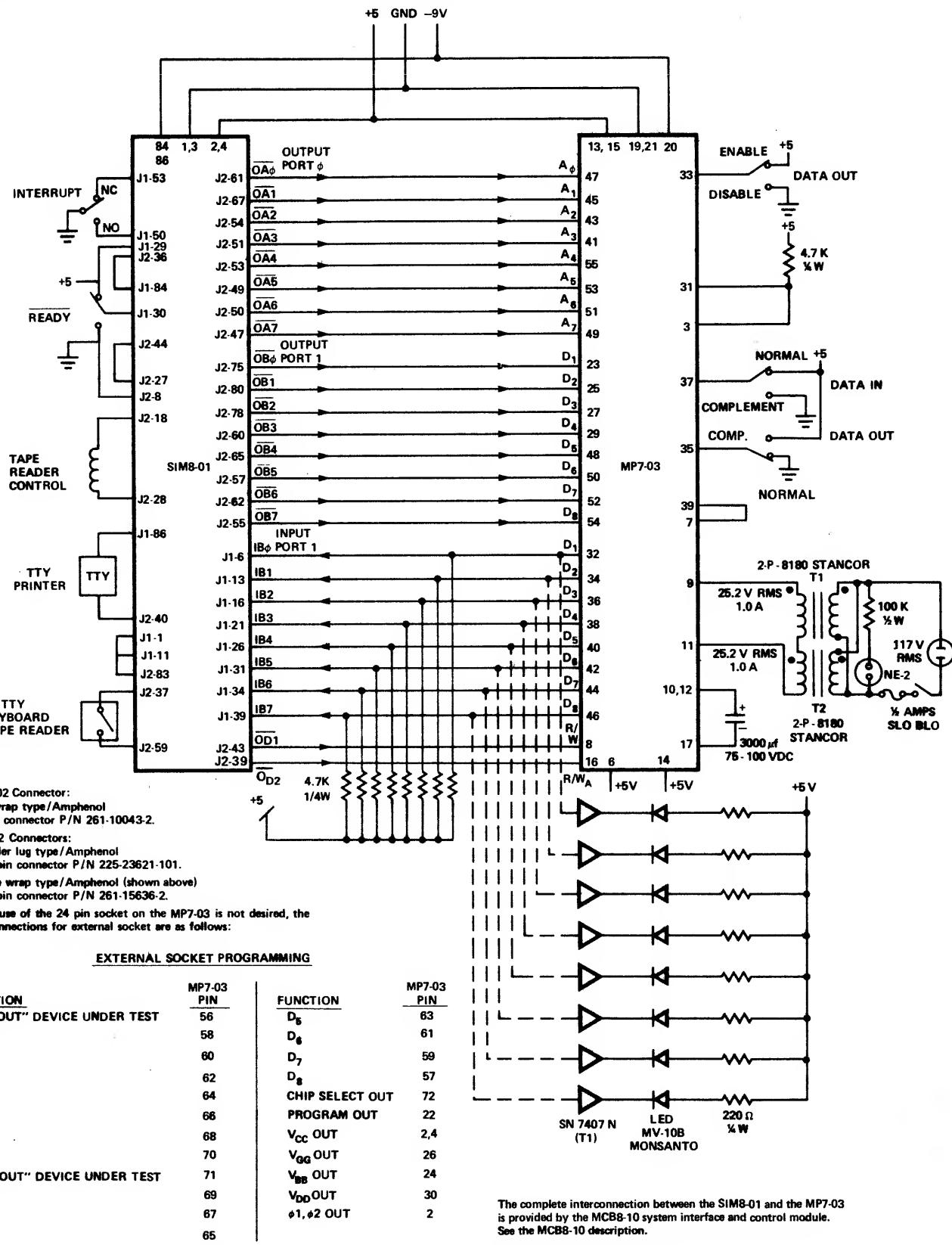
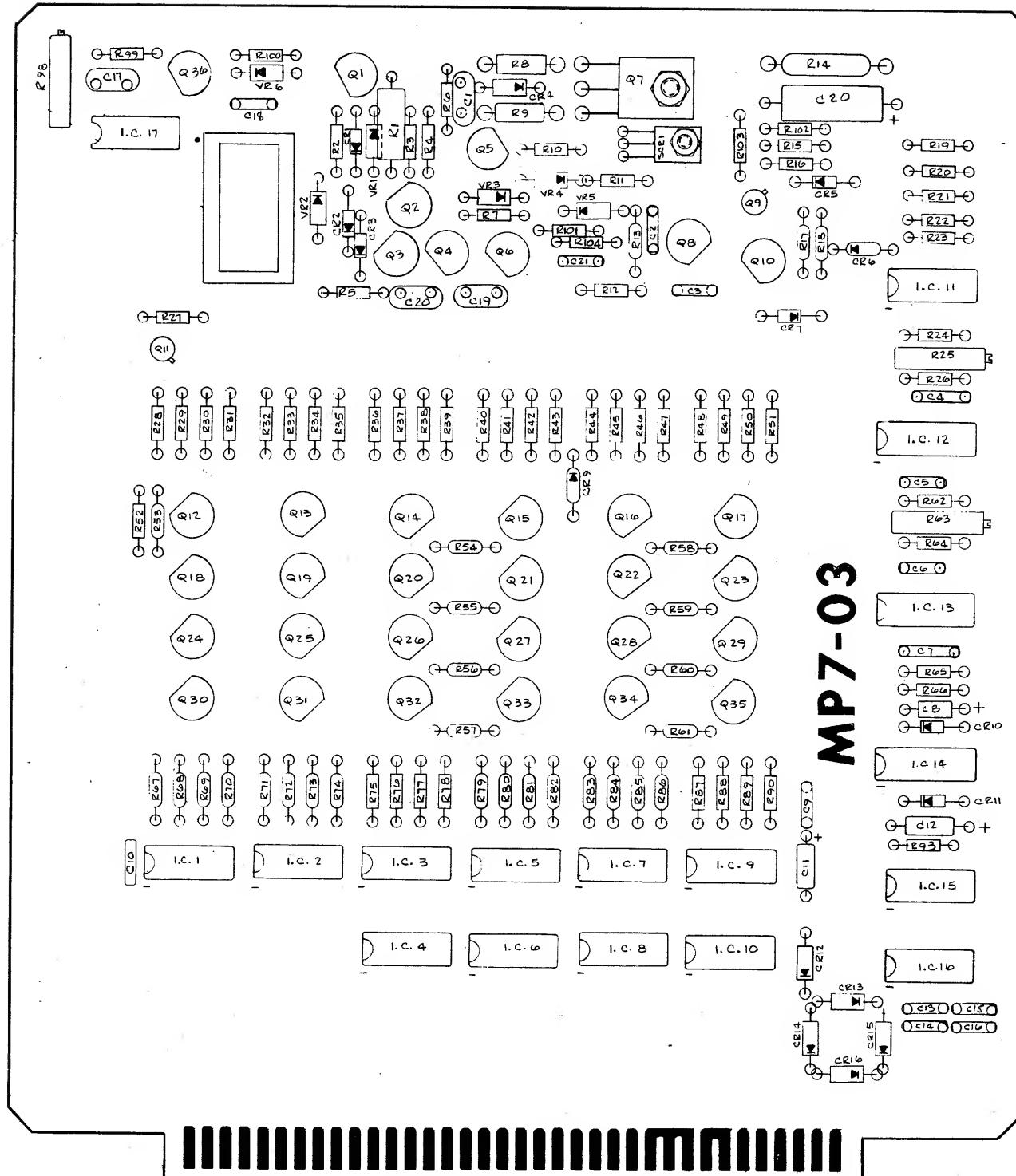


Figure 16. MP7-03/Sim8-01 PROM Programming System



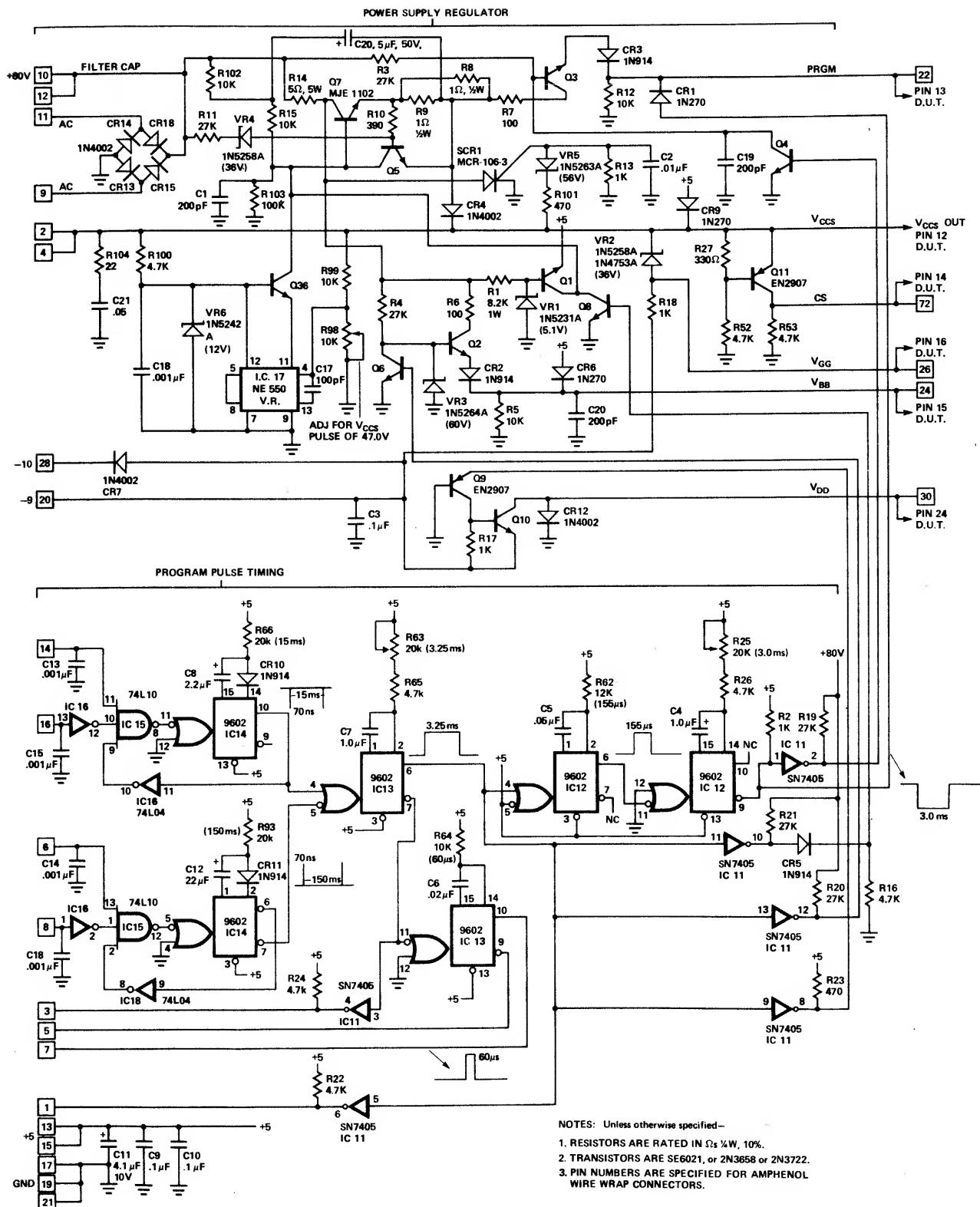
Solder Connector P/N 225-23621-101	R P N M L K J H F E D C B A Z Y X W V U T S R P N M L K J H F E D C B A	Amphenol
Wirewrap Connector P/N 261-15636-2	71 69 67 65 63 61 59 57 55 53 51 49 47 45 43 41 39 37 35 33 31 29 27 25 23 21 19 17 15 13 11 9 7 5 3 1	
Wirewrap Connector P/N VPB01E36E00A1	72 70 68 66 64 62 60 58 56 54 52 50 48 46 44 42 40 38 36 34 32 30 28 26 24 22 20 18 16 14 12 10 8 6 4 2	

Figure 17a. Component Side of MP7-03 Card



Solder Connector P/N 225-23621-101	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36	Amphenol
Wirewrap Connector P/N 261-15636-2	2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72	
Wirewrap Connector P/N VPB01E36E00A1	1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71	

Figure 17b. Pin Definition — Reverse Side of MP7-03 Card



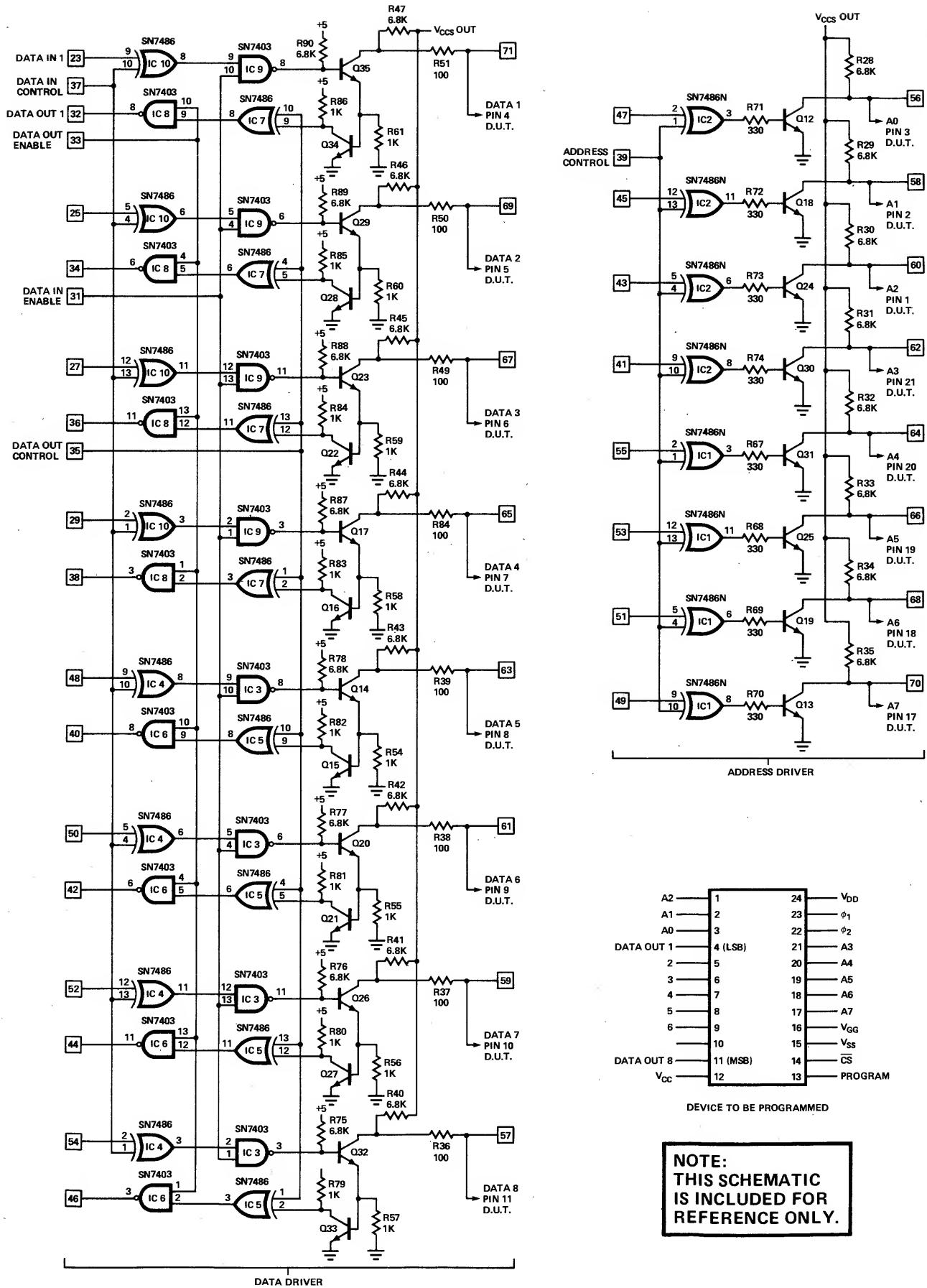


Figure 18. MP7-03 PROM Programmer Board Schematic

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 developmental 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.

```
/*
 * SAMPLE PROGRAM
 * LOCATE ALL PRIME NUMBERS BETWEEN 1 AND 50.
 * PUT RESULTS IN TRUTH TABLE AS FOLLOWS:
 * PRIME(1) = TRUE IF 1 IS A PRIME */
DECLARE PRIME(50) BYTE;
DECLARE (I,K) BYTE;
DECLARE TRUE LITERALLY '1'; FALSE LITERALLY '0';

PRIME(1) = TRUE; /* 1 IS A PRIME */

DO I = 2 TO 50;
  PRIME(I) = FALSE; /* INITIALIZE TABLE TO FALSE */
  K = 2;
  DO WHILE I MOD K >> 0; /* LOOP UNTIL TEST FOR PRIME FAILS */
    K = K + 1;
  END;
  IF K = 1 THEN
    DO; /* FOUND A PRIME */
      PRIME(I) = TRUE;
    END;
  ENDI;

EOF /* END OF PROGRAM */
```

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."

```
MCS8 MACRO ASSEMBLER: PAGE 1
/*
 * SAMPLE PROGRAM
 * LOCATE ALL PRIME NUMBERS BETWEEN 1 AND 50.
 * PUT RESULTS IN TRUTH TABLE AS FOLLOWS:
 * PRIME(1) = TRUE IF 1 IS A PRIME */
; DEFINE REGISTERS
B030 A EDU 0
B021 B EDU 1
B022 C EDU 2
B233 D LDU 3
B224 E EDU 4
B193 F EDU 5
B196 G EDU 6
B027 H EDU 7
; START!
; PRIME(1) = TRUE; /* 1 IS A PRIME */
B089 2601 HVI 1
B082 36472E00 LXI H,PRIME
B086 B6 ADD L
B087 C8 MOV A,B
B088 D0 MOV H,A
B089 3C07 ACI 0
B0AB F1 MOV B,L
B08C ED MOV A,H
B08D 3C01 HVI H,1
; DO I = 2 TO 50;
B08F 36422E03 LXI H,I
B013 3C72 HVI H,2
LOOPB1: LODPB1;
B015 6632 HVI A,50
B117 36422E03 LXI H,I
B018 B7 CMP H
B01C 606E00 JC DONE
; PRIME(1) = FALSE; /* INITIALIZE TABLE TO FALSE */
B01F 36472E00 LXI H,PRIME
B023 B6 ADD L
B025 C8 MOV A,B
B026 D0 MOV H,A
B028 F1 MOV B,L
B029 FB MOV A,H
B02A 3L00 HVI H,0
; K = 2;
B02C 36A12E20 LXI H,K
B030 3C72 HVI H,2
; DO WHILE I MOD K >> 0; /* LOOP UNTIL TEST FOR PRIME FAILS */
LOOPB1: LODPB1;
B032 36A22E03 LXI H,I
B036 C7 MDV H,A
MCS8 MACRO ASSEMBLER: PAGE 2
B037 36A32E00 LXI H,K
B038 CF SUB B
B03C 91 DCR B
B03D 3C00H JLC LOPD2
B041 B1 ADD B
B041 3C93 CPI 0
B043 684CJA JR LOPD3
; K = K + 1;
B044 CF HSC H,N,B
B045 36 IHR B
B046 F9 MOV B,H
; ENDI;
B049 4437B0 JMP LOPD1
B04C LUDP3;
; IF K = 1 THEN
;   DO; /* FOUND A PRIME */
B04C 36A32E00 LXI H,K
B050 C7 MDV H,A
B051 31 DCR L
B052 31 CPI H
B053 4C54B0 JR LOPD4
; PRIME(1) = TRUE;
B056 C7 LUDP2;
B057 3C72E00 LXI H,PRIME
B058 C5 ADD L
B059 C5 MOV A,B
B05A C5 MOV H,A
B055 3C00H ACI 0
B056 F1 MOV B,L
B051 E0 MOV A,H
B052 3C21 HVI H,1
; ENDI;
B054 LOPD4;
B054 36A72E00 LXI H,I
B055 CF MDV H,B
B059 30 IHR B
B060 F9 MOV B,H
; ENDI;
B060 4415D3 JMP LOPD0
DONE: ; END /* END OF PROGRAM */
B06E 02 HLT
;
; DECLARE PRIME(50) BYTE;
; DECLARE (I,K) BYTE;
; DECLARE TRUE LITERALLY '1'; FALSE LITERALLY '0';
;
B033 PRIME1 DS $1
B021 II DS 1
B021 KI DS 1
END
```

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 auxilliary 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)^[1] 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 auxilliary "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.

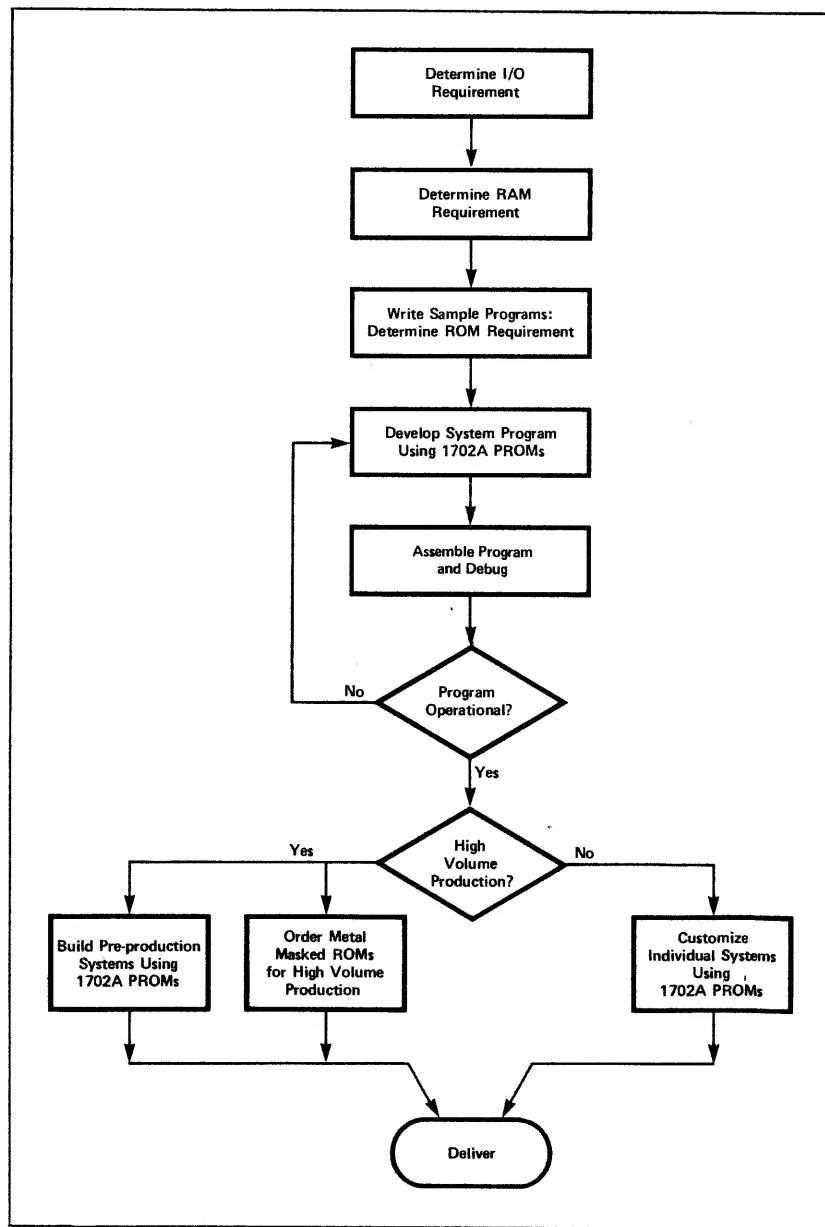
- ★ MCS-8 bootstrap loader and control program and PROM programming systems routine for the SIM8-01 and SIM8-01/MP7-03 PROM programming system (A0860, A0861, A0863) [1].
- Floating point multiply routine for the MCS-8.
- ★ Fixed point multiply routine for the MCS-8.
- Fast Fourier transform program for 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 states 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.

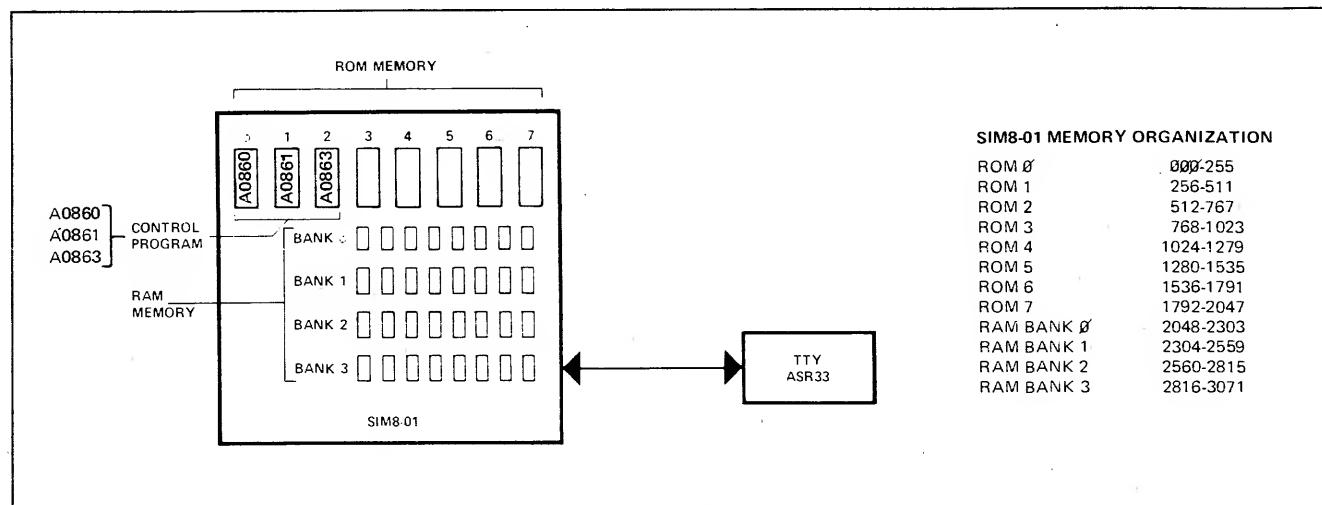


Figure 19. MCS-8 Operating System

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

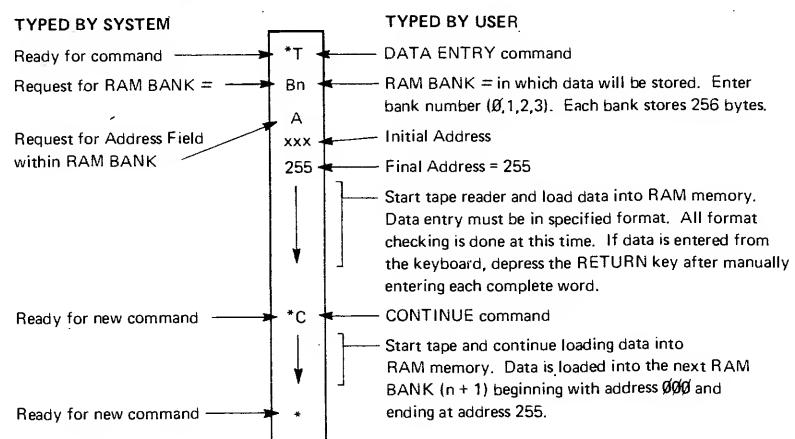
To use the SIM8-01 as the microcomputer controller for the bootstrap loading of a program from the TTY into RAM memory and the execution of programs stored in RAM, the following steps must be followed:

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.

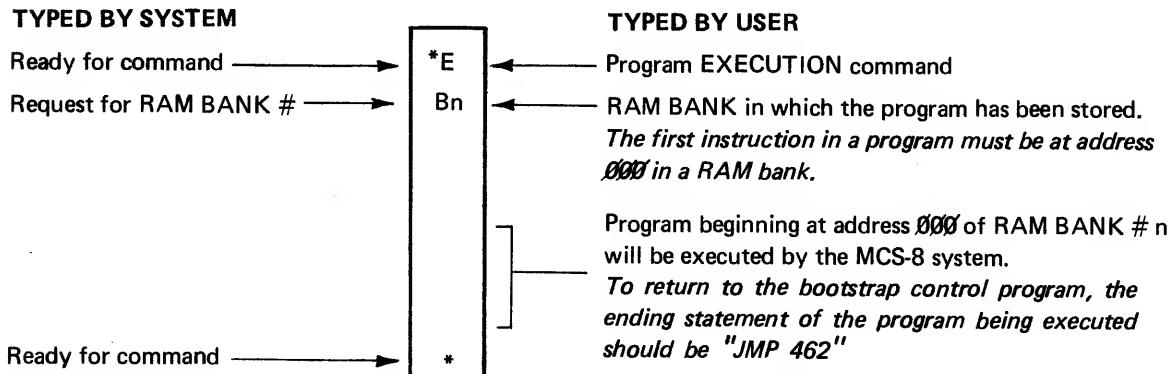
Note that the CONTINUE command should only be used when the subsequent RAM will be completely loaded with 256 bytes of data. For partial loading of RAM banks, always use the DATA ENTRY command. The content of a RAM bank may be edited by using the DATA ENTRY command and revising



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 **000** 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.



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:

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

The complete Bootstrap Loader Program is presented in Appendix V.

