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

Intellec® 8/MOD 80 Operators Manual

JUNE 1974

© Intel Corporation, 1974

98-003A

INTEL CORPORATION

INTELLEC 8/MOD 80
MICROCOMPUTER SYSTEM
OPERATOR'S MANUAL

-- TABLE OF CONTENTS --

THE INTELLEC 8I OPERATOR'S MANUAL

		Page No.
1.0	INTRODUCTION	1-1
2.0	HARDWARE ORGANIZATION	2-1
3.0	OPERATOR'S GUIDE	3-1
3.1	Control Console	3-1
3.1.1	Indicators	3-3
3.1.2	Switches	3-6
3.2	Operating the INTELLEC 8I	3-11
3.2.1	Starting the INTELLEC 8I	3-11
3.2.2	Loading Data Into Memory	3-11
3.2.3	Examining Memory	3-13
3.2.4	Executing An Instruction Supplied By The Console	3-14
3.2.5	Running A Program (Using Reset)	3-16
3.2.6	Search/Wait And Pass Count	3-18
3.3	I/O System	3-20
3.3.1	Logical And Physical Devices	3-20
3.3.2	I/O Subroutines	3-22
3.3.3	User Supplied Devices	3-27
3.4	ASR-33 Teletype Description	3-29
3.5	Teletype Operation	3-31
3.5.1	Loading The Tape Reader	3-31
4.0	SYSTEM MONITOR	4-1
4.1	System Monitor Implementation And Execution	4-3
4.1.1	System Monitor Implementation	4-3
4.1.2	Starting System Monitor	4-3
4.2	System Monitor Operand And Commands	4-3
4.2.1	A Command (Assign I/O Devices)	4-3

			Page No.
	4.2.2	B Command (BNPF Output)	4-4
	4.2.3	C Command (Compare PROM With Memory)	4-6
	4.2.4	D Command (Display Data)	4-8
	4.2.5	F Command (Fill Memory With Constant)	4-11
	4.2.6	G Command (Go To)	4-12
	4.2.7	H Command (Hexadecimal Arithmetic)	4-14
	4.2.8	L Command (Load BNPF Tape)	4-15
	4.2.9	M Command (Move Memory)	4-16
	4.2.10	R Command (Read Hex File)	4-19
	4.2.11	S Command (Substitute Memory)	4-21
	4.2.12	X Command (Examine And Modify Registers)	4-22
	4.2.13	E Command (End File)	4-23
	4.2.14	W Command (Write Memory)	4-23
	4.2.15	N Command (Null Punch)	4-25
	4.2.16	T Command (Transfer From PROM To Memory)	4-25
	4.2.17	P Command (Program PROM)	4-26
5.0	THE INTELLEC 8I TEXT EDITOR		5-1
	5.1	Loading The Text Editor	5-1
	5.2	Text Editor Operation And Commands	5-2
	5.2.1	Input Commands A (Append) And I (Insert)	5-4
	5.2.2	Buffer Pointer Manipulation Commands B (Beginning), C (Character), L (Line), Z (End of Workspace)	5-6
	5.2.3	Output Commands W (Write), T (Type), E (End)	5-8
	5.2.4	Data Manipulation Commands D (Delete), F (Find), S (Substitute), K (Kill)	5-13
	5.2.5	Command Strings	5-19
	5.2.6	Use of Tabs	5-21
	5.2.7	Command Iteration	5-22
6.0	THE INTELLEC 8I ASSEMBLER		6-1
	6.1	Loading The Assembler	6-1
	6.2	Execution of the Assembler	6-1
	6.3	Assembler Limits	6-2
	6.4	SYNTAX Restrictions	6-3
	6.5	Assembler Error Messages	6-4

		Page No.
Appendix A.	Instruction Summary	A-1
Appendix B.	Instruction Machine Code Summary	B-1
Appendix C.	Software Operating Commands And Messages	C-1
Appendix D.	Hexadecimal Program Tape Format	D-1
Appendix E.	Conversion Tables	E-1
Appendix F.	ASCII Codes	F-1
Appendix G.	I/O Port Assignment and Sample Device Drivers	G-1
Appendix H.	Example of a Program Listing	H-1

LIST OF FIGURES

Figure 2-1	THE INTELLEC 8I BLOCK DIAGRAM	2-2
Figure 3-1	THE INTELLEC 8I CONSOLE (FRONT PANEL)	3-2
Figure 3-2	ASR-33 TELETYPE KEYBOARD AND CONTROLS	3-30
Figure 4-1	NORMAL USE AND EXECUTION SEQUENCE FOR SYSTEM SOFTWARE	4-2

The INTELLEC 8 can be supplied with either an 8008 or an 8080 CPU chip. This manual uses the term INTELLEC 8I to mean the INTELLEC 8 with 8080 CPU.

This manual assumes that the reader has read and understood the 8080 Programming Manual.

--TERMS AND ABBREVIATIONS--

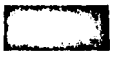
TERMS:

TERM	DESCRIPTION
Address	A 16 bit number assigned to a memory location corresponding to its sequential position.
Bit	The smallest unit of information which can be represented. (A bit may be in one of two states, 0 or 1).
Byte	A group of 8 contiguous bits occupying a single memory location.
Console	The INTELLEC 8I front panel, containing switches and indicators that allow a user to operate the computer and monitor program execution.
Instruction	The smallest single operation that the computer can be directed to execute.
Object Program	A program which can be loaded directly into the computer's memory and which requires no alteration before execution. An object program is usually on paper tape, and is produced by assembling (or compiling) a source program. Instructions are represented by binary machine code in an object program.
Program	A sequence of instructions which, taken as a group, allow the computer to accomplish a desired task.
Source Program	A program which is readable by a programmer but which must be transformed into object program format before it can be loaded into the computer and executed. Instructions in an assembly language source program are represented by their assembly language mnemonic.

TERMS -- (Continued):

TERM	DESCRIPTION
System Program	A program written to help in the process of creating user programs.
User Program	A program written by the user to make the computer perform any desired task.
Word	A group of 16 contiguous bits occupying two successive memory locations. (2 bytes).

ABBREVIATIONS:

ABBREVIATION	DESCRIPTION
Cr	Carriage return
CPU	Central Processing Unit
Lf	Line feed
PROM	Programmable Read Only Memory
Sp	Space Bar
nnn B	nnn represents a number in binary format.
nnn D	nnn represents a number in decimal format.
nnn O	nnn represents a number in octal format.
nnn Q	nnn represents a number in octal format.
nnn H	nnn represents a number in hexadecimal format.
	Shaded portions of teletype/operator dialog represent teletype output.

1.0 INTRODUCTION

This manual describes how to operate and run programs on the INTELLEC 8I computer. Information provided by this manual can be broadly divided into console operations (Section 3), and use of system software (Sections 4, 5 and 6).

1.1 INTELLEC 8I PURPOSE AND CAPABILITIES

The INTELLEC 8I provides a stand alone computer built around the INTEL 8080 CPU. It has been designed primarily as a system development tool for INTEL 8080 microcomputer users.

Since the entire 8080 CPU is enabled on one LSI chip, certain departures from common computer design are present in the INTELLEC 8I, and to the operator this is most apparent in the INTELLEC 8I console, as described in Section 2.

An input/output system is provided which enables the programmer to use standard devices, such as an ASR-33 teletype, or to supply his own devices for input, output, and listing. In addition, the INTELLEC 8I console has a connector for Programmable Read Only Memories (PROM), thus allowing programs to be loaded directly into PROMs.

Combining the INTELLEC 8I system hardware with system software (summarized in Section 1.2), the user is provided with the capability to generate executable programs, and to load programs into PROMs.

For users who anticipate extensive programming on the INTELLEC 8I, cross assemblers are available, so the programmer may generate object programs on a larger and more powerful computer (any computer having a FORTRAN compiler whose standard integer size is 32 bits or greater), using the INTELLEC 8I for final program checkout and PROM loading only.

1.2 SYSTEM SOFTWARE

INTELLEC 8I system software consists of three programs designed to help a user write, debug and prepare his own programs for execution. The three system programs are:

(1) The MONITOR:
This is the user's principle interface to the INTELLEC 8I. The monitor is pre-loaded on PROMs, and is ready to use as soon as power is applied to the INTELLEC 8 I. It controls the loading and execution of other system and user programs.

(2) The EDITOR:
The editor, used by a programmer to create and correct source program text.

(3) The ASSEMBLER:
The assembler is used to convert source programs into object programs which may be loaded into PROMs or random access memory and executed.

Before operating the INTELLEC 8I computer, a user should throughly understand console operations as described in Sections 2 and 3. Next, execute the system monitor (Section 4), and note the operations that can be executed either via the system monitor, or via the console. Only when console operations and monitor execution are clearly understood should a user execute the editor or assembler.

2.0 HARDWARE ORGANIZATION

In order to understand INTELLEC 8I console operations, it is necessary first to appreciate some aspects of INTELLEC 8I hardware design.

Since the INTELLEC 8I CPU is realized on one LSI chip (INTEL's 8080), external logic cannot interrogate registers or busses that are internal to the LSI chip. Thus, the INTELLEC 8I console does not have any direct access to CPU working registers, the program counter, or the CPU/memory interface; instead the console must access busses external to the CPU in order to perform any operation.

Figure 2-1 provides a very simple illustration of INTELLEC 8I hardware organization as seen by the operator. The console is most accurately visualized as a peripheral device, placed in parallel with the CPU, rather than as an adjunct of the CPU.

Bus (a) carries memory addresses from the console or the CPU to memory, and carries output data from the console or CPU to output ports, and thence to external peripheral devices (e.g., a teletype printer).

Bus (b) carries data from the console or CPU to the memory.

Bus (c) carries data from memory to the console of the CPU.

Bus (d) carries data from input ports to the CPU but not the console. This data is accessable to the console, however, by a process described later.

Bus (e) allows the console to transmit a program interrupt to the CPU.

Bus (f) is a control bus which is used to control instruction execution. Since the console is connected to the control bus, instruction execution can be controlled from the console.

Busses (a), (b), (c), (d) and (f) may be visualized as having three way switches that allow information to be routed to/from the CPU or the console. Since the console is a device designed in parallel to the CPU it contains a considerable amount of parallel logic, including its own data and address registers. Thus, there are certain states in which the CPU remains in control but the console has temporarily suspended operations and there are other states in which the console can completely take over machine operations.

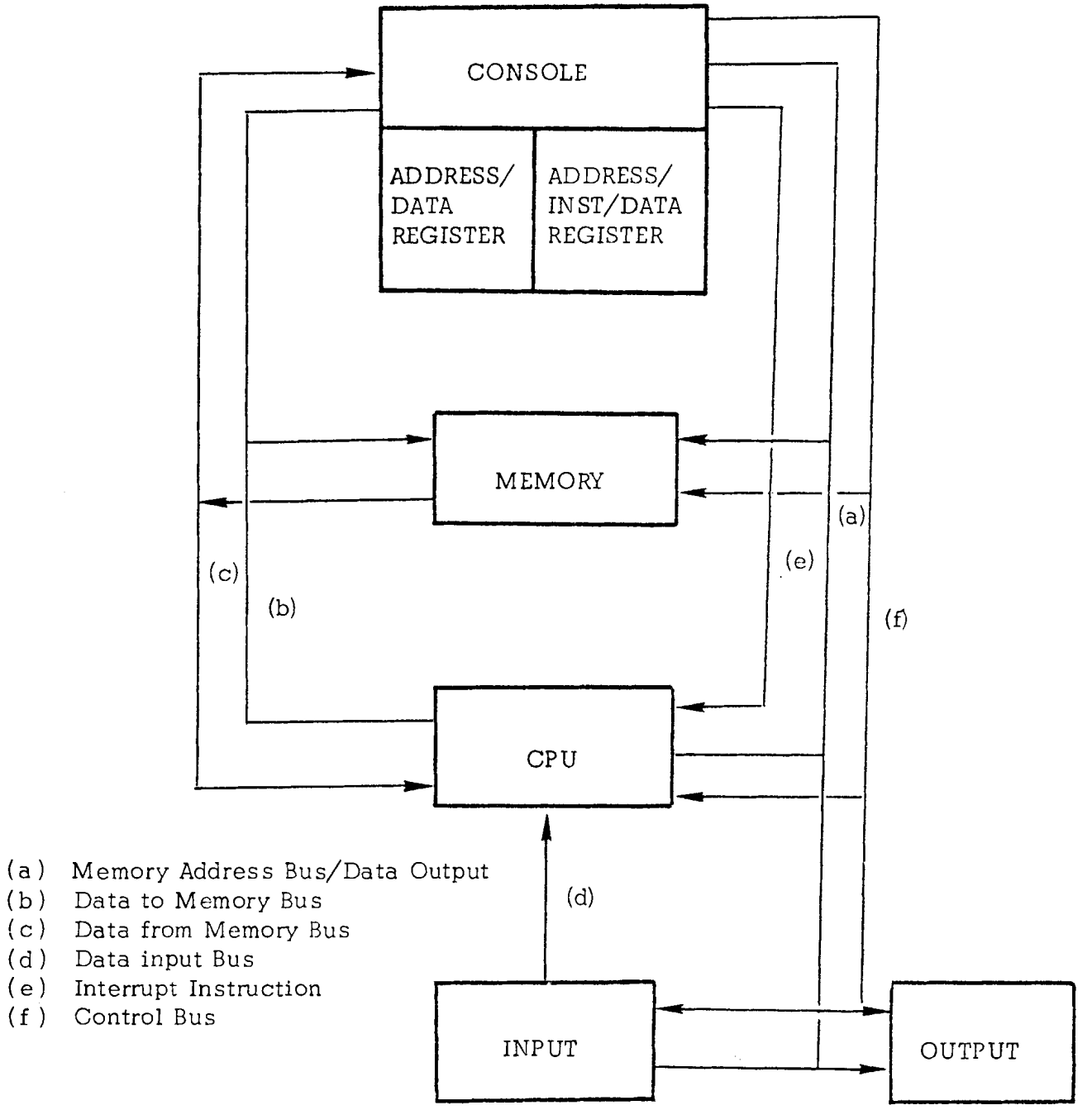


FIGURE 2-1.

A Simplified INTELLEC 8I Block Diagram
Showing the Computer as Seen by the Operator.

Consider now how some operations must be performed, given the hardware organization illustrated in Figure 2-1. Operations are described conceptually below, to provide an introduction to rigorous procedures given later in this manual.

Since the console has its own address and data registers, and since there is a bi-directional bus link between the console and memory, data can be read from memory to console, and written from console to memory directly.

Although there is no direct path for data from input ports to the console, performing an input access operation from the console causes the input data to be sent through the CPU and onto bus (c), where it is displayed on the console.

There is no direct link between the condition bits (carry, sign, zero, parity) and the console.

There is no direct link between CPU registers and the console.

In order to deposit data into a CPU register, it is necessary to deposit the new data into memory, then execute instructions that move the data to the required CPU register.

Sections 4, 5 and 6 provide step-by-step INTELLEC 8I operating instructions, but first the INTELLEC 8I console switches and indicators are described in Section 3.

3.0 OPERATOR'S GUIDE

The INTELLEC 8I is controlled and operated by means of the front panel console and a terminal, which on the standard system is an ASR-33 Teletype. By using these devices, the operator can exercise complete control over the loading, running, and debugging of programs, and can also monitor the status of the INTELLEC 8I as programs run. An optional high-speed paper tape reader allows the operator greater ease in loading programs and data from punched paper tapes.

This section will describe the appearance and operation of the control console, Teletype, and high-speed paper tape reader, and will include information on operating features available to the INTELLEC 8I user.

3.1 CONTROL CONSOLE

Console switches and indicators are used by an operator to control the INTELLEC 8I computer. Indicators provide the status of processor operations, console registers, and memory locations; the switches provide a means of manually loading data into the various console registers, input/output ports, and memory locations. Figure 3-1 is a picture of the INTELLEC 8I front panel. A basic description of the operation and function of each of the switches and indicators is given below.

The summary description of switches and indicators given in Section 3.1.1 and 3.1.2 do not attempt to explain terms or concepts described later in this manual. However, the "SEARCH/WAIT" is one feature of the INTELLEC 8I which must be understood in order to avoid confusion when first reading the switch and indicator descriptions provided below.

The "SEARCH/WAIT" allows the operator to cause program execution to halt upon encountering a memory address which has been passed some fixed number of times in the course of program execution. For a complete description of this feature, see Section 3.2.6.

3.1.1 INDICATORS

All of the indicators on the INTELLEC 8I are Light-Emitting Diodes, or LEDs, and are lit when true.

STATUS INDICATORS

These indicators show the processor status.

1.	RUN	Indicates that the processor is running.
2.	WAIT	Indicates that the processor is waiting for memory or input/output facilities to become available, or that the WAIT switch is set.
3.	HALT	Indicates that the processor is stopped.
4.	HOLD	Indicates that the processor is in the hold state, allowing a memory access or I/O access to be performed by the console. This state may be entered only while the processor is in the WAIT or HALT state.
5.	SRCH COMPL	Indicates that the memory address referred to in a "SEARCH/WAIT" operation has been encountered the required number of times.
6.	ACCESS REQ	Indicates that a console memory access or I/O access operation has been attempted and is waiting to be executed.
7.	INT REQ	Indicates that a console interrupt request has been generated, but has not yet been acknowledged.
8.	INT DISABLE	Indicates that the CPU interrupt system is disabled.

CYCLE INDICATORS

These indicators provide a continuous display of the processor's machine cycle status.

1.	FETCH	Indicates that the processor is performing an instruction fetch from memory.
2.	MEM	Indicates that the processor is performing a memory read or write operation, or that a memory access operation is being performed manually from the control console.
3.	I/O	Indicates that the processor is performing an input/output read/write operation, or that a manual input/output operation is being performed from the control console.
4.	DA	Indicates that a direct access to memory or an I/O port is being performed.
5.	READ/INPUT	Indicates that a memory read or input operation is being performed.
6.	WRITE/OUTPUT	Indicates that a memory write or output operation is being performed.
7.	INT	Indicates that an interrupt operation is in progress.
8.	STACK	Indicates that a stack operation is being performed.

ADDRESS INDICATORS

These indicators display memory and I/O addresses during processor operations.

0-15	These indicators display the memory address being accessed during FETCH, READ, WRITE and manual memory access operations.

0-7	These indicators display the I/O address during a programmed or manual Input/ Output access operation.

INSTRUCTION/DATA INDICATORS

These indicators display the instruction or data passed between the processor and memory, or between the processor and an input/output device.

REGISTER/FLAG DATA INDICATORS

0-7	These indicators display the data byte being written to output port 255 (FFH); thus it is a programmable display.

3.1.2 SWITCHES

The switches on the INTELLEC 8I are considered to be true, or at a logic 1 level, when the top of the switch is pushed in.

ADDRESS/INSTRUCTION/DATA SWITCHES

MEM ADDRESS LOW/INT INST/DATA/PASS COUNT

These switches are used for four functions: to hold the lower eight bits of the memory address during a manual memory access, or "SEARCH/WAIT" operation; to hold the interrupt instruction used with the console interrupt; to hold the data which is to be deposited into memory or an output port; and to hold the pass count during a "SEARCH/WAIT" operation.

ADDRESS/DATA SWITCHES

MEM ADDRESS HIGH/ I/O ADDRESS/SENSE DATA

These switches are used for three functions: to hold the higher 8 bits of a memory address during a manual memory access; to hold an I/O port number during a console I/O access; and to hold SENSE DATA to be used by INPUT instructions when the SENSE switch is set.

ADDRESS CONTROL SWITCHES

These switches control the loading of address or pass count data into the various console registers.

1. LOAD PASS — When depressed, loads the pass count contained in the Address/Instruction/Data switches into the pass count register.