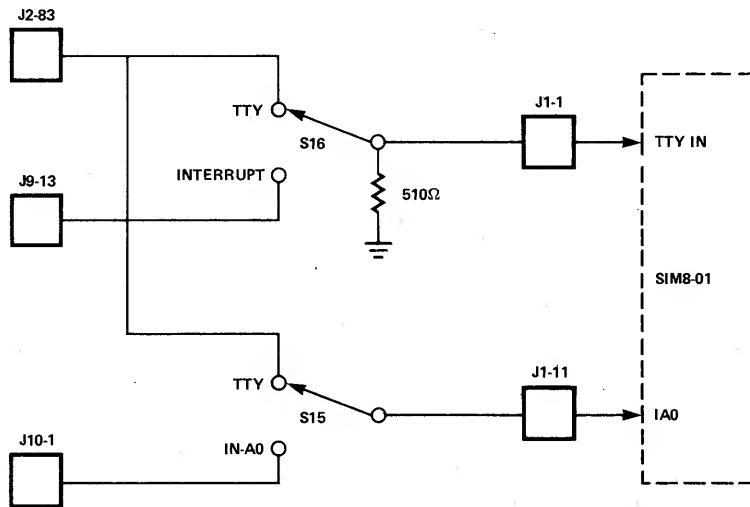
X. MCB8-10 MICRO COMPUTER INTERCONNECT AND CONTROL MODULE

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:



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

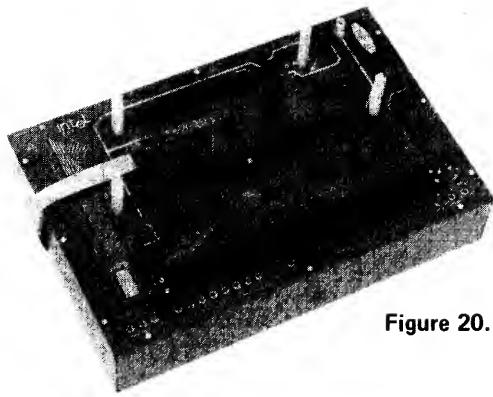


Figure 20. MCB8-10

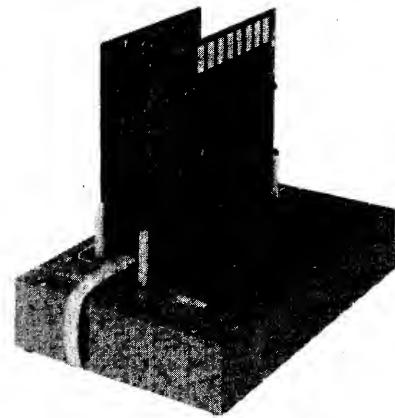


Figure 21. MCB8-10/MP7-03/SIM8-01 System

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).

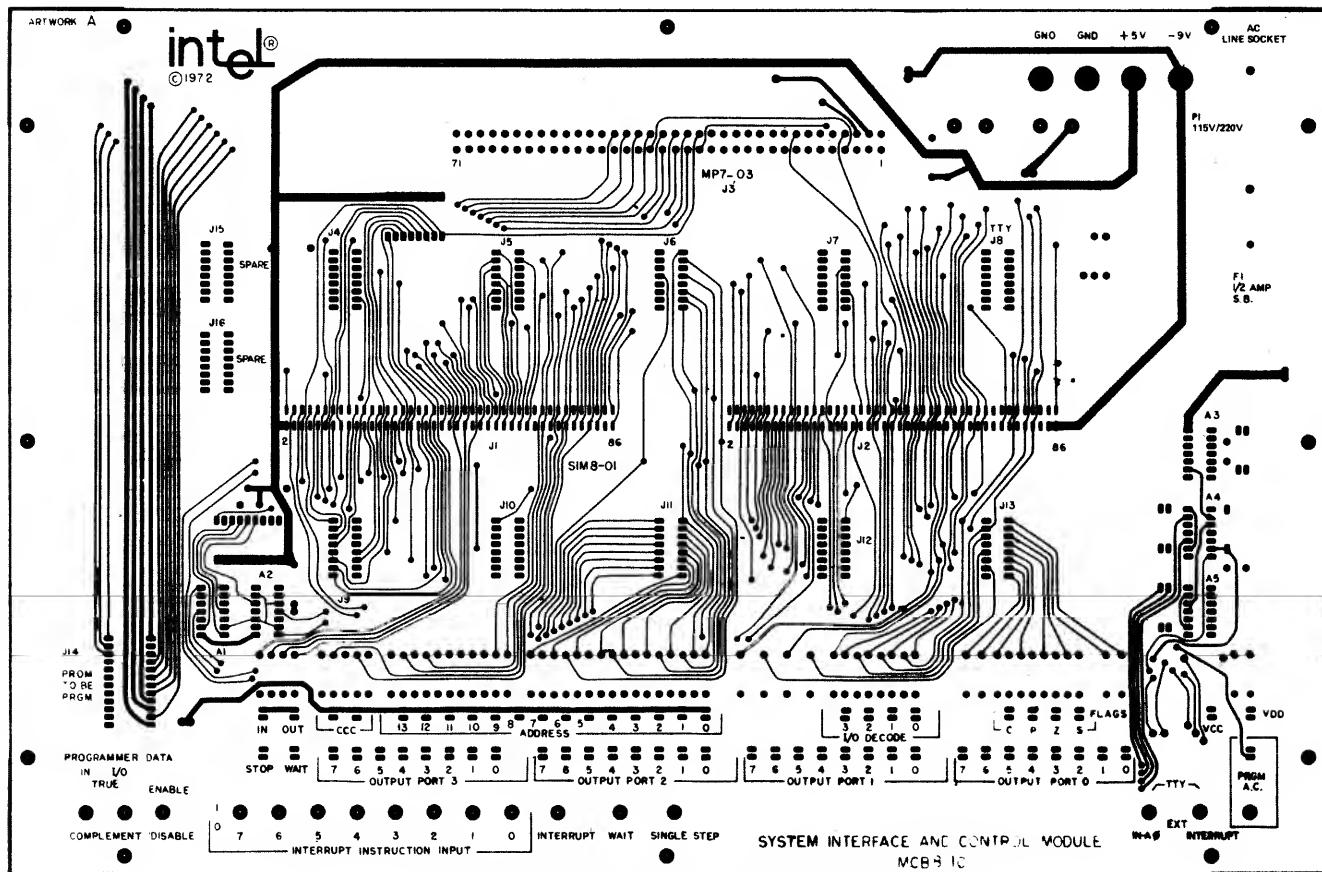


Figure 22. MCB8-10 Printed Circuit Board

B. Programming System

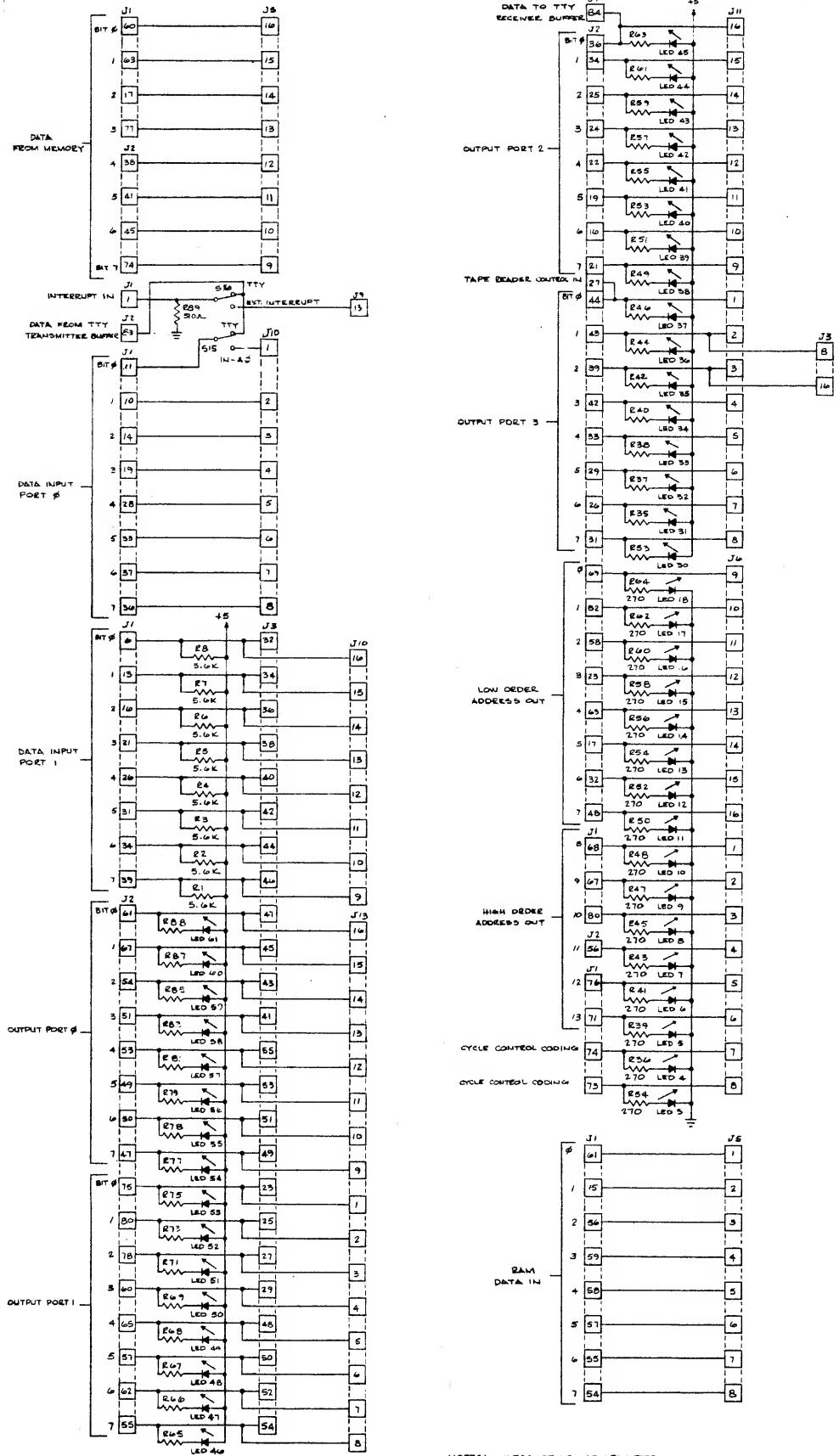
Consider the actual programming (in the hardware sense) of the 1702A PROM in the example above. The system can perform this function with the addition of an MP7-03 board inserted into the MP7-03 connector. An automatic programming system which allows data entry from a keyboard or paper tape, automatic 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)



NOTES: UNLESS OTHERWISE SPECIFIED.
 1. ALL RESISTOR VALUES ARE 450 Ω , 1/4W, 10%.
 2. ALL LEDs ARE PART NO. GE SSL-50, RED.
 3. J4 TO J15 ARE FLAT CABLE SOCKET.
 4. C1, T1, T2, 220V SOCKET ARE MOUNTED ON CHASSIS.

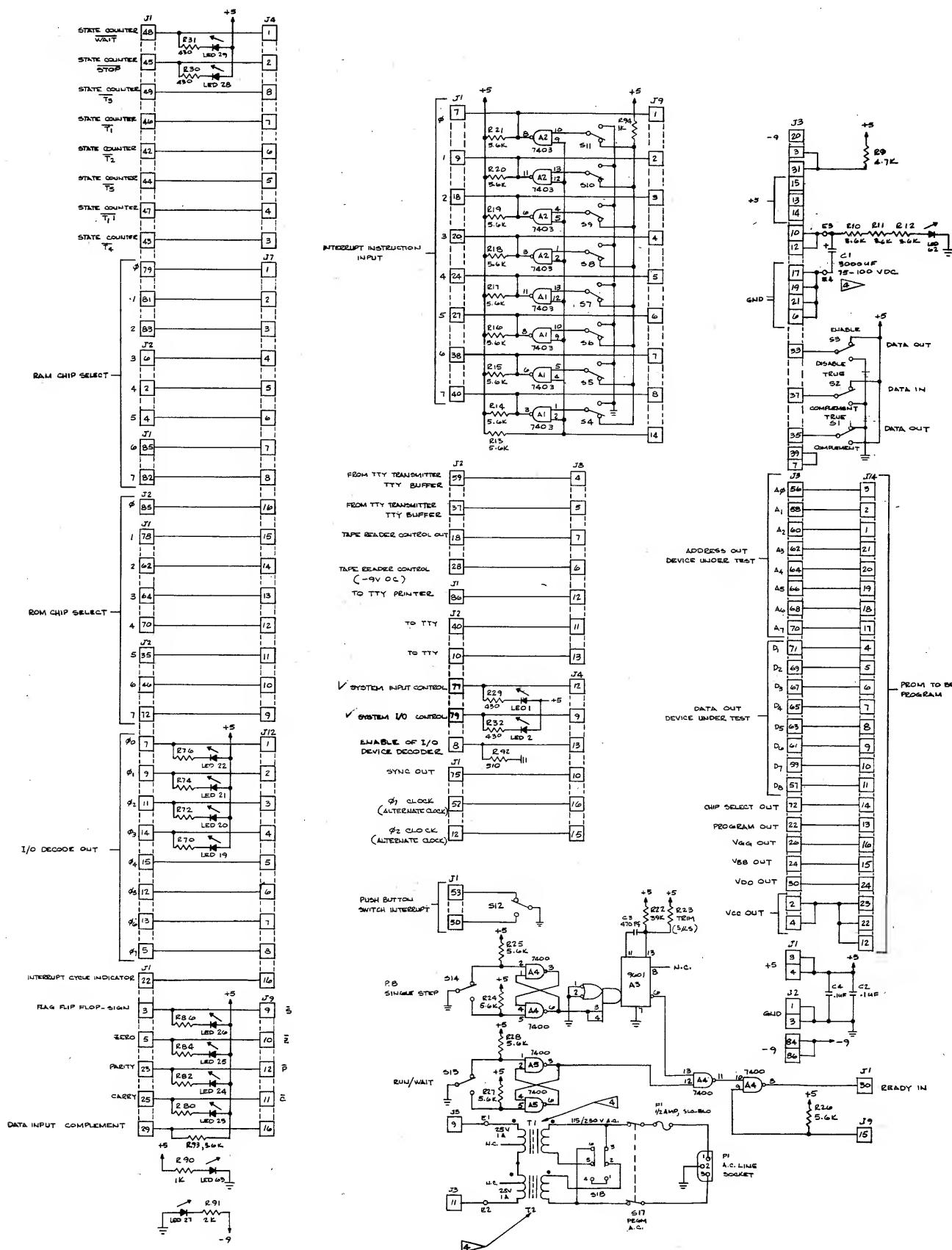


Figure 23. MCB8-10 Schematic (No. 00026)

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

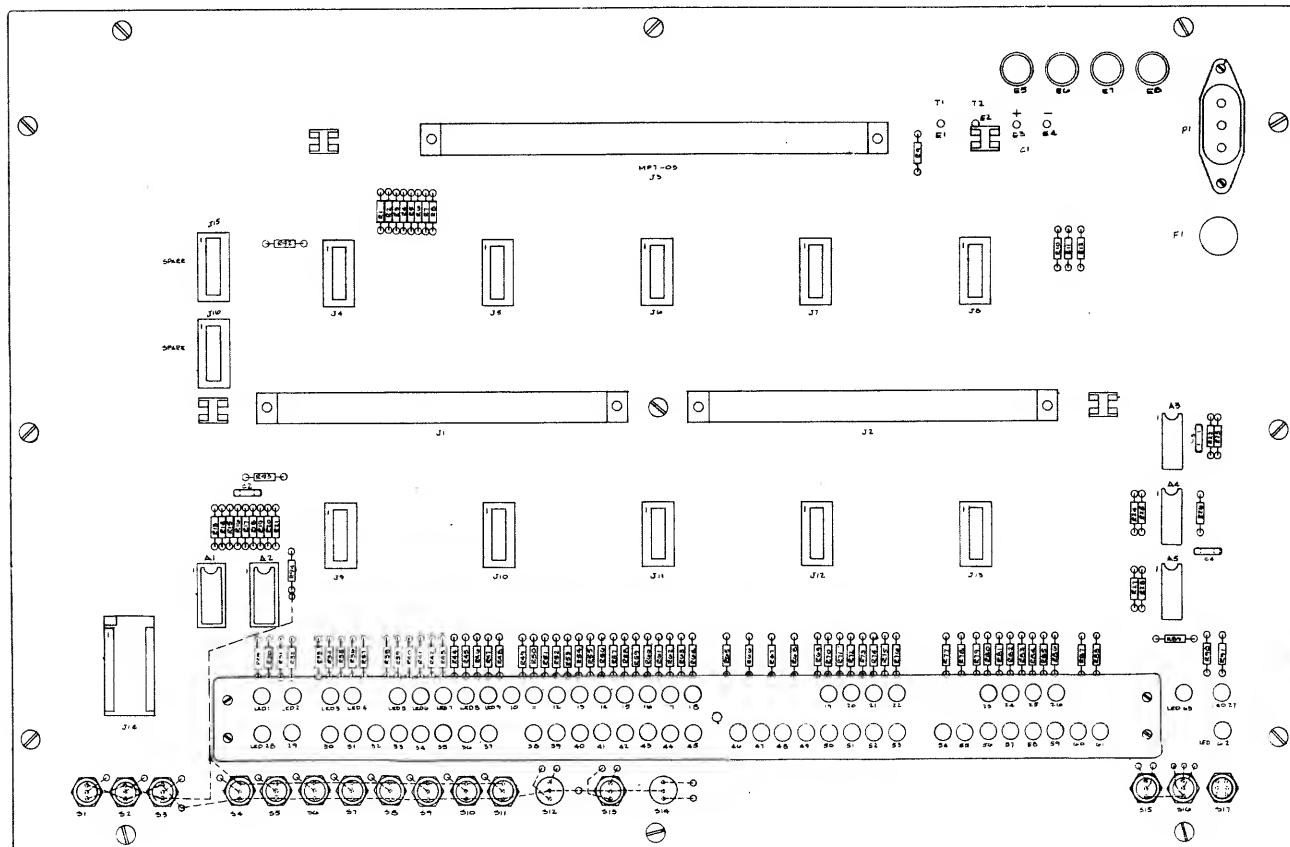


Figure 24. MCB8-10 Assembly Drawing

MCB8-10 INTERCONNECT AND CONTROL MODULE

SIM8-01				MCB8-10 Connection	SIM8-01				MCB8-10 Connection
Pin No.	Connector	Symbol	Description		Pin No.	Connector	Symbol	Description	
2,4	J1	+5V	+5VDC POWER SUPPLY		57	J1	D ₅	RAM DATA IN D ₅	J5-6
84 & 86	J2	-9V	-9VDC POWER SUPPLY		55	J1	D ₆	RAM DATA IN D ₆	J5-7
1,3	J2	GND	GROUND		54	J1	D ₇	RAM DATA IN D ₇	J5-8
60	J1	MD ₀	DATA FROM MEMORY 0 BIT 0	J5-16	48	J1	WAIT	STATE COUNTER	J4-1
63	J1	MD ₁	DATA FROM MEMORY 1 BIT 1	J5-15	49	J1	T ₃	STATE COUNTER	J4-8
17	J1	MD ₂	DATA FROM MEMORY 2 BIT 2	J5-14	46	J1	T ₁	STATE COUNTER	J4-7
77	J1	MD ₃	DATA FROM MEMORY 3 BIT 3	J5-13	45	J1	STOP	STATE COUNTER	J4-2
38	J2	MD ₄	DATA FROM MEMORY 4 BIT 4	J5-12	42	J1	T ₂	STATE COUNTER	J4-6
41	J2	MD ₅	DATA FROM MEMORY 5 BIT 5	J5-11	44	J1	T ₅	STATE COUNTER	J4-5
45	J2	MD ₆	DATA FROM MEMORY 6 BIT 6	J5-10	47	J1	T ₁	STATE COUNTER	J4-4
74	J2	MD ₇	DATA FROM MEMORY 7 BIT 7	J5-9	43	J1	T ₄	STATE COUNTER	J4-3
11	J1	IA ₀	DATA INPUT PORT 0 BIT 0	(S15) J10-1	79	J1	C _M ₀	RAM CHIP SELECT 0	J7-1
10	J1	IA ₁	DATA INPUT PORT 0 BIT 1	J10-2	81	J1	C _M ₁	RAM CHIP SELECT 1	J7-2
14	J1	IA ₂	DATA INPUT PORT 0 BIT 2	J10-3	83	J1	C _M ₂	RAM CHIP SELECT 2	J7-3
19	J1	IA ₃	DATA INPUT PORT 0 BIT 3	J10-4	6	J2	C _M ₃	RAM CHIP SELECT 3	J7-4
28	J1	IA ₄	DATA INPUT PORT 0 BIT 4	J10-5	2	J2	C _M ₄	RAM CHIP SELECT 4	J7-5
33	J1	IA ₅	DATA INPUT PORT 0 BIT 5	J10-6	4	J2	C _M ₅	RAM CHIP SELECT 5	J7-6
37	J1	IA ₆	DATA INPUT PORT 0 BIT 6	J10-7	85	J1	C _M ₆	RAM CHIP SELECT 6	J7-7
36	J1	IA ₇	DATA INPUT PORT 0 BIT 7	J10-8	82	J1	C _M ₇	RAM CHIP SELECT 7	J7-8
6	J1	IB ₀	DATA INPUT PORT 1 BIT 0	J10-16	85	J2	CS ₀	ROM CHIP SELECT 0	J7-16
13	J1	IB ₁	DATA INPUT PORT 1 BIT 1	J10-15	78	J1	CS ₁	ROM CHIP SELECT 1	J7-15
16	J1	IB ₂	DATA INPUT PORT 1 BIT 2	J10-14	62	J1	CS ₂	ROM CHIP SELECT 2	J7-14
21	J1	IB ₃	DATA INPUT PORT 1 BIT 3	J10-13	64	J1	CS ₃	ROM CHIP SELECT 3	J7-13
26	J1	IB ₄	DATA INPUT PORT 1 BIT 4	J10-12	70	J1	CS ₄	ROM CHIP SELECT 4	J7-12
31	J1	IB ₅	DATA INPUT PORT 1 BIT 5	J10-11	35	J2	CS ₅	ROM CHIP SELECT 5	J7-11
34	J1	IB ₆	DATA INPUT PORT 1 BIT 6	J10-10	46	J2	CS ₆	ROM CHIP SELECT 6	J7-10
39	J1	IB ₇	DATA INPUT PORT 1 BIT 7	J10-9	72	J2	CS ₇	ROM CHIP SELECT 7	J7-9
61	J2	DA ₀	OUTPUT PORT 0 BIT 0	J13-16	5	J2	O ₇	I/O DECODE OUT O ₇	J12-8
67	J2	DA ₁	OUTPUT PORT 0 BIT 1	J13-15	13	J2	O ₆	I/O DECODE OUT O ₆	J12-7
54	J2	DA ₂	OUTPUT PORT 0 BIT 2	J13-14	12	J2	O ₅	I/O DECODE OUT O ₅	J12-6
51	J2	DA ₃	OUTPUT PORT 0 BIT 3	J13-13	15	J2	O ₄	I/O DECODE OUT O ₄	J12-5
53	J2	DA ₄	OUTPUT PORT 0 BIT 4	J13-16	14	J2	O ₃	I/O DECODE OUT O ₃	J12-4
49	J2	DA ₅	OUTPUT PORT 0 BIT 5	J13-11	11	J2	O ₂	I/O DECODE OUT O ₂	J12-3
50	J2	DA ₆	OUTPUT PORT 0 BIT 6	J13-10	9	J2	O ₁	I/O DECODE OUT O ₁	J12-2
47	J2	DA ₇	OUTPUT PORT 0 BIT 7	J13-9	7	J2	O ₀	I/O DECODE OUT O ₀	J12-1
75	J2	OB ₀	OUTPUT PORT 1 BIT 0	J13-1	3	J1	S	FLAG FLIP FLOP-Sign	J9-9
80	J2	OB ₁	OUTPUT PORT 1 BIT 1	J13-2	5	J1	Z	FLAG FLIP FLOP-Zero	J9-10
78	J2	OB ₂	OUTPUT PORT 1 BIT 2	J13-3	23	J1	P	FLAG FLIP FLOP-Parity	J9-12
60	J2	OB ₃	OUTPUT PORT 1 BIT 3	J13-4	25	J1	C	FLAG FLIP FLOP-Carry	J9-11
65	J2	OB ₄	OUTPUT PORT 1 BIT 4	J13-5	7	J1	D ₀	INTERRUPT INSTRUCTION INPUT 0	J9-1
57	J2	OB ₅	OUTPUT PORT 1 BIT 5	J13-6	9	J1	D ₁	INTERRUPT INSTRUCTION INPUT 1	J9-2
62	J2	OB ₆	OUTPUT PORT 1 BIT 6	J13-7	18	J1	D ₂	INTERRUPT INSTRUCTION INPUT 2	J9-3
55	J2	OB ₇	OUTPUT PORT 1 BIT 7	J13-8	20	J1	D ₃	INTERRUPT INSTRUCTION INPUT 3	J9-4
36	J2	OC ₀	OUTPUT PORT 2 BIT 0	J11-16	24	J1	D ₄	INTERRUPT INSTRUCTION INPUT 4	J9-5
34	J2	OC ₁	OUTPUT PORT 2 BIT 1	J11-15	27	J1	D ₅	INTERRUPT INSTRUCTION INPUT 5	J9-6
25	J2	OC ₂	OUTPUT PORT 2 BIT 2	J11-14	38	J1	D ₆	INTERRUPT INSTRUCTION INPUT 6	J9-7
24	J2	OC ₃	OUTPUT PORT 2 BIT 3	J11-13	40	J1	D ₇	INTERRUPT INSTRUCTION INPUT 7	J9-8
22	J2	OC ₄	OUTPUT PORT 2 BIT 4	J11-12	59	J2	FROM TTY TRANSMITTER IN	{ TTY BUFFER	J8-4
19	J2	OC ₅	OUTPUT PORT 2 BIT 5	J11-11	37	J2	FROM TTY TRANSMITTER OUT	} TTY BUFFER	J8-5
16	J2	OC ₆	OUTPUT PORT 2 BIT 6	J11-10	83	J2	DATA FROM TTY TRANSMITTER BUFFER		J11, S16
21	J2	OC ₇	OUTPUT PORT 2 BIT 7	J11-9	27	J2	TAPE READER CONTROL IN		J11-1
44	J2	OD ₀	OUTPUT PORT 3 BIT 0	J11-1	18	J2	TAPE READER CONTROL OUT		J8-7
43	J2	OD ₁	OUTPUT PORT 3 BIT 1	J11-2	28	J2	TAPE READER CONTROL (-9VDC)		J8-6
39	J2	OD ₂	OUTPUT PORT 3 BIT 2	J11-3	84	J1	DATA TO TTY RECEIVER BUFFER		J11-16
42	J2	OD ₃	OUTPUT PORT 3 BIT 3	J11-4	10	J2	TO TTY RECEIVER OUT		J8-13
33	J2	OD ₄	OUTPUT PORT 3 BIT 4	J11-5	86	J1	TO TTY RECEIVER OUT	{ TTY BUFFER	J8-12
29	J2	OD ₅	OUTPUT PORT 3 BIT 5	J11-6	40	J2	TO TTY RECEIVER OUT		J8-11
26	J2	OD ₆	OUTPUT PORT 3 BIT 6	J11-7	81	J2	READ/WRITE		
31	J2	OD ₇	OUTPUT PORT 3 BIT 7	J11-8	72	J1	I _B	MULTIPLEXER CONTROL LINES N8263	
69	J2	A ₀	LOW ORDER ADDRESS OUT	J6-9	41	J1	SL _B	MULTIPLEXER CONTROL LINES N8267	
82	J2	A ₁	LOW ORDER ADDRESS OUT	J6-10	69	J1	I ₁	MULTIPLEXER CONTROL LINES N8263	
58	J2	A ₂	LOW ORDER ADDRESS OUT	J6-11	8	J1	SL ₁	MULTIPLEXER CONTROL LINES N8267	
23	J2	A ₃	LOW ORDER ADDRESS OUT	J6-12	29	J1	DATA COMPLEMENT		
63	J2	A ₄	LOW ORDER ADDRESS OUT	J6-13	52	J1	θ_1	θ_1 CLOCK (alternate clock)	J9-16
17	J2	A ₅	LOW ORDER ADDRESS OUT	J6-14	12	J1	θ_2	θ_2 CLOCK (alternate clock)	J4-16
32	J2	A ₆	LOW ORDER ADDRESS OUT	J6-15	75	J1	SYNC	SYNC OUT	J4-15
48	J2	A ₇	LOW ORDER ADDRESS OUT	J6-16	30	J1	READY	READY IN	J4-10
68	J1	A ₈	HIGH ORDER ADDRESS OUT	J6-1	1	J1	INTERRUPT	INTERRUPT IN	JTY, S16
67	J1	A ₉	HIGH ORDER ADDRESS OUT	J6-2	8	J2	I/O ENABLE	I/O DEVICE DECODER	J4-13
80	J1	A ₁₀	HIGH ORDER ADDRESS OUT	J6-3	79	J2	I/O	SYSTEM I/O CONTROL	J4-9
56	J2	A ₁₁	HIGH ORDER ADDRESS OUT	J6-4	50	J1	N.O.	PUSH BUTTON SWITCH	S12
76	J1	A ₁₂	HIGH ORDER ADDRESS OUT	J6-5	53	J1	N.C.	PUSH BUTTON SWITCH	S12
71	J1	A ₁₃	HIGH ORDER ADDRESS OUT	J6-6	52	J2	W ₀	OUTPUT LATCH STROBE PORT 0	
74	J1	CC ₀	CYCLE CONTROL CODING	J6-7	71	J2	W ₁	OUTPUT LATCH STROBE PORT 1	
73	J1	CC ₁	CYCLE CONTROL CODING	J6-8	20	J2	W ₂	OUTPUT LATCH STROBE PORT 2	
61	J1	D ₀	RAM DATA IN D ₀	J5-1	30	J2	W ₃	OUTPUT LATCH STROBE PORT 3	
15	J1	D ₁	RAM DATA IN D ₁	J5-2	22	J1	INT CYCLE	INTERRUPT CYCLE INDICATOR	
56	J1	D ₂	RAM DATA IN D ₂	J5-3	32	J1	T ₃ _A	ANTICIPATED T ₃ OUTPUT	
59	J1	D ₃	RAM DATA IN D ₃	J5-4	35	J1	T ₃ _A	ANTICIPATED T ₃ OUTPUT	
58	J1	D ₄	RAM DATA IN D ₄	J5-5					J12-16

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 auxilliary 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.

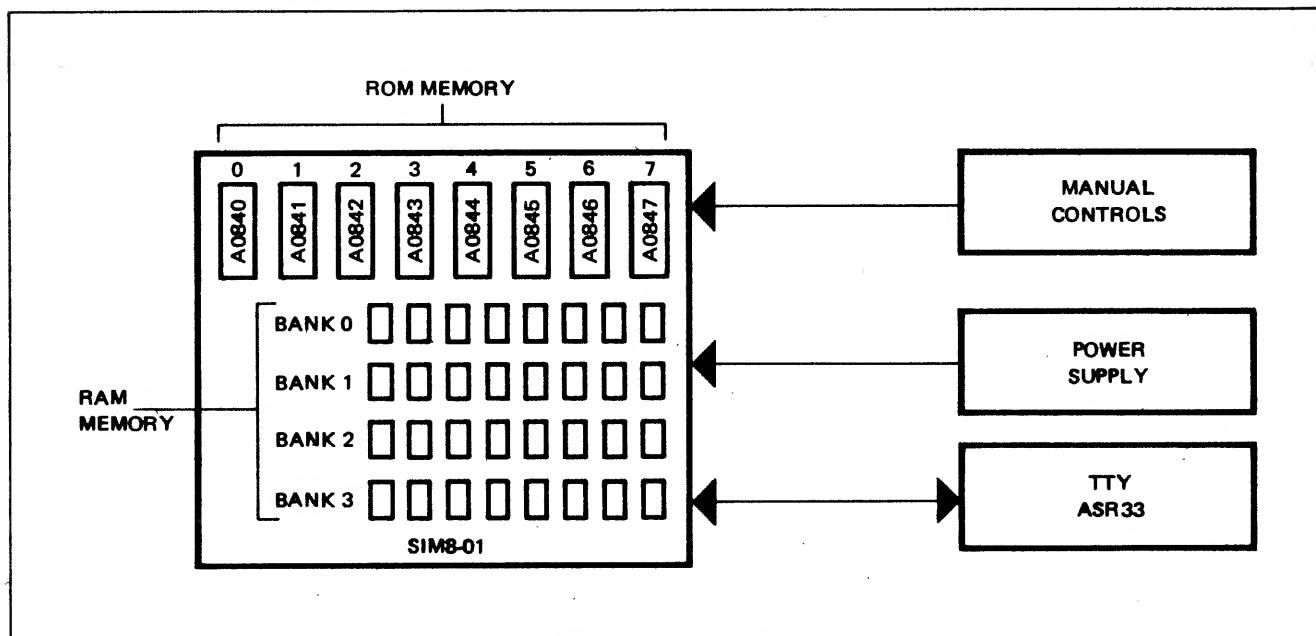


Figure 1.1. SIM8 Hardware Assembler System Configuration

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 auxilliary "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 SIM8-01 portion of the MP7-02/SIM8-01 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 SIM8-01 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 "SIM8 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 auxilliary 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.

ASTST	LAB
	LCM
JMP ASTST	
END	

Figure 2.1. Listing of Source Tape

KEYBOARD → ASSEMBLE: 032: 000:	
PASS 1	032 000 ASTST LAB
	032 001 LCM
	032 002 JMP ASTST
	032 004 END
	ASTST 032 00

KEYBOARD → LIST:

PASS 2	LOAD: 032: 000:
Octal Object	032 000 301: 327: 104: 032: 000:
	Code

Figure 2.2. Assembly Listing

KEYBOARD → LOAD: 013: 000:	
Tape Generator	013 000 106: 326: 000: 106: 237: 000: 354: 066:
	•
	•
	•
	013 150 153: 007: 050: 357: 361: 007:

Figure 2.3. Load of Tape Generator

KEYBOARD → LOAD: 012: 000:	
Octal Object	032 000 301: 327: 104: 032: 000: ...
	Code

Figure 2.4. Loading of Octal Object Code

KEYBOARD → BEGIN: 013: 000:	
"BNPF" Object Code	012:
	000 BPPNNNNNPF
	001 •
	002 •
	003 •
	004 BNPNPNNPPF

Figure 2.5. Execution of Tape Generator

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 alphabetics, 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:
Page Char.

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:

DUMP:	<u>Last Address</u>	<u>First Address</u>
	<u>011:</u> Page	<u>008:</u> Char.
	<u>011:</u> Page	<u>000:</u> Char.

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:

ASSEMBLE:	<u>Origin</u>	
	<u>032:</u> Page	<u>000:</u> Char.

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:

BEGIN:	<u>Address Modifier</u>	
	<u>032:</u> Page	<u>000:</u> Char.

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.

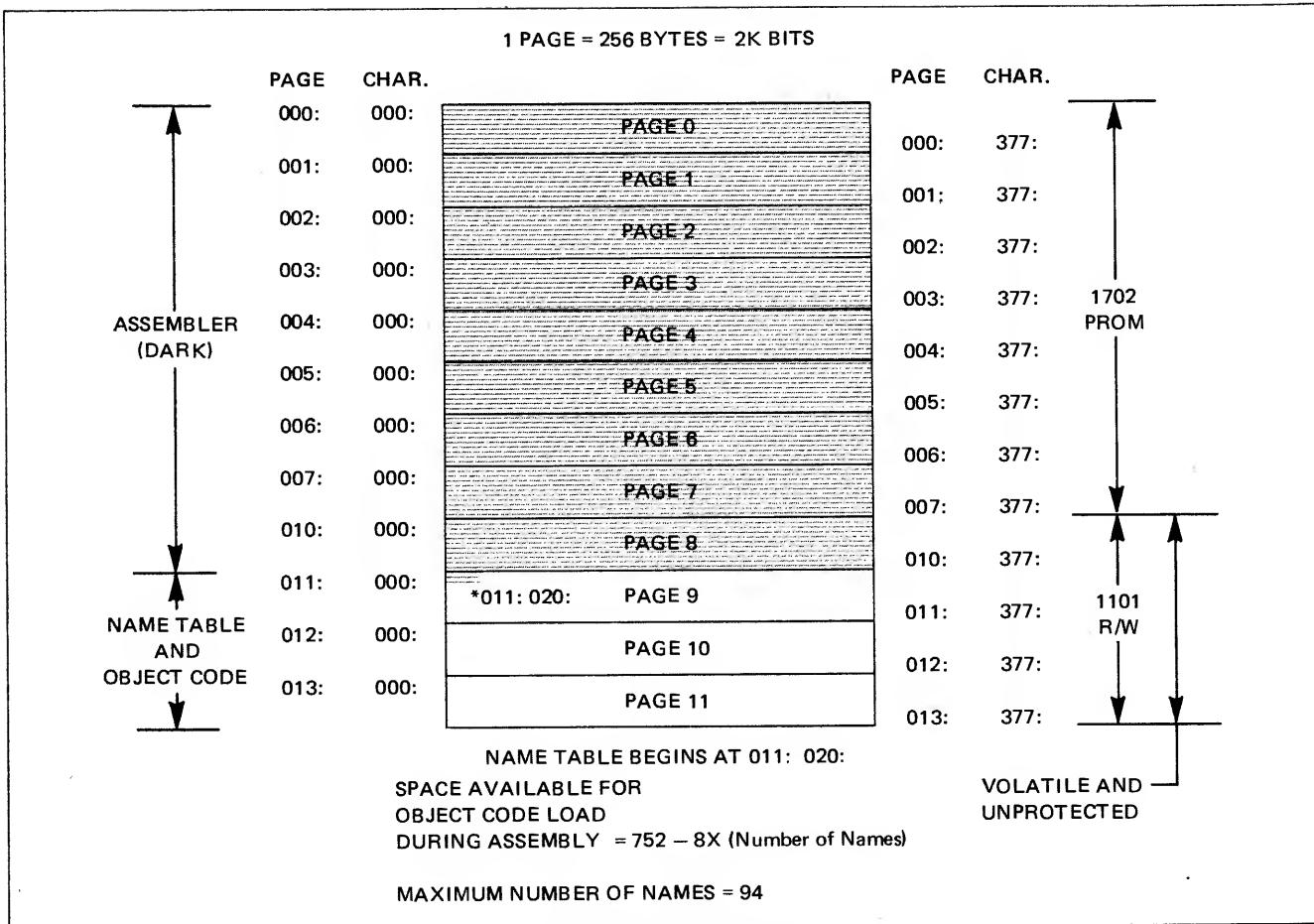


Figure 5.1 Memory Map

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.

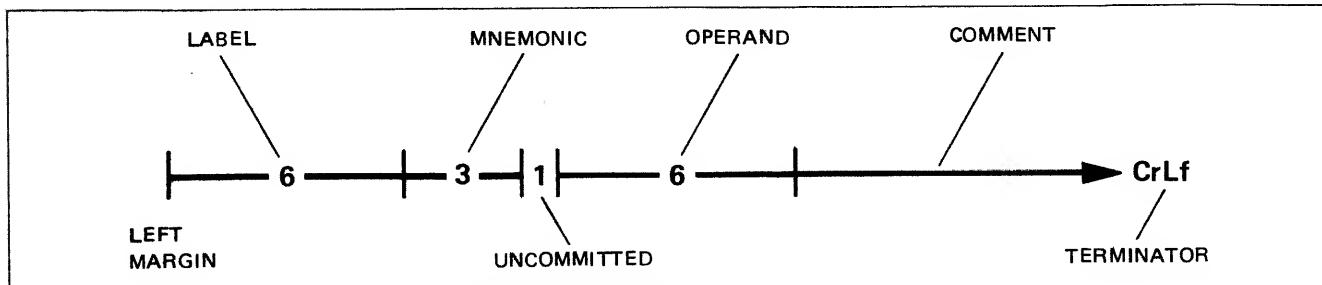


Figure 6.1 Source Line Statement Format

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.

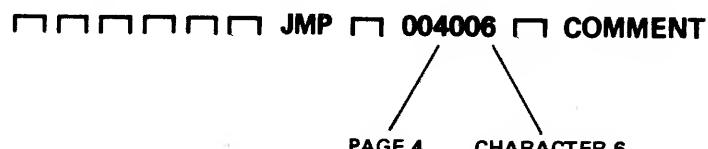
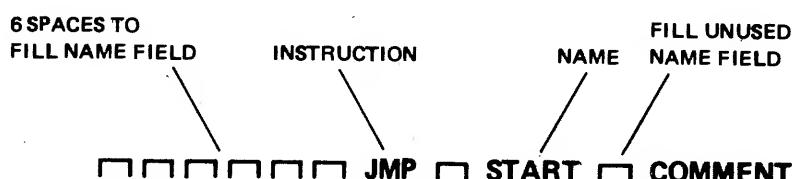
1. The name must be left justified in the field.
2. The name can contain any character except the commercial at sign, @.
3. All unused positions in the field must be filled with spaces.
4. The name must appear exactly once in a label field of the source text.
5. The total number of names for a single 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.

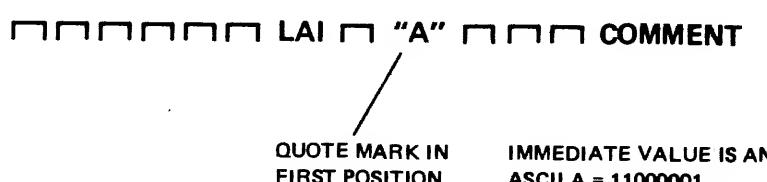
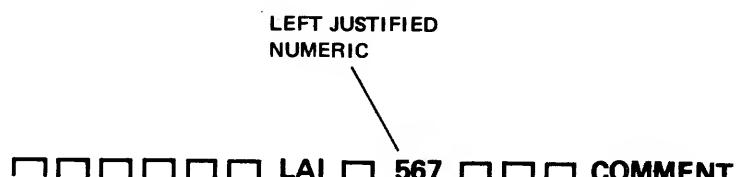
Further explanation and qualifications related to some of the instructions is given below.

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:



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.

IMMEDIATES: 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:



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

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 octal 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 "offline" 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:
SIGNIFIES SPACE ORIGIN

FIRST CHARACTER
MUST HAVE NAME

```
ASSEMBLE: 032: 000:  
ASTST LAB  
LCM  
LND  
LEI 123  
LMI "-"  
INH  
DCL  
ADA  
ACB  
SUC  
SSD  
NDE  
XCH  
OWL  
CPM  
ADI "A"  
ACI "B"  
SUI "C"  
SBI "D"  
NDI "E"  
XRI "F"  
ORI "G"  
CPI "H"  
RLC  
RRC  
RAL  
RAR  
JMP JMP 020000  
JFC JMP  
JFZ JMP  
JFS JMP  
JFP JMP  
JTC JMP  
JTZ JMP  
JTS JMP  
JTP JMP  
CAL CAL CAL  
CFG CAL  
CFZ CAL  
CFS CAL  
CFP CAL  
CTC CAL  
CTZ CAL  
CTS CAL  
CTR 010010  
RET  
RTC  
RTZ  
RTS  
RTP  
RFC  
RFZ  
RFS  
RFP  
RST 000070  
RST TTYOT  
TTYOT ADR 000060  
INP 003  
INP TYIN  
OUT 003  
OUT TYOT  
HLT  
HLT 123  
HLT "A"  
PAM TSTAY  
PAM 020123
```

THE FOLLOWING INSTRUCTIONS ARE IN ERROR 0

```
INA  
INM  
DCA  
DCM  
LAI  
JMP ASTSY  
CAL  
RES TYOT  
RES 000001  
LFM  
CAL CAL  
ADI 'A'
```

CONTROL CHARACTER
"DELETING" LINE

THE FOLLOWING INSTRUCTIONS ARE NONEXECUTABLE 0

```
TSTLC LOC 000001  
TSTAY LOC 001000  
ENDLG LOC 000001  
TYIN ADR 000000  
TYOT ADR 003000  
END
```

PASS
TERMINATOR

Figure 9.1 Source Listing

LINE ADDRESSES
ASSIGNED BY
ASSEMBLER

ASSEMBLE: 032: 000:
KEYBOARD INPUT

032 000 ASTST LAB
032 001 LCM
032 002 LMD
032 003 LEI 123
032 005 LNI ":"
032 007 INH
032 010 DCL
032 011 ADA
032 012 ACB
032 013 SUC
032 014 SBD
032 015 NDE
032 016 XRH
032 017 ORL
032 020 CPM
032 021 ADI "A"
032 023 ACI "B"
032 025 SUI "C"
032 027 SBI "D"
032 031 NDI "E"
032 033 XNI "F"
032 035 ORI "G"
032 037 CPI "H"
032 041 RLC
032 042 RRC
032 043 RAL
032 044 RAR
032 045 JMP #20000
032 050 JFC JMP
032 053 JFZ JMP
032 056 JFS JMP
032 061 JFP JMP
032 064 JTC JMP
032 067 JTZ JMP
032 072 JTS JMP
032 075 JTP JMP
032 100 CAL CAL
032 103 CFC CAL
032 106 CFZ CAL
032 111 CPS CAL
032 114 CFP CAL
032 117 CTC CAL
032 122 CTZ CAL
032 125 CTS CAL
032 130 CTP #10010
032 133 RET
032 134 RTC
032 135 RTZ
032 136 RTS
032 137 RTP
032 140 RPC
032 141 RPZ
032 142 RPS
032 143 RPP
032 144 RST #000070
032 145 RST TTYOT
032 146 TTYOT ADR 000060
032 146 INP #03
032 147 INP TYIN
032 150 OUT #03
032 151 OUT TYOT
032 152 HLT
032 153 HLT 123
032 154 HLT "A"
032 155 PAM TSTAY
032 161 PAM #20123

232 165 THE FOLLOWING INSTRUCTIONS ARE IN ERROR :
032 165 INA
032 166 INN
032 167 DCA
032 170 DCM
032 171 LAI
032 173 JMP ASTSY
032 176 CAL
032 201 RES TYOT
032 202 RES 000001
032 203 LFM
032 204 CAL CAL
032 207 ADI 'A'

032 211 THE FOLLOWING INSTRUCTIONS ARE NONEXECUTABLE :
032 211 TSTLC LOC 000001
032 212 TSTAY LOC 001000
033 212 ENDLC LOC 000001
033 213 TYIN ADR 003000
033 213 TYOT ADR 033000
033 213 END

ASTST 032 000
JMP 032 045
CAL 032 100
TTYOT 000 060
TSTLC 032 211 }
TSTAY 032 212 }
ENDLC 033 212 }
TYIN 003 000 }
TYOT 033 000 }

SYMBOL TABLE

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 pass1 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.

KEYBOARD INPUT LIST:	<pre> LOAD: 032: 000: 032 000 301: 327: 373: 046: 123: 076: 256: 359: 032 010 061: 200: 211: 222: 233: 244: 255: 266: 032 020 277: 004: 391: 014: 332: 024: 383: 034: 032 030 344: 344: 345: 054: 306: 264: 307: 074: 032 040 310: 002: 012: 022: 032: 104: 200: 020: 032 050 109: 045: 032: 110: 045: 032: 120: 245: 032 060 032: 130: 245: 032: 140: 045: 032: 150: 032 070 245: 032: 160: 045: 032: 170: 045: 032: 032 100 106: 100: 032: 132: 100: 032: 112: 100: 032 110 032: 122: 130: 032: 132: 100: 032: 142: 032 120 100: 032: 152: 100: 032: 162: 100: 032: 032 130 172: 010: 013: 007: 043: 053: 463: 073: 032 140 003: 013: 023: 033: 275: 065: 107: 107: 032 150 147: 167: 000: 123: 381: 356: 032: 066: 032 160 212: 055: 020: 066: 123: INA ? INM ? DCA ? DCM ? LAI ? 104: JMP ASTSY ? 106: CAL ? RES TYOT ? RES 000001? LFM ? 032 170 106: CAL CAL ? 004: ADI 'A' ? </pre>
PARTIAL OUTPUT FOR LAI (OPERAND IS MISSING)	OCTAL OBJECT CODE
	RESULT OF DIAGNOSTICS
	ERROR FLAG

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.

<p>KEYBOARD INPUT</p> <p>LOAD: 013: 000:</p> <p>ASSEMBLER OUTPUT FOR LATER USE</p> <pre> LOAD: 013: 000: 013 000 106: 326: 000: 106: 237: 000: 354: 466: 013 010 000: 046: 100: 006: 377: 065: 041: 110: 013 020 013: 013: 307: 106: 033: 013: 060: 110: 013 030 022: 013: 307: 325: 316: 056: 013: 066: 013 040 156: 371: 060: 110: 047: 013: 050: 372: 013 050 256: 013: 066: 160: 370: 056: 013: 066: 013 060 161: 276: 010: 106: 326: 000: 056: 013: 013 070 066: 156: 367: 106: 347: 000: 005: 302: 013 100 055: 055: 013: 066: 160: 377: 222: 378: 013 110 120: 121: 013: 066: 320: 065: 104: 124: 013 120 013: 026: 316: 065: 056: 013: 066: 161: 013 130 317: 011: 371: 110: 101: 013: 026: 326: 013 140 265: 056: 013: 066: 156: 317: 068: 110: 013 150 153: 013: 050: 357: 361: 007: </pre>	<p>"BNPF TAPE GENERATOR" OBJECT CODE, THIS IS LOADED INTO MEMORY BEGINNING WITH LOCATION 013: 000:</p>
<p>KEYBOARD INPUT</p> <p>LOAD: 012: 000:</p> <pre> LOAD: 012: 000: 032 000 301: 327: 373: 046: 123: 076: 256: 050: 032 010 061: 209: 211: 222: 233: 244: 255: 266: 032 020 277: 004: 301: 014: 302: 024: 303: 034: 032 030 304: 044: 305: 254: 306: 264: 307: 374: 032 040 310: 002: 012: 022: 032: 134: 220: 020: 032 050 102: 045: 232: 110: 245: 032: 120: 045: 032 060 032: 130: 045: 032: 140: 045: 032: 150: 032 070 045: 032: 160: 045: 032: 170: 045: 032: 032 100 106: 100: 032: 102: 100: 032: 112: 100: 032 110 032: 122: 100: 032: 132: 100: 032: 142: 032 120 100: 032: 152: 100: 032: 162: 100: 032: 032 130 172: 210: 010: 007: 043: 053: 263: 073: 032 140 003: 013: 023: 033: 075: 065: 107: 137: 032 150 167: 167: 009: 123: 301: 056: 032: 066: 032 160 212: 056: 020: 066: 123: </pre> <p>INA ?</p> <p>INM ?</p> <p>DCA ?</p> <p>DCM ?</p> <p>006: LAI ?</p> <p>104: JMP ASTST ?</p> <p>106: CAL ?</p> <p>RST TYOT ?</p> <p>RST 000001?</p> <p>LFM ?</p> <p>032 170 106: CAL CAL ?</p> <p>004: ADI 'A' ?</p>	<p>ASSEMBLY OBJECT CODE (ERRORS INCLUDED). THIS IS LOADED INTO MEMORY BEGINNING WITH LOCATION 012: 000:</p>

Figure 9.4. Loading of "BNPF Tape Generator" and Object Code

START OF PROGRAM
TO BE EXECUTED

KEYBOARD INPUT - [BEGIN: 913: 000:] ← START OF DATA BLOCK

000	BPPNNNNNPF
001	BPPNPNNPPF
002	BPPPPPNPPF
003	BNNPNPNPNF
004	BNPNPNNPPF
005	BNNPPPPPNNF
006	BPNPNPPPNNF
007	BNNPNPNNNF
010	BNNPPNNNPF
011	BPNNNNNNNNF
012	BPNNNPNNNPF
013	BPNNPNNPNF
346	BNNNNNNNNNF
347	BNNNNNNNNNF
350	BNNNNNNNNNF
351	BNNNNNNNNNF
352	BNNNNNNNNNPF
353	BNNNNNNNNNF
354	BNNNNNNNNNF
355	BNNNNNNNNNF
356	BNNNNNNNNNF
357	BNNNNNNNNNF
360	BNPNNNNNNNF
361	BWPNNNNNNNF
362	BNNNNNNNNPNF
363	BNNNNNNNNNF
364	BWPNNNNNNNF
365	BNNNNNNNNNF
366	BNNNNNPNNF
367	BNNNNNNNNNF
370	BNNNNNPNNPF
371	BNNNNNPNNNF
372	BNNNNNNNNNF
373	BNPNNNNNNNF
374	BNPNNNNNNNF
375	BNNNNNNNPF
376	BNPNNNNNNNF
377	BPPPPPPPFF

OUTPUT

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 auxilliary 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.

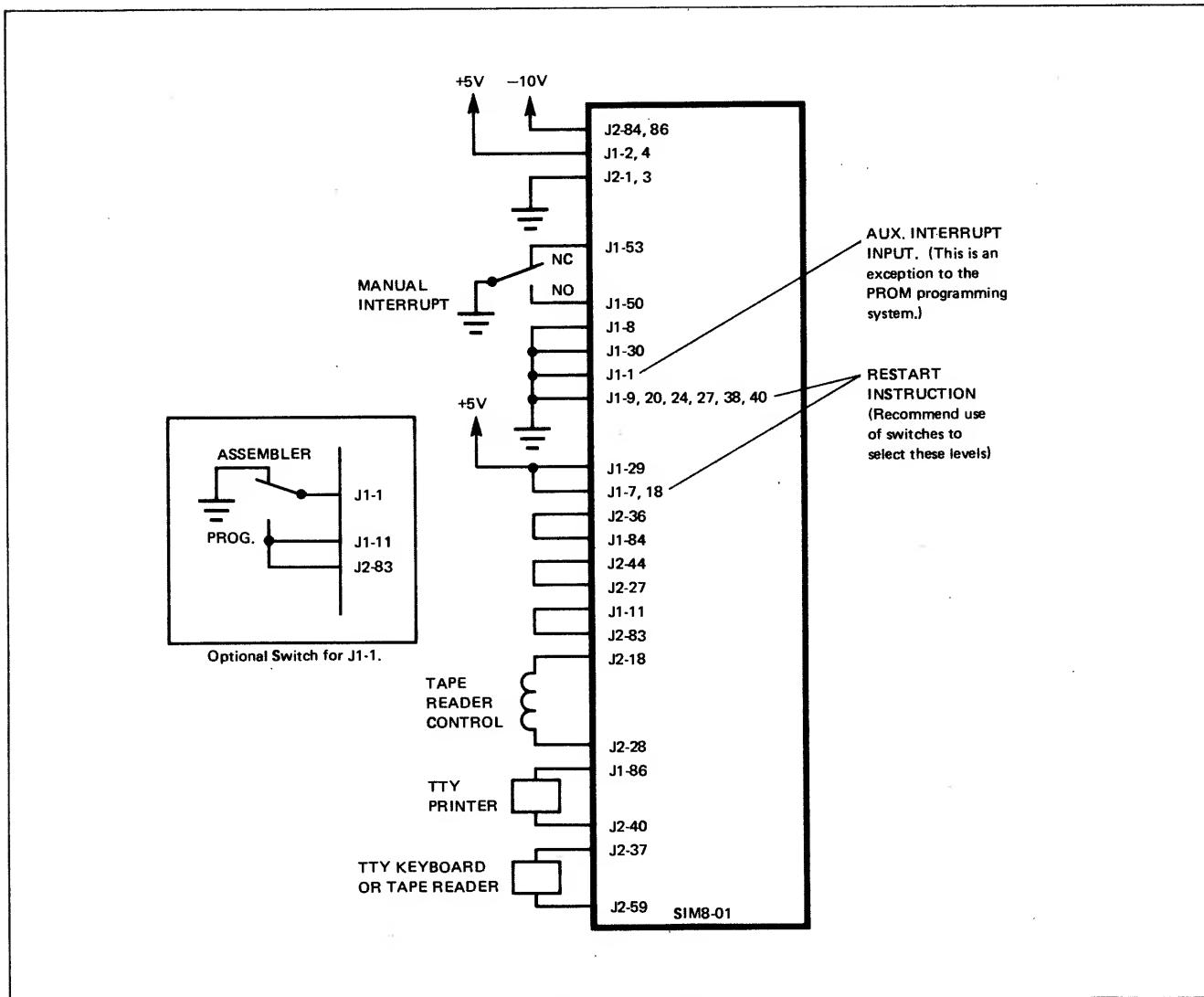


Figure 10.1. SIM8-01 Minimum Configuration Requirement

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 source program; 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. 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
```

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:

DANI	MIKE2-MIKE4 + 1
143B	773B + X2
1869	*-1
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:

Label	— Optional statement label;
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.
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.

(Label)		L r_1r_2		data		(Comment)
Move r_2 to r_1 .						

Examples:

Label		LEH		Comment
Move H to E.				

Label		LAM		Comment
Load A from memory.				

Label		LMB		Comment
Move B to memory.				

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)	XRr	data	(Comment)
---------	-----	------	-----------

Exclusive OR r with A.

3.2.7

(Label)	ORr	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 \neq 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 \neq 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 \neq 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.

(Label) | HLT | (Comment)

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.

(Label) | RST | start | (Comment)

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.

(Label) | SHL | address | (Comment)

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		Instruction starts in LOC 0000
LCD		
.		
.		
.		
ORG 1000B		
SAM LCD		Instruction stored in LOC 1000
.		
.		
.		
ORG 5000B		
SALLY DEF 1, 4, 777B, 7000B		Data starts in LOC 5000
END		

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 two 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, SEXPR, 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.

\$ERROR\$ 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.

\$ERROR\$ MULTIPLY DEFINED SYMBOL XXXXXX

The symbol XXXXXX has been defined more than one time.

\$ERROR\$ UNDEFINED SYMBOL XXXXXX

The symbol XXXXXX has been used but never defined.

\$ERROR\$ ILLEGAL NUMERIC CONTAINS CHARACTER X

An octal number includes an illegal digit (such as 8 or 9) or the numeric contains non numeric characters.

\$ERROR\$ ILLEGAL OPCODE XXX

The operation code XXX is not one of the acceptable mnemonics.

\$ERROR\$ MISSING OPERAND FIELD

No operand found for an operation code which requires one.

\$ERROR\$ ILLEGAL VALUE = YYYYYYY, 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= 37777B for 2 byte operands

XXXXXX= 37B for output device numbers

XXXXXX= 7 for input device numbers

YYYYYYY= given operand value

\$ERROR\$ ILLEGAL SYMBOL

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

\$ERROR\$ MISSING LABEL

The label, which is required by the EQU pseudo operation, is missing.

\$ERROR\$ SYMBOL TABLE OVERFLOW, MAXIMUM = XXXXXX

Too many symbols in source program to fit into allocated symbol table.

\$ERROR\$ LINE OVERFLOW, MAXIMUM = XXXX

Input line exceeds 48 characters; or missing carriage return.

\$ERROR\$ ERRONEOUS LABEL

Opcodes END and ORG may not have a label.

\$ERROR\$ ILLEGAL ORIGIN XXXXXX is less than XXXXXX

Value of new origin is less than current program count.

\$ERROR\$ 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

```
===== SYMBOL VALUE =====
1: MUL 00000
2: MUL000 00013
3: MUL001 00025
4: UMUL 00036
5: UMULS 00040
6: UDIV 00042
7: UDIV01 00054
8: DIV 00061
9: DIV000 00076
10: DIV001 00110
11: DIV002 00140
12: DIVVS 00144
13: UDIV 00146
14: UDIV00 00151
15: UDIV01 00173
16: DNEG 00204