2. DECR — When depressed, decrements the console memory address register by one.

3. INCR — When depressed, increments the console memory address register by one.

4. LOAD — When depressed, loads the address held in the Address switches into the console memory address register for manual memory access operations.

Example: To load the number 3A5F H into the console address register:

Set Address/Data switches 9,11,12, and 13 on, and set Address/Instruction/Data switches 0,1,2,3,4, and 6 on, then press LOAD switch.

Address/Data	Address/Instruction Data
15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
0 0 1 1 1 0 1 0	0 1 0 1 1 1 1 1 = 3A5F
	(1 = set, zero = reset)

If the INCR switch is now pressed, the console address register will contain 3A60 H (3A5F+1). This, of course, will not change the settings of the address switches. If the INCR switch is pressed again, the console address register will contain 3A61 H. If the LOAD switch is pressed again, the number 3A5F H held in the address switches is placed again in the console address register.

The state of the console address register during each of these steps is as follows:

	Console Address Register	
Set switches	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
Press LOAD	0 0 1 1 1 0 1 0 0 1 0 1 1 1 1 1	= 3A5F H
Press INCR	0 0 1 1 1 0 1 0 0 1 1 0 0 0 0 0	= 3A60 H
Press INCR	0 0 1 1 1 0 1 0 0 1 1 0 0 0 0 1	= 3A61 H
Press LOAD	0 0 1 1 1 0 1 0 0 1 0 1 1 1 1 1	= 3A5F H

Example: To load a pass count of 11_{10} into the console pass count register (See Section 3.2.6) for a complete description of SEARCH/WAIT - PASS COUNT) :

Set Address/Instruction/Data switches 0,1, and 3 on, then LOAD PASS.

Set switches: Address/Instruction/Data
7 6 5 4 3 2 1 0
0 0 0 0 1 0 1 1 = 0B H = 11
(1 = set, 0 = reset)

Pass Count Register unchanged

Press LOAD PASS: Pass Count Register = 00001011

MODE SWITCHES

These switches control the mode of operation of the INTELLEC 8I.

1.	SENSE	When depressed, an input instruction (IN) will take its data from the Address/(SENSE DATA) switches rather than from the input port.
2.	I/O ACCESS	When depressed, provides access to any of the I/O ports if the processor is in the WAIT or HALT state. The accessed port is that whose number appears in Address/Data switches 8-15.
3.	MEM ACCESS	When depressed, provides access to memory.
4.	SRCH-WAIT	When depressed, causes the processor to execute a program up to a certain location, then forces the processor into a WAIT state. (See Section 3.2.6).
5.	WAIT	When depressed, forces the processor into a WAIT state.

CONTROL SWITCHES

These switches provide operator control of the processor.

1. STEP/CONT

   If the processor was forced into the WAIT mode by depressing the WAIT switch, pressing the STEP/CONT switch causes the next instruction to be executed, followed by a return to the WAIT mode.

   After the Search Complete condition forces the processor into the WAIT mode (see Section 3.2.6), pressing the STEP/CONT switch causes normal program execution to continue.

2. DEP

   Deposits the data held in the Address/ Instruction/Data switches into the memory address held in the console address register, or the I/O port whose number is held in Address/Data switches 8-15.

3. DEP AT HLT

   If this switch is set, and the DEP switch is set, and either the MEM ACCESS or I/O ACCESS switch is set, and a program HALT is executed, then the 8 bit data word held in the Address/Instruction/Data switches will be deposited at the memory address held in the console address register or the I/O port whose number is held in Address/Data switches 8-15.

4. INT

   When depressed, this switch causes the processor to execute an interrupt cycle, using the interrupt instruction contained in the Address/Instruction/Data switches.

5. RESET

   When depressed, this switch forces the processor to begin program execution at location zero by executing an RST 0 instruction which is a CALL to location 0. The UART is also reset. (See the INTELLEC 8 Reference Manual).

Example: If the SENSE switch is set, and Address/Data Switches 10,11,13, and 15 are set, then any input instruction will place the number AC H into the accumulator:

```
Address/Data
15 14 13 12 11 10 9 8
 1  0  1  0  1  1 0 0   = AC H
```

Example: If the WAIT and I/O ACCESS switches are set, Address/Instruction/Data switches 1,3, and 4 are set, and Address/Data switches (I/O address) 10, 11, and 13 are set, then pressing the DEP switch will cause the number 1A H to be sent to output port number 2C H:

```
Address/Data                          Address/Instruction/Data
15 14 13 12 11 10 9 8                 7 6 5 4 3 2 1 0
 0  0  1  0  1  1 0 0                 0 0 0 1 1 0 1 0
 \_____________/                       \_____________/
 2C H= output port number              1A H = DATA SENT
```

Example: If the WAIT switch is set, the console address register contains 20A1 H, and the Address/Instruction/Data switches are set as in the preceding example, then pressing the DEP switch will cause the number 1A H to be placed at memory location 20A1 H.

Example: If Address/Instruction/Data switches 0,1,2,4,6, and 7 are set, pressing the INT switch will cause the processor to begin an interrupt cycle, executing the instruction RST 2, which is encoded as D7 H.

```
Address/Instruction/Data    (INT INST )
7 6 5 4 3 2 1 0
1 1 0 1 0 1 1 1  = D7 H = RST 2
```

POWER AND PROM PROGRAMMING SWITCHES

1.	PRGM PROM PWR	This switch provides the high voltage used by the PROM programmer.
2.	POWER	This switch is the key-operated main power switch.

PROM SOCKET

This socket is used to hold the 1602A or 1702A PROM (Intel's electronically programmable and ultraviolet erasable memory) to be programmed.

3.2 OPERATING THE INTELLEC 8I

3.2.1 STARTING THE INTELLEC 8I

Before attempting to start the INTELLEC 8I, ensure that the following conditions are met:

(1) The line cord must be inserted firmly into the socket, and the supply voltage must be at the proper level.

(2) All devices attached to the INTELLEC 8I must be turned OFF.

(3) All control and data switches should be switched OFF; (i.e., the bottom of the switch should be in.

(4) The PROM PRGM PWR switch should be off, and no PROM module should be in the PROM socket.

Insert the key into the POWER switch and turn it to the ON position. This applies power to the INTELLEC 8I. The INTELLEC 8I is now in the HALT state and ready for operation.

3.2.2 LOADING DATA INTO MEMORY

Any data which are to be loaded into memory may be loaded in one of two ways: by means of the System Monitor program, discussed in Section 4, or by hand from the console, as described below.

To load data into the INTELLEC 8I via the console follow these steps:

1. Press the MEM ACCESS switch, to enable memory access.

2. Set into the Address switches, the first memory byte address into which data is to be loaded.

3. Press the LOAD switch, to load the memory address into the console address register.

4. Set the data which you wish to load into the ADDRESS/INSTRUCTION/DATA switches.

5. Press the DEP switch, the data will be loaded into the addressed memory location.

6. If the next byte of data is to be loaded into the next sequential memory location, press the INCR switch once, to increment the memory address by one. If not, return to step 3.

7. Return to step 5.

The most efficient way to manually load large amounts of data into memory is to load it in sequential order, from the lowest memory location to the highest.

Example: Load the instruction JMP 3800 H into memory beginning at location 0. (This instruction will cause a jump to the INTELLEC 8I system monitor). The instruction is encoded as C30038H. The following process is used:

1. Press the MEM ACCESS switch, enabling a memory access.

2. Set the Address/Data and Address/Instruction/Data switches all to 0 (indicating memory location 0).

3. Press the LOAD switch, setting the console address register to 0.

4. Set the Address/Instruction/Data switches to C3 H (set switches 0, 1, 6, and 7).

5. Press the DEP switch. Location 0 now holds C3 H.

6. Press the INCR switch. The console address register now = 1.

7. Set the Address/Instruction/Data switches to 0.

8. Press the DEP switch. Location 1 now holds 00 H.

9. Press the INCR switch. The console address register now = 2.

10. Set the Address/Instruction/Data switches to 38 H (set switches 3, 4, 5).

11. Press the DEP switch. Location 2 now holds 38 H.

The entire instruction is now loaded.

3.2.3 EXAMINING MEMORY

To examine the contents of memory from the control console, perform the following steps:

1. Press the MEM ACCESS switch.

2. Set the address you wish to examine into the address switches.

3. Press the LOAD switch, loading the address into the console address register.

4. The contents of the selected memory location will be displayed in the Instruction/Data indicators.

5. If the next piece of data you wish to see is in the next sequential memory location, press INCR. The data from the next location will be displayed as in step 5. If not, return to step 3.

6. Return to step 3.

Example: Suppose that locations 0 through 2 contain the data C30038H as in the example of section 3.2.2. To verify this, use the following procedure:

1. Press the MEM ACCESS switch.

2. Set the Address/Data and Address/Instruction/Data switches all to 0 (indicating memory location 0).

3. Press the LOAD switch, setting the console address register to 0. Instruction/Data Indicators 0, 1, 6, and 7 will light, indicating that C3 H is the data in location 0.

4. Press the INCR switch. The console address register will now contain 1, and none of the Instruction/Data Indicators will light, indicating that location 1 contains 00 H.

5. Press the INCR switch. 38 H will appear in the Instruction/Data Indicators.

3.2.4 EXECUTING AN INSTRUCTION SUPPLIED BY THE CONSOLE

When the CPU recognizes a console interrupt request, it executes the instruction whose hexadecimal encoding is held in the Address/Instruction/Data switches. This enables the operator to supply the CPU with a one-byte instruction to be executed from the console. The procedure is as follows:

1. Set the WAIT switch.

2. Set the encoding of the instruction to be supplied by the console into the Address/Instruction/Data switches.

3. Press the INT switch.

4. Press the STEP/CONT switch repeatedly, until the INT and FETCH indicators are lit. (This enables the CPU to complete the instruction at which the WAIT occurred, and to fetch the interrupt instruction from the console).

5. Press the STEP switch once. The instruction set into the A/I/D switches has now been executed.

6. Reset the WAIT switch, and program execution resumes with the instruction following the one at which the WAIT occurred.

3.2.5 RUNNING A PROGRAM (USING RESET)

If your program has a starting address of zero, all that is necessary to begin execution is to press the reset switch. The INTELLEC 8I will automatically execute an RST 0 instruction, causing program execution to begin at memory address 0. Otherwise, use the following procedure to place a "jump" to your program's starting address at location 0. Pressing RESET will cause this jump to be executed, transferring control to the program.

1. Press the MEM ACCESS switch.

2. Place all of the Address switches into the 0 position, giving an address of 0000 H.

3. Press the LOAD switch, loading the 0000 H address into the console address register.

4. Place C3 H into the Address/Instruction/Data switches. This is the 8080 operation code for a Jump instruction.

5. Press the DEP switch to load the Jump code into memory location 0000 H.

6. Press the INCR switch one time to increment the memory address.

7. Place the low order eight bits of the starting address of your program into the Instruction/Data switches. The JUMP instruction format requires that the low order byte be given first.

8. Press the DEP switch.

9. Press the INCR switch once.

10. Place the high order eight bits of the starting address into the Address/ Instruction/Data switches.

11. Press the DEP switch.

12. Turn the MEM ACCESS switch to the OFF position.

13. Press the RESET switch. This will cause the INTELLEC 8I to begin execution at location 0000 H, which now contains a Jump instruction to the starting address of your program.

Example: Execute the System Monitor, which resides at location 3800 H.

1. Set the MEM ACCESS switch.

2. Set all the Address switches to 0, indicating memory location 0.

3. Press the LOAD switch.

4. Set the Address/Instruction/Data switches to C3 H.

5. Press the DEP switch.

6. Press the INCR switch.

7. Set the Address/Instruction/Data switches to 00 H.
(The low order byte of the jump address)

8. Press the DEP switch.

9. Press the INCR switch.

10. Set the Address/Instruction/Data switches to 38 H.
(The high order byte of the jump address)

11. Press the DEP switch.

12. Reset the MEM ACCESS switch.

13. Press the RESET switch. The "jump to 3800 H" instruction will be executed, transferring control to the System Monitor.

3.2.6 SEARCH/WAIT AND PASS COUNT

The PASS COUNT feature allows the INTELLEC 8I to count the number of times an operator-specified memory address is encountered and passed during execution of a program, and, when this count reaches a specified number, to enter the WAIT mode upon encountering this address again.

To use this feature, first set the WAIT switch to stop the processor. Then load a pass count (using the binary representation of the count) into the console's pass count register. This is done by setting the Address/Instruction/Data switches to the pass count, and pressing the LOAD PASS switch. Then, set the memory address to be monitored into the Address switches, load this address into the console memory register, set the SEARCH/WAIT switch, reset the WAIT switch, and begin execution of the program. It the pass count were set to N, the CPU would execute the instruction at the indicated address N times, and enter the WAIT state upon encountering the address for the N + 1st time.

Example: Suppose the following program section appears in memory:

MEMORY ADDRESS		INSTRUCTIONS		ASSEMBLED INSTRUCTIONS
0100		MVI H,02		2602
0102		MVI L,00		2E00
0104		MVI M,0	; STORE 0 IN 0200H	3600
0106		XRA A	; SET A=0	AF
0107	BACK:	ADI 1	; ADD 1 to A	C601
0109		MOV M,A	; STORE A IN 0200H	77
010A		JMP BACK		C30701

The first four statements cause memory location 0200 H and the accumulator to be set to 0. The ADI at location 0107 H adds one to the accumulator, and the MOV stores the accumulator at location 0200 H. Control then passes back to location 0107 H, where the accumulator is again incremented.

In order to cause the INTELLEC 8I to execute the ADI 14 times and to enter the WAIT state upon reaching it for the fifteenth time, use the following procedure:

1. Set the WAIT switch.

2. Set Address/Instruction/Data switches 1, 2, and 3 to 1 (representing E H = 14_{10}).

3. Press LOAD PASS, loading 14 into the console pass count register.

4. Set the Address switches to 0107 H, the memory address to be monitored.

5. Press the LOAD switch, setting the console address register to 0107 H.

6. Set the SEARCH/WAIT switch.

7. Reset the WAIT switch and begin execution of the program.

8. The WAIT and SEARCH COMPL indicators will light, indicating that the instruction at 0107 H has been executed and passed 14 times, and the CPU is waiting at this location.

If memory location 0200 H is displayed using the procedure of Section 3.2.3, it will be seen to contain 0E H.

3.3 I/O SYSTEM

The INTELLEC 8I can support a number of input/output devices, from the teletype and high speed paper tape reader to devices supplied by the user. In general, it may be convenient to have two devices which can perform the same function, but to use them for different purposes at various times. For example, if a program is being assembled, you might want the program listing to be written on one device, while any system messages not relevant to the assembly would be written on a separate device.

The I/O system described below permits this type of change. Devices may be assigned functions via a System Monitor command (see Section 4.2.1) or via the user's program. That is, it is possible to write programs which read from several different input devices and write to several different output devices of the program's choosing, without requiring any human intervention.

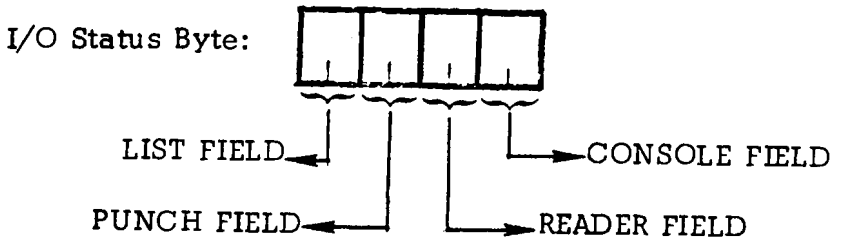
3.3.1 LOGICAL AND PHYSICAL DEVICES

Regardless of how many I/O devices a particular INTELLEC 8I has, there are only four operations which can be performed to any of them. For example, a punch operation can be performed either to the teletype punch or a high speed punch. All system programs and user-written programs, therefore, access four LOGICAL DEVICES (i.e., a PUNCH device) which are then translated to a PHYSICAL DEVICE (i.e., a high speed punch) by the I/O system.

The four logical devices available to programs are:

- CONSOLE — An interactive, character-oriented device used for both input and output.
- READER — A character-oriented, input-only device which transfers data on command and signals the program when there is no more data (an end-of-file condition).
- PUNCH — A character-oriented, output-only device which accepts a character from the program and records it on some external medium.
- LIST — A character-oriented, output-only device which accepts a character from the program and records it on some external medium in human readable form.

Each of these four logical devices may be associated with one of four physical devices at any instant, giving a total of 16 physical devices. The mapping from logical to physical devices is specified by an I/O status byte which resides in memory and is accessible to system and user programs via I/O subroutines described in the next section. The possible mappings appear as follows:



LOGICAL DEVICES	IOCHK FIELD	PHYSICAL DEVICES
	00	TTY
CONSOLE ——→	01	CRT
	10	BATCH (READER=CONSOLE INPUT LIST=CONSOLE OUTPUT)
	11	1 (user console device)
	00	TTY
READER ——→	01	PTR
	10	1 (user reader device 1)
	11	2 (user reader device 2)
	00	TTY
PUNCH ——→	01	PTP
	10	1 (user punch device 1)
	11	2 (user punch device 2)
	00	TTY
LIST ——→	01	CRT
	10	1 (user list device 1)
	11	2 (user list device 2)

At cold start or system reset, the I/O status byte is set equal to 00H, causing the teletype to be selected for all logical devices.

3.3.2 I/O SUBROUTINES

The way in which a program performs an I/O operation to any of the four logical devices is by calling the appropriate subroutine supplied by the I/O system. The available subroutines and their locations in memory are given in the following table:

ROUTINE	FUNCTION	MEMORY LOCATION
CI	Console input	3803H
RI	Reader input	3806H
CO	Console output	3809H
PO	Punch output	380CH
LO	List output	380FH
CSTS	Console input status	3812H
IOCHK	Check I/O mapping	3815H
IOSET	Set I/O mapping	3818H
MEMCK	Check size of memory	381BH

The rest of this section gives a description and examples of how to call these subroutines.

CI - CONSOLE INPUT

This routine returns a character received from the selected console device to the caller in the A register. The A register and the condition bits are affected by this operation.

Example:

Assembly Language

```
CI      EQU     3803H
        ...
        CALL    CI
        STA     DATA
        ...
```

PL/M

```
CI:
     PROCEDURE BYTE;
          DECLARE IOCI LITERALLY '3803H';
               GO TO IOCI;
     END CI;
               ...
     DATA = CI;
               ...
```

CO - CONSOLE OUTPUT

CO transmits a character, passed from the calling program in the C register, to the device selected for console output. The A and C registers and the condition bits are affected.

Example:

```
          Assembly Language

CO        EQU     3809H
          ...
          MVI     C,'.'
          CALL    CO              ; PRINT '.' ON CONSOLE
          ...
                PL/M

CO:
     PROCEDURE (CHAR);
          DECLARE CHAR BYTE;
          DECLARE IOCO LITERALLY '3809H';
               GO TO IOCO;
          END CO;

     CALL CO('.');
```

RI - READER INPUT

RI returns a character read from the reader device in the A register. If no character was read from the device (i.e., end of file), the CARRY condition bit is set equal to 1, and the A register is zeroed. If data is ready, the CARRY bit is zeroed. If no character is received from the physical device within a pre-established time, an end of file is simulated and control is returned to the calling program.

Example:

Assembly Language

```
RI          EQU     3806H
            ...
            CALL    RI
            JC      EOF       ; END OF FILE SENSED
            STA     DATA
            ...
```

PL/M

```
RI:
    PROCEDURE BYTE;
        DECLARE IORI LITERALLY '3806H';
            GO TO IORI;
        END RI;
            ...
    DATA = RI;
    IF CARRY THEN GO TO EOF$LOC;
```

PO - PUNCH OUTPUT

PO transmits a character from the calling program to the device selected as the punch device. PO is identical in format to CO, the only difference being the entry point address, '380CH'.