=====
LOC OBJECT CODE SOURCE STATEMENTS
=====
00000 * MUL - SIGNED INTEGER MULTIPLY
00000 * CALL: ARGUMENTS IN C & 0
00000 * EXIT: HI ORDER PRODUCT IN B
00000 * LO ORDER PRODUCT IN C
00000 * REGS: A,B,C,O,E, AND FLAGS ALTERED
00000 * TIME: 1074 TO 1490 MICROSECONDS (8000)
00000 250 MUL XRA 1) COUNT AND NEGATE
00001 340 LEA NEGATIVE ARGUMENTS
00002 222 SUC
00003 160 013 000 JTS MUL000
00006 150 013 000 JT2 MUL000
00011 320 LCA
00012 040 INE
00013 250 MUL000 XRA
00014 223 SUD
00015 160 025 000 JTS MUL001
00020 150 025 000 JT2 MUL001
00023 330 LOA
00024 040 INE
00025 304 MUL001 LAE 2) MOVE COUNT MOD 2
00026 032 RAR TO CARRY
00027 106 036 000 CAL UMUL 3) CALL 'UNSIGNED
00032 142 204 000 CTC ONEG MULTIPLY', IF CARRY
00035 007 RET NEGATE RESULT; EXIT
00036 * UMUL - UNSIGNED INTEGER MULTIPLY
00036 * CALL: ARGUMENTS IN C & 0
00036 * EXIT: HI ORDER PRODUCT IN B
00036 * LO ORDER PRODUCT IN C
00036 * REGS: A,B,0L, AND FLAGS EXCEPT CARRY ALTERED
00036 * TIME: 896 TO 1138 MICROSECONDS (8000)
00036 * UMULS - MULTI-PRECISION MULTIPLY ENTRY
00036 * (B&C := C + 0 + B)
00036 016 000 UMUL LBI 0
00040 046 011 UMULS LEI 9
00042 302 UMUL00 LAC 1) ROTATE CARRY INTO
00043 032 RAR PRODUCT = MULTIPLIER
```

00044 320 LCA SHARED REGISTER,
 00045 041 DCE FORCING NEXT LSB
 00046 053 RTZ TD CARRY
 00047 301 LAB 2) EXIT IF 8TH ITERATION
 00050 100 054 000 JFC UMUL01 3) IF STEP (1) SET CARRY
 00053 203 ADD ADD MULTIPLICAND TO
 00054 032 UMUL01 RAR PRODUCT
 00055 310 LBA 4) ROTATE MOST SIGNIFICA
 00056 104 042 000 JMP UHUL00 PRODUCT AND GO TO (1)
 00061 * DIV - SIGNED INTEGER DIVIDE
 00061 * CALL: HI ORDER DIVIDEND IN B
 00061 * LO ORDER DIVIDEND IN C
 00061 * DIVISDR IN D
 00061 * EXIT: QUOTIENT IN C
 00061 * REMAINDER IN B
 00061 * OVERFLOW FLAG IN CARRY (CY=0>UV)
 00061 * REGS: A,B,C,D,E, AND FLAGS ARE ALTERED
 00061 * TIME: 922 TD 1416 MICRSECONDS (8008)
 00061 250 DIV XRA 1) COUNT AND NEGATE
 00062 340 LEA LEA NEGATIVE ARGUMENTS
 00063 221 SUB
 00064 160 076 000 JTS DIV000
 00067 150 076 000 JTZ DIV000
 00072 040 INE
 00073 106 204 000 CAL DNEG
 00076 250 DIV000 XRA
 00077 223 SUD
 00100 160 110 000 JIS DIV001
 00103 150 110 000 JTZ DIV001
 00106 330 LDA
 00107 040 INE
 00110 304 DIV001 LAE 2) MOVE COUNT MOD 2
 00111 032 RAR TD CARRY
 00112 106 146 000 CAL UDIV 3) CALL 'UDIV'
 00115 032 RAR EXIT WITH CARRY
 00116 340 LEA * 0 IF OVERFLOW
 00117 250 XRA DCURRED
 00120 262 DRC
 00121 063 RTS
 00122 301 LAB
 00123 223 SUD
 00124 003 RFC
 00125 250 XRA 4) IF CARRY WAS
 00126 264 DRE SET IN STEP (2)
 00127 120 140 000 JFS DIV002 NEGATE QUOTIENT
 00132 250 XRA AND REMAINDER
 00133 222 SUC
 00134 320 LCA
 00135 250 XRA
 00136 221 SUB
 00137 310 LBA
 00140 006 200 DIV002 LAI 200B 5) SET CARRY AND
 00142 022 RAL EXIT
 00143 007 RET
 00144 * UDIV - UNSIGNED INTEGER DIVIDE
 00144 * CALL: HI ORDER DIVIDEND IN B
 00144 * LO ORDER DIVIDEND IN C
 00144 * DIVISDR IN D
 00144 * EXIT: QUOTIENT IN C
 00144 * REMAINDER IN B
 00144 * NOTE: OVERFLOW IF B >= D
 00144 * REGS: A,B,C,E, AND FLAGS EXCEPT CARRY ALTERED
 00144 * TIME: 724 TD 1298 MICROSECONDS (8008)
 00144 * UDIVS - SINGLE PRECISION DIVIDEND ENTHY
 00144 016 200 UDIVS LBI 0
 00146 046 011 UDIV LEI 9
 00150 301 LAB
 00151 310 UDIV00 LBA
 00152 302 LAC 1) ROTATE CARRY INTD
 00153 022 RAL DIVIDEND - QUOTIENT
 00154 320 LCA SHARED REGISTER,
 00155 041 DCE FORCING NEXT MSB
 00156 150 173 000 JTZ UDIV01 TD CARRY
 00161 301 LAB 2) ROTATE MSB INTD
 00162 022 RAL HI ORDER QUOTIENT
 00163 223 SUD 3) SUBTRACT DIVISDR IF
 00164 100 151 000 JFC UDIV00 LESS THAN HI ORDER QU
 00167 203 ADD GD TD (1)
 00170 104 151 000 JHP UDIV00 ELSE ADD IT BACK
 00173 022 UDIV01 RAL AND GD TD (1)
 00174 340 LEA 4) COMPLEMENT QUOTIENT
 00175 006 377 LAI 377B AND EXIT
 00177 252 XRC
 00200 320 LCA
 00201 304 LAE
 00202 032 RAR
 00203 007 RET
 00204 * DNEG - DOUBLE PRECISION NEGATE
 00204 * CALL: HI ORDER IN B
 00204 * LO ORDER IN C
 00204 * EXIT: HI ORDER IN B
 00204 * LO ORDER IN C
 00204 * REGS: A,B,C, AND FLAGS ARE ALTERED
 00204 * TIME: 76 MICROSECONDS (8008)
 00204 * NOTE: -32768 CANNOT BE NEGATED
 00204 DNEG XRA
 00205 222 SUC
 00206 320 LCA
 00207 006 000 LAI 0
 00211 231 SBB
 00212 310 LBA
 00213 007 RET.
 00214 END

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:

Number	Value
11011B	11011_2
28D	28_{10}
33O	33_8
33Q	33_8
1CH	$1C_{16}$
28	28_{10}

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, $EF2_{16}$ must be expressed as 0EF2H.

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.

Command	Purpose
LOAD	Causes symbol tables and code to be loaded into the simulated MCS-8 memory.
GO	Starts execution of the loaded 8008 code.
INTER	Simulates an 8008 interrupt.
TIME	Displays time used in the 8008 simulation.
CYCLE	Allows the simulated CPU to be stopped after a given number of cycles.
TRACE	Enables tracing feature when particular portions of the program are executed.
REFER	Causes the CPU simulation to stop when a particular storage location is referenced.
ALTER	Causes the CPU simulation to stop when the contents of a particular memory location is altered.
CONV	Displays the values of numbers converted to the various number bases.
DISPLAY	Displays memory locations, CPU registers, symbolic locations, and IO ports.
SET	Allows the values of memory locations, CPU registers, and IO ports to be altered.
BASE	Allows the default number base used for output to be changed.
PUNCH	Causes output of machine code in BPNF format.
END	Terminates execution of an 8008 program.

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:

START	(Same as 10)
START+6	(Same as 16)
START-101B	(Same as 5)
10 TO INCR	(Same as 10 TO 32)
START+3 TO	
INCR-2	(Same as 13 TO 30)

The range-list may also contain a reference to the current value of the program counter of the simulated 8008 CPU. The symbol "*" represents this value. If the value of the program counter is 16, for example, the following is a valid range-list:

START TO * (Same as 10 TO 16)

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:

range-element { TO range-element }

where a range-element is defined as:

[number {[+ number]}]
symbolic-name {[- number]}
*

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:

range-element { TO 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. **[ALTER]** range list { , range-list, range-list, . . . , range-list } .

[NOALTER]

The ALTER command is an operand breakpoint command which causes the execution of the 8008 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:

ALTER 0
ALTER 0 TO 10
ALTER 10 T INCR.
ALTER START + 2 TO INCR - 0AH
AL 5, START, X2, 7 T 10, INCR-3

4. BASE {
 BIN
 OCT
 DEC
 HEX}

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, 28Q, 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 IO 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 \$WIDTH command of Section D).

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

CY	(carry)	D	PS (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
DISP 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 IO ports can be displayed by using a display element of the form:

PART 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 (\pm 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
 BIN
 OCT
 DEC
 HEX}

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 { [*] } .

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 *, 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 { 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 20Q).

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 "BNPF" 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. [REFER] range-list { , range-list, . . . , range-list } .
[NOREFER]

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 * .

14. SET. set-element { , set-element, . . . , 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:

register = [number
 *]

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 = 11111000B 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 [NOTRACE] 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 + 111B.
NOTRACE START, INCR, FOUND TO FOUND+3, 7Q.

17. PUNCH range 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:

<u>INTERP/8 No.</u>	<u>Device</u>	<u>PDP-10 Device</u>	<u>TYMSHARE</u>	<u>GE</u>
1	User's Console	TTY 5		
2	Card Reader	CDR 2		
3	Paper Tape	PAP 6		
4	Magnetic Tape	MAG 16		
5	Magnetic Tape	DEC 9		
6	Disk	DISK 20	FOR20.DAT	LOGOUT
7	Disk	DISK 21	FOR21.DAT	LOGBIN
<u>OUTPUT</u>				
<u>INTERP/8 No.</u>	<u>Device</u>	<u>PDP-10 Device</u>	<u>File Name</u>	
1	User's Console	TTY5		
2	Printer	PTR 3		
3	Paper Tape	PAP 7		
4	Magnetic Tape	MAG 17		
5	Magnetic Tape	DEC 10		
6	Disk	DISK 22	FOR22.DAT	Disk φ1
7	Disk	DISK 23	FOR23.DAT	Disk φ2

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 "=" followed by a decimal integer. The character following the "\$" and the integer value affect the I/O formatting functions as follows:

<u>Control</u>	<u>Meaning</u>	<u>Initial Value</u>
\$COUNT = n	Start the output line count at the value n.	1
\$DELETE = n	Delete all characters after column n of the output	120
\$EOF = 1	End-of-file on this device	0
\$INPUT = n	Read subsequent input from file number n	1
\$LEFT = n	Ignore character positions 1 through n-1 of the input.	1
\$OUTPUT = n	Write subsequent output to file number n.	1
\$PRINT = n	Controls listing of the output. If n = 0, input lines are not printed; otherwise input is echoed.	0
\$RIGHT = n	Ignore all character positions beyond column n of the input.	80
\$TERMINAL = n	INTERP/8 assumes conversational usage if n = 1; otherwise batch processing is assumed.	1
\$WIDTH = n	This command sets the width of the output line. Note that this affects the format of the DISPLAY MEMORY command.	72

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

```

E R R O R   M E S S A G E S
EXECUTION ERRORS
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

```

```

10 IO FORMAT COMMAND ERROR (TOGGLE HAS VALUE OTHER THAN 0 OR 1)
11 UNUSED
...
13 INVALID SEARCH PARAMETER IN DISPLAY SYMBOL COMMAND (MUST BE
SYMBOLIC NAME, ADDRESS, OR *)
14 DISPLAY SYMBOLS COMMAND INVALID SINCE NO SYMBOL TABLE EXISTS
15 UNUSED
16 UNRECOGNIZED COMMAND OR INVALID FORMAT IN COMMAND MODE
17 MISSING . OR EXTRA CHARACTERS FOLLOWING COMMAND
18 LOWER BOUND EXCEEDS UPPER BOUND OR IS LESS THAN ZERO
IN RANGE LIST
19 THE FORMAT OF THE SYMBOL TABLE IS INVALID (MUST BE A
SEQUENCE OF THE FORM N SY AD, WHERE N IS AN INTEGER,
SY IS THE SYMBOLIC NAME, AND AD IS THE ADDRESS (IN OCTAL))
20 INVALID CHARACTER IN MACHINE CODE FILE.
21 UNUSED

```

```

22 UNRECOGNIZED DISPLAY ELEMENT OR INVALID DISPLAY FORMAT
23 SYMBOLIC NAME NOT FOUND IN SYMBOL TABLE
24 INVALID ADDRESS OR NO SYMBOL TABLE PRESENT IN DISPLAY SYMBOL
COMMAND
25 OUTPUT DEVICE WIDTH TOO NARROW FOR DISPLAY MEMORY COMMAND
(USE SWIDTH = N IO FORMAT COMMAND TO INCREASE WIDTH)
26 INVALID RADIX IN MEMORY DISPLAY COMMAND (MUST BE CODE, BIN,
OCT, OR DEC)
27 UNRECOGNIZED SET ELEMENT IN SET COMMAND
28 MISSING SET LIST IN SET COMMAND
29 INVALID SET LIST OR SET VALUE IN SET COMMAND
30 MISSING OR MISPLACED = IN SET COMMAND
31 MISSING PROGRAM STACK ELEMENT NUMBER IN SET PS N
COMMAND
32 INVALID INTERRUPT CODE SPECIFICATION (EITHER MORE THAN THREE
BYTES, OR ELEMENT EXCEEDS 255)

```

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.

```

.R INTB
BEGIN
/* THIS IS AN EXAMPLE OF THE USE OF THE INTERP/8 SYSTEM.

IN THIS EXAMPLE, THE BASIC COMMANDS WILL BE DEMONSTRATED
AND A SIMPLE PROGRAM WILL BE CONSTRUCTED AND EXECUTED */
/* THE NUMBER CONVERSION COMMAND IS USED FIRST */

CCNV 10.

1010B 12Q 10 AH
CON 100.

1000B 100 8 8H
CCN 3 TO 8.

```

```

11B 30 3 3H
100B 4C 4 4H
101B 5Q 5 5H
110B 6Q 6 6H
111B 7Q 7 7H
1000B 10Q 8 8H

```

```

/* NEXT, THE VARIOUS DISPLAY AND SET COMMANDS ARE DEMONSTRATED */

DISPLAY CPU.

CYZSP A B C D E H L HL SP PS0
*000*000*000*000*000*000*000*000*000*000*000*000
DISP A,D,HL.

A = 0
D = 0
HL = 0
DIS PORT 4, PS 0, MEM 5.

P4=0
PS0 = 0
/* MEMORY LOCATION 5 WAS NOT DISPLAYED SINCE NO PROGRAM HAS BEEN
LOADED */
SET H = 5, L=10Q. DISP CPU.

```

```

SET OK
CYZSP A B C D E H L HL SP PS0
0000 000 000 000 000*000*008*01288 000 00000
/* NOTE THAT THE ELEMENTS WHICH HAVE CHANGED SINCE THE LAST DISPLAY
ARE PRECEDED BY AN ASTERISK */

SET HL = 0EEFH. DIS CP.

```

```

SET OK
CYZSP A B C D E H L HL SP PS0
0000 000 000 000 000 000 000*014*239*03823 000 00000
CONV 03823.

11101110111B 7357Q 3823 EEFH
/* NOW CHANGE THE DEFAULT NUMBER BASE TO HEXADECIMAL */

BASE HEX. DISP CPU.

```

```

HEX BASE OK
CYZSP A B C D E H L HL SP PS0
0000 00H 00H 00H 00H 00H 0EH EFH 0EEFH 00H 0000H
/* THEN CHANGE BASE TO OCTAL */

BASE OC. DI CP.

```

```

OCT BASE OK
CYZSP A B C D E H L HL SP PS0
0000 000Q 000Q 000Q 000Q 000Q 016Q 357Q 07357Q 000Q 00000Q.

```

```

/* NOW PLACE A SIMPLE PROGRAM INTO MEMORY STARTING AT LOCATION 10.
THIS PROGRAM WILL ALTER THE VALUE OF MEMORY CELL 200 BY ADDING 1
TO THE CURRENT VALUE OF THE CELL. IN SYMBOLIC FORM, THE PROGRAM
IS AS FOLLOWS... LHI 0, LLI 200, LBM, INB, LMB, HLT.

THE LOAD OPERATION BELOW IS A 'DUMMY' OPERATION SO THAT MEMORY IS
INITIALIZED PROPERLY. */

LOAD I.

```

```

0Q LOAD OK
DISPLAY MEMORY 10 TO 20.

00012Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q
BASE DEC.

```

```

DEC BASE OK
SET MEM 10 TO 20 = 00101110B 0 /* THIS IS LHI 0 */
00110110B 200 /* LLI 200 */
1100111B /* LBM */ 00001000B /* INB */
11111001B /* LMB */ 0 /* HLT */

```