LO - LIST OUTPUT

LO performs the same function to the selected list device as CO and PO do to their selected devices. It's entry point address is '380FH'.

CSTS - CONSOLE INPUT STATUS

In many applications, there is a need to 'poll' the console device to see if the operator wishes to interrupt the current task and do something else. The CSTS routine allows the caller to test the console to see if a character is ready for input. CSTS returns a logical value in the A register (true = 0FFH, false = 0) indicating whether or not a key has been depressed. The user may then use CI to read the character.

Example:

Assembly Language

```
CSTS        EQU     3812H
            ...
            CALL    CSTS
            RRC
            JNC     CONTINUE        ; NO CHARACTER
            CALL    CI
            CPI     BREAK
            JZ      BREAKIT         ; INTERRUPT PROCESSING
CONTINUE:
            ...
```

PL/M

```
CSTS:
    PROCEDURE BYTE;
        DECLARE IOCS LITERALLY '3812H';
            GO TO IOCS;
        END CSTS;
            ...
        DO WHILE NOT CSTS;
            ...
        END;
        IF CI<>BREAK THEN
        DO;
            ...
        END;
```

IOCHK - CHECK I/O SYSTEM CONFIGURATION

IOCHK returns an 8-bit value in the A register which describes the current assignment of physical devices to logical devices. The calling program can test selected bits in the byte returned and determine the current status of the system. For a detailed description of this I/O status byte, see section 3.3.1.

Example:

Assembly Language

```
IOCHK       EQU     3815H
            ...
            CALL    IOCHK
            ANI     CMSK    ; MASK ALL BUT CONSOLE
            SUI     CCRT    ; IS IT A CRT?
            JZ      CRTO    ; YES
```

PL/M

```
IOCHK:
      PROCEDURE BYTE;
          DECLARE IOCHECK LITERALLY '3815H';
               GO TO IOCHECK;
          END IOCHK;
                ...
      IF (IOCHK AND CMSK) = CCRT THEN
      DO;
                ...
      END;
```

IOSET - SET I/O SYSTEM CONFIGURATION

IOSET allows the user to modify the I/O status byte, thus changing the assignment of physical devices to logical devices.

Example:

Assembly Language

```
IOSET       EQU     3818H
            ...
            CALL    IOCHK           ; GET STATUS
            ANI     NOT CMSK        ; CLEAR CONSOLE BITS
            ORI     CTTY            ; CONSOLE = TTY
            MOV     C,A
            CALL    IOSET           ; STORE MODIFIED I/O BYTE
```

PL/M

```
IOSET:
      PROCEDURE (CHAR);
          DECLARE IOS LITERALLY '3818H';
          DECLARE CHAR BYTE;
               GO TO IOS;
          END IOSET;
                ...
      CALL IOSET ((IOCHK AND NOT CMSK) OR CTTY);
```

MEMCK - DETERMINE SIZE OF RAM MEMORY

This subroutine gives the user the ability to determine the highest RAM address currently available in the system. The MEMCK routine is equivalent to a PL/M address function and the 16 bit value it returns in the A and B registers is the highest address available to the user after the system monitor has allocated its own storage at the top of memory.

Example:

Assembly Language

```
MEMCK       EQU     381BH
            ...
            CALL    MEMCK
            LXI     H,MAXMEM
            MOV     M,A             ; LOW-ORDER BYTE IN A
            INX     H
            MOV     M,B             ; HIGH-ORDER BYTE IN B
```

PL/M

```
MEMCK:
    PROCEDURE ADDRESS;
        DECLARE MEMSIZ LITERALLY '381BH';
            GO TO MEMSIZ;
        END MEMCK;
            ...
    DECLARE MAXMEM ADDRESS;
            ...
    MAXMEM = MEMCK;
```

3.3.3 USER-SUPPLIED DEVICES

This section describes the necessary steps in hooking up a user-supplied I/O device to the I/O system.

The I/O subroutines described in Section 3.3.2 assume that programs (called drivers) exist which perform the actual transfer of data between I/O devices and the CPU. For instance, when the console input routine is called, it checks to see which physical device is assigned to the console, and then branches to the driver appropriate to the device. (Examples of drivers for the teletype are given in Appendix G). Therefore, when the user supplies his own device, he must:

1) Write a program to perform the data transfer, making sure that the program saves and restores any CPU registers it uses that are not specifically changed by the I/O subroutine.

2) Store a JMP to this driver's address in the appropriate location as defined in the following table:

MEMORY LOCATION	USE
3700H	USER DEFINED CONSOLE INPUT
3703H	USER DEFINED CONSOLE OUTPUT
3706H	USER DEFINED READER (1)
3709H	USER DEFINED READER (2)
370CH	USER DEFINED PUNCH (1)
370FH	USER DEFINED PUNCH (2)
3712H	USER DEFINED LIST (1)
3715H	USER DEFINED LIST (2)
3718H	USER DEFINED CONSOLE STATUS

Thus, if the user supplied a custom built listing device, he would write a driver to transfer data to it in an appropriate manner, then store the JMP to the driver's address at location 3712H. By assigning LIST=1, his device would receive any listing output generated.

3.4 ASR-33 TELETYPE DESCRIPTION

The basic input-output device for the INTELLEC 8I is the ASR-33 Teletype, which consists of a printer, keyboard, paper tape reader and paper tape punch.

The ASR-33 Teletype is pictured in Figure 3-2 and has its major controls labeled. The function of each of the controls is listed below:

CONTROL KNOB	3-position knob. OFF: turns off Teletype console. LINE: Teletype is on and attached to INTELLEC 8I as an Input-Output device. LOCAL: Teletype is on but not attached to INTELLEC 8I.
KEYBOARD	The keyboard is illustrated in Figure 3-2 and is similar to a typewriter keyboard. Several non-printing control characters are included and can be used by depressing the CONTROL key and then depressing the selected character key.
PRINTER	The printer produces a typed copy of input and output at a maximum rate of ten characters per second. When the control knob is in the LINE position, the printer types outputs transmitted from the computer. When the control knob is in the LOCAL position, the printer prints in response to keyboard operation.
TAPE PUNCH	The tape punch is used to record information on punched paper tape at a rate of 10 characters per second.
REL	Disengages the paper tape to allow loading or unloading of tape.

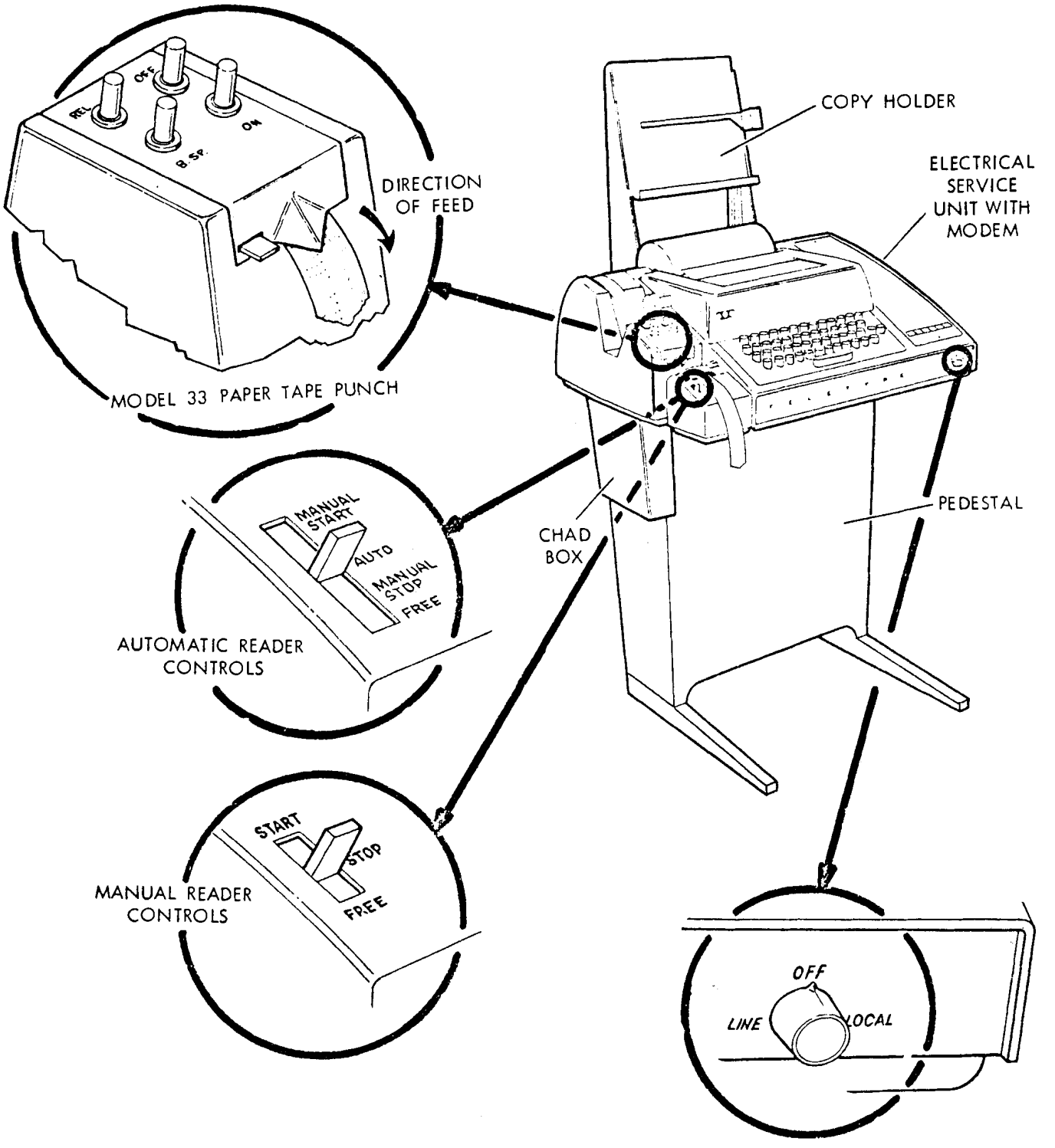


FIGURE 3-2.

ASR-33 Teletype Keyboard and Controls.

B.SP.	Backspaces the paper tape one character for each depression.
ON	Engages the tape punch and enables punching of tape.
OFF	Disengages the tape punch.
TAPE READER	The tape reader reads information punched on paper tape at a ten character per second rate.
START	Begins tape reading.
STOP	Stops tape reading.
FREE	Disengages paper tape in the reader and allows tape to be pulled through reader.

3.5 TELETYPE OPERATION

Operation of the ASR-33 Teletype is a simple task. There are few operational controls that are not self-explanatory, and so operation can be learned in an extremely short time. The more sophisticated operations are described in this section.

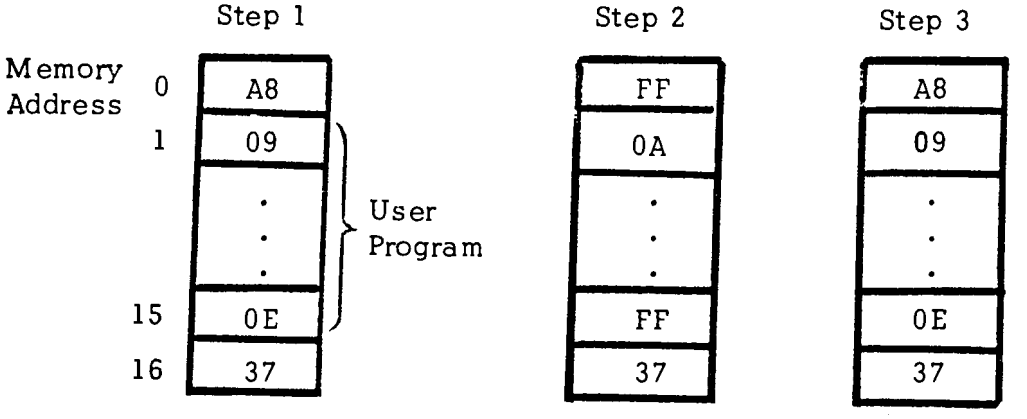
3.5.1 LOADING THE TAPE READER

To load the tape reader, first lift the plastic cover over the sprocket wheel. Place the control switch into the FREE position, and then place your tape onto the tape reader, with the sprocket holes to your left. Close the tape reader cover, ensuring that the tape is still in place.

4.0 SYSTEM MONITOR

The INTELLEC 8I System Monitor enables the operator to easily manipulate the contents of memory, read and produce paper tapes, execute programs, and read or initialize PROMs.

The System Monitor, and all INTELLEC 8I system software in general, use the first 16 memory locations for storage of temporary data. Therefore if the operator runs a program beginning in these locations, and then uses the System Monitor, Text Editor, or Assembler, he must re-load the first 16 bytes of his program before running it again. Alternatively, programs could be written beginning at location 16. Then system programs and user programs could be executed in any order, without requiring the re-load operation.



Step 1 shows the first 17 bytes of memory while a user program beginning at location 0 is being run.

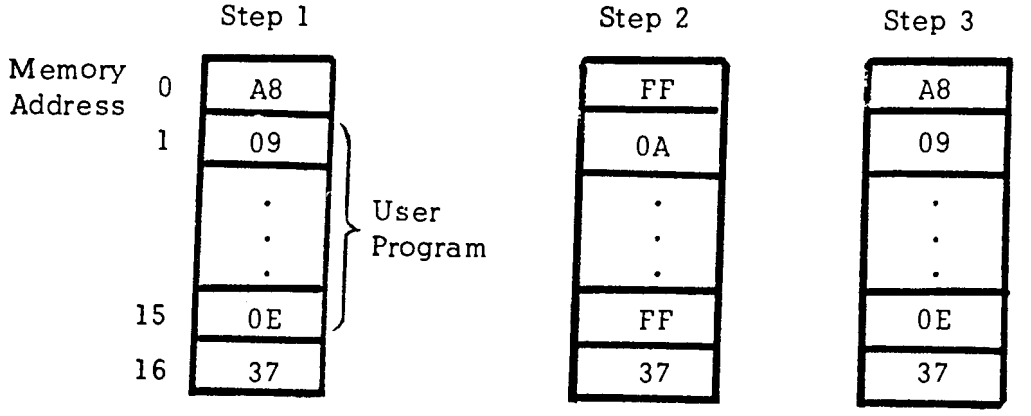
Step 2 shows these bytes after the System Monitor has been run, changing the first 16 bytes of memory (0-15). Bytes 16 on are unchanged.

Step 3 shows that the user has reloaded the first 16 bytes of his program, and is ready to execute the program again.

4.0 SYSTEM MONITOR

The INTELLEC 8I System Monitor enables the operator to easily manipulate the contents of memory, read and produce paper tapes, execute programs, and read or initialize PROMs.

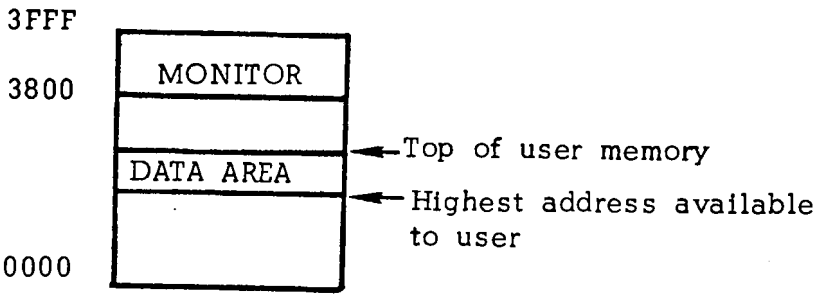
The System Monitor, and all INTELLEC 8I system software in general, use the first 16 memory locations for storage of temporary data. Therefore if the operator runs a program beginning in these locations, and then uses the System Monitor, Text Editor, or Assembler, he must re-load the first 16 bytes of his program before running it again. Alternatively, programs could be written beginning at location 16. Then system programs and user programs could be executed in any order, without requiring the re-load operation.



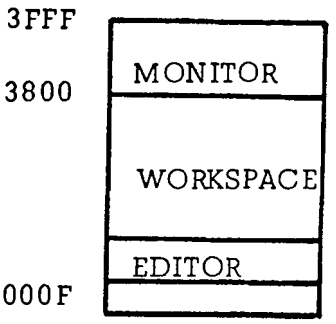
Step 1 shows the first 17 bytes of memory while a user program beginning at location 0 is being run.

Step 2 shows these bytes after the System Monitor has been run, changing the first 16 bytes of memory (0-15). Bytes 16 on are unchanged.

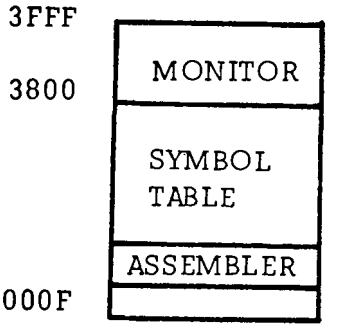
Step 3 shows that the user has reloaded the first 16 bytes of his program, and is ready to execute the program again.



a) Initial power-on



b) Load editor and generate source code in workspace.



c) Load assembler and convert source code to object code.

Figure 4-1: Normal Use and Execution Sequence for System Software

The System Monitor is the operator's interface to INTELLEC 8I, and controls loading and execution of the editor and assembler, loading and execution of user programs, and to some extent the debugging of user programs. Figure 4-1 illustrates memory utilization during various stages of system software use. While the System Monitor is running, it uses an area at the top of memory for data storage and scratch work. The user can determine how large this area is via the I/O subroutine MEMCK (described in section 3.3.2), which returns the highest memory address available for the user.

4.1 SYSTEM MONITOR IMPLEMENTATION AND EXECUTION

4.1.1 SYSTEM MONITOR IMPLEMENTATION

The INTELLEC 8I System Monitor program is implemented on eight PROM modules, which are pre-installed into each INTELLEC 8I. This allows System Monitor to be used with great ease, as it is not necessary to wait for lengthy paper-tape loading operations. All that is required to go on-line with System Monitor is to start the INTELLEC 8I, turn the Teletype on-line, perform a program jump to the execution address of System Monitor (3800), and begin execution.

4.1.2 STARTING SYSTEM MONITOR

To begin operating System Monitor, place a "Jump to 3800" command (C30038 H) into the first three locations in the INTELLEC 8I. Press the RESET button and the INTELLEC 8I will automatically jump to the starting address of System Monitor, 3800. (For the exact sequence, see Section 3.2.2).

4.2 SYSTEM MONITOR OPERATION AND COMMANDS

A monitor command consists of a single letter typed into the Teletype keyboard followed by a number of arguments, possibly none. The arguments are separated, if there are more than one, by spaces or commas. A command is terminated and executed by typing a carriage return or space, depending upon the command.

4.2.1 A COMMAND (ASSIGN I/O DEVICES)

The format of the A command is:

A ldev = pdev

ldev is one of the four logical I/O devices CONSOLE, READER, PUNCH or LIST. Only the first character is required, the rest being optional.

pdev is one of the four legal physical I/O devices corresponding to ldev, as shown in Table 4-1.

Description: The physical device pdev is assigned to logical device ldev. The following table gives all possible A commands, with optional characters shown in lower case. Numeric values for pdev indicate user-supplied devices.

AConsole = Tty
AConsole = Crt
AConsole = Batch
AConsole = 1
AReader = Tty
AReader = Ptr
AReader = 1
AReader = 2
APunch = Tty
APunch = Ptp
APunch = 1
APunch = 2
AList = Tty
AList = Crt
AList = 1
AList = 2

Table 4-1: I/O Assignment Commands

For a description of the I/O system, see Section 3.3.

Error Conditions:

If a selected physical device has not been readied or does not exist, the monitor's execution will be undefined, usually executing an infinite loop. This may be corrected by readying the device or pressing system reset.

4.2.2 B COMMAND (BNPF OUTPUT)

The format of the B command is:

B low address , high address

Low address is a valid 16-bit memory address.

High address is a valid 16-bit memory address equal to or greater than low address.

Description: The B command outputs the content of memory, from (low address) through (high address), to the Teletype printer/punch in BNPF form. Six inches of null tape are punched before and after the data.

BNPF form represents the data word in pure binary form. A B is punched to indicate the beginning of a word, P is punched to indicate a "1" bit, N is punched to indicate a "0" bit, and F is to indicate the end of a word. Thus, 'A5'H would be shown as follows:

BPNPNNPNPF

Beginning 10100101 End

For reference, the address is punched in decimal form every fourth byte. The reason the addresses are punched in decimal rather than in hexadecimal is to avoid confusion between the character 'B' punched on the tape to indicate the beginning of a byte, and the character 'B' which would have to be punched as part of some hexadecimal addresses. (See Section 4.2.7 for the legal BNPF program tape format.)

Example: If memory bytes 1 through 4 contain 2B 00 FF 55 then the command:

.B1,4(Cr)

typed into the teletype will cause the following to be punched and printed at the teletype:

```
1    BNNPNPNPPF    BNNNNNNNNF    BNPPPPPPPF
4    BNPNPNPNPF
```

Error conditions:

1. If low address or high address is greater than 16 bits, only the last 4 hex digits of the argument will be used as the address.

Example: The command:

.B3C03B00,3CA3C01 (Cr)

is equivalent to the command:

.B3B00,3C01 (Cr)

2. If low address is greater than high address, only the one byte at low address will be punched.

Example: The command:

.B4A00,3A02 (Cr)

is equivalent to the command:

.B4A00,4A00 (Cr)

3. Non-existent memory is equivalent to a string of bytes all containing FF H.

Example: If memory addresses 2000 H- 2003 H are not present in a particular INTELLEC 8I system, then the command:

.B2000,2003 (Cr)

will cause the following to be printed and punched at the teletype:

```
8192    BPPPPPPPPF    BPPPPPPPPF    BPPPPPPPPF
8195    BPPPPPPPPF
```

4. If low address or high address contains an invalid character, or if high address is omitted, the monitor will immediately type '*(Cr)(lf).' and await the next command.

Example: If the user tries to enter the number 3AGE as low address, the following will be printed:

4.2.3 C COMMAND (COMPARE PROM WITH MEMORY)

The format of the C command is:

C address

Address is a valid 16 bit memory address.

Description: The C command causes the monitor to read a PROM plugged into the console programming socket and compare its contents with a 256 byte area of memory starting at address. If the contents of the PROM and memory match completely, no message is given. Otherwise, for each location where a mismatch occurs, the memory address, memory data, and PROM data are printed.

Example: If the command

.C600(Cr)

produces the printout

0600	A3	FF
0605	6D	05

then the PROM contains the same data as memory locations 0600 H through 06FF H, except at location 0600 H and 0605 H. Memory contains A3 H and 6D H, while the PROM contains FF H and 05 H, respectively.

Error Conditions:

1. If address is greater than 16 bits, only the last 4 hex digits are used as the address.

Example: The command:

.C0ABC059(Cr)

is equivalent to the command:

.CC059(Cr)

2. If address specifies a non-existent memory range, the PROM contents are compared to 256 bytes of FF H.

Example: If locations 2000 H through 20FF H are non-existent, and the command

.C2000(Cr)

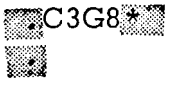
is issued, every PROM byte not equal to FF H will be printed.

3. If a PROM is not installed in the socket, memory contents are compared to 256 bytes of FF H.

Example: If a PROM is not plugged into the socket, any C command will print every byte in the memory range not equal to FF H.

4. If an invalid character is specified in address, the monitor will type '*(Cr)(lf).' and await the next command.

Example: If the user attempts to enter 3GAB as the address, the following will be printed:



4.2.4 D COMMAND (DISPLAY DATA)

The format of the D command is:

D low address , high address

Low address is a valid 16 bit memory address.

High address is a valid 16 bit memory address equal to or greater than low address.

Description: Upon execution of this command, memory data from (low address) to (high address) is displayed upon the list device (normally the Teletype). Data are displayed in hexadecimal form. Up to sixteen bytes per line are printed, preceded by the hexadecimal address of the first byte of that line. A carriage return is forced after a byte having a low order digit of F in its memory address is printed.

Example: Enter at the teletype the command:

.D10F,123(Cr)

and the teletype will type back:

```
010F   AA
0110   BB CC DD EE FF 11 22 33 44 55 66 77 88 99 AB CD
0120   EF 12 34 56
```

where memory locations 010F through 0123 are assumed to contain

AA BB CC DD EE FF 11 22 33 44 55 66 77 88 99 AB CD EF 12 34 56

The D command should be used only to examine memory contents. To punch the memory contents onto a paper tape, either the B command or the W command should be used. These commands produce a punched paper tape in the proper formats, while the D command causes only a simple sequence of characters to be output.

Error conditions:

1. If low address or high address is greater than 16 bits, only the last 4 hex digits of the argument will be used as the address.

Example: The command

.D30010,AB0013(Cr)

is equivalent to the command

.D0010,0013(Cr)

2. If low address is greater than high address, only the one byte at low address will be displayed.

Example: The command:

.D10,6

is equivalent to the command

.D10,10

3. Non-existent memory is equivalent to a string of bytes all containing FF H.

Example: If memory address 2000 H- 2010 H are invalid, then the command:

```
.D2000,2010
```

will cause the teletype to print:

```
2000   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2010   FF
```

4. If low address or high address contains an invalid character, or if high address is omitted, the monitor will immediately type '*(Cr)(lf). and await the next command.

Example: If the user attempts to enter the number 0G as an address, the following will be printed:

4.2.5 F COMMAND (FILL MEMORY WITH CONSTANT)

The format of the F command is:

F low address , high address , data

Low address is a valid 16 bit memory address.

High address is a valid 16 bit memory address equal to or greater than low address.

Data is an 8 bit data value.

Description: Execution of this command causes memory locations (low address) through (high address) to be filled with the constant (data).

Example: The command:

.F7,14,AA(Cr)

will set bytes 0007 through 0014 equal to AA H.

```
0007    AA AA AA AA AA AA AA AA AA
0010    AA AA AA AA AA
```

Error Conditions:

1. If low address or high address is greater than 16 bits (or data is greater than 8 bits), only the last 4 (or 2) hex digits will be used.

Example: The command:

.F7AB0007,0014,FFACAA(Cr)

is equivalent to the command:

.F0007,0014,AA(Cr)

2. If low address is greater than high address, data will replace only the byte at low address.

Example: If locations 7,8, and 9 contain AA H, BB H, and CC H, execution of the command:

.F7,1,33(Cr)

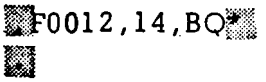
will cause memory to appear as follows:

0007 33 BB CC

3. If a non-existent memory address is specified, this command has no effect.

4. If low address, high address, or data contain an invalid character, the monitor will immediately type '*(Cr)(lf).' and await the next command.

Example: If the user tries to enter BQ as data, the following will be printed:



4.2.6 G COMMAND (GO TO)

The format of the G command is:

G address, bkptl, bkpt2

Address,bkptl, and bkpt2 are valid 16 bit hexadecimal memory addresses.

Description: The G command causes program control to be transferred to location address. If either bkptl or bkpt2 is specified, a breakpoint will be set in the program at the corresponding address(es). The specified address must correspond to the first byte of a program instruction. If either breakpoint is encountered during program execution, the System Monitor will save all program status (CPU registers and condition bits), clear all existing breakpoints, and take control. The user may then examine and/or modify registers or memory, or use any other monitor commands. This feature allows the user to debug portions of a program.

If address is not specified, the program status is restored and the saved value of the program counter is used as the new starting address.

NOTE: Encountering a breakpoint may be simulated at any time by setting the console Address/Instruction/Data switches to CFH and pressing the INT switch. All breakpoints will be cleared, status will be saved, and the System Monitor will take control.

Example: The command:

G24A

will cause program execution to begin at location 24AH, with no breakpoints being set.

The command:

G,12C

will cause a breakpoint to be set at 12CH, and program execution to resume at the address indicated by the saved value of the program counter.

The command:

G

will cause program execution to resume at the address indicated by the saved value of the program counter, with all status restored and no breakpoints set.

Error Conditions:

1. If address is greater than 16 bits, only the last 4 hex digits of the argument will be used as the address.

Example: The command:

.G3C0010(Cr)

is equivalent to the command

.G0010(Cr)

2. If address is a non-existent memory address, the system will attempt to transfer control and then stop with no response. The System Monitor must then be manually restarted.

4.2.7 H COMMAND (HEXADECIMAL ARITHMETIC)

The format of the H command is:

.H number , number Sp

Number is a 16 bit hexadecimal number.

Description: The H command is designed to aid the user in performing hexadecimal arithmetic while using the System Monitor. It causes the sum and difference its arguments to be printed in two's complement hexadecimal form. This command is terminated by a space, rather than by a carriage return.

Example:

.H1E,5C 007A FFC2

Error Conditions:

1. If either number is greater than 16 bits, only the last 4 hex digits are used.

Example: The command:

.H00ABC,23Sp

is equivalent to the command:

.H0ABC,23Sp

2. If number contains an invalid character, the monitor will immediately type '*(Cr)(lf).' and await the next command.

Example: If the user attempts to enter 01P, the following will be printed:

.H01P*

4.2.8 L COMMAND (LOAD BNPF TAPE)

The format of the L command is:

L low address , high address

Low address is a valid 16 bit memory address.

High address is a valid 16 bit memory address greater than or equal to low address.

Description: The L command loads a paper tape punched in BNPF format into memory starting at low address and continuing through high address. BNPF format represents a word in pure binary form. A B is punched to indicate the beginning of a word. Following the B, exactly eight P's and N's must be punched, P indicating a "1" bit and N indicating a "0" bit. The ninth character following the B must be an F, indicating the end of the word. All characters following the F are ignored until another B is encountered. This allows comments (not including the letter B) to appear between data words in BNPF format. For instance, the two hexadecimal data words 3AF0 could be represented on a BNPF tape as follows:

BNNPPPNPNF * COMMENT * BPPPPNNNNF

Example:

.L0,FF(Cr)

When executed, this would load the first page (256 bytes) from paper tape into memory.

Error Conditions:

1. If low address or high address is greater than 16 bits, only the last 4 hex digits of the argument will be used as the address.

Example: The command:

.L0,AF00FF(Cr)

is equivalent to the command:

.L0,00FF(Cr)

2. If low address is greater than high address, only one byte will be read from the paper tape and transferred into memory at low address.

Example: The command:

.L10,0(Cr)

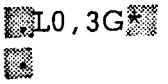
is equivalent to the command:

.L,0,0(Cr)

3. If non-existent memory is referenced, the tape will be read, but no other action will occur.

4. If low address or high address contains an invalid character, the monitor will immediately type '*(Cr)(lf).' and await the next command.

Example: If the user attempts to enter 3GAB as high address, the following will be printed:



4.2.9 M COMMAND (MOVE MEMORY)

The format of the M command is:

.M low address , high address , destination address

Low address is a valid 16 bit memory address.

High address is a valid 16 bit memory address equal to or greater than low address.

Destination address is a valid 16 bit memory address.

Description: The M command causes the block of memory from low address through high address to be moved to the locations in memory beginning at destination address.

Example: If memory appears as follows:

LOCATIONS		DATA
0300-0304	contain	01020304
0200-0204	contain	A1A2A3A4

then the command:

M200,204,300

will cause the following:

LOCATIONS		DATA
0300-0304	contain	A1A2A3A4
0200-0204	contain	A1A2A3A4

Note: The movement is performed byte by byte: the byte at low address is moved to destination address, then low address+1 is moved to destination address+1, etc. Therefore, the MOVE command may be used to fill memory with a byte or sequence of bytes.

Example: If location 0300 H contains FF H, the command

.M300,310,301(Cr)

will cause locations 300 through 310 to contain FF H. The FF at 300 is moved to 301, then the byte at 301 (which is now FF), is moved to 302, and so on.

Error Conditions:

1. If any address is greater than 16 bits, only the last 4 hex digits are used as the address.

Example: The command:

.M00302,303,00405(Cr)

is equivalent to the command:

.M302,303,405(Cr)

2. If low address is greater than high address, only one byte will be moved from low address to destination address.

Example: The command:

.M300,2F0,100(Cr)

is equivalent to the command:

.M300,300,100(Cr)

3. If low address through high address specifies a non-existent range of memory, bytes of FF H will be moved to the memory locations specified by destination address.

Example: If locations 2000 H through 2005 are non-existent, the command:

.M2000,2005,100(Cr)

will cause locations 0100 H through 0105 H to contain FF H.

4. If an invalid character is entered in an address, the monitor will type '*(Cr)(lf).' and await the next command.

Example: If the user attempts to enter 0BAG as the destination address, the following will be printed:

.M100,10F,0BAG
*

4.2.10 R COMMAND (READ HEX FILE)

The format of the R command is:

R bias address

Bias Address is a 16 bit two's complement hexadecimal number.

Description: This command loads paper tape punched in hexadecimal format (using the W command) into memory. The address at which the tape is loaded is determined by adding the address punched on the tape to the bias address using two's complement arithmetic. The bias may be negative, but in this case must be in two's complement form. If the tape was produced using an E command with a non-zero entry point address (see section 4.2.13), control will be transferred to that location in memory. Otherwise, the System Monitor will remain in control and request another command.

Example: If a tape was punched which began at location 0100 H, the following command:

.RFFB0(Cr)

will cause the tape to be read and loaded into location 50 H. (1000+FFB0= 50).

NOTE: If an error occurs while reading the tape (such as a checksum error), the monitor will immediately stop reading the tape, type '*(Cr)(Lf).' and await the next command. The operation may be retried by backing up the tape to any point before the last colon and issuing another R command, since each data word specifies the address at which it is to be loaded. The monitor will read up to the first colon it encounters, and then begin loading data.

Note that this means that, if you wish to change data in locations in memory, it is not necessary to regenerate an entirely new tape with the change; instead you may read in the original tape, then read in a patch tape which reloads only the erroneous locations.

Error Conditions:

1. If the bias address is greater than 16 bits, only the last 4 hex digits are used as the bias address.

Example: The command:

.R00FFB0 (Cr)

is equivalent to the command:

.RFFB0(Cr)

2. If an invalid character is present in the bias address, the monitor will immediately type '*(Cr)(lf).' and await the next command.

Example: If the user attempts to enter G00 as a bias address, the following will be printed:

4.2.11 S COMMAND (SUBSTITUTE MEMORY)

The S command is used to display and/or modify the contents of individual memory locations. It is used as follows:

1. Type an S, followed by the hexadecimal address of the first memory location you wish to display. Type space.

2. The data from the selected address is displayed, followed by a dash (-).

3. To modify memory, type in the new data followed by a space or a carriage return. If you do not wish to modify the contents of that location, do not type any data in, but only type a space or carriage return.

4. If a space was typed in step 3, the next memory location will be displayed as in step 2. If a carriage return was typed, operation will be returned to the System Monitor.

Example: The contents of the first four bytes of memory is 00 A1 CE FF. You wish to change it to 00 A3 CE 11.

.S0000Sp00Sp A1 -A3Sp CE -Sp FF -11Cr

User entries are unshaded. Printback is shaded.

Error Conditions:

1. If address is greater than 16 bits, or the data to be substituted is greater than 8 bits, only the last 4 or 2 hex digits respectively are used.

Example: The following sequence is equivalent to the previous example:

.S0AB0000Sp 00 -Sp A1 -BA3Sp CE -Sp FF -011Cr

2. If an invalid character is encountered, the monitor will immediately type '*(Cr)(lf). and await the next command.

4.2.12 X COMMAND (EXAMINE AND MODIFY REGISTERS)

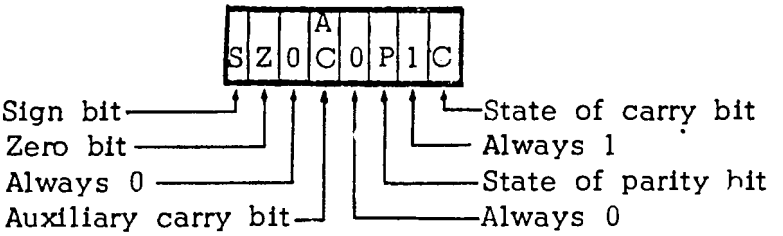
The format of the X command is:

X reg ident

Reg ident is a single character specifying a CPU register as follows:

A = A register
B = B register
C = C register
D = D register
E = E register
F = Flag byte, displayed in the form as it is stored by the instruction PUSH PSW
H = H register
L = L register
M = H and L registers combined (16 bits)
P = Program counter (16 bits)
S = Stack pointer (16 bits)

Note: The format of the flag byte F is:



Description: The X command is used to display and/or modify CPU registers. It operates similar to the S command, as follows:

1. Type an X, followed by the register identifier.
2. The data from the selected register is displayed, followed by a dash (-). Four hexadecimal digits are displayed for M, P, and S; two hex digits for the other register identifiers.
3. To modify the register, type in the new data followed by a space or a carriage return. If you do not wish to modify the register, type only the space or carriage return.
4. If a space was typed in step 3, the next register in alphabetical order is displayed. If carriage return was typed, the X command is terminated. If a space is typed after register S has been displayed, the command is terminated, this being the last register identifier in the list.

Example: The A, B, C, and D registers contain AAH, BBH, CCH, and DDH, respectively. You wish to change the B and C registers to 00H and FFH, respectively.

XASp AA-Sp BB-00Sp CC-FFSp DD-Cr

Note: Values set by the X-command will become the actual contents of the registers after execution of the next GO command.

The values displayed by the X-command are the contents of the registers prior to the execution of the last breakpoint set by the GO command. These displayed values, however, will reflect any changes of register "contents" made by the execution of X-commands since this last breakpoint.

Type an X followed by a carriage return. The contents of all internal registers will be printed out in the same form as above.

Note: Values set by the X-command will become the actual contents of the registers after execution of the next GO command.

The values displayed by the X-command are the contents of the registers prior to the execution of the last breakpoint set by the GO command. These displayed values, however, will reflect any changes of register "contents" made by the execution of X-commands since this last breakpoint.

Error Conditions:

1. If the data to be substituted is greater than 16 bits for registers M, P, S,or 8 bits for the other register identifiers, only the last 4 or 2 hex digits respectively are used.

2. If an invalid register identifier or character is encountered, the monitor will immediately type '*(Cr)(Lf).' and await the next command.

4.2.13 E COMMAND (END FILE)

The format of the E command is:

E address

Address is a valid 16 bit memory address.

Description: The E command causes an end-of-file mark and sixty null characters to be punched at the end of a hexadecimal output file. The end of file mark is hexadecimal record of length 00. (See Appendix D). If address is 0 or absent, the R command which loads the file will return control to the System Monitor. If address is non-zero, the R command will transfer control to that memory address immediately after loading the file.

4.2.14 W COMMAND (WRITE MEMORY)

The format of the W command is:

W low address ; high address

Low address is a valid 16 bit memory address.

High address is a valid 16 bit memory address equal to or greater than low address.

Description: The W command is used to output memory locations low address through high address to the system punch device in hexadecimal format. A series of W commands may be issued in order to punch various non-contiguous memory locations onto a continuous strip of tape.