```

SET OK
DI ME 10 TO 20.

```

00010 046 000 054 200 207 000 249 000 046 000 054
DI M 10 TO 20 CODE.

00010 LHI,00H LLI,C8H LBM INB LMB HLT LHI,00H LLI

/* NOTE THAT THE ',', SEPARATES ELEMENTS WHICH ARE PART OF THE
SAME INSTRUCTION (THE SECOND AND THIRD BYTES ARE IN HEX) */
CONV SC8H.

11001000B 3100 200 C8H

/* WE CAN NOW EXECUTE THE PROGRAM BY SETTING THE PROGRAM COUNTER
TO LOCATION 10 */
SET PC=10. DI CP.

SET OK
CYZSP A B C D E H L HL SP PS0
0000 000 000 000 000 000 614 239 63823 000+00010

SE HL=0.

SET OK

GO.

HLT CYCLE 56
DI CPU.

CYZSP A B C D E H L HL SP PS0
0000 000+001 000 000 000+000+200+00200 000+00017
DI MEM 200.

00200 001
/* MEMORY LOCATION 200 HAS BEEN INCREMENTED -- NOW TURN ON THE
TRACE AND EXECUTE THE PROGRAM AGAIN */

TRACE 0 TO 100. GO.

TRACE OK
0000 001 000 000 000 000 200 00200 000 00017
HLT
HLT CYCLE 68
/* CPU MUST FIRST BE INITIALIZED TO ZERO */ SET CPU. GO.

SET OK
0000 000 000 000 000 000+000+00000 000+00000
HLT
HLT CYCLE 4
DI CPU.

CYZSP A B C D E H L HL SP PS0
0000 000 000 000 000 000 00000 000 00000
/* FORGOT TO SET PC = 10, TRY AGAIN */ SET CPU, PC=10. GO.

SET OK
0000 000 000 000 000 000 000 00000 000+00010
LHI 0
0000 000 000 000 000 000 00000 000+00012
LLI 200
0000 000 000 000 000 000+200+00200 000+00014
LBM
0000 000+001 000 000 000 000 200 00200 000+00015
INB
0000 000+002 000 000 000 000 200 00200 000+00016
LMB
0000 000 002 000 000 000 000 200 00200 000+00017
HLT
HLT CYCLE 48
/* NOW TRY THE SAME EXECUTION WITH THE TRACE ENABLED OVER ONLY

PART OF THE PROGRAM */

NOTRACE 0 TO 100. TRACE 12 TO 14, 17.

TRACE OK
TRACE OK
SET CPU, PC=10. GO.

SET OK
0000 000+000 000 000 000 000+000+00000 000+00012
LLI 200
0000 000 000 000 000 000+200+00200 000+00014

LBM
00001 000+003 000 000 000 000 200 00200 000+00017
HLT
HLT CYCLE 48
/* SWITCH BACK TO FULL TRACE */ TR 0 TO 100.

TRACE OK
DISP MEM 200.

00200 003
/* NOW RUN THE CPU FOR ONLY A FEW INSTRUCTIONS AT A TIME. IN THIS
WAY THE EXECUTION CAN BE MONITORED EASILY */

GO 2.

GO OK
CYZSP A B C D E H L HL SP PS0
0001 000 003 000 000 000 000 200 00200 000 00017
HLT
HLT CYCLE 44
SET CPU, PC=10. GO 2.

SET OK
GO OK
0000 000+000 000 000 000 000+000+00000 000+00010
LHI 0
0000 000 000 000 000 000 000 000 00000 000+00012
LLI 200
CYCLE AT 14
DI CPU.

CYZSP A B C D E H L HL SP PS0
0000 000 000 000 000 000 000+200+00200 000+00014
GO 1.

GO OK
0000 000 000 000 000 000 000 200 00200 000 00014
LBM
CYCLE AT 15
DI CPU.

CYZSP A B C D E H L HL SP PS0
0000 000+003 000 000 000 000 200 00200 000+00015
GO *.

0000 000 003 000 000 000 000 200 00200 000 00015
INB
0000 000+004 000 000 000 000 200 00200 000+00016
LMB
0000 000 004 000 000 000 000 200 00200 000+00017
HLT
HLT CYCLE 40
DI CPU.

CYZSP A B C D E H L HL SP PS0
0000 000 004 000 000 000 000 200 00200 000 00017
/* WE CAN SET BREAK POINTS IN THE CODE SO THAT AT EXECUTION STOPS
WHEN A PARTICULAR INSTRUCTION IS Fetched. */

SET CPU,PC=10. TR 0 TO 100. REFER 12 TO 14.

SET OK
TRACE OK
REFER OK
GO.

*0000 000+000 000 000 000 000+000+00000 000+00010
LHI 0
0000 000 000 000 000 000 000 000 00000 000+00012
LLI 200
REFER AT 12
DI CPU.

CYZSP A B C D E H L HL SP PS0
0000 000 000 000 000 000 000 000 00000 000 00012

/* THE EXECUTION CAN ALSO BE STOPPED WHEN THE PROGRAM REFERS
TO MEMORY LOCATION 200 */

REFER 200. NOTRACE 8 TO 100. SET CPU,PC=10. GO.

REFER OK
TRACE OK
SET OK
REFER AT 14
DI CPU.

CY2SP A B C D E H L HL SP PSB
0000 000 000 000 000 000 000*200+00200 000+00014
DI MEM 14 CODE.

00014 LBM
GO 1. DI CP.

GO OK
CYCLE AT 15
CY2SP A B C D E H L HL SP PSB
0000 000+005 000 000 000 000 200 00200 000+00015

/* THIS SHOWS THE VALUE FETCHED FROM LOCATION 200. WE CAN STOP
THE PROGRAM ON A STORE INTO LOCATION 200 AS WELL */
NOREF 200. ALTER 200. SET CP, PC=10. GO.

REFER OK
ALTER OK
SET OK

ALTER AT 16
DI CPU.

CY2SP A B C D E H L HL SP PSB
*0001 000+006 000 000 000 000 200 00200 000+00016
D M 16 CO.

00016 LMB
/* THE REGISTER DUMP SHOWS THAT 6 WILL BE STORED AT LOCATION 200.
EXAMINE LOCATION 200, RUN THE MACHINE FOR ONE CYCLE, AND EXAMINE
THE CELL AGAIN */

DI MEM 200. GO 1. DI MEM 200.

00200 005
GO OK
CYCLE AT 17
00200 006

/* NOW GET A COMPLETE MEMORY DUMP IN BINARY */

DI MEM 8 TO 777Q BIN.

00000 00000000B 00000000B 00000000B 00000000B 00000000B 00000000B
00006 00000000B 00000000B 00000000B 00101110B 00000000B
00012 00110110B 11001000B 11001111B 00001000B 11111001B 00000000B
00018 00101110B 00000000B 00110100B 00000000B 00000000B 00000000B
00024 00000000B 00000000B 00000000B 00000000B 00000000B 00000000B

00198 00000000B 00000000B 00000110B 00000000B 00000000B 00000000B
00294 00000000B 00000000B 00000000B 00000000B 00000000B 00000000B
00510 00000000B 00000000B

/* AND THEN PUNCH THE CODE BETWEEN LOCATIONS 10 AND 26 (WE WILL USE
THE CONSOLE AS THE OUTPUT DEVICE) */

PUNCH 10 TO 26 1.

8 BNNNNNNNNF BNNNNNNNNF BNNPMPNNF BNNNNNNNNF
BNNPPNPNNF BPPNPNNNF BNNNNNNNF BNNPMPNNF
16 BPPPPNPNNF BNNNNNNNNF BNNNPMPNF BNNNNNNNNF
BNPMPNPNF BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF

END.

SEOF

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

THIS EXAMPLE SHOWS A COMPLETE ASSEMBLY AND INTERP/8 EXECUTION

TYPE ASMI-DAT
* SAMPLE MCS-8 PROGRAM (PAGE 47 OF 8088 MANUAL)
START LLI 200
LHI 0
LOOP LAM
CPI 46
JNZ FOUND
CAL INCR
LAL
CPI 228
JFZ LOOP
FOUND RET
INCR INL
RFZ
INH
RET
END
.R ASM8

PLEASE TYPE INPUT FILE NAME
ASMI

8085 INTEL ASSEMBLER

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

EXIT
*C

.RENAME FOR20.DAT = LOGOU.DAT, FOR21.DAT = LOGBI.DAT
FILES RENAMED:
LOGOU.DAT
LOGBI.DAT

.TYPE FOR20.DAT

SYMBOL VALUE

1: START 00000
2: LOOP 00004
3: FOUND 00023
4: INCR 00024

***** *C
*C

.TYPE DOR+U
TYPE FOR21.DAT

6 BNNPPNPNNF BPPNPNNNF BNNPMPNNF BNKNNNNNNF
BPPNNPPPF BNNPPPNNF BNPNNPPNF BNPPNPNNF
8 BNNNPMPFF BNNNNNNNNF BNPNNPPNF BNPPNPNNF
BNNNNNNNNF BPPNPNNNF BNPNNPPNF BNPPNPNNF
16 BPNPNNNNF BNNNNNNNNF BNNNNNNNNF BNKNNNNNNF
BNNPPNPNNF BNNNNNNNNF BNPNPNNNF BNKNNNNNNF
24 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNKNNNNNNF
BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNKNNNNNNF
32 BNNNNNNNNF BNNNNNNNNF BNNNNNNNNF BNKNNNNNNF
BNL+C

THE CODE FILE MUST BE TBRMINATED BY A \$ IN THE INPUT -- USE TECO
TECO FOR21.DAT

*N 32 SS
*OLSTSS
32 BNNNNNNNF BNNNNNNNF BNNNNNNNF

*IS
SS
*EX\$

.R INT8
LOADING

LOADER 10K CORE
EXECUTION

BEGIN
/* THE SYMBOL TABLE AND CODE WILL NOW BE LOADED */

LOAD 6 7.

32 LOAD OK
DI SYMBOLS.

000000Q 00000 0000H START
000004Q 00004 0004H LOOP
000023Q 00019 0013H FOUND
000024Q 00020 0014H INCR
DI SYMBOL LOOP.

000004Q 00004 0004H LOOP
DI SYMBOL ZAP.

(00027) ERROR 23 NEAR ZAP
/* ERROR MESSAGE HAS LINE NUMBER ERROR NUMBER AND ITEM IN ERROR. IN

THIS CASE, THE SYMBOL COULD NOT BE FOUND IN THE TABLE */

DI SY 13H.

FOUND
DI SY 12H.

FOUND-1
DI SY 8.

LOOP+4
DI SY *.

START
/* NOW TAKE A LOOK AT MEMORY IN HEXADECIMAL AND IN CODE FORMAT */

DI MEM 0 TO 100 HEX, MEM 0 TO 100 CODE.

00000 36H C8H 2EH 00H C7H 3CH 2EH 68H 13H 00H 46H 14H 00H C6H 3CH DCH
00016 48H 04H 00H 07H 30H 0BH 28H 07H 00H 00H 00H 00H 00H 00H 00H
00032 00H 00H

00096 00H 00H 00H 00H 00H
00000 LLI,C8H LHI,00H LAM CPI,2EH JTZ,13H,00H CAL,14H,00H LAL CPI,DCH
00016 JFZ,04H,00H RET INL RFZ INX RET HLT HLT HLT HLT HLT HLT HLT HLT
00032 HLT HLT

00096 HLT HLT HLT HLT HLT

/* THIS PROGRAM SEARCHES FOR A 46 STARTING AT LOCATION 200 IN
MEMORY. WE WILL START BY PLACING A SEQUENCE OF NUMBERS IN THESE

LOCATIONS /*

SET MEM 200 TO 210 = 43 46 48 20H 1111000B. DI MEM 200 TO 210.

SET OK

00200 043 046 048 032 120 043 046 048 032 120 043
/* GET A COMPLETE TRACE OF THE PROGRAM */ TR 0 TO 0100.

TRACE OK
GO.

CY2SP A B C D E H L HL SP PS0
*0000*0000*0000*0000*0000*0000*0000*0000*0000*0000*0000
LLI 200
0000 000 000 000 000 000 000*200*00200 000*00002
LHI 8
0000 000 000 000 000 000 000 200 00200 000*00004
LAM
0000*043 000 000 000 000 000 200 00200 000*00005
CPI 46
*1016 043 000 000 000 000 000 200 00200 000*00007
JTZ 19
1016 043 000 000 000 000 000 200 00200 000*00010
CAL 20
1016 043 000 000 000 000 000 200 00200*001*00013*00020
INL
*1011 043 000 000 000 000 000*201*00201 001 00013*00021
RFZ
1011 043 000 000 000 000 000 201 00201*000 00013

LAL
1011*201 000 000 000 000 000 201 00201 000*00014
CPI 220
CY2SP A B C D E H L HL SP PS0
1011 201 000 000 000 000 000 201 00201 000*00016
JFZ 4
1011 201 000 000 000 000 000 201 00201 000*00004
LAM
1011*046 000 000 000 000 000 201 00201 000*00005
CPI 46
*0101 046 000 000 000 000 000 201 00201 000*00007
JTZ 19
0101 046 000 000 000 000 000 201 00201 000*00019
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 MM\MM\EM 19 COD.

00019 RET

SET M 19 = 0. DI MEM 19 CO.

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

TRACE OK
SET OK
HLT CYCLE 117
DI CPU.

CY2SP A B C D E H L HL SP PS0
0101 046 000 000 000 000 000 201 00201 000 00019
/* 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 *.

0000000 00000 0000H START
0000040 00004 0004H LOOP
0000230 00019 0013H FOUND
0000240 00020 0014H 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

REFER OK
*1011*201 000 000 000 000 000 201 00201 000*00004
LAM
*0101*046 000 000 000 000 000 201 00201 000*00019
HLT
REFER AT 19
DI CP.

CY2SP A B C D E H L HL SP PS0
0101 046 000 000 000 000 201 00201 000 00019
SET CP. GO.

SET OK
*0000*000 000 000 000 000 000*000*00000 000*00000
LLI 200
0000 000 000 000 000 000 000*200*00200 000*00004
LAM
*1011*201 000 000 000 000 000 000*201*00201 000 00004
LAM
*0101*046 000 000 000 000 000 201 00201 000*00019
HLT
REFER AT 19
GO.

0101 046 000 000 000 000 201 00201 000 00019
HLT
HLT CYCLE 117

/* THE ONLY REMAINING COMMANDS TO ILLUSTRATE ARE THE SET AND
DISPLAY

POTS COMMANDS */

DI PORT 4.

P4=0
DI PORT 4, PO 3, PO 7 TO 100.

P4=0
P3=0
P7=0 P8=0
DI PO 20 TO 25.

P20=0 P21=0 P22=0 P23=0 P24=0 P25=0
SET PORT 5 = 11001100B, PO 10H = 550.

SET OK
DI POR 5 TO 17.

P5=204 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.

SEOF

APPENDIX IV

TELETYPE 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 internally. 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 thyrauctor 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.)

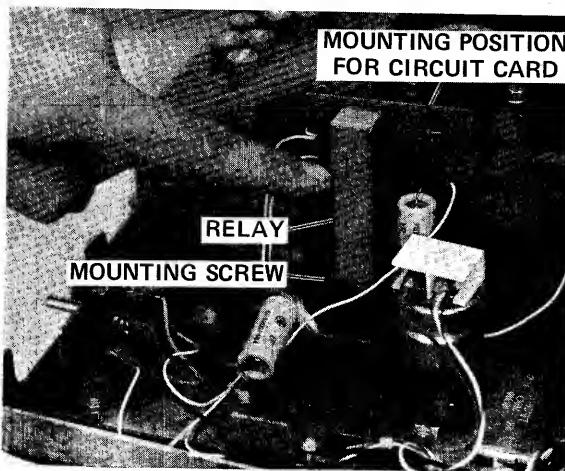


Figure 1. Relay Circuit (Alternate)

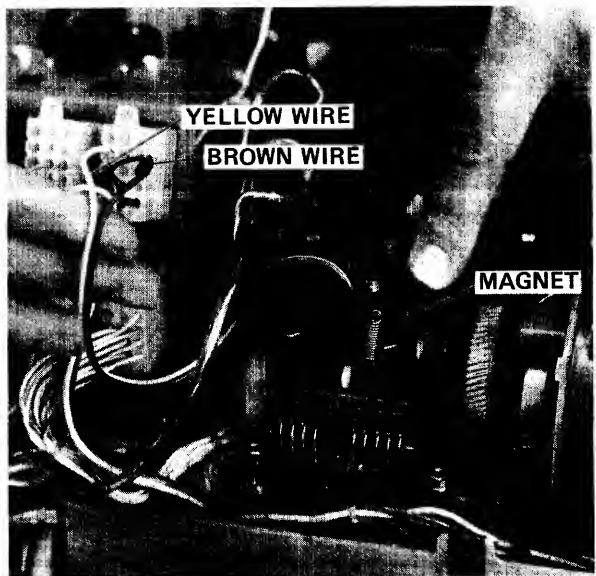


Figure 2. Distributor Trip Magnet

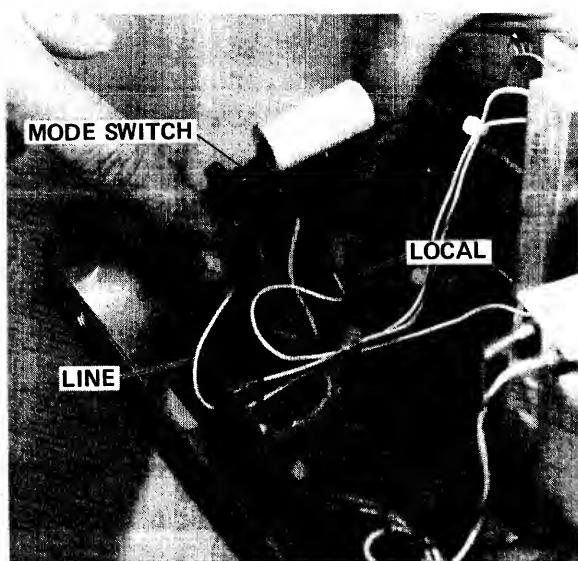


Figure 3. Mode Switch (Rear View)



Figure 4. Terminal Block

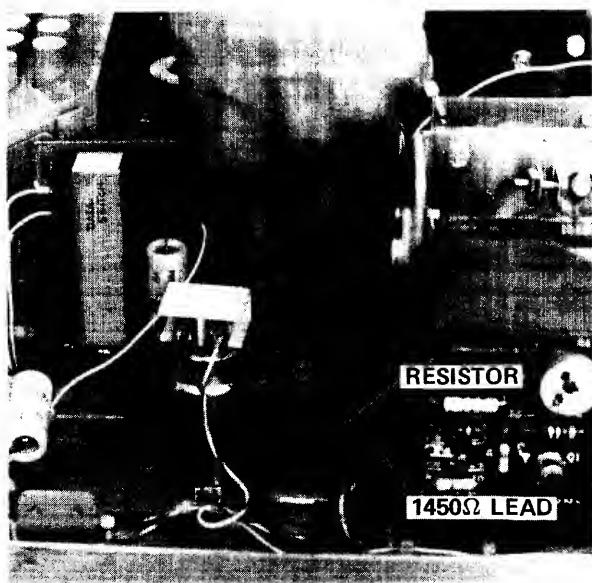


Figure 5. Current Source Resistor

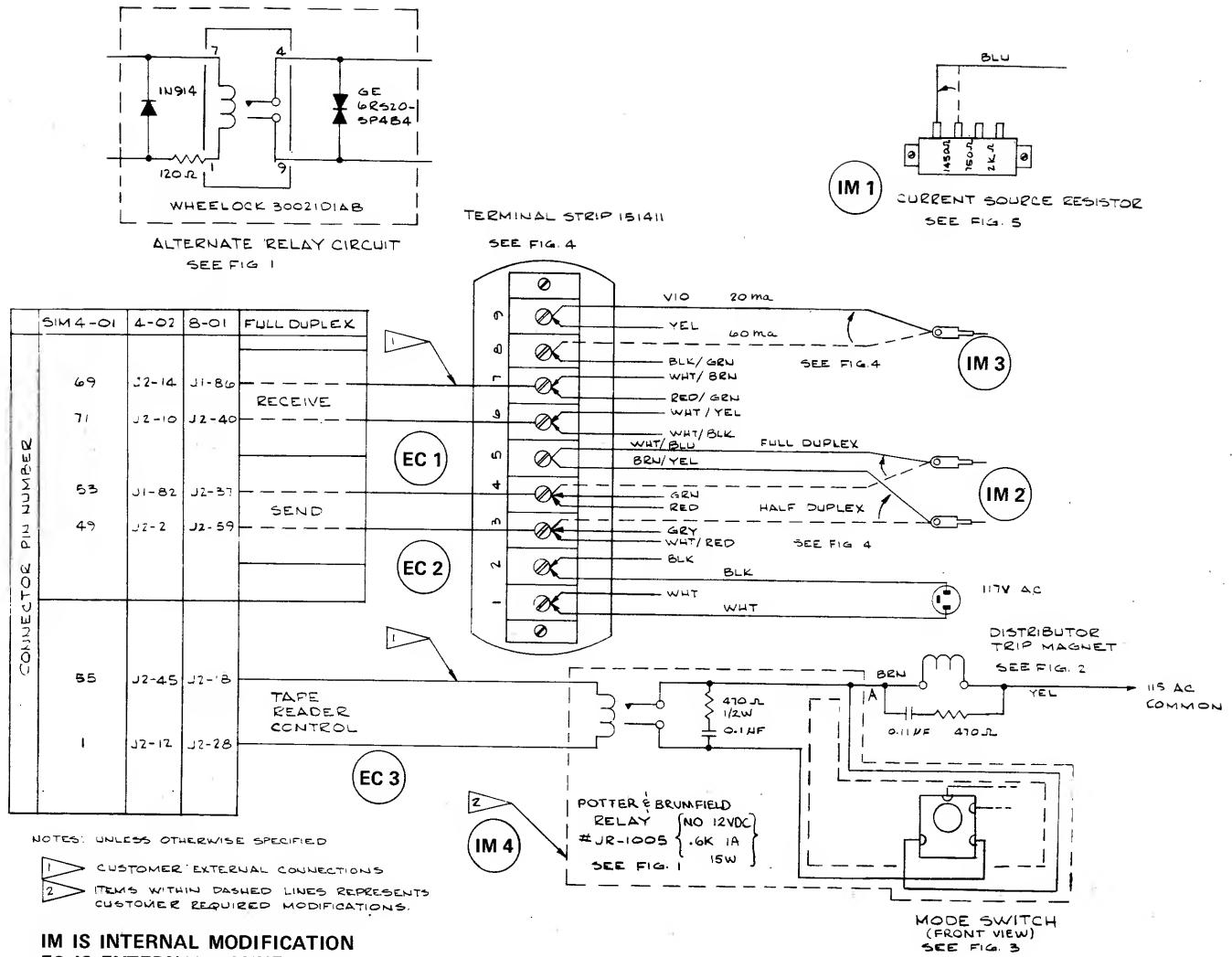


Figure 6. Schematic

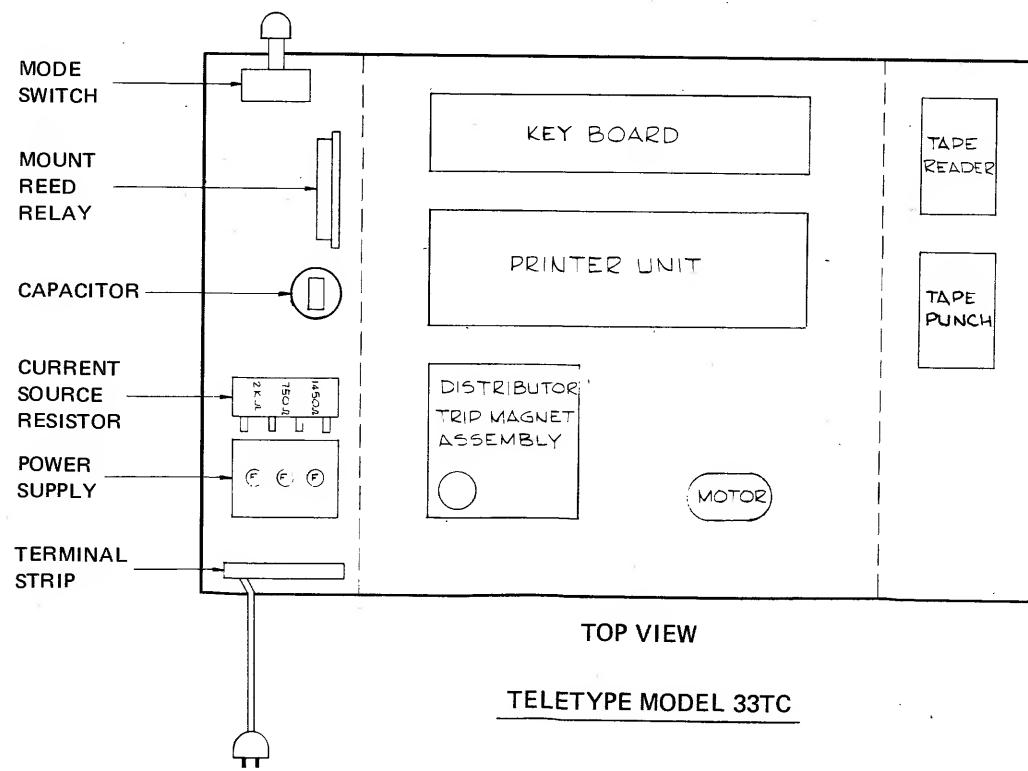
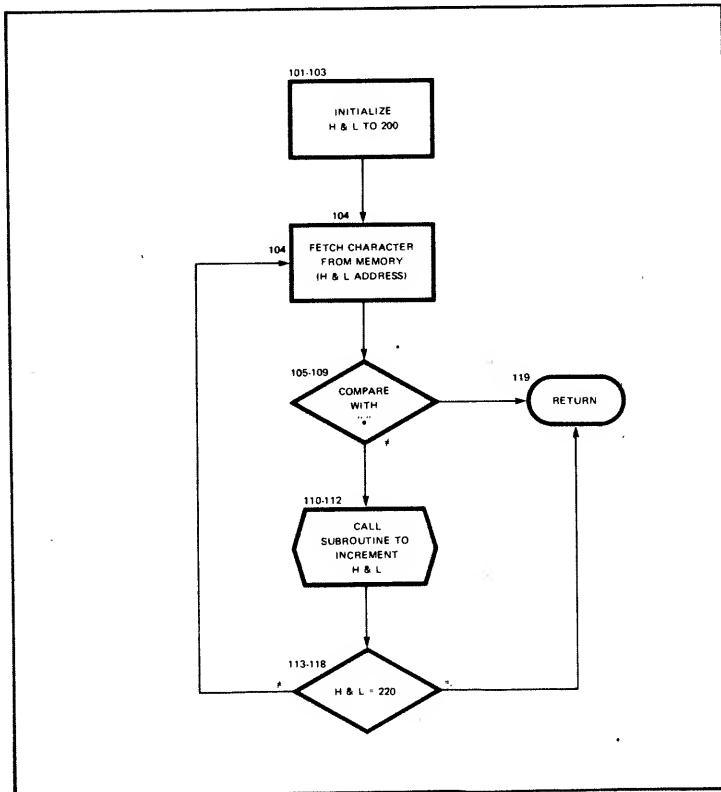


Figure 7. Block Diagram

APPENDIX V. PROGRAMMING EXAMPLES

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

MNEMONIC	OPERAND	EXPLANATION	BYTES	LOCATION	ROM CODE	COMMENT
Start:	LLI 200	Load L with 200	2	100 101	00110110 11001000	(200)
LHI	0	Load H with 0	2	102 103	00101110 00000000	(0)
Loop:	LAM	Fetch Character from Memory	1	104	11000111	ASC II
CPI	"."	Compare it with period	2	105 106	00111100 00101110	ASC II (•)
JTZ	Found	If equal go to return	3	107 108 109	01101000 01110111 00000000	(119)
CAL	INCR	Call increment H&L subroutine	3	110 111 112	01000110 00111100 00000000	(60)
LAL		Load L to A	1	113	11000110	
CPI	220	Compare it with 220	2	114 115	00111100 11011100	(220)
JFZ	Loop	If unequal go to loop	3	116 117 118	01001000 01101000 00000000	(104)
Found:	RET	Return	1	119	00000111	
INCR:	INL	Increment L	1	60	00110000	
RFZ		Return if not zero	1	61	00001011	
INH		Increment H	1	62	00101000	
RET		Return	1	63	00000111	



Subroutine to Search for Period.

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.
      JMP BEGIN

TAPE  LAI 1      TAPE READER ENABLE CODE
      OUT 13B    OUTPUT 3 - ENABLE TAPE READER
      CAL TTYD1  TAPE READER CONTROL DELAY
      HLT       WAIT FOR TTY START PULSE
      CAL TTYD2  TTY DELAY - 4.468 SEC.
      XRA       TAPE READER DISABLE CODE
      OUT 13B    OUTPUT 3, DISABLE TAPE READER
      INP 0B    INPUT 0, READ START PULSE
      LCI 255   COMPLEMENT TTY START PULSE
      XRC       EXCLUSIVE-OR REG. C
      OUT 12B    OUTPUT 2, OUTPUT START PULSE
      LEI 248   TTY DATA SAMPLING COUNTER
      CAL TTYD1  TTY DELAY - 9.012 SEC.
      INP 0B    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
      LBA       LOAD AC TO REG. B
      INE       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 128   REMOVE PARITY BIT
      LBA       STORE TTY INPUT DATA
      CAL TTYD1
      LAI 1
      OUT 12B    SUPPRESS TTY
      RET
      TTYD1 LDI 115  9.012 SEC. DELAY
      ST      IND     D = D + 1
      JFZ ST
      RET
      TTYD2 LDI 186  4.468 SEC. DELAY
      ST2     IND     D = D + 1
      JFZ ST2
      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 11B
      OUT 12B
      OUT 13B
      OUT 14B
      OUT 15B
      OUT 16B
      OUT 17B
      CAL DELAY  DELAY 16.436 SEC.
      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 REG. C
      LLI 252B  LOAD 252B(OCTAL) TO REC. C
      LHI 0     LOAD 0 TO REC. H
      CSTEST   LAH
      OUT 10B
      LAL       LOAD L TO AC
      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
D1      LDI 0      LOAD 0 TO REC. D
        IND
        JFZ D1
        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
      LM1      LMA       LOAD AC TO MEMORY
      INL
      CPL
      JFZ LM2   JUMP IF AC IS NOT ZERO
      INH          H = H + 1
      LAI 12    LOAD 12 TO AC
      CPH
      JFZ LM1   AC-H
      LHI 8     JUMP IF AC IS NOT ZERO
      REPT4    LAB
      OUT 10B   LOAD REC. B TO AC
      REPT3    LLC
      LAC
      OUT 13B   LOAD REC. C TO L
      LAI 255   LOAD REC. C TO AC
      LMA
      CPM
      JFZ ERROR  JUMP IF AC IS NOT ZERO
      REPT2    LAH
      OUT 10B   LOAD REC. H TO AC
      REPT5    XRA
      INL
      CPL
      JTZ REPT1 JUMP IF AC=0
      LAL       LOAD REC. L TO AC
      OUT 11B   CLEAR AC
      XRA
      CPM
      JFZ ERROR  JUMP IF AC IS NOT ZERO
      REPT1    INH          H = H + 1
      LAI 12
      CPH
      JTZ CONT
      XRA
      CPM
      JFZ ERROR
      JMP REPT2
      REPT1    INH
      LAI 12
      CPH
      JTZ CONT
      XRA
      CPM
      JFZ ERROR
      JMP REPT2
      REPT1    LHB       LOAD REC. B TO H
      XRA
      INC
      CPC
      JFZ REPT3
      INB
      LHB
      LAI 12
      CPB
      JFZ REPT4
      JMP BEGIN
      REPT1    LAI 240  LOAD 240 TO AC
      ADB
      OUT 10B
      LAL
      OUT 11B
      LAM
      OUT 12B
      LAC
      OUT 13B
      HLT
      END
      REPT1    LAL       LOAD REC. L TO AC
      OUT 11B
      LAM
      OUT 12B
      LAC
      OUT 13B
      HLT
      END
      REPT1    LAM       LOAD MEMORY TO AC
      OUT 12B
      LAC
      OUT 13B
      HLT
      END
      REPT1    LAC       LOAD REC. C TO AC
      OUT 13B
      HLT
      END
  
```

E. Bootstrap Loader Program

(Intel Tape Numbers A0860, A0861, A0863, Nov. 16, 1972)

0	6	1	BEGIN	ORC 0	SUPPRESS TTY	131 16	INC	C=C+1
2	85			OUT 12B	OUTPUT 2	132 68 126 0	JMP BD3	B = 10
3	168		XRA	CLEAR AC		135 14 10	LBI 10	AC=AC+B
4	87		OUT 13B	OUTPUT 3 - TAPE R		137 129	ADB	LOAD AC TO REG B
5	0		EADER CONTROL			138 200	LBA	A=A+43
6	68 206	1	HLT			139 6 48	LAI 48	A=A+C
9		*	JMP START			141 130	ADC	L=L+1
9		*	*TELETYPE TAPE READER & I/O CONTROL			142 48	INL	LOAD A TO M
9	6	1	TAPE LAI 1	TAPE READER ENABL		143 248	LMA	A=A+48
11	87	E CODE	OUT 13B	OUTPUT 3 - ENABLE		144 6 48	LAI 48	A=A+B
12	0	TTY HLT	TAPE READER	WAIT FOR TTY STAR		146 129	ADB	L=L+1
13	30 194	T PULSE	OUT 12B	TTY DELAY - 4 MSE		147 48	INL	LOAD A TO M
15	24	C.	LDI 194			148 248	LMA	RETURN
16	72	15 0	ST2 1ND			149 7	RET	
19	168	JFZ ST2	XRA	TAPE READER DISAB		150	*	
20	87	LE CODE	OUT 13B	OUTPUT 3, DISAPLE		150	*TTY OUTPUT ROUTINE	
21	85	TAPE READER	OUT 12B	OUTPUT 2, OUTPUT		150	*	
22	38 248	START PULSE	OUT 12B	TTY DATA SAMPLING		150 22 253	TTYOUT LC1 253	C=253
24	70 55 0	LEI 248	COUNTER	TTY DELAY - 8.7 M		152 70 55 0	TTYO CAL TTYDI	DELAY = 9.012 MSE
27	65	TTYIN CAL TTYDI	SEC.			155 16	INC	C=C+1
28	44 255	1NP OB	1NP OB	READ TTY DATA INP		156 72 152 0	JFZ TTYO	
30	85	UT	XRI 255	COMPLEMENT TTY DA		159 168	XRA	TTY START PULSE
31	26	TA	OUT 12B	OUTPUT 2, TTY DAT		160 85	OUT 12B	REG C=248
32	193	A OUT	RAR	STORE TTY DATA		161 22 248	LCI 248	TTY DELAY - 9.012
33	26	LAB	LAB	LOAD TTY DATA TO		163 70 55 0	MSEC.	LOAD DATA TO AC
34	200	REC. B	REC. B			166 193	TTY1 CAL TTYDI	OUTPUT DATA
35	32	RAR	RAR			167 85	LAB	STORE DATA IN CAR
36	72 24 0	LBA	LBA	LOAD AC TO REC. B		168 26	OUT 12B	
37		JFZ TTY1	IS NOT SET	E = E + 1		169 200	RAR	
39	193	LAB	LAB	JUMP IF ZERO F/F		170 6 0	RY	LOAD A TO F
40	36 127	ND1 127	ND1 127	LOAD REG. B TO AC		172 26	LBA	AC = 0
42	200	LBA	LBA	REMOVE PARITY BIT		173 129	LAI 0	RESTORE DATA BIT
43	70 55 0	ATA	ATA	STORE TTY INPUT D		174 200	RAR	RESTORE DATA
46	6	CAL TTYDI	CAL TTYDI			175 16	INC	STORE
48	85	LAI 1	SUPPRESS TTY			176 72 163 0	JFZ TTY1	C=C+1
49	7	OUT 12B				177 70 55 0	ZERO	JUMP IF AC IS NOT
50	192	RET		NOP		179 70 55 0	MSEC.	TTY DELAY - 9.012
51	192	LAA				182 6 1	LAI 1	A=A+1
52	192	LAA				184 85	OUT 12B	SUPPRESS TTY
53	192	LAA				185 7	RET	
54	192	LAA				186	*	
55		*	*TTY DELAY - 8.7 MSEC.			186	*CARRIAGE RETURN & LINE FEED	
55		*				186	*	
55	30 121	TTYDI LDI 121	8.7 MSEC. DELAY	D=D+1		186 14 141	CRLF LBI 215B	CARRIAGE RETURN -
57	24	ST 1ND				188 70 150 0	CR	TYPE CR
58	72 57 0	JFZ ST				191 14 138	CAL TTYOUT	LINE FEED - LF
61	7	RET				193 70 150 0	LF LBI 212B	TYPE LF
62		*	*BCD TO BINARY CONVERSION			196 7	CAL TTYOUT	
62		*				197	RET	
62	199	BCDBIN LAM	LOAD LSD TO A			197	*	
63	20 48	SUI 48	AC=AC-48			197	*ERROR SIGNAL	
65	200	LBA	LOAD A TO B			197	*	
66	49	DCL	L=L-1			197 14 191	ERROR LBI 277B	(?)
67	199	LAM	LOAD M TO A			199 70 150 0	CAL TTYOUT	TYPE (?)
68	20 48	SUI 48	A=A-48			202 7	RET	
70	224	LEA	LOAD A TO E			203	*	
71	104 82 0	BB1 JTZ BB2	IF A=0 JUMP			203	*TYPE B AND IDENTIFY RAM BANK	
74	6 10	LAI 10	AC=10			203	*	
76	129	ADB	AC=AC+B			203	ADRESH CAL CRLF	
77	200	LPA	LOAD AC TO REC. B			206 14 194	LBI 302B	LOAD (B)
78	33	DCE	E=E-1			208 70 150 0	CAL TTYOUT	TYPE (B)
79	68 71 0	JMP BB1	LOAD M TO A			211 70 12 0	CAL TTY	CALL FOR TTY KB I
82	49	BB2 DCL	A=A-48			214 249	INPUT LMB	STORE INPUT IN ME
83	199	LAM	LOAD A TO E			215 7	MORY PET	
84	20 48	SUI 48	AC=AC-48			216	*	
86	224	LEA	LOAD A TO E			216	*TYPE A AND IDENTIFY INITIAL AND FINAL LOCATION	
87	104 98 0	BB3 JTZ BB4	AC=100			216 70 186 0	ADRESL CAL CRLF	
90	6 100	LAI 100	AC=AC+B			219 14 193	LBI 301B	LOAD (A)
92	129	ADB	LOAD AC TO REG. B			221 70 150 0	CAL TTYOUT	TYPE (A)
94	33	LBA	E=F-1			224 70 186 0	ADI CAL CRLF	C=253
95	68 87 0	JMP BB3	LOAD M TO A			227 22 253	LC1 253	CALL FOR TTY KB I
98	7	BB4 RET	A=A-1			229 70 12 0	AD2 CAL TTY	
99		*	*BINN BCD LHI 11			232 48	INL LMB	L=L+1
99	46 11	LLI 241	LOAD M TO A			233 249	TO M	LOAD TTY KR INPUT
101	54 241	BNBD LC1 0	A=A-48			234 16	INC	C=C+1
103	22 0		LOAD A TO E			235 72 229 0	JFZ AD2	JUMP IF C IS NOT
105	193					238 7	ZERO	
106	20 100	BD1 SUI 100	AC=AC-100			239	*	
108	96 115 0	JTC BD2	JUMP IF AC<100			239 70 9 0	DATA1N CAL TAPE	READ TAPE
111	16	INC	C=C+1			242 6 66	LAI 102B	LOAD (B)
112	68 106 0	JMP BD1	LOAD 100 TO REC.			244 185	CPB	SEARCH FOR (B)
115	14 100	BD2 LBI 100	AC=AC-10			245 72 239 0	JFZ DATA1N	JUMP IF IT IS NOT
117	129	B	AC=AC+B			(B) 246 46 11	LAI 11	H=11
118	200	ADB	LOAD AC TO REG. B			250 54 255	LLI 255	L=255
119	6 48	LAI 48	A=A+48			252 6 248	LAI 248	DATA BIT COUNTER
121	130	ADC	A=A+C			254 248	LMA	STORE DATA BIT CO
122	228	LMA	LOAD A TO MEMORY					
123	22 0	LCI 0	CLEAR REC. C					
125	193	LAB	LOAD B TO A					
126	20 10	BD3 SUI 10	AC=AC-10					
128	96 135 0	JTC BD4	JUMP IF AC<10			260 6 80	LAI 120B	
						262 185	CPB	
						266 6 78	JTZ PDATA	
						268 185	CPB	
						269 104 49 1	LAI 116B	
						272 6 66	CPB	
						274 185	JTZ NDATA	
						275 104 248 0	LAI 102B	
						278 6 127	CPB	
							T INSTRUCTION	
							LAI 177B	
							LOAD (R0)	