Any series of W commands should be terminated with an E command in order to punch a termination character, so that when the tape is read it will be handled properly.

Example: If memory locations 1 through 3 contain 53F8EC, the command:

.W0001,0003(Cr)

produces:

:0300010053F8ECC5

(See Appendix D for an explanation of paper tape format.)

Error Conditions:

1. If low address or high address is greater than 16 bits, only the last 4 hex digits of the argument will be used as the address.

Example: The command:

WAB0010,100(Cr)

is equivalent to the command:

.W0010,100(Cr)

2. If low address is greater than high address, only the one byte at low address will be punched.

Example: The command:

.W10,0(Cr)

is equivalent to the command:

.W10,10(Cr)

3. Non-existent memory is equivalent to a string of bytes all containing FF H.

4. An invalid character in either address will cause the monitor to print '*(Cr)(lf).' and await the next command.

Example: If the user attempts to enter 3Z as low address, the following will be printed:

4.2.15 N COMMAND (NULL PUNCH)

The N command consists only of the letter N followed by a carriage return and causes 60 null characters to be punched.

4.2.16 T COMMAND (TRANSFER FROM PROM TO MEMORY)

The format of the T command is:

T address

Address is a valid 16 bit memory address.

Description: The T command causes the contents of a PROM in the front panel programming pocket to be transferred to memory beginning at address and continuing for 256 bytes.

Error Conditions:

1. If address is greater than 16 bits, only the last 4 hex digits will be used as the address.

Example: The command:

.TAB0100(Cr)

is equivalent to the command:

.T100(Cr)

2. If address specifies a non-existent memory range, no data is transferred.

3. If no PROM is present in the socket, 256 bytes of FF H will be transferred into memory.

Example: If no PROM is present, and the command:

.T100(Cr)

is issued, memory locations 0100 H through 01FF H will contain FF H.

4. If address contains an invalid character, the monitor will immediately type '*(Cr)(lf). and await the next command.

4.2.17 P COMMAND (PROGRAM PROM)

The format of the P command is:

P low address , high address , PROM address

Low Address is a valid 16 bit memory address.

High Address is a valid 16 bit memory address equal to or greater than low address.

PROM Address is an 8 bit data value.

Description: The P command causes the contents of memory from low address to high address to be programmed into a PROM module in the PROM programming socket.

Data are programmed into the PROM beginning at the PROM address.

Before attempting to program a PROM, ensure that the PROMPRGR POWER switch is ON, and that the PROM module is inserted firmly into the socket.

Error Conditions:

1. If low address or high address is greater than 16 bits, or if PROM address is greater than 8 bits, only the last 4 or 2 hex digits respectively will be used.

Example: The command:

.PAB0100,1FF,3FF(Cr)

is equivalent to the command

.P100,1FF,FF(Cr)

2. If low address is greater than high address, only one byte of data at low address will be transferred.

Example: The command:

.P400,300,0(Cr)

is equivalent to the command

.P400,400,0(Cr)

3. If low and high address refer to a non-existent memory range, a string of bytes containing FF H will be transferred to the PROM.

4. If an invalid character is typed in any address, the monitor will immediately type '*(Cr)(lf).' and await the next command.

Example: If the user attempts to type 3R as the PROM address, the following will be printed:

.P400,410,3R
*

5. If the PROM operation fails, the monitor will type a '$' and re-try the operation. Up to 3 retries will be performed, and, if still unsuccessful, the monitor will type the PROM address at which the failure occurred.

5.0 THE INTELLEC 8I TEXT EDITOR

The INTELLEC 8I Text Editor is used to create new source programs and to correct existing ones. It enables the user to create large amounts of alphanumeric text from either pre-existing paper tape or the assigned system CONSOLE device. Additions, deletions, and corrections may be made to the text on either a character-by-character or a line-by-line basis, and the resulting text may be output to either the assigned system CONSOLE device or the assigned system PUNCH device. For a complete description of the assignment of system I/O devices under the I/O System, see Section 4.2.1.

5.1 LOADING THE TEXT EDITOR

Prior to loading the Text Editor, system I/O devices should be assigned as desired by the user. If no assignments have been made, the Teletype unit will be used for all reading, printing, and punching operations. To load the Text Editor into the INTELLEC 8I memory, place the paper tape which includes the Text Editor onto the system READER device, as described in the appropriate subsection of Section 3. Enter the System Monitor, if you are not already using it, by loading C3H, 00 H, 38 H into the first three memory locations of the INTELLEC 8I and press the RESET button. (Manual loading of data is discussed in Section 3.2.2). Execute an R command; that is, type:

R (Cr)

The Text Editor source tape will automatically be loaded into the INTELLEC 8I memory.

When the paper tape has been loaded, the System Monitor will type a period in the left-hand column of the system CONSOLE device. Execute a Go To location 0010 command:

Type:

.G0010(Cr)

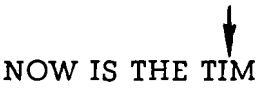
The Text Editor will then type:

INTELLEC 8 TEXT EDITOR, VERSION X.X

Start the system PUNCH device. The Text Editor will cause sixty null characters to be output to the PUNCH device and will then pause. Turn the PUNCH device off. The Text Editor will reply by typing an asterisk (*) in the left-hand column of the CONSOLE device; commands may now be entered.

5.2 TEXT EDITOR OPERATION AND COMMANDS

Text Editor operates on input from one of two sources: either the system CONSOLE device or the system READER device. The program stores the input into a memory buffer, called the workspace. A special register, called the buffer pointer, "points" to the location in the workspace from which operations are to be performed. The buffer pointer is always located between two characters. For example, suppose the workspace contains the following text (the buffer pointer is indicated by the arrow):



In this case, the buffer pointer is located between the I and the M of TIME.

The Text Editor divides the contents of the workspace by the two classifications: characters and lines. A line, to the INTELLEC 8I Text Editor, is the space between two line feed characters. A character is a single ASCII character. CARRIAGE RETURN and LINE FEED are considered to be characters by the text editor, and can be manipulated in the same way as any other characters as in the following example:

The workspace contains the following data:

ABCDEFGH(Cr)(Lf)
IJKLMNOP(Cr)(Lf)
QRSTUVWXYZ(Cr)(Lf)

Text editor divides this data into lines as follows: line 1, consisting of ABCDEFGH(Cr)(Lf); line 2, consisting of IJKLMNOP(Cr)(Lf); and line 3 consisting of QRSTUVWXYZ(Cr)(Lf).

If the first LINE FEED character were removed, Text Editor would consider the workspace as 2 lines:

line 1	ABCDEFGH(Cr)IJKLMNOP(Cr)(Lf)
line 2	QRSTUVWXYZ(Cr)(Lf)

Text Editor commands are single letters typed into the CONSOLE device keyboard in response to the asterisk printed by the program. They may have arguments associated with them, and are terminated and executed with two ESCAPE or ALT MODE characters. These characters may differ for different devices; if they do not appear on a particular device, check its manual for the equivalent character.

```
*3D4L$$
       \_____________ALT MODE KEY STRUCK TWICE
```

The ESCAPE character (rather than CARRIAGE RETURN) is used to terminate commands because of the manner in which the Text Editor processes the CARRIAGE RETURN character. Carriage returns and line feeds are treated as data by the editor, so in order to create a clear distinction between data strings and command strings, ESCAPE is used as the command terminator while LINE FEED is used as the internal line terminator.

Note: When lines are being typed into the workspace, the Editor automatically supplies a LINE FEED with each CARRIAGE RETURN typed. It is not necessary to manually insert a LINE FEED unless text is prepared off-line.

The execution of a command may be terminated at any time by typing a BREAK character. This will cause the editor to cease command execution, and return to an input mode:

```
*757T$$
 NOW IS THE
 TIME FOR ALL
 GOOD MEN
 TO ( break typed by operator )
*
```

As described in Section 5.2.1 to 5.2.4, Editor commands may be divided into four functional groups: Input commands, output commands, data manipulation commands, and buffer pointer manipulation commands.

All text editor examples in Section 5 will assume that the workspace holds the following data (unless specifically stated otherwise):

NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THEIR
COUNTRY
ABCDEFG
HIJKLMNOP
QRSTUV
WXYZ
123456
7890

The buffer pointer is assumed to be immediately preceding the word NOW, or at the beginning of the workspace.

Editor commands are presented in groups so that the reader will quickly identify commands by the purpose they serve in the normal sequence of source program generation or correction. Also, the format in which Editor commands are described differs from that used in Section 4 for System Monitor commands, since Editor commands are highly interactive and require more complex examples, most of which are in Sections 5.2.3, 5.2.4 and 5.2.5.

5.2.1 INPUT COMMANDS A (APPEND) AND I (INSERT)

Input commands cause text to be input to the workspace. There are two input commands: A, or Append, and I, or Insert.

A COMMAND (APPEND):

The format of the Append command is:

A$$

Description: The Append command causes text to be read from the assigned system READER device and to be appended to the workspace until one of the following conditions are met.

1. End of tape
2. End of file character (control-Z) read
3. Workspace full
4. 50 lines read

The Append command may be repeated until an entire input tape is read.

To execute an Append command, simply load the input tape onto the system READER device, turn the device on, and type an "A" followed by two ESCAPE characters on the CONSOLE device keyboard in response to the Text Editor asterisk. The tape will automatically begin to load in. Text Editor will type an asterisk at the left margin of the CONSOLE device when loading is finished. If the paper tape has not been loaded correctly, or if the READER device has not been turned on, Text Editor will issue an asterisk in the left-hand column, and the command must be reentered after clearing the error condition.

If there is no end-of-file on the tape, the Append will supply its own when the end of the tape is reached. In any case, no null characters or end-of-file characters will ever be put in the workspace; only the actual text characters will be put in the workspace.

I COMMAND (INSERT):

The format of the Insert command is:

I inserted text $$

Description: The Insert command causes text to be input from the CONSOLE device keyboard into the workspace. Text is inserted at the location pointed to by the buffer pointer and the buffer pointer is positioned after the last character of the inserted text.

In use, the Insert command itself is followed by an alphanumeric argument made up of the text to be inserted. For example, a typical appearance of the Insert command might be:

*INOW IS THE TIME FOR ALL GOOD

In this case, executing the I command would cause NOW IS THE TIME FOR ALL GOOD to be inserted into the workspace at the location of the buffer pointer. The buffer pointer would then be positioned immediately after the D in GOOD.

An example showing the relationship of the Insert command to the buffer pointer is given below.

The workspace originally contained the following:

THE INTELLEC 8 IS A COMPUTER

DESIGNED FOR DEVELOPMENT OF MICROCOMPUTERS

The following command was issued:

*IESPECIALLY (Sp) (Cr) (Lf)$$

The workspace now contains the following:

THE INTELLEC 8 IS A COMPUTER

DESIGNED ESPECIALLY

FOR DEVELOPMENT OF MICROCOMPUTERS

Note that the inclusion of the CARRIAGE RETURN and LINE FEED characters causes the formation of an entirely new line. If those characters had been omitted, the workspace would contain:

```
THE INTELLEC 8 IS A COMPUTER
DESIGNED ESPECIALLY FOR DEVELOPMENT OF MICROCOMPUTERS
```

The argument to an Insert command may be of any length up to the number of character spaces remaining in the workspace, and may be made up of any characters except the ESCAPE, ALT MODE, or BREAK characters.

When the inserted text is long enough to fill the workspace, the Text Editor stops echoing the inserted characters, instead echoing the BELL character. The operator should then delete characters by using the RUBOUT key, terminate the command, and either store the text on backup storage or edit the existing text as desired.

5.2.2 BUFFER POINTER MANIPULATION COMMANDS B (BEGINNING), C (CHARACTER), L (LINE), Z (END OF WORKSPACE)

Buffer pointer manipulation commands move the buffer pointer to different locations in the workspace, allowing operations to take place at any specified point in the workspace. There are four buffer pointer manipulation commands: B, or Beginning; C, or Character; L, or Line; and Z, or end of workspace.

B COMMAND (BEGINNING):

The format of the Beginning command is:

```
B$$
```

Description: The B, or Beginning, command moves the buffer pointer to the beginning of the workspace. This is useful when data is to be inserted at the front of the workspace or in order to type out the entire contents of the workspace.

Z COMMAND (END OF WORKSPACE):

The format of the End of Workspace command is:

Z$$

Description: The Z command is used to position the buffer pointer to the end of the workspace. This command is used before appending text to the end of the workspace.

C COMMAND (CHARACTER):

The format of the Character command is:

nC$$

n is a decimal number from -254 to +255. If not in this range, n is evaluated modulo 256. If not present, n is assumed to be positive 1.

Description: The C or Character command moves the buffer pointer forward or backward the number of characters specified by n. A negative argument causes the buffer pointer to move backward, and a positive argument causes the buffer pointer to move forward. If n is greater than the number of characters between the buffer pointer and the end (or beginning) of the workspace, the buffer pointer is moved to the end (or beginning) of the workspace. If n is 0, no movement occurs.

Example: Suppose the workspace contains:

ABCDEFGHIJKLMNOP

(Buffer pointer is indicated by the arrow).

A command of 3C would move the buffer pointer to between the O and P. A command of -4C would move it to between the H and I. A command of 10C would move the buffer pointer to the end of the workspace; a command of -20C would move it to the beginning of the workspace.

L COMMAND (LINE):

The format of the Line command is:

nL$$

n is a decimal number from -254 to +255. If not in this range, n is evaluated modulo 256. If not present, n is assumed to be positive 1.

Description: The L or Line command moves the buffer pointer the number of lines specified by n. A positive argument causes the buffer pointer to move forward, and a negative argument causes the buffer pointer to move backward. An argument of 0 causes the buffer pointer to move backward to the first previous LINE FEED, i.e., the beginning of the current line. If n is greater than the number of lines between the buffer pointer and the end (or beginning) of the workspace, the buffer pointer is moved to the end (or beginning) of the workspace.

Text Editor considers a line to be a character string ending with a LINE FEED character:

ABCDEFGHIJKLMNOP(Cr) is not a line, as no LINE FEED character exists.

ABCDEFG(Cr)(Lf) is a line. Note that the text editor will automatically supply a LINE FEED with a CARRIAGE RETURN. It is not necessary to manually insert a LINE FEED unless text is prepared off-line.

Numerous examples of Buffer Pointer manipulation commands use are given in Section 5.2.3 and 5.2.4.

5.2.3 OUTPUT COMMANDS T (TYPE), W (WRITE), E (END), N (NULL)

Output commands cause the text in the workspace to be output to either the system CONSOLE device or the system PUNCH device, where it is either printed or punched, depending upon the command. There are four output commands: T, or Type; W, or Write; E, or End; and N, or Null.

T COMMAND (TYPE)

The format of the Type command is:

nT$$

n is a decimal number from -254 to +255. If n is out of range, it is evaluated modulo 256. If not present, n is assumed to be positive 1.

Description: The T, or Type command causes the number of lines specified by n to be output to the system CONSOLE device. Typing starts at the current location of the buffer pointer. n indicates the number of lines to be typed, if positive, or the number of previous lines to be typed if negative. If 0, the characters from the previous LINE FEED to the current buffer pointer location will be typed. If n is greater than the number of lines existing after (or before) the buffer pointer, only the existing lines are typed.

EXAMPLE 1: TYPE OUT

Note that for this example, the buffer pointer is initially in front of the sixth line in the workspace.

```
(1) *T$$
    WXYZ
(2) *-5T$$
    NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THEIR
    COUNTRY
    ABCDEFG
    HIJKLMNOP
    QRSTUV
(3) *-4T$$
    COUNTRY
    ABCDEFG
    HIJKLMNOP
    QRSTUV
(4) *4T$$
    WXYZ
    123456
    7890
(5) *3T$$
    WXYZ
    123456
(6) 7890*
```

(1) T command - types one line after buffer pointer.

(2) -5T - types the 5 lines preceding the buffer pointer.

(3) -4T - types the 4 lines preceding the buffer pointer.

(4) 4T - types the 4 lines after the buffer pointer: Note that if the existing number of lines is less than the argument, only the existing line will be typed.

(5) 3T - types the 3 lines after the buffer pointer.

(6) Control is returned to Editor. (Note that, since the line "7890" does not end with a Cr Lf the asterisk is printed on the same line).

W COMMAND (WRITE):

The format of the Write command is:

nW$$

n is a decimal number from -254 to +255. A negative number is treated as if it were positive by this command. n is evaluated modulo 256. If n is not present, it is assumed to be positive 1.

Description: The W, or Write command causes the specified number of lines to be output to the system PUNCH device. If the Teletype is assigned as the PUNCH device, the lines will be printed as well as punched.

EXAMPLE 1: PUNCH 4 LINES

```
(1) *4W$$
(2) START PUNCH, TYPE CHAR
(3) NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THEIR
    COUNTRY
    ABCDEFG
    HIJKLMNOP
(4) *4T$$
(5) QRSTUV
    123456
(6) 7890*
```

(1) W command--punch 4 lines.

(2) Typed by Editor only if the teletype is assigned as the system PUNCH device. Allows punch to be started. Any character may be input to start punching.

(3) 4 lines are typed and punched, then Editor pauses for tape punch shut-off, if the teletype is assigned as the system PUNCH device.

(4) T command -- types 4 lines.

(5) 4 lines typed -- new beginning of workspace.

(6) Control returned to Editor.

EXAMPLE 2: NEGATIVE ARGUMENT

```
*-1W$$
NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THEIR
*
```

Note that the argument is -1, but +1 lines are typed/punched since negative arguments are treated as positive by this command.

E COMMAND (END):

The format of the End command is:

E$$

Description: The E, or End command causes the entire workspace to be output to the system PUNCH device, the workspace to be cleared, the remaining input on the input tape to be copied onto the PUNCH device, and the Editor to be reinitialized. If the Teletype is assigned as the system PUNCH device, the output will be printed as well as punched.

EXAMPLE: E COMMAND

```
(1)  *E$$

(2)  START PUNCH, TYPE CHAR
     NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THEIR
     COUNTRY
     ABCDEFG
(3)  HIJKLMNOP
     QRSTUV
     WXYZ
     123456
     7890

(4)
(5)  INTELLEC 8 TEXT EDITOR, VERSION X.X
     START PUNCH, TYPE CHAR
(6)  *
```

(1) E Command -- initiates punching or workspace.

(2) First line typed only if teletype is assigned as the system PUNCH device. Allows punch to be turned on. Any character may be input to start punching.

(3) Workspace is output to the system PUNCH device. Note that if there is a paper tape in the system READER device, and if the READER device is turned on, the paper tape would be appended to the end of the workspace output.

(4) A character must be typed to give control back to Text Editor. Turn punch off before returning control to the Text Editor.

(5) Text Editor is initialized and sixty null characters are punched. The second line is typed only if the TTY is the system PUNCH device.

(6) Control is passed to the Editor.

N COMMAND (NULL):

The format of the Null command is:

N$$

Description: The N, or Null command causes sixty null characters to be output to the system PUNCH device. This command may be used to punch leader or trailer paper tape.

EXAMPLE:

(1) [illegible]N$$
(2) [illegible]
(3) [illegible]

(1) N command - initiates punching
(2) Typed only if the teletype is assigned as the system PUNCH device. Allows PUNCH device to be turned on. After character is typed, Text Editor punches sixty null characters, and pauses to allow the PUNCH device to be turned off. Another character must be typed to return control to Text Editor.
(3) Control is passed to Editor.

5.2.4 DATA MANIPULATION COMMANDS D (DELETE), F(FIND), S (SUBSTITUTE), K(KILL)

Data manipulation commands are used to actually do the editing of the workspace material. Characters or lines may be deleted, changed or searched out. There are four data manipulation commands: D, or Delete; F, or Find, K, or Kill, and S, or Substitute.

D COMMAND (DELETE):

The format of the Delete command is:

nD$$

n is a decimal number from -254 to +255. If not in this range, n is evaluated modulo 256. If not present, n is assumed to be positive 1.

Description: The D, or Delete command causes a specified number of characters to be deleted from the workspace. The numeric argument which precedes the command determines the number of characters to be deleted, and its sign determines which direction is taken. A negative argument deletes characters preceding the buffer pointer, and a positive argument deletes characters following the buffer pointer. If n is 0, no action occurs. If n is greater than the number of characters between the buffer pointer and the end (or beginning) of the workspace, the appropriate characters are deleted and the buffer pointer is left at the end (or beginning) of the workspace.

Consider the following text in the workspace:

If the command 4D were executed, the end result would be:

COMPUTER PROGRA

If the command -6D were executed, the end result would be:

COMPUTER MMER

(Remember, the space is considered to be a character by the Editor).

EXAMPLE 1:

```
(1)  *3L$$
(2)  *4C$$
(3)  *-2D$$
(4)  *B$$
(5)  *4T$$
     NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THEIR
(6)  COUNTRY
     ABCDEFG
     HILMNOP
(7)  *
```

(1) Buffer pointer is moved three lines down; i.e., to the beginning of HIJKLMNOP
(2) Buffer pointer is moved forward four characters; i.e., to just before L.
(3) Two characters immediately before the buffer pointer are deleted.
(4) Buffer pointer is moved to the beginning of the workspace.
(5) Four lines are typed, to observe the changes made in steps 1-3.
(6) Note that J and K were deleted.
(7) Editor is again ready to accept commands.

EXAMPLE 2:

```
(1)  *2L$$
(2)  *3C$$
(3)  *3D$$
(4)  *B$$
(5)  *3T$$
     NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THEIR
(6)  COUNTRY
     ABCG
(7)  *
```

(1) Buffer pointer is moved two lines down to the beginning of ABCDEFG.
(2) Buffer pointer is moved three characters forward.
(3) Three characters are deleted.
(4) The buffer pointer is moved to the beginning of the workspace.
(5) Three lines are typed.
(6) Note the deletion of D, E and F.
(7) Editor is ready to accept commands.

K COMMAND (KILL):

The format of the Kill command is:

nK$$

n is a decimal number from -254 to +255. If not in this range, n is evaluated modulo 256. If not present, n is assumed to be positive 1.

Description: The K, or Kill command performs the same operation as the D command, except that it works on a line-by-line basis instead of a character-by-character basis. If n is 0, the characters from the buffer pointer to the first previous line feed will be deleted. If n is greater than the number of lines between the buffer pointer and the end (or beginning) of the workspace, the lines will be deleted and the buffer pointer will be at the end (or beginning) of the workspace.

EXAMPLE 1:

```
(1)  *2K$$
(2)  *3T$$
     ABCDEFG
(3)  HIJKLMNOP
     QRSTUV
(4)  *1L$$
(5)  *2K$$
(6)  *B$$
(7)  *3T$$
     ABCDEFG
(8)  WXYZ
     123456
(9)  *
```

(1) Two lines are deleted.
(2) Three lines are typed.
(3) Note that NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THEIR and COUNTRY have been deleted.
(4) The buffer pointer is moved down one line.
(5) Two lines are deleted.
(6) The buffer pointer is moved to the beginning of the workspace.
(7) Three lines are typed.
(8) Note the deletion of HIJKLMNOP and QRSTUV.
(9) Editor is ready to accept more commands.

EXAMPLE 2:

```
(1) *4L$$
(2) *-2K$$
(3) *B$$
(4) *3T$$
    NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THEIR
(5) COUNTRY
    QRSTUV
(6) *
```

(1) The buffer pointer is moved forward four lines; i.e., to the beginning of QRSTUV.
(2) The two lines preceding the buffer pointer are deleted.
(3) The buffer pointer is moved to the beginning of the workspace.
(4) Three lines are typed.
(5) Note the deletion of ABCDEFG and HIJKLMNOP
(6) Editor is ready to accept new commands.

F COMMAND (FIND):

The format of the Find command is:

Ftext$$

text is a string of length 16 characters or less, including any characters except ESCAPE, ALT MODE, or BREAK.

Description: The F, or Find command, causes the Editor to search for the first occurrence of a character string in the workspace. The string to be searched out appears as an alphanumeric argument after the initial F. The Editor will begin its search at the present location of the buffer pointer and will conclude upon reaching either of the following points:

(1) A match is found. The buffer pointer is then positioned immediately after the matched character string.

(2) The end of workspace is encountered. Text Editor then types the message "CANNOT FIND" and the search string on the system CONSOLE device. The buffer pointer is not affected. If the string to be found is greater than 16 characters, a match will never occur.

EXAMPLE 1:

NOTE: Line 2 has been changed to PARTY instead of COUNTRY, and the buffer pointer is an undetermined position.

```
(1)   *FPARTY$$
(2)   CANNOT FIND "PARTY"
(3)   *B$$
(4)   *FPARTY$$
(5)   *IAS WELL AS THEIR COUNTRY.$$
(6)   *B$$
(7)   *2T$$
(8)   NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THEIR
      PARTYAS WELL AS THEIR COUNTRY. . .
(9)   *FPARTY$$
(10)  *I $$
(11)  *B$$
(12)  *2T$$
(13)  NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THEIR
      PARTY AS WELL AS THEIR COUNTRY.
(14)  *
```

(1) Editor is told to search for the string "PARTY".
(2) "PARTY" was not found. Either it does not exist within the workspace or the buffer pointer was located after the last occurrence of "PARTY".
(3) The buffer pointer is moved to the beginning of the workspace.
(4) "PARTY" is searched for again.
(5) "PARTY" is found and "AS WELL AS. . ." is inserted immediately after it.
(6) The buffer pointer is reset to the beginning.
(7) Two lines are typed.

(8) Note that since the buffer pointer was left, after the search, at the end of "PARTY" and no space was inserted, "AS" is concatenated with "PARTY".
(9) "PARTY" is searched for again.
(10) A space is inserted immediately after "PARTY".
(11) The buffer pointer is reset.
(12) Two lines are typed.
(13) The new line is checked.
(14) Editor is ready to accept more commands.
(15) *FPARTY$$
(16) *10D$$
(17) *B$$
*2T$$
(18) NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THEIR PARTYS THEIR COUNTRY.

(15) "PARTY" is searched for again.
(16) The ten characters following the buffer pointer are deleted.
(17) The first two lines are typed.
(18) Note that the first ten characters after the Y in PARTY are deleted, including spaces.

S COMMAND (SUBSTITUTE):

The format of the Substitute command is:

S search string $ substitute string $$

search string and substitute string are strings of characters including any characters except ALT MODE, ESCAPE or BREAK.

Description: The S or Substitute command searches the workspace for the first occurrence of a specified character string of up to 16 characters and substitutes another string for it, if the search is successful. The search begins at the present location of the buffer pointer, and ends either when the string is found, or at the end of the workspace, in which case the message "CANNOT FIND" and the search string is typed on the system CONSOLE device. If the search if successful, the buffer pointer is positioned after the substituted string. If the search is unsuccessful, the buffer pointer is unaffected.

If search string is greater than 16 characters, only the first 16 are used.

Even if substitute string is greater than 16 characters, the entire substitute string will replace the search string.

Example:

```
(1) *SCOUNTRY$PARTY$$
(2) *-1T$$
(3)  NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THEIR
     PARTY
     *
```

(1) The workspace is searched for the string "COUNTRY" and, when "COUNTRY" is found, substitutes the string "PARTY" for it.
(2) The line immediately preceding the buffer pointer is typed.
(3) Note the occurrence of "PARTY" rather than "COUNTRY".

5.2.5 COMMAND STRINGS

Text Editor commands may be chained together to form command strings. Strings may be made up of any number of commands and will be executed as individual commands from left to right. An example of a command string is

```
*B3L1K4DICORRECTION
```

which would, in order of execution:

Position the buffer pointer at the beginning of the workspace;

Move it down three lines;

Delete one line;

Delete four characters;

Insert the string "CORRECTION" into the workspace.

Commands may optionally be separated from each other with ESCape or ALT MODE characters (echoed as dollar signs); S, F and I commands must be separated from other commands in this manner, for otherwise Text Editor would have no means of distinguishing between the text to be inserted or searched and the next command: a string such as

```
IINSERT3L4D
```

would cause INSERT3L4D to be placed into the workspace, while

```
IINSERT$3L$4D
```

would cause INSERT to be inserted, the buffer pointer to be advanced three lines and four characters to be deleted.

Example 1:

```
*B3T$$
NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THEIR
COUNTRY
ABCDEFG
*
```

The command string B3T causes the buffer pointer to be reset to the beginning of the workspace and then types the three lines immediately after the buffer pointer. The net effect is to type the first three lines in the workspace.

Example 2:

```
      *4L$$
      *-2C$$
      *-4D$$
      *-2T$$
       COUNTRY
       ABCDEFG
(1)    HIJKL*B3L3T$$
       HIJKL
       ORSTUV
       WXYZ
(2)    *B4L-2CIMNOP$$
(3)    *B3L3T$$
       HIJKLMNOP
       ORSTUV
       WXYZ
       *
```

(1) The string B3L3T causes the third, fourth, and fifth lines to be typed.
(2) The string B4L-2XIMNOP causes the buffer pointer to be positioned immediately after the L in the third line (B4L-2C), and the string MNOP to be inserted. Note that a line is ended with the two characters CARRIAGE RETURN and LINE FEED.
(3) The first three lines are typed out.

The commands listed as 2 and 3 could have been combined effectively. The resulting command string B4L-2CIMNOP$B3L3T would cause the same series of operations to be performed as the two separate strings B4L-2CIMNOP and B3L3T. Note, however, that the I command is separated from the B command by an ESCape or ALT MODE character, echoed as a dollar sign.

5.2.6 USE OF TABS

The Text Editor allows the use of the horizontal tab character, (created by typing the CTRL and I keys simultaneously) wherever a space could be used. This allows the operator to produce a more readable listing. Tab stops are located every eight positions.

Example:

The editor command:

```
*ILABEL"tabMOVtabA,Btab;SpCOMMENTS$$
```

is printed as:

```
COL     0       1       2
01234567890123456789012345678 90

LABEL:  MOV     A,B     ; COMMENT
        JMP     HOME
```

5.2.7 COMMAND ITERATION

The Text Editor allows a specified command string to be automatically executed a specified number of times. This is done by command iteration.

In order to cause a command string to be automatically executed more than once, angle brackets (< and >) should be used to surround the command:

n< command string >$$

The number immediately preceding the left bracket (<), n, determines the number of times the command is executed. n is evaluated modulo 256.

EXAMPLE:

(1) *4<IHELLO.CrLf
R>$B4T$$
(2) HELLO
HELLO
HELLO
HELLO
(3) *

(1) The command is entered. This command will insert the string HELLO. CrLf four successive times, move the buffer pointer to the beginning of the workspace, and type the first four lines of the workspace.
(2) The first four lines of the workspace are typed. Note that the string HELLO.CrLf has, indeed, been inserted four times.
(3) Control is returned to the Editor.

Command iterations may be nested up to eight levels deep, by placing a bracketed command string inside of another bracketed string.

EXAMPLE:

(1) *2<IA$2< IB$>>0LT$$
(2) ABBABB
(3) *

(1) The command is entered. This command will cause, in order, "A" to be inserted, "B" to be inserted twice, "A" to be inserted, "B" to be inserted twice more, the buffer pointer to be moved to the beginning of the line, and the line to be typed.
(2) The line is typed. Note that the command 2<IB$> caused B to be inserted twice each time it was encountered, and that the overall command IA$2<IB$> was executed twice.

(3) Control is returned to the Editor.

If command iterations are nested more than eight deep, the Text Editor will print the error message "ITERATION STACK FAULT" on the system CONSOLE device, and the command will have to be reentered with the error condition removed.

EXAMPLE:

```
(1) *2<IA$2<IB$2<IC$2<ID$2<IE$2<IF$2<IG$2<IH$2<II$>>>>>>>>>$$
(2) ITERATION STACK FAULT
(3) *2<IA$2<IB$2<IC$2<ID$2<IE$2<IF$2<IG$2<IHII$>>>>>>>>$$
(4) *0L$T$$
(5) ABCDEFGHIIHIIGHIIHIIFGHIIHIIEFGHIIHIIDEFGHIIHIIGDEFGHIIHIBBCDEFGHIIHII
    ABCDEFGHIIHII
(6) *
```

(1) The command is entered. Note that there are nine levels of iteration.
(2) Text Editor types the error message, and terminates command execution.
(3) Text Editor types an asterisk. The new command is entered. Note that there are now only eight levels of iteration.
(4) The buffer pointer is reset, and the line is typed.
(5) The typed line.
(6) Control is returned to the Text Editor.

6.0 THE INTELLEC 8I ASSEMBLER

This section will deal with the loading and execution of the INTELLEC 8I assembler source tape. For more information on the assembly language itself, consult the 8080 Programmer's Manual. See Appendix A for a summary of pseudo operations and the assembly language instruction set.

6.1 LOADING THE ASSEMBLER

To load the Assembler into the INTELLEC 8I memory, place the paper tape containing the Assembler onto the system READER device as described in the appropriate subsection of Section 3. Ensure that the READER device is turned on. Enter the System Monitor, if it is not already running, by loading the data C3H, 00 H, and 38 H into the first three locations in memory, and pressing the RESET button. When System Monitor is running, assign I/O devices to the system READER, CONSOLE, LIST and PUNCH devices as desired. For complete details of I/O device assignment under the I/O System, see Section 4.2.1. Then execute an R command; that is, type:

.R(Cr)

The Assembler source tape will then be loaded into the INTELLEC 8I memory.

When the paper tape is completely loaded into memory, System Monitor will type a period (.) in the left-hand column of the CONSOLE device. When this occurs, enter a Go To 0010 command:

.G0010 (Cr)

The INTELLEC 8I Assembler will begin operations.

6.2 EXECUTION OF THE ASSEMBLER

The INTELLEC 8I Assembler is a two or three pass assembler, depending upon the devices selected as the system LIST and system PUNCH devices. If these devices can be operated independently, then two passes are necessary for complete assembly; that is, the input paper tape must be read in twice in order to produce the final paper tape containing the machine-coded instructions. If the LIST device and the PUNCH device cannot be operated independently, as is the case if the Teletype is

assigned to both devices, the Assembler operates as a three-pass assembler and the input paper tape must be read in three times.

When the Assembler begins operations, it will print P= on the CONSOLE device. Load the input paper tape into the READER device, make sure that the READER device is turned on, and type 1 on the CONSOLE device.

The input tape will be read in, and the assembler will process the first pass, which sets up the symbol tables for the program.

When the Assembler has finished processing the first pass, it will again print P= on the CONSOLE device. Reload the input paper tape on the READER device, and type in either 2 (if the LIST and PUNCH devices cannot be operated independently) or 4 (if the LIST and PUNCH devices may be operated independently). The input paper tape will be read in again, and the Assembler will begin processing the second pass.

If the second pass was started by typing 4 in response to the Assembler query, it will perform two functions: first, it will create a listing of the program, including any error messages which may have been generated during Assembly. This listing will be printed on the LIST device. Second, it will output the machine-coded instructions to the PUNCH device to create a hexadecimal format tape containing the assembled program.

If 2 was entered at the beginning of the second pass, then the second pass will only output the program listing to the LIST device. The Assembler will again print P= on the CONSOLE device. The input paper tape must be reloaded, and 3 typed on the CONSOLE device. The input paper tape will be read again, and the hexadecimal format machine-coded program will be output to the PUNCH device.

6.3 ASSEMBLER LIMITS

The following facts should be remembered while creating source programs for the INTELLEC 8I Assembler:

(1) There is no limit as to the number of characters in any one assembler statement; however only 72 characters per line will be printed on the teletype.

(2) There is no limit as to the number of characters in any one symbol; however, the first five characters of any symbol must be unique.

(3) The maximum number of parameters that may be associated with any macro is 126.

(4) Only a finite amount of memory is available to the assembler in which to store its symbol table. Therefore, assemblies containing excessive numbers of symbols and macros may cause an overflow of the symbol table, invalidating the assembly.

Each symbol requires 8 bytes of memory in the symbol table, and every character of every macro definition is stored in the symbol table. In addition, every macro reference requires some symbol table memory (the precise amount varies with the environment of the macro reference).

If assemblies are being made on an INTELLEC 8I system with limited memory, then, the macro definitions and number of symbols should be as small as possible.

6.4 SYNTAX RESTRICTIONS

The following rules must be observed when writing source programs for the INTELLEC 8I Assembler:

(1) Symbols which appear as names in EQU or SET operations must be defined before being used in expressions or operands.

Thus the instruction sequence:

```
FIRST:    DB      SYM*3
SYM       EQU     2
```

is invalid.

(2) A macro must be defined before any references to it are made. Thus the instruction sequence:

```
REF:      LOAD    A,B       ; MACRO REFERENCE
          :
LOAD      MACRO   P1,P2     ; MACRO DEFINITION
          :
          ENDM
```

is invalid.

(3) Macros may not be defined within other macro definitions; that is, a MACRO pseudo instruction may not appear between any MACRO and ENDM pseudo instructions.

Thus the sequence:

```
MAC1        MACRO
              .
              .
              .
MAC2        MACRO
              .
              .
              .
            ENDM
              .
              .
              .
            ENDM
```

is invalid.

(4) The parameters listed in the operand field of a MACRO pseudo-instruction may not be symbols previously defined in the assembly.

Thus the sequence:

```
P1:         EQU       5
              .
              .
              .
MAC1        MACRO     P1,P2
```

is invalid.

6.5 ASSEMBLER ERROR MESSAGES

Errors detected by the assembler are indicated by single letter codes printed at the teletype during pass 2. These codes are as follows:

A ADDRESS ERROR:

Description: This error indicates that the address referenced by a JUMP or CALL instruction is not in the range 0 to 65535.

Example:

```
JUMP        65536D      ; A ERROR
JUMP        -2          ; A ERROR
```

B BALANCE ERROR:

Description: This error indicates that the parentheses in an expression are unbalanced, or that the quotes in an ASCII string are unbalanced.

Example:

```
MVI     H,3*(2+(4)      ; B Error
DB      ''A'            ; B Error
```

E EXPRESSION ERROR:

Description: This error indicates a badly constructed expression. It usually occurs due to a missing operator, omitted comma, or a misspelled opcode.

Example:

```
MVI     H,3 (2+4)       ; E Error, Missing Operator
MVO     H,M             ; E Error, Mispelled Opcode
```

F FORMAT ERROR:

Description: This error indicates an error in the format of a statement. It is usually caused by a missing operand or an extraneous operand.

Example:

```
MOV     ,D              ; F Error, Missing Operand
MOV     A,B,C           ; F Error, Extraneous Operand
```

I ILLEGAL CHARACTER:

Description: This error indicates that an invalid ASCII character is present in the statement. It is also caused by a numeric character which is too big for the base of the number in which it occurs.

Example:

```
MVI     H,02B           ; I Error, 2 Invalid in Binary
MVI     H,79Q           ; I Error, 9 Invalid in Octal
```

M MULTIPLE DEFINITION:

Description: This error indicates that two or more labels exist which are identical or not unique in the first five characters.

NOTE: Identical labels generated by macro references are legal.

Example: The following segment of code is illegal:

```
LABEL1:     INC     B
            :
            :
LABEL2:     INC     C           ;M ERROR
```

N NESTING ERROR:

Description: This error indicates that an IF, ENDIF, MACRO, or ENDM statement is improperly nested.