280 185		CPB	SEARCH FOR RUEOUT	436 7	IN M	
281 72 34 1	T	JFZ FMEROR	JUMP IF NOT RUEOUT	437	*	RET
284 70 90 1		CAL RUEOUT	CALL FOR RUEOUT R	437	*SET ADDRESS TO 1101 RAM	
287 68 255 0	OUTINE	JMP DATA2	CALL FOR FORMAT E	437	*	
290 70 98 1	FMEROR CAL FORMAT	ROR ROUTINE	CALL FOR FORMAT E	437	SETMA LHI 11	H=11
293 68 89 1	JMP DATAEN	RROR ROUTINE	REPLACE (P) WITH	437	LLI 252	L=252
296 6 1	PDATA LAI 1	(1)	ROTATE RIGHT	437 46 11	LDM	BANK NO TO D
298 26	RAR		ROTATE LEFT	439 54 252	INL	L=L+1=253
299 199	LAM		ROTATE LEFT	441 223	LAM	INIT ADR TO E
300 18	RAL		ROTATE LEFT	442 48	OUT 10B	WRITE ADDRESS TO
301 248	LMA		ROTATE LEFT	443 199	LLA	LOAD AC TO L
302 68 53 1	JMP DATA3		ROTATE LEFT	444 81	LHD	D TO H = BANK NO
305 168	NDATA XRA Y		CLEAR AC AND CARR	445 240	RET	
306 199	LAM		CLEAR AC AND CARR	446 235		
307 18	RAL		ROTATE LEFT	447 7		
308 248	LMA		ROTATE LEFT	448	*	
309 54 255	DATA3 LLI 255		ROTATE LEFT	448 46 11	*ADDRESS CHECKING	
311 207	LBM		ROTATE LEFT	450 54 254	ACHECK LHI 11	H=11
312 8	INB		ROTATE LEFT	452 199	LLI 254	L=254
313 249	LMB		LOAD M TO B	453 49	LAM	LOAD FINAL ADRES.
314 72 255 0	JFZ DATA2		INC DATA BIT COUN	454 191	TO AC	
317 70 9 0	ZERO		JUMP IF B IS NOT	455 104 205 1	DCL	L=EL-1=253
320 6 70	FDATA CAL TAPE		CALL FOR TAPE INP	456 215	CPM	COMPARE:AF-AI
322 185	UT		CALL FOR TAPE INP	459 16	JTZ CHECK	JUMP IF AF-AI=0
323 104 88 1	JTZ DATA4		CALL FOR TAPE INP	460 250	LCM	LOAD AI TO AC
326 6 66	IS (F)		CALL FOR TAPE INP	461 7	INC	AI=AI+1
328 185	LAI 102B		CALL FOR TAPE INP	462	LMC	LOAD AI TO MEMORY
329 104 248 0	JTZ DATA1		CALL FOR TAPE INP	462 70 186 0	CHECK RET	
332 6 127	UCTION IF IT IS (B)		CALL FOR TAPE INP	465 14 170	*	
334 185	LAI 177B		CALL FOR TAPE INP	467 70 150 0	*PROGRAM BEGINS	
335 72 34 1	JFZ FMEROR		CALL FOR TAPE INP	470 70 12 0	START CAL CRLF	
338 70 90 1	TINE		CALL FOR TAPE INP	473 6 84	LBI 252B	
341 68 255 0	JMP DATA2		CALL FOR TAPE INP	475 185	CAL TTYOUT	
344 168	DATA4 XRA Y		CALL FOR TAPE INP	476 104 3 2	CAL TTY	
345 7	DATAEN RET		CLEAR AC AND CARR	479 6 69	INPUT	
346 *			CLEAR AC AND CARR	481 185	LAI 124B	LOAD (T) TO AC
346 *			CLEAR AC AND CARR	482 104 31 2	CPB	AC-E
346 *			CLEAR AC AND CARR	485 6 82	JTZ TAPEIN	JUMP IF AC-B=0
346 192	RUBOUT LAA		CLEAR AC AND CARR	486 104 6 2	LAI 105B	AC=105B, (E)
347 192	LAA		CLEAR AC AND CARR	487 185	CPB	AC-B
348 192	LAA		CLEAR AC AND CARR	491 6 67	JTZ EXECUT	JUMP IF AC-B=0
349 192	LAA		CLEAR AC AND CARR	493 185	LAI 122B	AC=122B, (R)
350 192	LAA		CLEAR AC AND CARR	494 104 77 2	CPB	AC-B
351 192	LAA		CLEAR AC AND CARR	497 6 76	JTZ READIN	JUMP IF AC-B=0
352 192	LAA		CLEAR AC AND CARR	499 185	LAI 103B	AC=103B, (C)
353 7	RET		CLEAR AC AND CARR	500 104 94 2	CPB	AC-B
354 *			CLEAR AC AND CARR	503 6 80	JTZ LISTIN	JUMP IF AC-B=0
354 *			CLEAR AC AND CARR	505 185	LAI 120B	AC=120B, (P)
354 *			CLEAR AC AND CARR	506 104 181 2	CPB	AC-B
354 *			CLEAR AC AND CARR	509 70 197 0	JTZ PROGRAM	JUMP IF AC-B=0
354 *			CLEAR AC AND CARR	512 68 206 1	CAL ERROR	TYPE (?)
354 *			CLEAR AC AND CARR	515	JMP START	
354 *			CLEAR AC AND CARR	515	*	
354 *			CLEAR AC AND CARR	515	*LOAD DATA INPUT TO 1101 RAM	
354 *			CLEAR AC AND CARR	515	*	
354 *			CLEAR AC AND CARR	516 70 141 1	TAPEIN CAL ENTRA	ENTER ADDRESS
354 *			CLEAR AC AND CARR	516 70 239 0	READIN CAL DATAIN	READ TAPE INPUT R
354 14 160	FORMAT LBI 240B		LOAD (SP)	521 26	OUTLINE	
356 70 150 0	CAL TTYOUT		TYPE (SP)	522 96 206 1	RAR	CHECK FOR FE FLAG
359 14 198	LBI 306B		LOAD (F)	525 54 250	JTC START	JUMP IF CARRY=1
361 70 150 0	CAL TTYOUT		TYPE (F)	527 215	LLI 250	L=250
364 14 197	LBI 305B		LOAD (E)	528 70 181 1	LCM	LOAD MEMORY TO C
366 70 150 0	CAL TTYOUT		TYPE (E)	531 194	CAL SETMA	SET MEMORY ADDRES
369 70 186 0	LISTA CAL CRLF		NOP	532 83		
372 54 253	PRINTA LLI 253		NOP	533 246		
374 207	LBM		NOP	534 70 192 1	RY	LOAD DATA TO MEMO
375 70 99 0	CAL BINBCD		NOP	537 104 206 1	CAL ACHECK	COMPARE AF AND AI
378 38 253	LE1 253		NOP	540 6 6 2	JTZ START	JUMP IF A=0
380 49	E=253		NOP	543 46 11	JMP READIN	READ INPUT DATA
381 49	DCL		NOP	545 54 240	EXECUT LHI 11	H=11
382 199	FM1 LAM		NOP	547	LLI 240	L=240
383 4 128	ADI 128		NOP	549	BANK0 EQU 4000B	BANK 0 LOCATION
385 200	LBA		NOP	551	BANK1 EQU 4400B	BANK 1 LOCATION
386 70 150 0	CAL TTYOUT		NOP	552	BANK2 EQU 5000B	BANK 2 LOCATION
389 48	INL		NOP	554	BANK3 EQU 5400B	BANK 3 LOCATION
390 32	INE		NOP	557 189	CAL ADRESH	ENTER BANK NO
391 72 126 1	JFZ FM1		NOP	558 199	CAL CRLF	
394 6 1	0		NOP	559 20 48	LAM	LOAD MEMORY TO AC
396 7	LAI 1		NOP	560 4 8	SUI 48	AC=AC-48
397 *	RET		NOP	562 189	ADI 8	AC=AC+8
397 *			NOP	565 6 9	LHA	LOAD AC TO H
397 46 11	ENTERA LHI 11		NOP	566 104 0 8	CPH	AC=8
399 54 240	LLI 240		NOP	567 189	JTZ BANK0	AC=AC-H
401 70 203 0	ENTERH CAL ADRESH		NOP	568 104 0 9	LAI 9	JUMP IF AC=0
404 70 216 0	ENTERL CAL ADREL		NOP	571 6 10	CPH	
407 70 224 0	RESS		NOP	573 189	JTZ BANK1	
410 70 186 0	CAL ADI		NOP	574 104 0 10	LAI 10	
413 54 246	S5		NOP	577 6 11	CPH	
415 70 62 0	CAL CRLF		NOP	578 232	JTZ BANK2	
418 209	LLI 246		NOP	580 104 0 11	LAI 11	
419 49	CAL BCDBIN		NOP	583 70 197 0	CPH	
420 70 62 0	Y		NOP	586 68 206 1	JMP START	
423 49	ARY		NOP	589 46 11	CONTIN LHI 11	
424 199	DCL		NOP	591 54 252	LLI 252	
425 20 48	LAM		NOP	593 223	LDM	
427 4 8	SUI 48		NOP	594 24	IND	D=D+1
429 54 252	ADI 8		NOP	595 251	LMD	BANK=BANK+1
431 248	LLI 252		NOP	596 48	INL	L=L+1
432 48	LMA		NOP	597 168	XRA	CLEAR AC
433 249	M		NOP	598 248	LMA	INITIAL ADRES=0
434 48	INL		NOP	599 48	INL	
435 250	LMB		NOP	600 6 255	LAI 255	
435 250	ES IN M		NOP	602 248	LMA	
436 48	INL		NOP	603 68 6 2	JMP READIN	FINAL ADRES=255
437 250	LMC		NOP			

				R=4	
606	*	*PROM LISTINC ROUTINE'		690	JMP LIST1
606	*			693	*
606				693	*PROM PROGRAMMER
606 46 11	LISTIN	LHI 11	H=11	693	*
608 S4 240		LLI 240	L=240	693	PHOCRM CAL ENTRA
610 70 148 1		CAL ENTERL	ENTER INITIAL & F	ESS	ENTER MEMORY ADDR
	1NAL ADR.			PC1	REPPOCRAM CONTR.
613 70 186 0	LISTER	CAL CRLF	L=2S1	696	AC=253
616 54 251		LLI 251	NO. OF INSTR. PER	698	LOAD AC TO MEMORY
618 6 2S2		LAI 252		700	CAERIACF RETURN
	LINE			701	
620 248		LMA	LOAD AC TO MEMORY	703	CAL TTYOUT
621 70 116 1	LIST1	CAL PRINTA	PRINT ADDRESS	706	CAL SETMA
624 14 160		LLI 240B	LOAD [SP]	706	SET ADDRESS TO 17
626 70 150 0		CAL TTYOUT	PRINT [SP]	709	COMPLEMENT INPUT
629 14 194		LLI 302B	LOAD [B]	711	LOAD DATA TO AC
631 70 150 0		CAL TTYOUT	PRINT [B]	712	WRITE DATA TO OUT
634 S4 2S3		LLI 253	L=253		
636 199		LAM	LOAD AI TO AC	713	AC=4, DELAY
637 81		OUT 10B	OUTPUT AI TO OUT	715	PROGCRAM PULSE ENA
	0	LEI 248	READ DELAY/DATA B	BLE	
638 38 248	IT CONTR	INP 1B	READ INPUT FROM 1	716	E=197, DELAY = 52
640 67		702		716	
641 18	LIST2	RAL	L=249	718	0 MSEC.
642 S4 249		LLI 249	SAVE INPUT DATA	718	PC4 CAL TTYDI
644 248		LMA	PRINT [P] IF CARR	721	C*
645 96 144 2		JTC PRINTP		722	INE
	Y=1		LOAD [N]	722	JFZ PC4
648 14 206		LLI 316B	PRINT [N]	725	AC=0
650 70 150 0		CAL TTYOUT		727	DISABLE PROGRAM P
652 68 149 2		JMP LIST3		728	DELAY APPROX. 9
656 14 208	PRINTP	LLI 320B	LOAD [P]	729	READ DATA FROM 17
658 70 150 0		CAL TTYOUT	PRINT [P]	730	COMPARE DATA
661 199	LIST3	LAM	LOAD DATA TO AC	731	JUMP IF COMPARED
662 32		INE	E=E+1	734	LOAD [\$]
663 72 129 2		JFZ LIST2	JUMP IF E IS NOT	734	PRIN[\$]
	0		LOAD [F]	736	
666 14 198		LLI 306B	PRINT [F]	739	
668 70 150 0		CAL TTYOUT	LOAD [SP]	741	
671 14 160		LLI 240B	PRINT [SP]	743	
673 70 150 0		CAL TTYOUT	AF - A1	744	
676 70 192 1		CAL ACHECK	LOAD LINE CONTR.	745	
679 104 206 1		JTZ START	LOAD MEMORY TO C	746	
682 54 2S1	TO AC	LLI 2S1	C=C+1	749	
			JUMP IF LINE CONT	752	
684 215		LCM		755	
685 16		INC		758	
686 2S0		LMC		761	
687 104 101 2		JTZ LISTER		764	CONTINUE PROC. NE
				767	XT INSTR.
					END

intellecTM 8

Bare Bones 8 and Microcomputer Modules

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 12.5 μ s.

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
-------------------------	-----------------------	--------------------

- 9k bytes of Memory (expandable to 16,384 bytes – Intellec 8)

- 5k bytes of Memory (expandable to 16,384 bytes – Bare Bones 8)

- Direct Access to Memory and I/O

- Four 8-bit input ports (expandable to eight)

- Four 8-bit output ports (expandable to twenty-four)

- Universal Asynchronous Transmitter Receiver for serial communications interface

- Real time interrupt capability

- Crystal controlled master system clock

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 imm6-28 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 imm6-26 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 imm6-26 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.

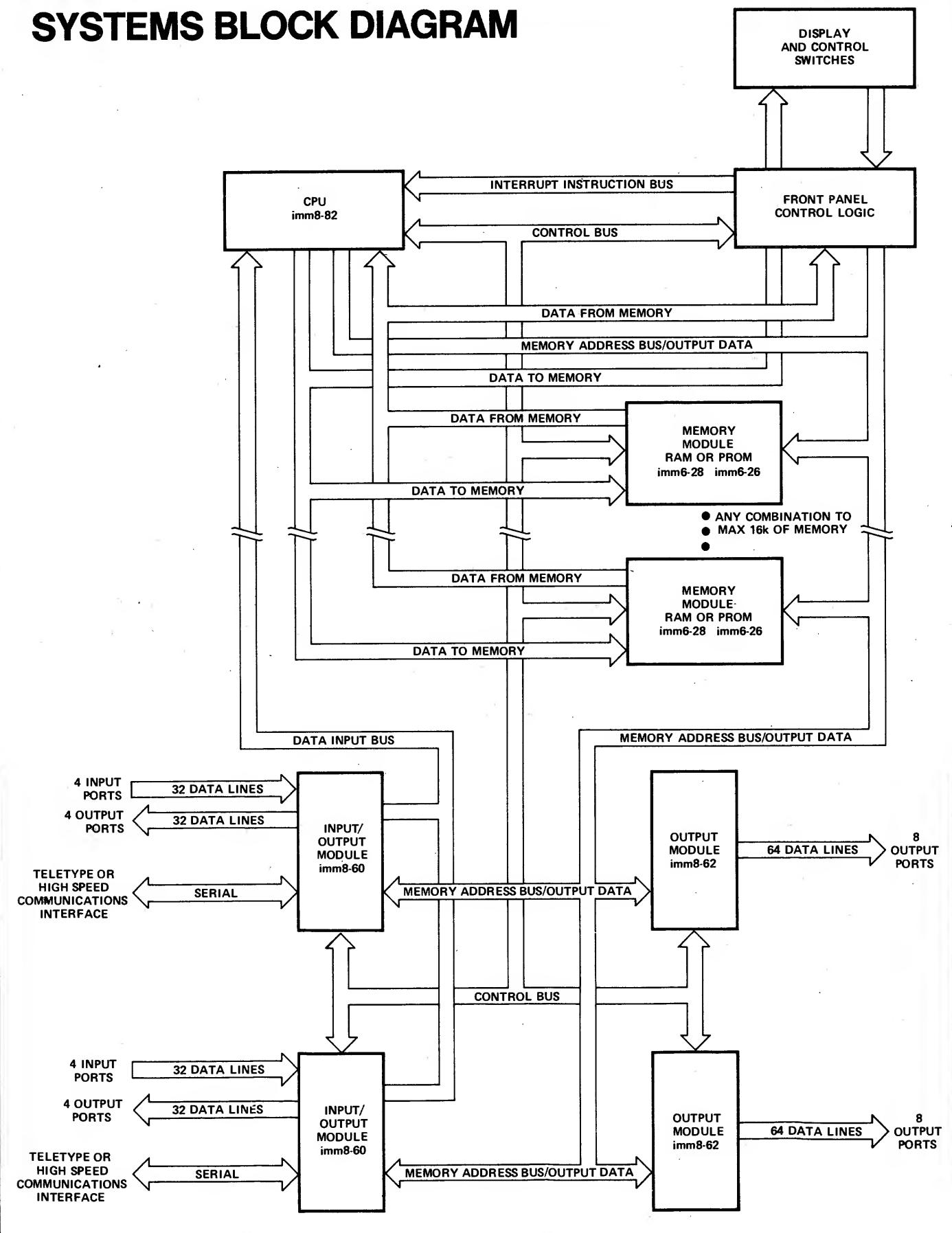
Every Intellec 8 comes with three basic pieces of software, the systems monitor, a resident program located in the upper 1280 bytes of memory, a symbolic assembler and a text editor. The resident systems monitor allows the operator to punch and load tapes, display and alter memory, and execute programs.

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.

SYSTEMS BLOCK DIAGRAM



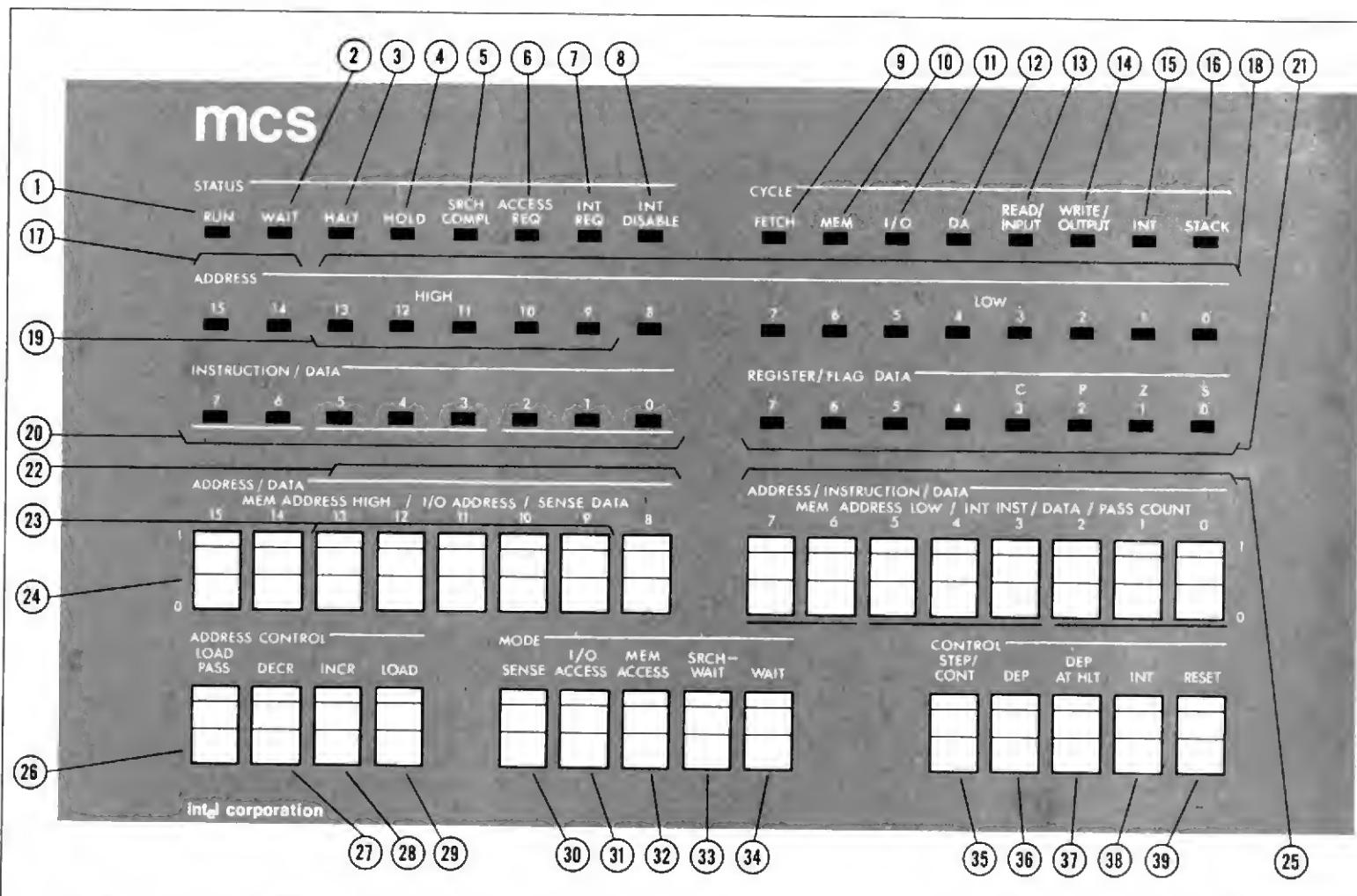
INTELLEC 8 CONTROL CONSOLE AND DISPLAY

The Control Console directs and monitors all activities of the Intellic 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.

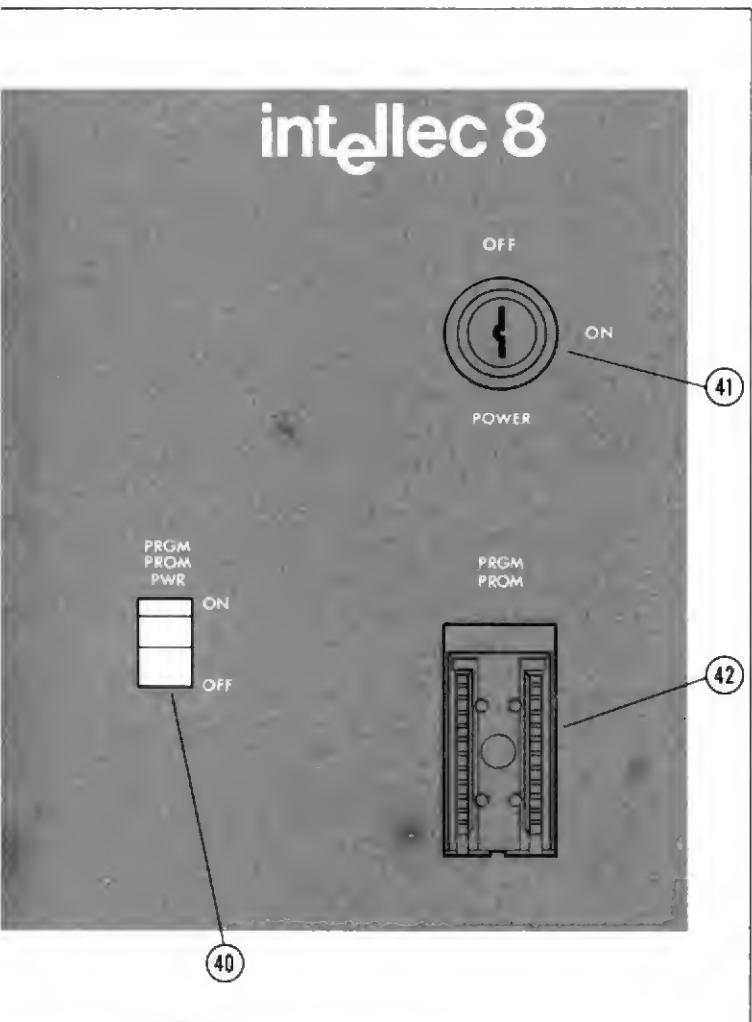
■ **STATUS** is a display of the operating mode of the processor.

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 COMPL 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. ACCESS REQ 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 for deposit 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 HALT (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 Intellic 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.

TEXT EDITOR

- Edits symbolic data from paper tape with data from operator's terminal
- Edited output is available via paper tape
- Appends text to editor input buffer
- Moves pointer to any desired location
- Finds and inserts or substitutes strings
- Deletes lines selectively

The Text Editor allows the operator to edit his source code, making corrections and additions. He may append code, delete code, locate strings, insert strings, substitute strings and output edited code via paper tape. The text editor runs on a minimum Intellic 8 system with teletype I/O. (Requires a minimum of 8k x 8 of RAM.)

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 Intellic 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 time-sharing services for access to the programs. The output from both PL/M and the MCS-8 Assembler may be run directly on the Intellic 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	Weight:	30 lb.
Memory Size:	9k bytes Intellic 8/5k bytes Bare Bones expandable to 16k bytes	Standard Software:	System Monitor Resident Assembler Text Editor
Instruction Set:	48, including: conditional branching, binary arithmetic, logical, register-to- register and memory reference operations	Support Software:	PL/M Compiler Cross Assembler Simulator
Machine Cycle Time:	12.5 μ s		written in FORTRAN IV
System Clock:	Crystal controlled at 800kHz $\pm 0.01\%$		
I/O Channels:	4 expandable to 8 input ports TTL 4 expandable to 24 output ports Compatible		
Interrupt:	Single Level		
Direct Access to Memory:	Standard via control console		
Memory Cycle Time:	1 μ s		
Operating Temperature:	0°C to 55°C		
DC Power Supplies:	$V_{CC} = 5V$, $I_{CC} = 12A^*$ (standard Intellic 8) $V_{DD} = -9V$, $I_{DD} = 1.8A^*$ $V_{GG} = -12V$, $I_{GG} = 0.06A$		
DC Power Requirement:	$V_{CC} = 5V \pm 5\%$, $I_{CC} = 11A$ max., .6A typ. $V_{DD} = -9 \pm 5\%$, $I_{DD} = 1A$ max., 0.5A typ. $V_{GG} = -12V \pm 5\%$, $I_{GG} = 0.03A$ max., 0.016A typ.		
AC Power Requirement: (standard Intellic 8)	60Hz, 115 VAC, 200 Watts *Larger power supplies may be required for expanded systems.		
Physical Size:	Intellic 8: 7" x 17 1/8" x 12 1/4" (table top only) Bare Bones 8: 6 3/4" x 17" x 12" (suitable for mounting in standard REMTA 7" x 19" panel space)		

STANDARD SYSTEMS and OPTIONAL MODULES

Intellic 8 (imm8-80A) Standard System includes the following
Modules and Accessories:

- Central Processor Module
- Input/Output Module
- PROM Memory Module
- RAM Memory Modules (Two)
- Chassis with Mother Board
- Power Supplies
- Control and Display Panel
- Finished Cabinet
- Standard Software:
System Monitor
Resident Assembler
Text Editor
- PROM Programming Module

Bare Bones 8 (imm8-81) Standard System includes the following
Modules:

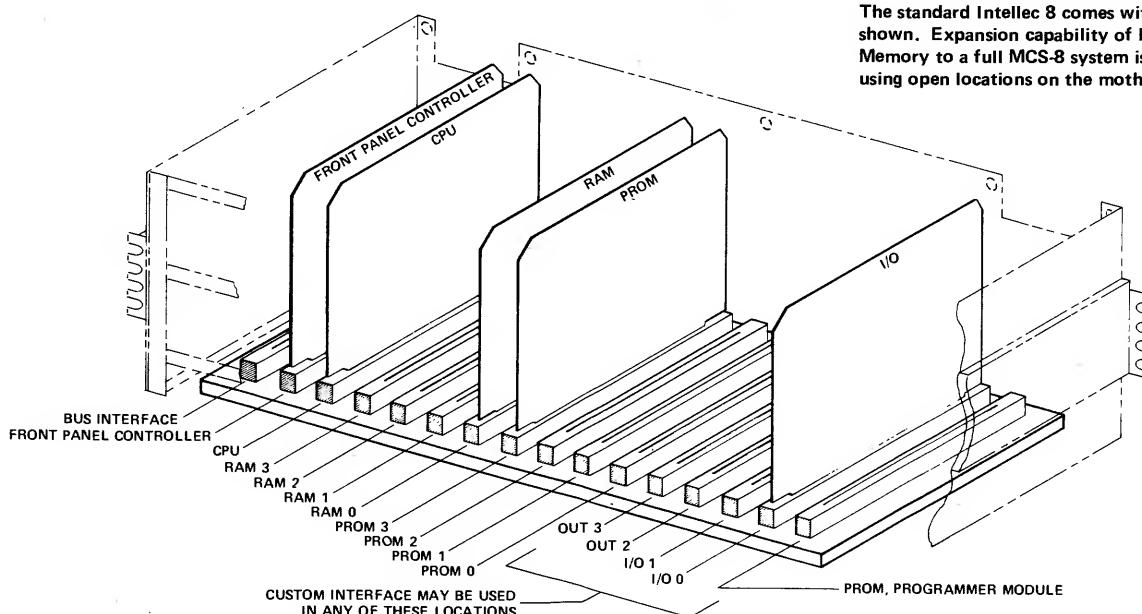
- Central Processor Module
- Input/Output Module
- PROM Memory Module
- RAM Memory Module
- Chassis (rack mountable
with Mother Board)
- Standard Software:
System Monitor
Resident Assembler *
Text Editor *

*Requires a minimum of
8k of RAM

Optional Modules available for the Intellic 8 and Bare Bones 8:

- Additional I/O or Output Modules
- Additional RAM Memory Modules
- Universal Prototype Module
- Module Extender
- Rack mounting kit for Intellic 8

The standard Intellic 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.



Intellic 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
- 12.5 μ s 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.

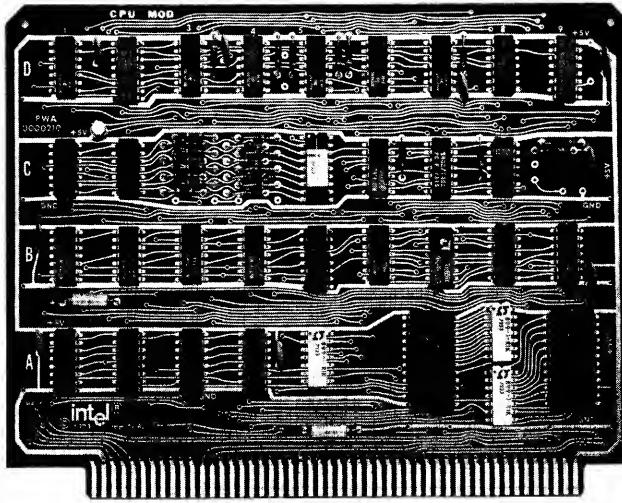
The imm8-82 has a fourteen bit TTL compatible memory address bus, an 8-bit data output bus and an 8-bit memory data input bus. Memory read and write signals and the wait request signal provide interface at TTL levels to any type of memory (including PROM, ROM, and RAM). Asynchronous interface to slower speed memories (access > 1 μ s) 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 12.5 μ s ± 0.01% and provides an accurate timing source for software delay loops and other timing requirements.