Example: The statement:

```
            ENDIF
```

will cause an N error if no IF statement precedes it in the program.

P PHASE ERROR:

Description: This error indicates that the value of an element being defined changed between pass one and pass two of the assembly.

Example: The following segment of code will cause every label in the assembly to produce a P error:

```
            ORG     BEGIN
            :
            :
BEGIN       EQU     5
```

During pass one, the symbol BEGIN is undefined when the ORG is reached. The assembler will assume it to be 0, and begin assembling the program at 0. During pass two, the symbol BEGIN is equal to 5. Therefore the location assigned to every label in the program will have increased by 5, producing a P error.

Q QUESTIONABLE SYNTAX:

Description: This error is usually caused by omitting an opcode.

Example:

```
LOC         1234H,5AB3H         ;Opcode DW was omitted.
```

R REGISTER ERROR:

Description: This error indicates that a register specified for an operation is invalid for that operation.

Example:

```
        PUSH        A           ; R ERROR, A INVALID FOR PUSH
```

S STACK OVERFLOW:

Description: This error indicates that the assembler's internal expression evaluation stack became too large and overflowed the memory available to the assembler. It may be caused by using extremely long character strings, too many nested macros, too many nested IF statements, or expressions which are too complex.

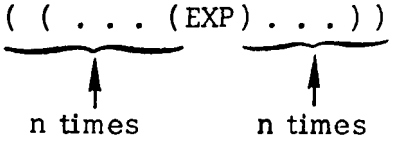
Example: A nested IF statement is one which occurs between another IF - ENDIF pair. Thus long sequences of the form:

```
IF          EXP
:
IF          EXP
:
IF          EXP
:
ENDIF
ENDIF
ENDIF
:
```

may cause an S error.

The assembler can evaluate complex expressions as long as they do not contain unnecessary parentheses.

An expression of the form:



will cause an S error when n becomes large enough.

T TABLE OVERFLOW:

Description: This error indicates that the assembler's symbol table space has been exhausted. This may be caused by using too many symbols, overly lengthly macro definitions, or too many macro references in an assembly.

U UNDEFINED IDENTIFIER:

Description: This error indicates that a symbol used in an operand field was never defined by appearing in the label field of another instruction.

Example: If the statement

```
        MVI     H,LAB1
```

appears is an assembly, and the symbol LAB1 does not appear in the label field of any other statement, it will cause a U error.

V ILLEGAL VALUE:

Description: This error indicates that the value of an operand or expression exceeds the range required for a particular operation.

Example: The statement:

```
        RST  8
```

will cause a V error, because the operand of a RST instruction must be in the range 0 to 7.

The statements:

```
                ORG     256
LOC:            INCR    B
                :
                :
                MVI     H,LOC
```

will cause a V error, since the value of the second operand of an MVI instruction must be an 8 bit quantity, while the value of LOC is a 16 bit address which cannot be held in 8 bits.

APPENDIX A

-- INSTRUCTION SUMMARY --

This appendix provides a summary of 8080 assembly language instructions. Abbreviations used are as follows:

A	The accumulator (register A)
A_n	Bit n of the accumulator contents, where n may have any value from 0 to 7 and 0 is the least significant (rightmost) bit.
ADDR	Any memory address
Aux. carry	The auxiliary carry bit
Carry	The carry bit
CODE	An operation code
DATA	8 bits (one byte) of data
DATA16	16 bits (2 bytes) of data
DST	Destination register or memory byte
EXP	A constant or mathematical expression
INTE	The 8080 interrupt enable flip-flop
LABEL:	Any instruction label
M	A memory byte
Parity	The parity bit
PC	Program Counter
PCH	The most significant 8 bits of the program counter
PCL	The least significant 8 bits of the program counter
REGM	Any register or memory byte

RP	**A register pair. Legal register pair symbols are:** **B for registers B and C** **D for registers D and E** **H for registers H and L** **SP for the 16 bit stack pointer** PSW for condition bits and register A
RP1	**The first register of register pair RP**
RP2	**The second register of register pair RP.**
sign	**The sign bit**
SP	**The 16-bit stack pointer register**
SRC	**Source register or memory byte**
zero	**The zero bit**
XY	**The value obtained by concatenating the values X and Y**
[ ]	**An optional field enclosed by brackets**
( )	**Contents of register or memory byte enclosed by parentheses**
←	**Replace value on lefthand side of arrow with value on right-hand side of arrow**

CARRY BIT INSTRUCTIONS

Format:

[LABEL:] CODE

CODE	DESCRIPTION		
STC	(carry) ←	1	Set carry
CMC	(carry) ←	$\overline{\text{(carry)}}$	Complement carry

Condition bits affected: Carry

SINGLE REGISTER INSTRUCTIONS

Format:

```
[LABEL:]    INR     REGM
            -or-
[LABEL:]    DCR     REGM
            -or-
[LABEL:]    CMA
            -or-
[LABEL:]    DAA
```

Code	Description	
INR	(REGM) ← (REGM)+1	Increment register REGM
DCR	(REGM) ← (REGM)-1	Decrement register REGM
CMA	(A) ← $\overline{(A)}$	Complement accumulator
DAA	If $(A_0\text{-}A_3) > 9$ or (aux. carry)=1, (A) ← (A)+6 Then if $(A_4\text{-}A_7) > 9$ or (carry)= 1 (A) = (A) + 6 * 2^4	Convert accumulator contents to form two decimal digits

Condition bits affected: INR,DCR : Zero, sign, parity
CMA : None
DAA : Zero, sign, parity, carry, aux. carry

NOP INSTRUCTION

Format:

```
[LABEL:]    NOP
```

Code	Description
NOP	- - - - - - - - No operation

Condition bits affected: None

DATA TRANSFER INSTRUCTIONS

Format:

```
[LABEL:]    MOV     DST,SRC
            -or-
[LABEL:]    CODE    RP
```

NOTE: SRC and DST not both = M

NOTE: RP = B or D

Code	Description	
MOV	(DST) ← (SRC)	Load register DST from register SRC
STAX	((RP)) ← (A)	Store accumulator at memory location referenced by the specified register pair
LDAX	(A) ← ((RP))	Load accumulator from memory location referenced by the specified register pair

Condition bits affected: None

REGISTER OR MEMORY TO ACCUMULATOR INSTRUCTIONS

Format:

```
[LABEL:]    CODE    REGM
```

Code	Description	
ADD	(A) ← (A)+(REGM)	Add REGM to accumulator
ADC	(A) ← (A)+(REGM)+(carry)	Add REGM to accumulator with carry
SUB	(A) ← (A)-(REGM)	Subtract REGM from accumulator
SBB	(A) ← (A)-(REGM)-(carry)	Subtract REGM from accumulator with borrow
ANA	(A) ← (A) AND (REGM)	AND accumulator with REGM
XRA	(A) ← (A) XOR (REGM)	EXCLUSIVE-ORaccumulator with REGM

Code	Description	
ORA	(A) ← (A) OR (REGM)	OR accumulator with REGM
CMP	Condition bits set by (A)-(REGM)	Compare REGM with accumulator

Condition bits affected:

ADD, ADC, SUB, SBB: Carry, sign, zero, parity, aux. carry
ANA, XRA, ORA: Sign, zero, parity. Carry is zeroed.
CMP: Carry, sign, zero, parity, aux. carry. Zero set if (A)=(REGM)
Carry reset if (A) < (REGM)
Carry set if (A) ≥ (REGM)

ROTATE ACCUMULATOR INSTRUCTIONS

Format:

[LABEL:] CODE

Code	Description	
RLC	(carry) ← A_7, A_{n+1}, ← A_n, A_0 ← A_7	Set carry = A_7, rotate accumulator left
RRC	(carry) ← A_0, A_n ← A_{n+1}, A_7 ← A_0	Set carry = A_0, rotate accumulator right
RAL	A_{n+1} ← A_n, (carry) ← A_7, A_0 ← (carry)	Rotate accumulator left through the carry
RAR	A_n ← A_{n+1}, (carry) ← A_0, A_7 ← (carry)	Rotate accumulator right through carry

Condition bits affected: Carry

REGISTER PAIR INSTRUCTIONS

Format:

[LABEL:] CODE1 RP
-or-
[LABEL:] CODE2

Note: For PUSH and POP, RP=B,D,H, or PSW
For DAD, INX, and DCX, RP=B,D,H, or SP

Code1	Description	
PUSH	((SP)-1)←(RP1), ((SP)-2)←(RP2), (SP)←(SP)-2	Save RP on the stack RP=A saves accumulator and condition bits.
POP	(RP1)←((SP)+1), (RP2)←((SP)), (SP)←(SP)+2	Restpre RP from the stack RP=A restores accumulator and condition bits.
DAD	(HL)←(HL) + (RP)	Add RP to the 16-bit number in H and L.
INX	(RP)←(RP)+1	Increment RP by 1
DCX	(RP)←(RP)-1	Decrement RP by 1

Code2	Description	
XCHG	(H)←→(D), (L)←→(E)	Exchange the 16 bit number in H and L with that in D and E.
XTHL	(L)→((SP)), (H)→((SP)+1)	Exchange the last values saved in the stack with H and L.
SPHL	(SP)←(H):(L)	Load stack pointer from H and L.

Condition bits affected:

PUSH, INX, DCX, XCHG, XTHL, SPHL: None
POP : If RP=PSW, all condition bits are restored from the stack, otherwise none are affected.
DAD : Carry

IMMEDIATE INSTRUCTIONS

Format:

```
[LABEL:]    LXI     RP, DATA16
            -or-
[LABEL:]    MVI     REGM, DATA
            -or-
[LABEL:]    CODE    REGM
```

Note: RP=B,D,H, or SP

CODE	DESCRIPTION	
LXI	(RP)—— DATA 16	Move 16 bit immediate Data into RP
MVI	(REGM)———DATA	Move immediate DATA into REGM
ADI	(A)—— (A) + DATA	Add immediate data to accumulator
ACI	(A)—— (A) + DATA + (carry)	Add immediate data to accumulator with carry
SUI	(A)—— (A) - DATA	Subtract immediate data from accumulator
SBI	(A)—— (A) - DATA - (carry)	Subtract immediate data from accumulator with borrow
ANI	(A)—— (A) AND DATA	AND accumulator with immediate data
XRI	(A)—— (A) XOR DATA	EXCLUSIVE-OR ccumulator with immediate data
ORI	(A)—— (A) OR DATA	OR accumulator with immediate data
CPI	Condition bits set by (A)-DATA	Compare immediate data with accumulator

Condition bits affected:

LXI, MVI: None
ADI, ACI, SUI, SBI: Carry, sign, zero, parity, aux. carry
ANI, XRI, ORI: Zero, sign, parity. Carry is zeroed.
CPI: Carry, sign, zero, parity, aux. carry.

Zero set if (A) = DATA
Carry reset if (A) < DATA
Carry set if (A) $\geqslant$ DATA

DIRECT ADDRESSING INSTRUCTIONS

Format:

[LABEL:] CODE ADDR

CODE	DESCRIPTION	
STA	(ADDR) ——— (A)	Store accumulator at location ADDR
LDA	(A) —— (ADDR)	Load accumulator from location ADDR
SHLD	(ADDR) — (L), (ADDR+1) — (H)	Store L and H at ADDR and ADDR+1
LHLD	(L)—— (ADDR), (H)— (ADDR+1)	Load L and H from ADDR and ADDR+1

Condition bits affected: None

JUMP INSTRUCTIONS

Format:

[LABEL:] PCHL

-or-

[LABEL:] CODE ADDR

CODE	DESCRIPTION	
PCHL	(PC)←— (HL)	Jump to location specified by register H and L
JMP	(PC)←——ADDR	Jump to location ADDR
JC	If (carry) = 1, (PC)←— ADDR If (carry) = 0, (PC)←— (PC)+3	Jump to ADDR if carry set
JNC	If (carry) = 0, (PC)←— ADDR If (carry) = 1, (PC)←— (PC)+3	Jump to ADDR if carry reset
JZ	If (zero) = 1, (PC)←— ADDR If (zero) = 0, (PC)←— (PC)+3	Jump to ADDR of zero set
JNZ	If (zero) = 0, (PC)←— ADDR If (zero) = 1, (PC)←— (PC)+3	Jump to ADDR if zero reset
JP	If (sign) = 0, (PC)←— ADDR If (sign) = 1, (PC)←— (PC)+3	Jump to ADDR if plus
JM	If (sign) = 1, (PC)←— ADDR If (sign) = 0, (PC)←— (PC)+3	Jump to ADDR if minus
JPE	If (parity)= 1, (PC)←— ADDR If (parity)= 0, (PC)←— (PC)+3	Jump to ADDR if parity even
JPO	If (parity)= 0, (PC)←— ADDR If (parity)= 1, (PC)←— (PC)+3	Jump to ADDR if parity odd

Condition bits affected: None

CALL INSTRUCTIONS

Format:

[LABEL:] CODE ADDR

CODE	DESCRIPTION
CALL	((SP)-1)← (PCH), ((SP)-2)←(PCL), (SP)←(SP)+2, (PC) ← ADDR Call subroutine and push return address onto stack
CC	If (carry) = 1, ((SP)-1)←(PCH), ((SP)-2)←(PCL), (SP)←(SP)+2, (PC) ← ADDR If (carry) = 0, (PC) ← (PC)+3 Call subroutine if carry set
CNC	If (carry) = 0, ((SP)-1)←(PCH), ((SP)-2)←(PCL), (SP)←(SP)+2, (PC) ← ADDR If (carry) = 1, (PC) ← (PC)+3 Call subroutine if carry reset
CZ	If (zero) = 1, ((SP)-1)←(PCH), ((SP)-2)← (PCL), (SP)←(SP)+2, (PC)←ADDR If (zero) = 0, (PC) ← (PC)+3 Call subroutine if zero set
CNZ	If (zero) = 0, ((SP)-1) ← (PCH), ((SP)-2)← (PCL), (SP)← (SP)+2, (PC) ← ADDR If (zero) = 1, (PC) ← (PC)+3 Call subroutine if zero reset
CP	If (sign) = 0, ((SP)-1)←(PCH), ((SP)-2)←(PCL), (SP)←(SP)+2, (PC) ← ADDR If (sign) = 1, (PC) ← (PC)+3 Call subroutine if sign plus
CM	If (sign) = 1, ((SP)-1)← (PCH), ((SP)-2)←(PCL), (SP)←(SP)+2, (PC) ← ADDR If (sign) = 0, (PC) ← (PC)+3 Call subroutine if sign minus
CPE	If (parity)= 1, ((SP)-1)←(PCH), ((SP)-2)←(PCL), (SP)← (SP)+2, (PC) ← ADDR If (parity)= 0, (PC) ← (PC)+3 Call subroutine if parity even
CPO	If (parity)= 0, ((SP)-1)←(PCH), ((SP)-2)←(PCL), (SP)← (SP)+2, (PC)← ADDR If (parity)= 1, (PC)← (PC)+3 Call subroutine if parity odd

Condition bits affected: None

RETURN INSTRUCTIONS

Format:

[LABEL:] CODE

CODE	DESCRIPTION
RET	(PCL)←((SP)), (PCH)←((SP)+1), (SP)←(SP)+2 Return from subroutine
RC	If (carry) = 1, (PCL)←((SP)), (PCH)←((SP)+1), (SP)←(SP)+2 If (carry) = 0, (PC)←(PC)+3 Return if carry set
RNC	If (carry) = 0, (PCL)←((SP)), (PCH)←((SP)+1), (SP)←(SP)+2 If (carry) = 1, (PC)←(PC)+3 Return if carry reset
RZ	If (zero) = 1, (PCL)←((SP)), (PCH)←((SP)+1), (SP)←(SP)+2 If (zero) = 0, (PC)←(PC)+3 Return if zero set
RNZ	If (zero) = 0, (PCL)←((SP)), (PCH)←((SP)+1), (SP)←(SP)+2 If (zero) = 1, (PC)←(PC)+3 Return if zero reset
RM	If (sign) = 1, (PCL)←((SP)), (PCH)←((SP)+1), (SP)←(SP)+2 If (sign) = 0, (PC)←(PC)+3 Return if minus
RP	If (sign) = 0, (PCL)←((SP)), (PCH)←((SP)+1), (SP)←(SP)+2 If (sign) = 1, (PC)←(PC)+3 Return if plus
RPE	If (parity)= 1, (PCL)←((SP)), (PCH)←((SP)+1), (SP)←(SP)+2 If (parity)= 0, (PC)←(PC)+3 Return if parity even
RPO	If (parity)= 0, (PCL)←((SP)), (PCH)←((SP)+1), (SP)←(SP)+2 If (parity)= 1, (PC)←(PC)+3 Return if parity odd

Condition bits affected: None

RST INSTRUCTION

Format:

[LABEL:] RST EXP

Note: 0 EXP 7

CODE	DESCRIPTION
RST	((SP)-1)←(PCH), ((SP)-2)←(PCL), (SP)←(SP)+2 (PC)←0000000000EXP000B Call subroutine at address specified by EXP

Condition bits affected: None

INTERRUPT-FLIP FLOP INSTRUCTIONS

Format:

[LABEL:] CODE

CODE	DESCRIPTION	
EI	(INTE) ← 1	Enable the interrupt system
DI	(INTE) ← 0	Disable the interrupt system

Condition bits affected: None

INPUT/OUTPUT INSTRUCTIONS

Format:

[LABEL:] CODE EXP

CODE	DESCRIPTION	
IN	(A) ← input device	Read a byte from device EXP into the accumulator
OUT	output device ← (A)	Send the accumulator contents to device EXP

Condition bits affected: None

HLT INSTRUCTION

Format:

[LABEL:] HLT

CODE	DESCRIPTION	
HLT	----------	Instruction execution halts until an interrupt occurs.

Condition bits affected: None

PSEUDO - INSTRUCTIONS

ORG PSEUDO - INSTRUCTION

Format:

ORG EXP

Code	Description	
ORG	LOCATION COUNTER ← EXP	Set Assembler location counter to EXP

EQU PSEUDO- INSTRUCTION

Format:

NAME EQU EXP

Code	Description	
EQU	NAME ← EXP	Assign the value EXP to the symbol NAME

SET PSEUDO - INSTRUCTION

Format:

NAME SET EXP

Code	Description	
SET	NAME ⟵ EXP	Assign the value EXP to the symbol NAME., which may have been previously SET.

END PSEUDO - INSTRUCTION

Format:

END

Code	Description
END	End the assembly.

CONDITIONAL ASSEMBLY PSEUDO - INSTRUCTIONS

Format:

IF EXP

-and-

ENDIF

Code	Description
IF	If EXP =0, ignore assembler statements until ENDIF is reached. Otherwise, continue assembling statements.
ENDIF	End range of preceding IF.

MACRO DEFINITION PSEUDO - INSTRUCTIONS

Format:

NAME MACRO LIST

-and-

ENDM

Code	Description
MACRO	Define a macro named NAME with parameters LIST
ENDM	End macro definition

APPENDIX B

--INSTRUCTION EXECUTION TIMES AND BIT PATTERNS--

This appendix summarizes the bit patterns and number of time states associated with every 8080 CPU instruction.

When using this summary, note the following symbology:

1) DDD represents a destination register. SSS represents a source register. Both DDD and SSS are interpreted as follows:

DDD or SSS	Interpretation
000	Register B
001	Register C
010	Register D
011	Register E
100	Register H
101	Register L
110	A memory register
111	The accumulator

2) Instruction execution time equals number of time periods multiplied by the duration of a time period.

   A time period may vary from 480 nanosecs to 2 μsec.

   Where two numbers of time periods are shown (eg. 5/11), it means that the smaller number of time periods will be required if a condition is not met, and the larger number of time periods will be required if the condition is met.