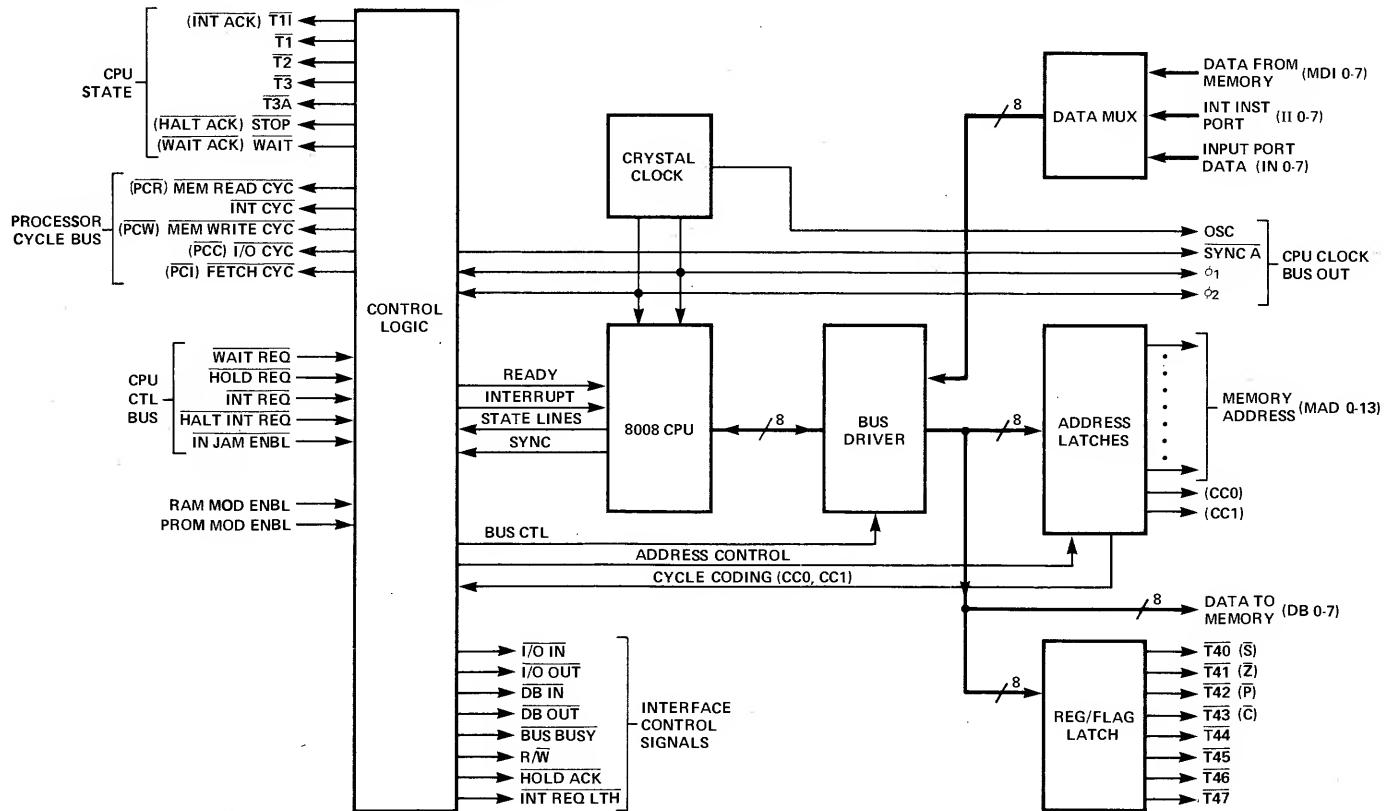


Central Processor Module

Central Processor Module Specifications

Word Size:	Instruction: 8, 16, or 24 bits Data: 8 bits	System Clock:	Crystal controlled, 800kHz ± 0.01% Processor cycle time: 12.5µs
Central Processor:	8008-1 CPU, 8 bit accumulator, six 8-bit registers, subroutine nesting to seven levels, interrupt capability, asynchronous operation with memory	Connector:	Dual 50-pin on 0.125 in. centers. Connectors in rack must be positioned on 0.5 in. centers min.
Instruction Set:	48 including conditional branching, binary arithmetic, logical operations, register-to-register transfers, and I/O	Board Dimensions:	Wirewrap P/N C800100 from SAE P/N VPB01C50E00A1 from CDC
Memory Addressing:	Any combination of PROM, ROM and RAM up to 16,384 bytes	Operating Temp :	6.18 in. x 8.0 in. x 0.062 in. Board to be on 0.5 in. centers minimum 0°C to +55°C
Memory Interface:	Address: 14-bits TTL latching bus Data: 8-bit TTL bus to and from memory	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
I/O Addressing:	Input: Eight 8-bit input ports Output: twenty-four 8-bit latching output ports	Support Software:	PL/M Compiler Cross Assembler Simulator
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)		Written in FORTRAN IV

imm8-82 Block Diagram

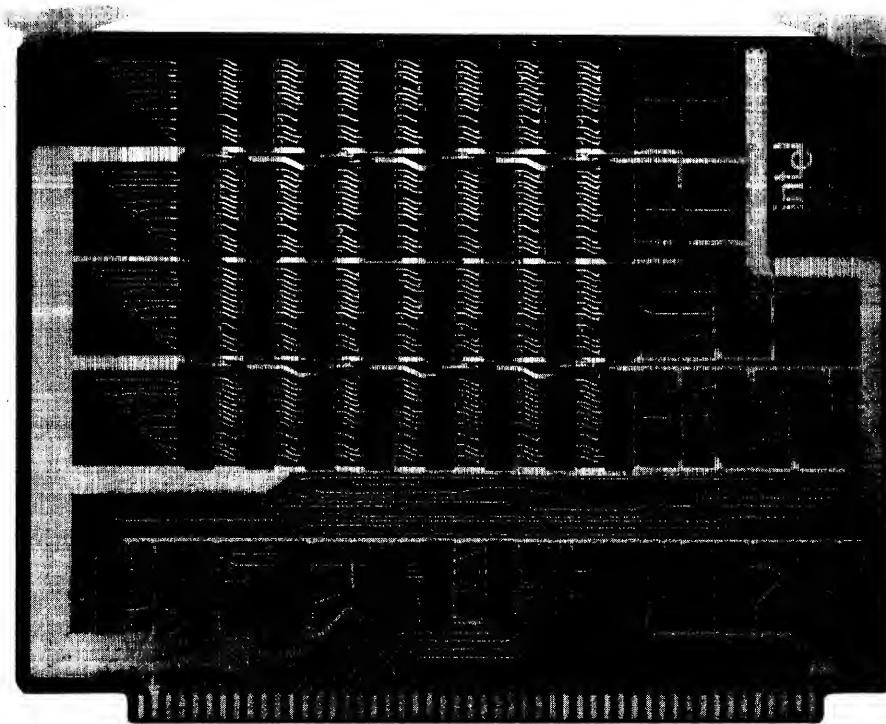


imm6-28 RAM MEMORY MODULE

- 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.

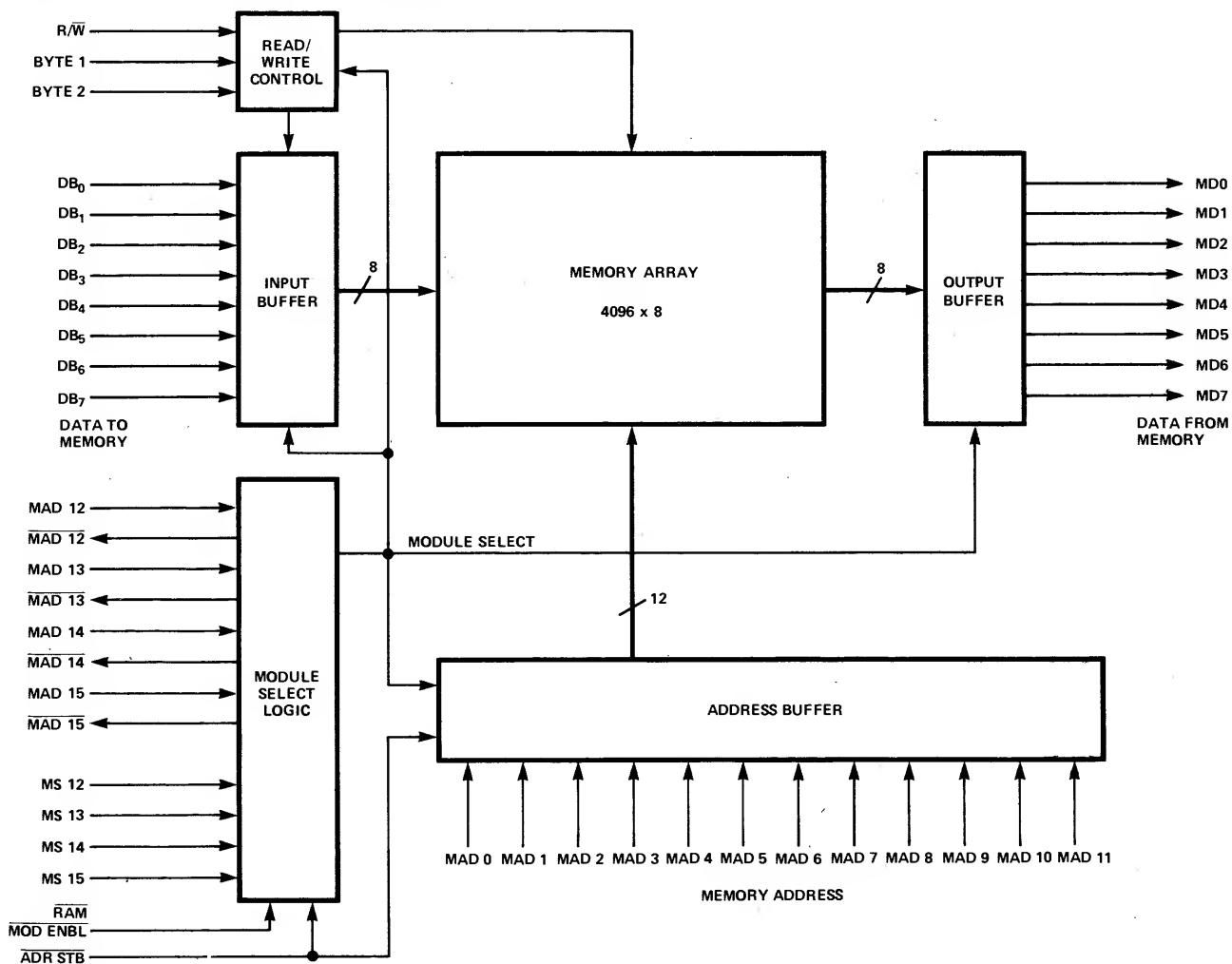


RAM Memory Module

RAM Memory Module Specifications

Memory Size: 4k bytes
Word Size: 8 bits
Memory Expansion: To 65k bytes (16 modules)
Cycle Time: 1 μ s
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 VPB01C50E00A1 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 ± 5%, I_{CC} = 2.5A max., 1.25A typical

imm6-28 Block Diagram



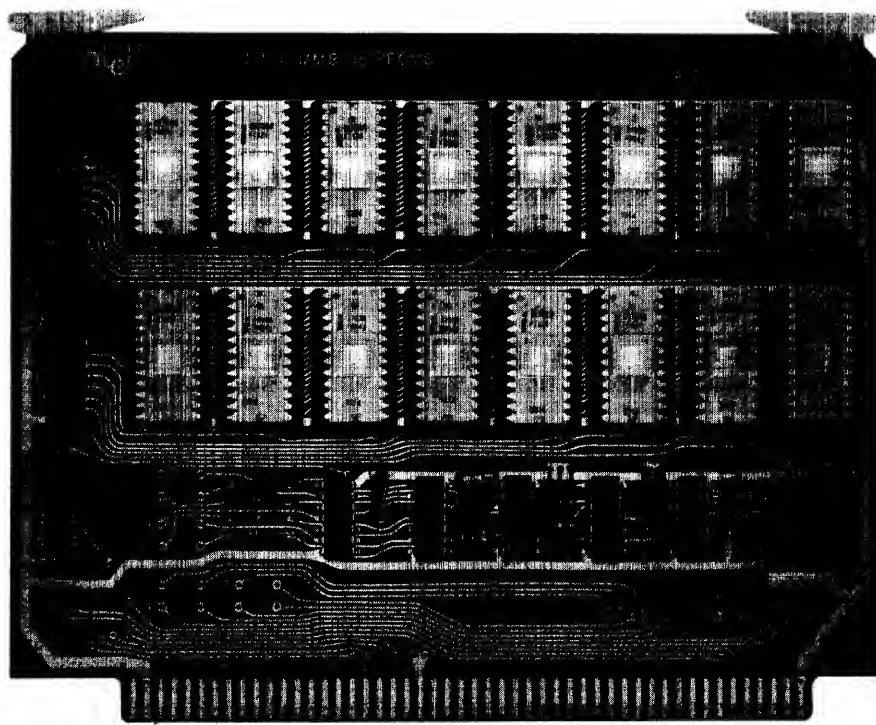
imm6-26 PROM MEMORY MODULE

- 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.



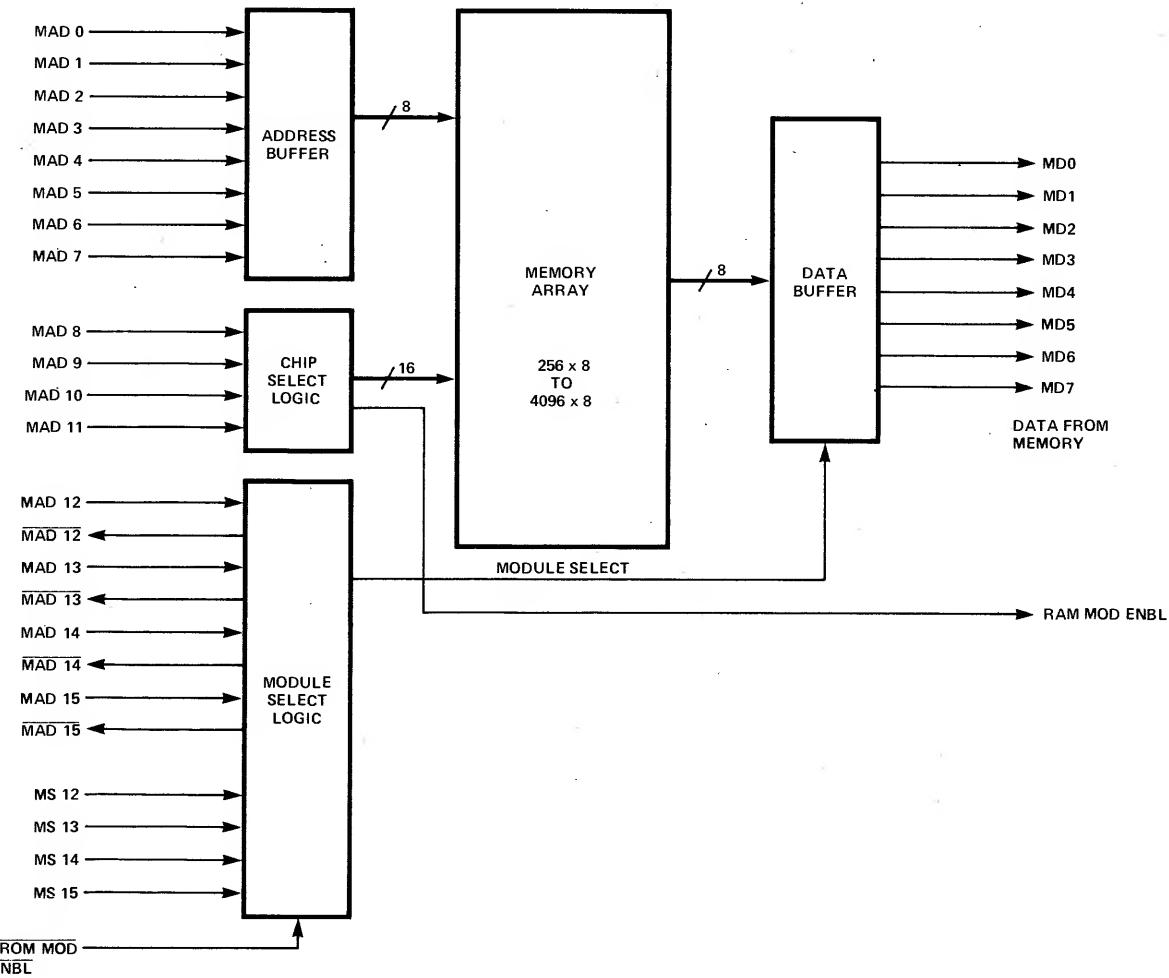
PROM Memory Module

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 VPB01C50E00A1 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:	$V_{CC} = +5V \pm 5\%$ $I_{CC} = 1.6A \text{ max., } 1.1A \text{ typical}^{(1)}$ $V_{DD} = -9V \pm 5\%$ $I_{DD} = 1.6A \text{ max., } 1.0A \text{ typical}^{(1)}$

(1)Board loaded with all 16 PROMs.

imm6-26 Block Diagram



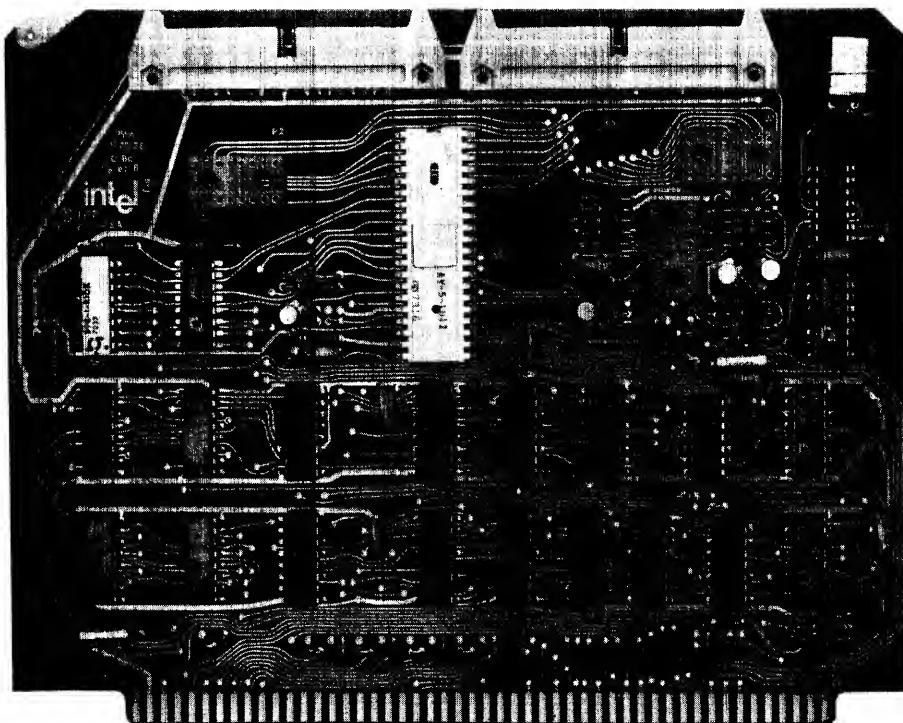
imm8-60 INPUT/OUTPUT MODULE

- 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.

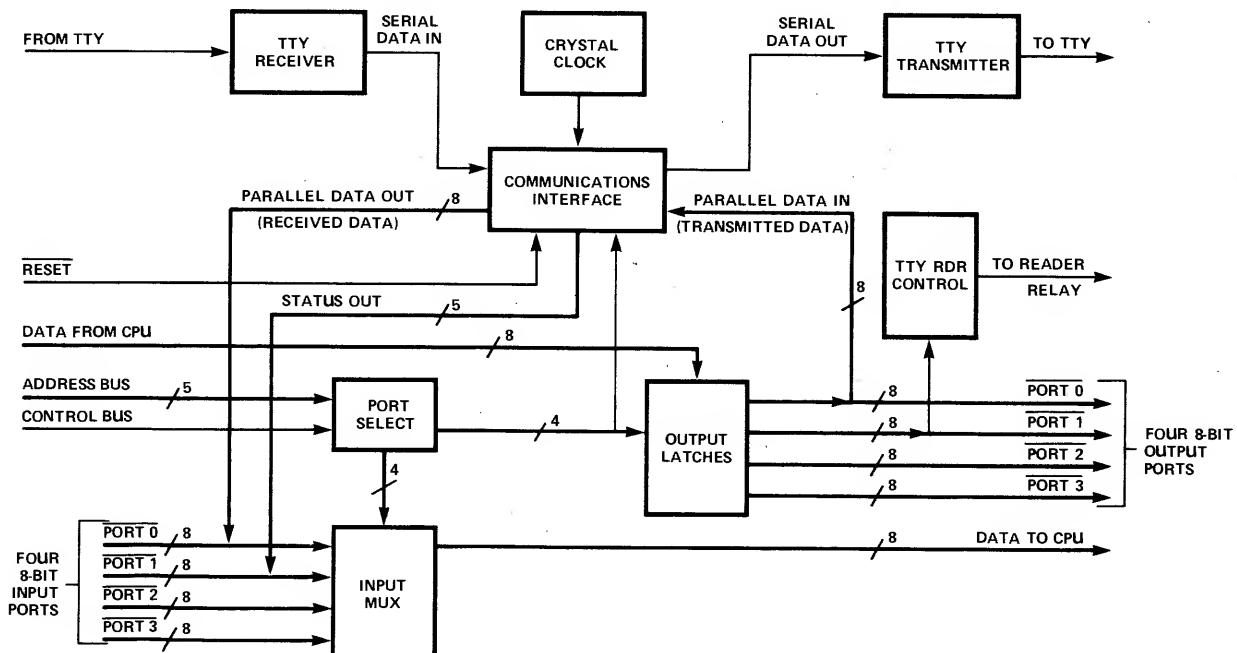


I/O Module

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 VPB01C50E00A1 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: $V_{CC} = +5V \pm 5\%$, $I_{CC} = 0.820A$ max., $0.478A$ Typical
 $V_{DD} = -9V \pm 5\%$, $I_{DD} = 0.080A$ max., $0.050A$ Typical
 $V_{GG} = -12V \pm 5\%$, $I_{GG} = 0.030A$ max., $0.016A$ Typical

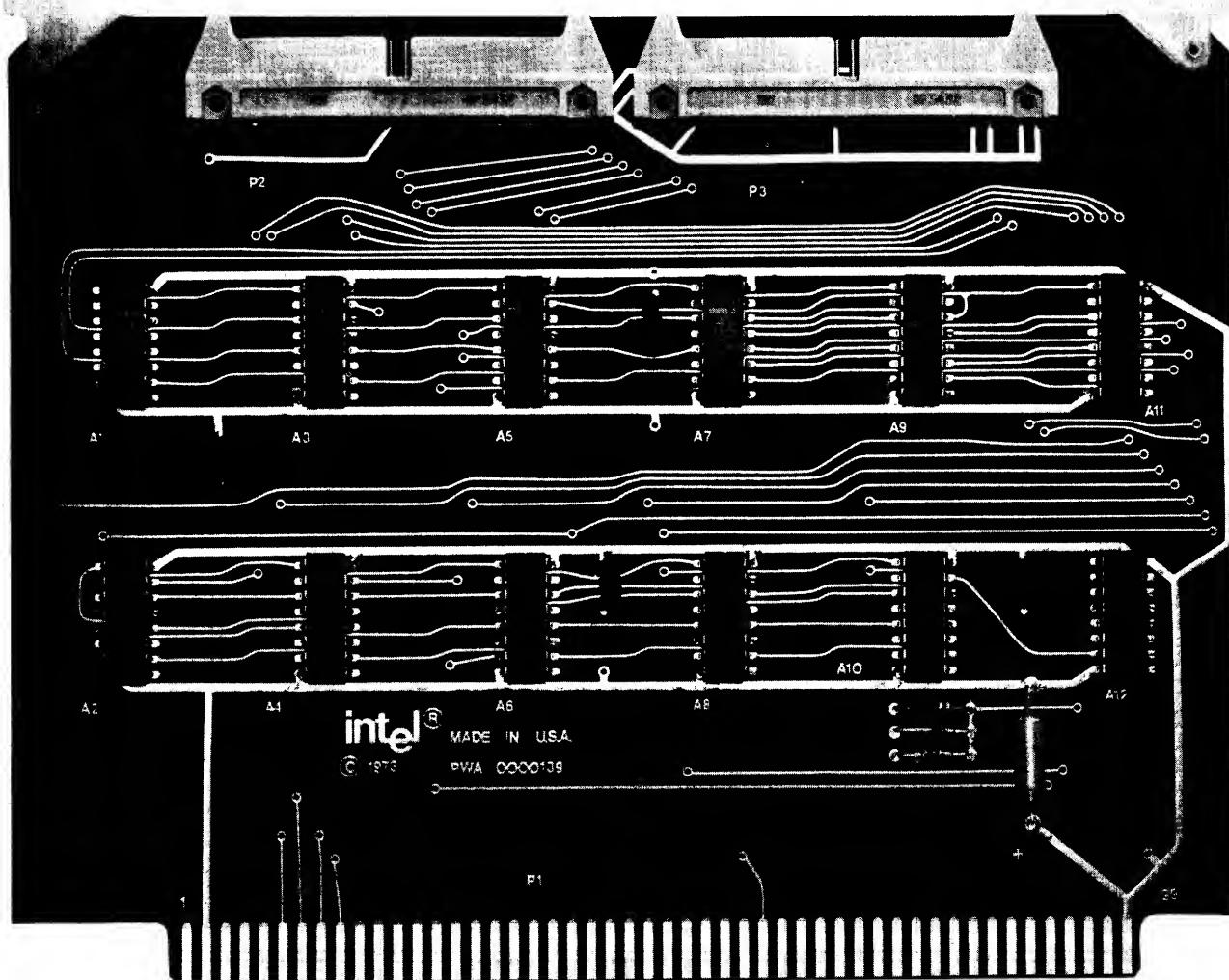
imm 8-60 Block Diagram



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.

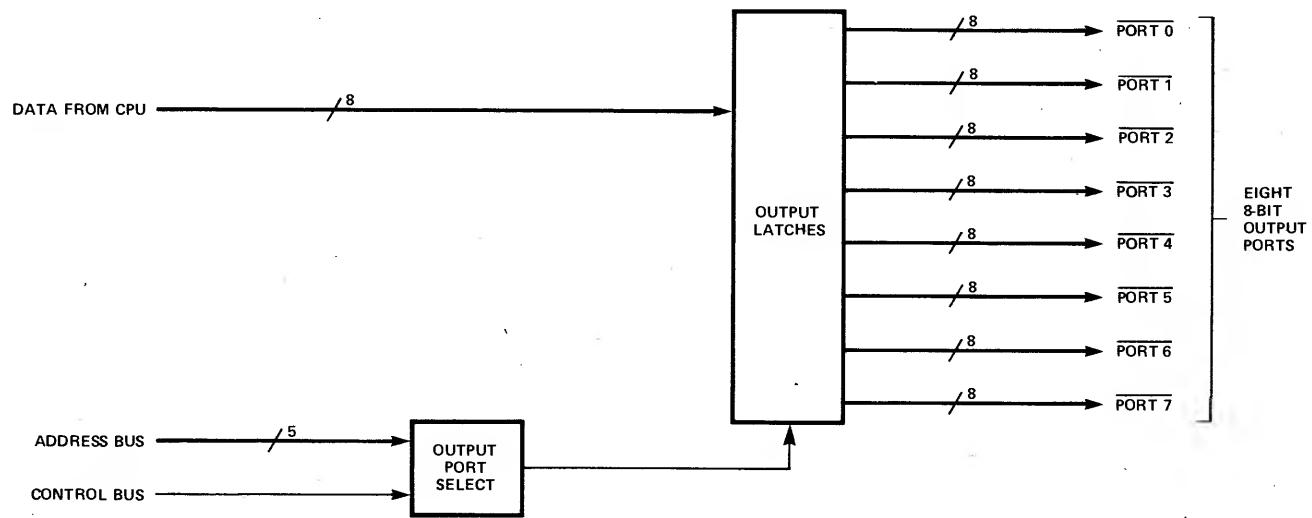


Output Module

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 VPB01C50E00A1 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 ± 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

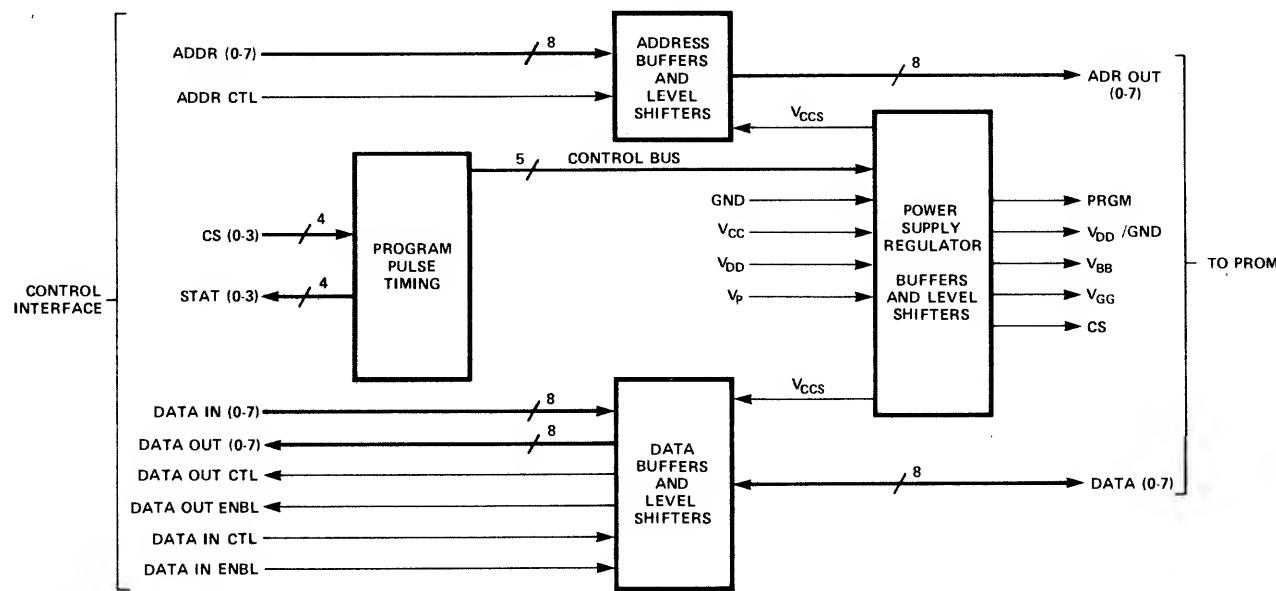
The imm6-76 PROM Programmer Module provides all necessary hardware and software to add PROM programming capability to the Intellec 8 microcomputer development system.

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

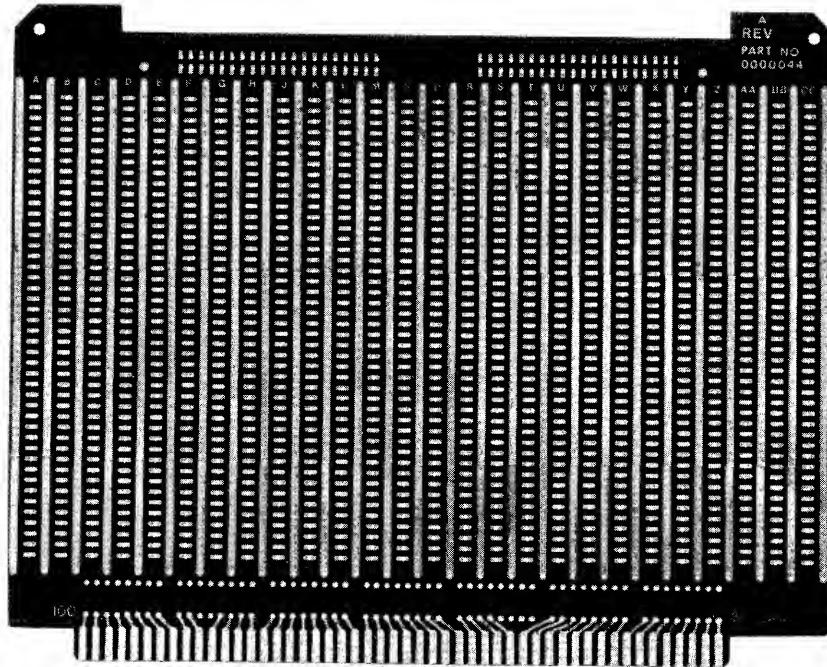
System Interface:	All inputs and outputs are TTL compatible and available at the ribbon connector at the top of the module. Control for either "True" or "False" data is provided. Direct interface to Intellec 8.
Control Software:	Included in the Intellec 8 executive monitor.
Connector:	Dual 50-pin on 0.125 in. centers. Connectors in rack must be positioned on 0.5 in. centers min. Wirewrap P/N C800100 from SAE P/N VPB01C50E00A1 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 ± 5%, I _{CC} = 0.8A max., 0.5A typical V _{DD} = -9V ± 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 Intellic 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

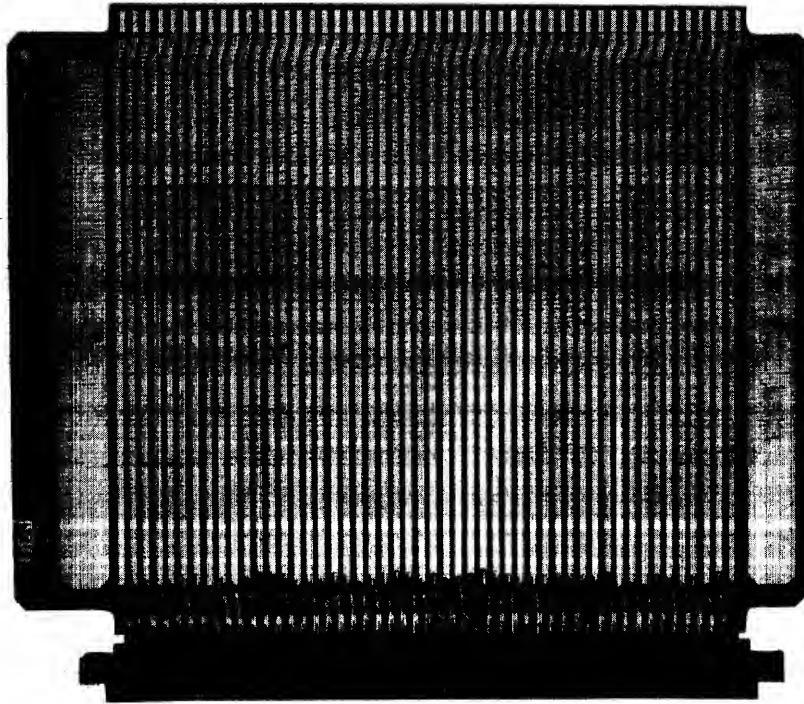
Universal Prototype Module Specifications

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 VPB01C50E00A1 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

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

3065 Bowers Avenue
408/246-7501, TWX: 910-338-0026
Telex: 34-6372
*Santa Clara, California 95051

NATIONAL SALES MANAGER

Hank O'Hara
3065 Bowers Avenue
408/246-7501, TWX: 910-338-0026
Telex: 34-6372
*Santa Clara, California 95051

U.S. REGIONAL SALES MANAGERS' OFFICES

WESTERN

William T. O'Brien
17291 Irvine Blvd., Suite 262
714/838-1126, TWX: 910-595-1114
*Tustin, California 92680

MID-AMERICA

Mick Carrier
13333 N. Central Expressway
Suite 110
214/234-1109, TWX: 910-867-4763
*Dallas, Texas 75231

NORTHEAST

James Saxon
2 Militia Drive, Suite 4
617/861-1136, Telex: 92-3493
*Lexington, Massachusetts 02173

MID-ATLANTIC

Hank Smith
30 South Valley Road
215/647-2615, TWX: 510-668-7768
*Paoli, Pennsylvania 19301

U.S. SALES OFFICES

ARIZONA

Sales Engineering, Inc.
7155 E. Thomas Road, No. 6
602/945-5781, TWX: 910-950-1288
Scottsdale 85252

CALIFORNIA

Intel Corp.
3065 Bowers Avenue
408/246-7501, TWX: 910-338-0026
*Santa Clara 95051
Intel Corp.
17291 Irvine Blvd., Suite 262
714/838-1126, TWX: 910-595-1114
*Tustin 92680

COLORADO

Intel Corp.
1341 South Lima St.
303/755-1335
*Aurora 80010

CANADA

Multilek, Inc.
4 Barran Street
613/825-4695
Ottawa, Ontario K2C 3H2

FLORIDA

Semtronic Associates, Inc.
P.O. Box 1449
305/771-0010
Pompano Beach 33061

Semtronic Associates, Inc.
685 Cheesecake Road
305/831-8233
Longwood 32750

Mar-Con Associates, Inc.
4836 Main Street
312/675-6450
Skokie 60076

MARYLAND

Barnhill and Associates
1931 Greenspring Drive
301/252-5610
Timonium 21093

Barnhill and Associates
P.O. Box 251
301/252-5610
Glen Arm 21057

MASSACHUSETTS

Intel Corp.
2 Militia Drive, Suite 4
617/861-1136, Telex: 92-3493
*Lexington 02173

Datcom
7A Cypress Drive
617/273-2990
Burlington 01803

MICHIGAN

Sheridan Associates, Inc.
33708 Grand River Avenue
313/477 3800
Farmington 48024

MINNESOTA

Intel Corp.
800 Southgate Office Plaza
5001 West 78th Street
612/835-6722

*Bloomington 55437

E.C.R., Inc.
5280 W. 74th Street
612/831-4547, TWX: 910-576-3153
Minneapolis 55435

MISSOURI

Sheridan Associates, Inc.
110 S. Highway 140, Suite 10
314/837-5200

NEW JERSEY

Addem
Post Office Box 231
516/567-5900
Keasbey 08832

NEW YORK

Ossmann Components Sales Corp.
395 Cleveland Drive
716/832-4271
Buffalo 14215

Addem
37 Pioneer Blvd.
516/567-5900
Huntington Station, L.I. 11746

NEW YORK (Continued)

Ossmann Components Sales Corp.
280 Metro Park
716/442-3290
Rochester 14623

Ossmann Components Sales Corp.
1911 Vestal Parkway E.
607/785-9949

Vestal 13850

Ossmann Components Sales Corp.
132 Pickard Building
315/454-4477

Syracuse 13211

Ossmann Components Sales Corp.
411 Washington Avenue
914/338-5505

Kingston 12401

NORTH CAROLINA

Barnhill and Associates
6030 Bellow Street
919/787-5774

Raleigh 27602

OHIO

Sheridan Associates, Inc.
10 Knollcrest Drive
513/761-5432, TWX: 810-461-2670

Cincinnati 15237

Sheridan Associates, Inc.
7800 Wall Street
216/524-8120

Cleveland 44125

Sheridan Associates, Inc.
Shiloh Bldg., Suite 250
5045 North Main Street
513/277-8911

Dayton 45405

PENNSYLVANIA

Vantage Sales Company
21 Bala Avenue
215/667-0990

Bala Cynwyd 19004

Intel Corp.
30 South Valley Road
215/647-2615, TWX: 510-668-7768

*Paoli, Pennsylvania 19301

Sheridan Associates, Inc.
4268 North Pike,
North Pike Pavilion
412/373-1070

Monroeville 15146

TENNESSEE

Barnhill and Associates
206 Chicasaw Drive
615/928-0184

Johnson City 37601

Evans and McDowell Associates
13333 N. Central Expressway
Room 180
214/238-7157, TWX: 910-867-4763

Dallas 75222

VIRGINIA

Barnhill and Associates
P.O. Box 1104
703/846-4624

Lynchburg 24505

WASHINGTON

SD.R2 Products and Sales
14040 N.E. 8th Street
206/747-7424, TWX: 910-443-2305

Bellevue 98007

*Direct Intel Office

EUROPEAN MARKETING OFFICES

DENMARK

John Johansen
Intel Office
Vester Farimagsgade 7
45-11 5644, Telex: 19567
DK 1606 Copenhagen V

FRANCE

Bernard Giroud
Intel Office
Cidex R-141
(1) 677-60-75, Telex: 27475
94-534 Rungis

ENGLAND

Keith Chapple
Intel Office
Broadfield House
4 Between Towns Road
771431, Telex: 837203
Cowley, Oxford

GERMANY

Erling Holst
Intel Office
Wolfratshauserstrasse 169
798923, Telex: 5-212870
D8 Munchen 71

AUSTRALIA

A.J. Ferguson (Adelaide) PTY. Ltd.
125 Wright Street
51-6895
Adelaide 5000

AUSTRIA

Bacher Elektronische Geräte GmbH
Meidlinger Haupstrasse 78
0222-9301 43, Telex: (01) 1532
A 1120 Vienna

BELGIUM

Inelco Belgium S.A.
Avenue Val Duchesse, 3
(02) 60 00 12, Telex: 25441
B-1160 Bruxelles

DENMARK

Scandinavian Semiconductor
Supply A/S
20, Nannasgade
Telex: 19037
DK-2200 Copenhagen N

GERMANY

Alfred Neye Enatachnik GmbH
Schillerstrasse 14
041 06/612-1, Telex: 02-13590
2085 Quickborn-Hamburg

ISRAEL

Telsys Ltd.
54, Jabotinsky Road
25 28 39, Telex: TSEE-IL 333192
Ramat - Gan 52 464

ITALY

Eledra 3S
Via Ludovico da Viadana 9
(02) 86-03-07
20122 Milano

NETHERLANDS

Inelco N.V.
Weerdestein 205
Postbus 7815
0204416 66, Telex: 12534
Amsterdam 1011

NORWAY

Nordisk Elektronik (Norge) A/S
Mustads Vei 1
602590, Telex: 16963
Oslo 2

SOUTH AFRICA

Electronic Building Elements
P.O. Box 4609
78-9221, Telex: 30181 SA
Pretoria

SWEDEN

Nordisk Elektronik AB
Fack
08-24-83-40, Telex: 10547
S-103 Stockholm 7

SWITZERLAND

Industrade AG
Gemenstrasse 2
Postcheck 80 - 21190
01-60-22-30, Telex: 56788
8021 Zurich

UNITED KINGDOM

Walmore Electronics Ltd.
11-15 Betterton Street
Drury Lane
01-836-0201, Telex: 28752
London WC2H 9BS

ORIENT MARKETING OFFICES

ORIENT MARKETING HEADQUARTERS

JAPAN

Y. Magami
Intel Japan Corp.
Kashara Building
1-6-10 Uchikanda, Chiyoda-Ku
03-295 5441, Telex: 781-28426
Tokyo 101

ORIENT DISTRIBUTORS

JAPAN

Pan Elektron Inc.
No. 1 Higashikata-Machi
045-471-8321, Telex: 781-4773
Midori-Ku, Yokohama 226

U.S. DISTRIBUTORS

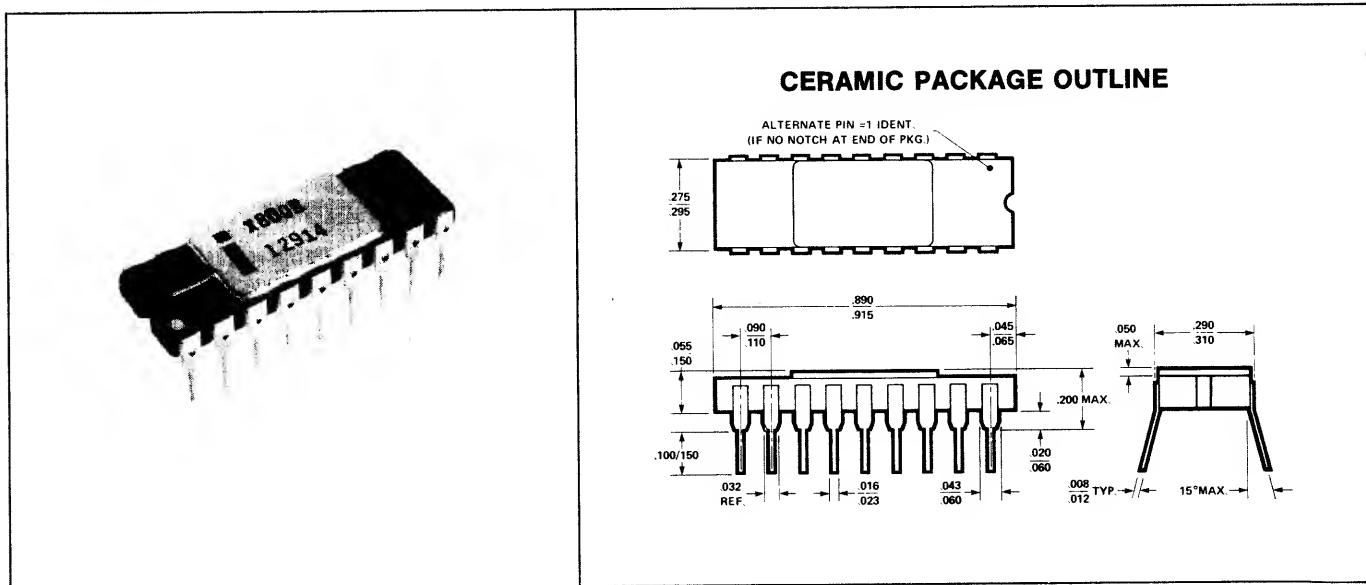
WEST	MID-AMERICA	NORTHEAST	SOUTHEAST
ARIZONA Hamilton/Avnet Electronics 2615 South 21st Street 602/275-7851 Phoenix 85034 Cramer/Arizona 2816 N. 16th Street 602/263-1112 Phoenix 85006	ILLINOIS Cramer/Chicago 1911 South Busse Road 312/593-8230 Mt. Prospect 60056 Hamilton/Avnet Electronics 3901 North 25th Avenue 312/678-6310 Schiller Park 60176	OHIO Cramer/Tri-States, Inc. 666 Redna Terrace 513/771-6441 Cincinnati 45215 Hamilton/Avnet Electronics 118 West Park Road 513/433-0610 Dayton 45459 Sheridan Sales Co. 10 Knollcrest Drive 513/761-5432 Cincinnati 45237 Cramer/Cleveland 5835 Harper Road 216/248-7740 Cleveland 44139 Sheridan Sales Co. 7800 Wall Street 216/524-8120 Cleveland 44125 Sheridan Sales Co. Shiloh Blvd., Suite 250 5045 North Main Street 513/277-8911 Dayton 45405	CONNECTICUT Hamilton/Avnet Electronics 643 Danbury Road 203/762-0361 Georgetown 06829 Cramer/Connecticut 36 Dodge Avenue 203/239-5641 North Haven 06473
CALIFORNIA Hamilton/Avnet Electronics 340 E. Middlefield Road 415/961-7000 Mountain View 94041 Cramer/San Francisco 720 Palomar Avenue 408/739-3011 Sunnyvale 94086 Hamilton Electro Sales 10912 W. Washington Blvd. 213/870-7171 Culver City 90230 Cramer/Los Angeles 17201 Daimler Street 714/979-3000 Irvine 92705 Hamilton/Avnet Electronics 8817 Complex Drive 714/279-2421 San Diego 92123 Cramer/San Diego 8975 Complex Drive 714/565-1881 San Diego 92123	KANSAS Hamilton/Avnet Electronics 37 Lenexa Industrial Center 913/888-8900 Lenexa 66215 MICHIGAN Sheridan Sales Co. 33708 Grand River Avenue 313/477-3800 Farmington 48204 Cramer/Detroit 13193 Wayne Road 313/425-7000 Livonia 48150 Hamilton/Avnet Electronics 12870 Farmington Road 313/522-4700 Livonia 48150	TEXAS Cramer Electronics 2970 Blystone 214/350-1355 Dallas 75220 Hamilton/Avnet Electronics 4445 Sigma Road 214/661-8661 Dallas 75240 Hamilton/Avnet Electronics 1216 West Clay 713/526-4661 Houston 77019	MARYLAND Cramer/EW Baltimore 9224 Patapsco Avenue 301/354-0100 Baltimore 21230 Cramer/EW Washington 16021 Industrial Drive 301/948-0110 Gaithersburg 20760 Hamilton/Avnet Electronics 7255 Standard Drive 301/796-5000 Hanover 20176
COLORADO Cramer/Denver 5465 E. Evans Place at Hudson 303/758-2100 Denver 80227 Hamilton/Avnet Electronics 5921 N. Broadway 303/534-1212 Denver 80216	MINNESOTA Cramer/Bonn 7275 Bush Lake Road 612/941-4860 Edina 55435 Hamilton/Avnet Electronics 2850 Metro Drive 612/854-4800 Minneapolis 55420 Industrial Components, Inc. 5280 West 74th Street 612/831-2666 Minneapolis 55435	MISSOURI Sheridan Sales Co. 110 South Highway 140, Suite 10 314/837-5200 Florissant 63033 Hamilton/Avnet Electronics 392 Brookes Drive 314/731-1144 Hazelwood 63042	MASSACHUSETTS Cramer Electronics, Inc. 85 Wells Avenue 617/969-7700 Newton 02159 Hamilton/Avnet Electronics 185 Cambridge Street 617/273-2120 Burlington 01803
NEW MEXICO Cramer/New Mexico 137 Vermont, N.E. 505/265-5767 Albuquerque 87108 Hamilton/Avnet Electronics 2450 Baylor Drive S.E. 505/765-1500 Albuquerque 87117	WISCONSIN Cramer/Wisconsin 430 West Rawson 414/764-1700 Oak Creek 53154	NEW JERSEY Hamilton Electro Sales 218 Little Falls Road 201/239-0800 Cedar Grove 07009 Cramer/New Jersey No. 1 Barrett Avenue 201/935-5600 Monachie 07074 Hamilton/Avnet Electronics 113 Gaither Drive East Gate Industrial Park 609/234-2133 Mt. Laurel 08057 Cramer/Pennsylvania, Inc. 7300 Route 130 North 609/662-5061 Pennsauken 08110	NORTH CAROLINA Cramer Electronics 938 Burke Street 919/725-8711 Winston-Salem 27102
OREGON Almac/Strom Electronics 8888 S.W. Canyon Road 503/292-3534 Portland 97225		ONTARIO Cramer/Binghamton 3220 Watson Boulevard 607/754-6661 Endwell 13760 Cramer/Rochester 3000 Winton Road South 716/275-0300 Rochester 14623 Cramer/Syracuse 6716 Joy Road 315/437-6671 East Syracuse 13057 Hamilton/Avnet Electronics 6400 Joy Road 315/437-2642 Syracuse 13057 Cramer/Long Island 29 Oser Avenue 516/231-5600 Hauppauge , L.I. 11787 Hamilton/Avnet Electronics 70 State Street 516/333-5800 Westbury , L.I. 11590	CANADA BRITISH COLUMBIA L.A. VARAH Ltd. 2077 Alberta Street 604/373-3211 Vancouver 10
UTAH Cramer/Utah 391 W. 2500 South 801/487-3681 Salt Lake City 84115 Hamilton/Avnet Electronics 647 W. Billings Road 801/262-8451 Salt Lake City 84115		QUEBEC Hamilton/Avnet Electronics 935 Monte De Liesse 514/735-6393 St. Laurent , Montreal 377	
WASHINGTON Hamilton/Avnet Electronics 13407 Northrup Way 206/746-8750 Bellevue 98005 Almac/Strom Electronics 5811 Sixth Avenue South 206/763-2300 Seattle 98108 Cramer/Seattle 5602 Sixth Avenue South 206/772-5755 Seattle 98108		PENNSYLVANIA Sheridan Sales Co. 4268 North Pike North Pike Pavilion 412/373-1070 Monroeville 15146	

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 electrically 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 individual 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 interconnection between the SIM8-01 and MP7-03. In addition, the MCB8-10 provides the 50 Vrms power supply for PROM programming, 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 instructions 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
 - imm8-60 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^{T.M.} 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.

MNEMONIC	MINIMUM STATES REQUIRED	INSTRUCTION CODE D ₇ D ₆ D ₅ D ₄ D ₃ D ₂ D ₁ D ₀	DESCRIPTION OF OPERATION
(1) Lr ₁ r ₂	(5)	1 1 D D D S S S	Load index register r ₁ with the content of index register r ₂ .
(2) LrM	(8)	1 1 D D D 1 1 1	Load index register r with the content of memory register M.
LMr	(7)	1 1 1 1 1 S S S	Load memory register M with the content of index register r.
(3) Lrl	(8)	0 0 D D D 1 1 0 B B B B B B B B	Load index register r with data B . . . B.
LMI	(9)	0 0 1 1 1 1 0 B B B B B B B B	Load memory register M with data B . . . B.
INr	(5)	0 0 D D D 0 0 0	Increment the content of index register r (r ≠ A).
DCr	(5)	0 0 D D D 0 0 1	Decrement the content of index register r (r ≠ A).

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.

ADr	(5)	1 0 0 0 0 S S S	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.
ADM	(B)	1 0 0 0 0 1 1 1	
ADI	(8)	0 0 0 0 0 1 0 0 B B B B B B B B	
ACr	(5)	1 0 0 0 1 S S S	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.
ACM	(8)	1 0 0 0 1 1 1	
ACI	(B)	0 0 0 0 1 0 0 B B B B B B B B	
SUr	(5)	1 0 0 1 0 S S S	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.
SUM	(B)	1 0 0 1 0 1 1	
SUI	(8)	0 0 0 1 0 1 0 0 B B B B B B B B	
SBr	(5)	1 0 0 1 1 S S S	
SBM	(B)	1 0 0 1 1 1 1 1	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.
SBI	(B)	0 0 0 1 1 1 0 0 B B B B B B B B	
NDr	(5)	1 0 1 0 0 S S S	Compute the logical AND of the content of index register r, memory register M, or data B . . . B with the accumulator.
NDM	(8)	1 0 1 0 0 1 1 1	
NDI	(B)	0 0 1 0 0 1 0 0 B B B B B B B B	
XRr	(5)	1 0 1 0 1 S S S	Compute the EXCLUSIVE OR of the content of index register r, memory register M, or data B . . . B with the accumulator.
XRM	(8)	1 0 1 0 1 1 1	
XRI	(B)	0 0 1 0 1 1 0 0 B B B B B B B B	
ORr	(5)	1 0 1 1 0 S S S	Compute the INCLUSIVE OR of the content of index register r, memory register m, or data B . . . B with the accumulator.
ORM	(8)	1 0 1 1 0 1 1 1	
ORI	(B)	0 0 1 1 0 1 0 0 B B B B B B B B	
CPr	(5)	1 0 1 1 1 S S S	Compare the content of index register r, memory register M, or data B . . . B with the accumulator. The content of the accumulator is unchanged.
CPM	(8)	1 0 1 1 1 1 1	
CPI	(B)	0 0 1 1 1 1 0 0 B B B B B B B B	
RLC	(5)	0 0 0 0 0 0 1 0	Rotate the content of the accumulator left.
RCR	(5)	0 0 0 0 1 0 1 0	Rotate the content of the accumulator right.
BAL	(5)	0 0 0 1 0 0 1 0	Rotate the content of the accumulator left through the carry.
RAR	(5)	0 0 0 1 1 0 1 0	Rotate the content of the accumulator right through the carry.

PROGRAM COUNTER AND STACK CONTROL INSTRUCTIONS

(4) JMP	(11)	0 1 X X X 1 0 0 B ₂ B ₂ B ₂ B ₂ B ₂ B ₂ X X B ₃ B ₃ B ₃	Unconditionally jump to memory address B ₃ . . . B ₃ B ₂ . . . B ₂ .
(5) JFc	(9 or 11)	0 1 0 C ₄ C ₃ 0 0 0 B ₂ B ₂ B ₂ B ₂ B ₂ B ₂ X X B ₃ B ₃ B ₃	Jump to memory address B ₃ . . . B ₃ B ₂ . . . B ₂ if the condition flip-flop c is false. Otherwise, execute the next instruction in sequence.
JTc	(9 or 11)	0 1 1 C ₄ C ₃ 0 0 0 B ₂ B ₂ B ₂ B ₂ B ₂ B ₂ X X B ₃ B ₃ B ₃	Jump to memory address B ₃ . . . B ₃ B ₂ . . . B ₂ if the condition flip-flop c is true. Otherwise, execute the next instruction in sequence.
CAL	(11)	0 1 X X X 1 1 0 B ₂ B ₂ B ₂ B ₂ B ₂ B ₂ X X B ₃ B ₃ B ₃	Unconditionally call the subroutine at memory address B ₃ . . . B ₃ B ₂ . . . B ₂ . Save the current address (up one level in the stack).
CFc	(9 or 11)	0 1 0 C ₄ C ₃ 0 1 0 B ₂ B ₂ B ₂ B ₂ B ₂ B ₂ X X B ₃ B ₃ B ₃	Call the subroutine at memory address B ₃ . . . B ₃ B ₂ . . . B ₂ if the condition flip-flop c is false, and save the current address (up one level in the stack.) Otherwise, execute the next instruction in sequence.
CTc	(9 or 11)	0 1 1 C ₄ C ₃ 0 1 0 B ₂ B ₂ B ₂ B ₂ B ₂ B ₂ X X B ₃ B ₃ B ₃	Call the subroutine at memory address B ₃ . . . B ₃ B ₂ . . . B ₂ if the condition flip-flop c is true, and save the current address (up one level in the stack.) Otherwise, execute the next instruction in sequence.
RET	(5)	0 0 X X X 1 1 1	Unconditionally return (down one level in the stack).
RFc	(3 or 5)	0 0 0 C ₄ C ₃ 0 1 1	Return (down one level in the stack) if the condition flip-flop c is false. Otherwise, execute the next instruction in sequence.
RTc	(3 or 5)	0 0 1 C ₄ C ₃ 0 1 1	Return (down one level in the stack) if the condition flip-flop c is true. Otherwise, execute the next instruction in sequence.
RST	(5)	0 0 A A A 1 0 1	Call the subroutine at memory address AAA000 (up one level in the stack).

INPUT/OUTPUT INSTRUCTIONS

INP	(B)	0 1 0 0 M M M 1	Read the content of the selected input port (MMM) into the accumulator.
OUT	(6)	0 1 R R M M M 1	Write the content of the accumulator into the selected output port (RRMM, RR ≠ 00).

MACHINE INSTRUCTION

HLT	(4)	0 0 0 0 0 0 0 X	Enter the STOPPED state and remain there until interrupted.
HLT	(4)	1 1 1 1 1 1 1	Enter the STOPPED state and remain there until interrupted.

NOTES:

(1) SSS = Source Index Register These registers, r_i, are designated A(accumulator=000),

DDD = Destination Index Register B(001), C(010), D(011), E(100), H(101), L(110).

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

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

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

(5) Flag flip-flops are defined by C₄C₃: carry (00-overflow or underflow), zero (01-result is zero), sign (10-MSB of result is "1"), parity (11-parity is even).



intel® Microcomputers. First from the beginning.

INTEL CORPORATION • 3065 Bowers Avenue, Santa Clara, California 95051 • (408) 246-7501