MNEMONIC	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Number of Time Periods
CALL	1	1	0	0	1	1	0	1	17
CC	1	1	0	1	1	1	0	0	11/17
CNC	1	1	0	1	0	1	0	0	11/17
CZ	1	1	0	0	1	1	0	0	11/17
CNZ	1	1	0	0	0	1	0	0	11/17
CP	1	1	1	1	0	1	0	0	11/17
CM	1	1	1	1	1	1	0	0	11/17
CPE	1	1	1	0	1	1	0	0	11/17
CPO	1	1	1	0	0	1	0	0	11/17
RET	1	1	0	0	1	0	0	1	10
RC	1	1	0	1	1	0	0	0	5/11
RNC	1	1	0	1	0	0	0	0	5/11
RZ	1	1	0	0	1	0	0	0	5/11
RNZ	1	1	0	0	0	0	0	0	5/11
RP	1	1	1	1	0	0	0	0	5/11
RM	1	1	1	1	1	0	0	0	5/11
RPE	1	1	1	0	1	0	0	0	5/11
RPO	1	1	1	0	0	0	0	0	5/11
RST	1	1	A	A	A	1	1	1	11
IN	1	1	0	1	1	0	1	1	10
OUT	1	1	0	1	0	0	1	1	10
LXI B	0	0	0	0	0	0	0	1	10
LXI D	0	0	0	1	0	0	0	1	10
LXI H	0	0	1	0	0	0	0	1	10
LXI SP	0	0	1	1	0	0	0	1	10
PUSH B	1	1	0	0	0	1	0	1	11
PUSH D	1	1	0	1	0	1	0	1	11
PUSH H	1	1	1	0	0	1	0	1	11
PUSH A	1	1	1	1	0	1	0	1	11
POP B	1	1	0	0	0	0	0	1	10
POP D	1	1	0	1	0	0	0	1	10
POP H	1	1	1	0	0	0	0	1	10
POP A	1	1	1	1	0	0	0	1	10
STA	0	0	1	1	0	0	1	0	13
LDA	0	0	1	1	1	0	1	0	13
XCHG	1	1	1	0	1	0	1	1	4
XTHL	1	1	1	0	0	0	1	1	18
SPHL	1	1	1	1	1	0	0	1	5
PCHL	1	1	1	0	1	0	0	1	5
DAD B	0	0	0	0	1	0	0	1	10
DAD D	0	0	0	1	1	0	0	1	10
DAD H	0	0	1	0	1	0	0	1	10
DAD SP	0	0	1	1	1	0	0	1	10
STAX B	0	0	0	0	0	0	1	0	7
STAX D	0	0	0	1	0	0	1	0	7
LDAX B	0	0	0	0	1	0	1	0	7
LDAS D	0	0	0	1	1	0	1	0	7
INX B	0	0	0	0	0	0	1	1	5
INX D	0	0	0	1	0	0	1	1	5
INX H	0	0	1	0	0	0	1	1	5
INX SP	0	0	1	1	0	0	1	1	5

MNEMONIC	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Number of Time Periods
MOV r_1,r_2	0	1	D	D	D	S	S	S	5
MOV M,r	0	1	1	1	0	S	S	S	7
MOV r,M	0	1	D	D	D	1	1	0	7
HLT	0	1	1	1	0	1	1	0	7
MVI r	0	0	D	D	D	1	1	0	7
MVI M	0	0	1	1	0	1	1	0	10
INR	0	0	D	D	D	1	0	0	5
DCR	0	0	D	D	D	1	0	1	5
INR A	0	0	1	1	1	1	0	0	5
DCR A	0	0	1	1	1	1	0	1	5
INR M	0	0	1	1	0	1	0	0	10
DCR M	0	0	1	1	0	1	0	1	10
ADD r	1	0	0	0	0	S	S	S	4
ADC r	1	0	0	0	1	S	S	S	4
SUB r	1	0	0	1	0	S	S	S	4
SBB r	1	0	0	1	1	S	S	S	4
NDA r	1	0	1	0	0	S	S	S	4
XRA r	1	0	1	0	1	S	S	S	4
ORA r	1	0	1	1	0	S	S	S	4
CMP r	1	0	1	1	1	S	S	S	4
ADD M	1	0	0	0	0	1	1	0	7
ADC M	1	0	0	0	1	1	1	0	7
SUB M	1	0	0	1	0	1	1	0	7
SBB M	1	0	0	1	1	1	1	0	7
NDA M	1	0	1	0	0	1	1	0	7
XRA M	1	0	1	0	1	1	1	0	7
ORA M	1	0	1	1	0	1	1	0	7
CMP M	1	0	1	1	1	1	1	0	7
ADI	1	1	0	0	0	1	1	0	7
ACI	1	1	0	0	1	1	1	0	7
SUI	1	1	0	1	0	1	1	0	7
SBI	1	1	0	1	1	1	1	0	7
NDI	1	1	1	0	0	1	1	0	7
XRI	1	1	1	0	1	1	1	0	7
ORI	1	1	1	1	0	1	1	0	7
CPI	1	1	1	1	1	1	1	0	7
RLC	0	0	0	0	0	1	1	1	4
RRC	0	0	0	0	1	1	1	1	4
RAL	0	0	0	1	0	1	1	1	4
RAR	0	0	0	1	1	1	1	1	4
JMP	1	1	0	0	0	0	1	1	10
JC	1	1	0	1	1	0	1	0	10
JNC	1	1	0	1	0	0	1	0	10
JZ	1	1	0	0	1	0	1	0	10
JNZ	1	1	0	0	0	0	1	0	10
JP	1	1	1	1	0	0	1	0	10
JM	1	1	1	1	1	0	1	0	10
JPE	1	1	1	0	1	0	1	0	10
JPO	1	1	1	0	0	0	1	0	10

MNEMONIC	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Number of Time Periods
DCX B	0	0	0	0	1	0	1	1	5
DXC D	0	0	0	1	1	0	1	1	5
DCX H	0	0	1	0	1	0	1	1	5
DCX SP	0	0	1	1	1	0	1	1	5
CMA	0	0	1	0	1	1	1	1	4
STC	0	0	1	1	0	1	1	1	4
CMC	0	0	1	1	1	1	1	1	4
DAA	0	0	1	0	0	1	1	1	4
SHLD	0	0	1	0	0	0	1	0	17
LHLD	0	0	1	0	1	0	1	0	17
EI	1	1	1	1	1	0	1	1	4
DI	1	1	1	1	0	0	1	1	4
NOP	0	0	0	0	0	0	0	0	4

APPENDIX C
SOFTWARE OPERATING COMMANDS AND MESSAGES

C.1 SYSTEM MONITOR

STARTING ADDRESS - 3800

All arguments are in hexadecimal form.

A ASSIGN I/O DEVICE

A ldev,pdev

Physical device pdev is assigned to logical device ldev

B PUNCH IN BNPF FORMAT

B low address, high address

Memory from low address to high address is punched in BNPF form.

C COMPARE PROM WITH MEMORY

C address

256 bytes of memory beginning at address are compared with a PROM in the programming socket, and miscompares are printed.

D DISPLAY IN HEXADECIMAL FORMAT

D low address, high address

Memory from low address to high address is displayed in hexadecimal form.

E END

E address

Endfile mark is created and punched; 60 null characters are punched.

F FILL MEMORY

F low address, high address, data

Memory from low address to high address is filled with data.

G GO TO

G Address, bkptl, bkpt2

Program control is transferred to address. Breakpoints are set at bkptl and bkpt2.

H HEXADECIMAL ARITHMETIC

H number, number Sp

The sum and difference of the two numbers is printed in hexadecimal.

L LOAD BNPF FORMAT

L low address, high address

A BNPF format tape is read and stored in low address through high address.

M MOVE

M low address, high address, destination address

A block of memory from low address to high address is moved to location destination address.

N PUNCH NULL

N

Sixty null characters are punched.

P PROGRAM PROM

P low address, high address, PROM address

A 1602A or 1702A PROM is programmed with data from memory location low address through high address. Programming begins at PROM location PROM address MOD 256.

R READ HEXADECIMAL TAPE

R bias address

A hexadecimal format tape is read into memory at tape address plus bias address.

S SUBSTITUTE

S address Sp

Memory at address is displayed, and can be modified by typing in new data. Termination with space opens next sequential address, termination with carriage return ends command.

X EXAMINE REGISTERS OR MEMORY

X reg ident

Register is displayed, and can be modified as in the S command.

T TRANSFER PROM TO MEMORY

T address

The contents of a PROM are loaded into memory starting at address and continuing for 256 bytes.

W WRITE HEXADECIMAL

W low address, high address

Memory from low address to high address is punched in hexadecimal format.

MESSAGES

. Monitor ready to accept commands

* Error. Reenter command

C.2 TEXT EDITOR

STARTING ADDRESS - 0010

All commands are terminated with two ESCAPE or ALT MODE characters.

A	Appends text to workspace.
B	Resets buffer pointer to beginning of workspace.
nC	Moves buffer pointer n characters.
nD	Deletes n characters.
E	Punches workspace and input tape, creates end of file.
F string	Searches for string and places buffer pointer after string.
I string	Inserts string at location of buffer pointer.
nK	Deletes n lines.
nL	Moves buffer pointer n lines.
N	Writes 60 null characters
S search string $ substitute string $	Searches for search string. If found, substitutes substitute string and leaves buffer pointer at end of substitute string.
nT	Types n lines on the teletype.
nW	Punches n lines from the beginning of the workspace and deletes the lines typed.
Z	Moves the buffer pointer to the end of the workspace.
n <command string > $$	Executes command string n times. May be nested up to eight levels.

MESSAGES

*	Editor is ready to accept commands.
START PUNCH, TYPE CHARACTER	Allows tape punch to be started prior to punching operations.
"CHARACTER" ILLEGAL IN THIS CONTEXT	Error - reenter command or command string.
WORKSPACE FULL	Self-explanatory
CANNOT FIND "string"	A 'F' or 'S' command search was unsuccessful
ESCAPE/ALT MODE	Terminates or separates commands
BREAK	Terminates command execution
LINE FEED	Identifies end of line

APPENDIX D
HEXADECIMAL PROGRAM TAPE FORMAT

The hexadecimal tape format used by the INTELLEC 8 system is a modified memory image, blocked into discrete records. Each record contains record length, record type, memory address, and checksum information in addition to data. A frame by frame description is as follows:

Frame 0	Record Mark. Signals the start of a record. The ASCII character colon (":" HEX 3A) is used as the record mark.
Frames 1,2 (0-9,A-F)	Record Length. Two ASCII characters representing a hexadecimal number in the range 0 to 'FF'H (0 to 255). This is the count of actual data bytes in the record type or checksum. A record length of 0 indicates end of file.
Frames 3 to 6	Load Address. Four ASCII characters that represent the initial memory location where the data following will be loaded. The first data byte is stored in the location pointed to by the load address, succeeding data bytes are loaded into ascending addresses.
Frames 7, 8	Record Type. Two ASCII characters. Currently all records are type 0, this field is reserved for future expansion.
Frames 9 to 9+2* (Record Length) -1	Data. Each 8 bit memory word is represented by two frames containing the ASCII characters (0 to 9, A to F) to represent a hexadecimal value 0 to 'FF'H (0 to 255).

Frames 9+2*(Record Length) to 9+2*(Record Length) +1	Checksum. The checksum is the negative of the sum of all 8 bit bytes in the record since the record mark (":") evaluated modulus 256. That is, if you add together all the 8 bit bytes, ignoring all carries out of an 8-bit sum, then add the checksum, the result is zero.

Example: If memory locations 1 through 3 contain 53F8EC, the format of the hex file produced when these locations are punched is:

:0300010053F8ECC5

APPENDIX "E"

-- BINARY-DECIMAL-HEXADECIMAL CONVERSION TABLES --

HEXADECIMAL ARITHMETIC

ADDITION TABLE

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
2	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11
3	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12
4	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
5	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14
6	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15
7	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16
8	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17
9	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18
A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19
B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A
C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

MULTIPLICATION TABLE

1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
3	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
4	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

POWERS OF TWO

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5
2 199 023 255 552	41	0.000 000 000 000 454 747 350 886 464 118 957 519 531 25
4 398 046 511 104	42	0.000 000 000 000 227 373 675 443 232 059 478 759 765 625
8 796 093 022 208	43	0.000 000 000 000 113 686 837 721 616 029 739 379 882 812 5
17 592 186 044 416	44	0.000 000 000 000 056 843 418 860 808 014 869 689 941 406 25
35 184 372 088 832	45	0.000 000 000 000 028 421 709 430 404 007 434 844 970 703 125
70 368 744 177 664	46	0.000 000 000 000 014 210 854 715 202 003 717 422 485 351 562 5
140 737 488 355 328	47	0.000 000 000 000 007 105 427 357 601 001 858 711 242 675 781 25
281 474 976 710 656	48	0.000 000 000 000 003 552 713 678 800 500 929 355 621 337 890 625
562 949 953 421 312	49	0.000 000 000 000 001 776 356 839 400 250 464 677 810 668 945 312 5
1 125 899 906 842 624	50	0.000 000 000 000 000 888 178 419 700 125 232 338 905 334 472 656 25
2 251 799 813 685 248	51	0.000 000 000 000 000 444 089 209 850 062 616 169 452 667 236 328 125
4 503 599 627 370 496	52	0.000 000 000 000 000 222 044 604 925 031 308 084 726 333 618 164 062 5
9 007 199 254 740 992	53	0.000 000 000 000 000 111 022 302 462 515 654 042 363 166 809 082 031 25
18 014 398 509 481 984	54	0.000 000 000 000 000 055 511 151 231 257 827 021 181 583 404 541 015 625
36 028 797 018 963 968	55	0.000 000 000 000 000 027 755 575 615 628 913 510 590 791 702 270 507 812 5
72 057 594 037 927 936	56	0.000 000 000 000 000 013 877 787 807 814 456 755 295 395 851 135 253 906 25
144 115 188 075 855 872	57	0.000 000 000 000 000 006 938 893 903 907 228 377 647 697 925 567 626 953 125
288 230 376 151 711 744	58	0.000 000 000 000 000 003 469 446 951 953 614 188 823 848 962 783 813 476 562 5
576 460 752 303 423 488	59	0.000 000 000 000 000 001 734 723 475 976 807 094 411 924 481 391 906 738 281 25
1 152 921 504 606 846 976	60	0.000 000 000 000 000 000 867 361 737 988 403 547 205 962 240 695 953 369 140 625
2 305 843 009 213 693 952	61	0.000 000 000 000 000 000 433 680 868 994 201 773 602 981 120 347 976 684 570 312 5
4 611 686 018 427 387 904	62	0.000 000 000 000 000 000 216 840 434 497 100 886 801 490 560 173 988 342 285 156 25
9 223 372 036 854 775 808	63	0.000 000 000 000 000 000 108 420 217 248 550 443 400 745 280 086 994 171 142 578 125

TABLE OF POWERS OF SIXTEEN$_{10}$

16^n	n	16^{-n}
1	0	0.10000 00000 00000 00000 $\times 10$
16	1	0.62500 00000 00000 00000 $\times 10^{-1}$
256	2	0.39062 50000 00000 00000 $\times 10^{-2}$
4 096	3	0.24414 06250 00000 00000 $\times 10^{-3}$
65 536	4	0.15258 78906 25000 00000 $\times 10^{-4}$
1 048 576	5	0.95367 43164 06250 00000 $\times 10^{-6}$
16 777 216	6	0.59604 64477 53906 25000 $\times 10^{-7}$
268 435 456	7	0.37252 90298 46191 40625 $\times 10^{-8}$
4 294 967 296	8	0.23283 06436 53869 62891 $\times 10^{-9}$
68 719 476 736	9	0.14551 91522 83668 51807 $\times 10^{-10}$
1 099 511 627 776	10	0.90949 47017 72928 23792 $\times 10^{-12}$
17 592 186 044 416	11	0.56843 41386 08080 14870 $\times 10^{-13}$
281 474 976 710 656	12	0.35527 13678 80050 09294 $\times 10^{-14}$
4 503 599 627 370 496	13	0.22204 46049 25031 30808 $\times 10^{-15}$
72 057 594 037 927 936	14	0.13877 78780 78144 56755 $\times 10^{-16}$
1 152 921 504 606 846 976	15	0.86736 17379 88403 54721 $\times 10^{-18}$

TABLE OF POWERS OF 10_{16}

10^n	n	10^{-n}
1	0	1.0000 0000 0000 0000
A	1	0.1999 9999 9999 999A
64	2	0.28F5 C28F 5C28 F5C3 $\times 16^{-1}$
3E8	3	0.4189 374B C6A7 EF9E $\times 16^{-2}$
2710	4	0.68DB 8BAC 710C B296 $\times 16^{-3}$
1 86A0	5	0.A7C5 AC47 1B47 8423 $\times 16^{-4}$
F 4240	6	0.10C6 F7A0 B5ED 8D37 $\times 16^{-4}$
98 9680	7	0.1AD7 F29A BCAF 4858 $\times 16^{-5}$
5F5 E100	8	0.2AF3 1DC4 6118 73BF $\times 16^{-6}$
3B9A CA00	9	0.44B8 2FA0 9B5A 52CC $\times 16^{-7}$
2 540B E400	10	0.6DF3 7F67 5EF6 EADF $\times 16^{-8}$
17 4876 E800	11	0.AFEB FF0B CB24 AAFF $\times 16^{-9}$
E8 D4A5 1000	12	0.1197 9981 2DEA 1119 $\times 16^{-9}$
916 4E72 A000	13	0.1C25 C268 4976 81C2 $\times 16^{-10}$
5AF3 107A 4000	14	0.2D09 370D 4257 3604 $\times 16^{-11}$
3 8D7E A4C6 8000	15	0.480E BE7B 9D58 566D $\times 16^{-12}$
23 8652 6FC1 0000	16	0.734A CA5F 6226 F0AE $\times 16^{-13}$
163 4578 5D8A 0000	17	0.B877 AA32 36A4 B449 $\times 16^{-14}$
DE0 B6B3 A764 0000	18	0.1272 5DD1 D243 ABA1 $\times 16^{-14}$
8AC7 2304 89E8 0000	19	0.1D83 C94F B6D2 AC35 $\times 16^{-15}$

HEXADECIMAL-DECIMAL INTEGER CONVERSION

The table below provides for direct conversions between hexadecimal integers in the range 0–FFF and decimal integers in the range 0–4095. For conversion of larger integers, the table values may be added to the following figures:

Hexadecimal	Decimal	Hexadecimal	Decimal
01 000	4 096	20 000	131 072
02 000	8 192	30 000	196 608
03 000	12 288	40 000	262 144
04 000	16 384	50 000	327 680
05 000	20 480	60 000	393 216
06 000	24 576	70 000	458 752
07 000	28 672	80 000	524 288
08 000	32 768	90 000	589 824
09 000	36 864	A0 000	655 360
0A 000	40 960	B0 000	720 896
0B 000	45 056	C0 000	786 432
0C 000	49 152	D0 000	851 968
0D 000	53 248	E0 000	917 504
0E 000	57 344	F0 000	983 040
0F 000	61 440	100 000	1 048 576
10 000	65 536	200 000	2 097 152
11 000	69 632	300 000	3 145 728
12 000	73 728	400 000	4 194 304
13 000	77 824	500 000	5 242 880
14 000	81 920	600 000	6 291 456
15 000	86 016	700 000	7 340 032
16 000	90 112	800 000	8 388 608
17 000	94 208	900 000	9 437 184
18 000	98 304	A00 000	10 485 760
19 000	102 400	B00 000	11 534 336
1A 000	106 496	C00 000	12 582 912
1B 000	110 592	D00 000	13 631 488
1C 000	114 688	E00 000	14 680 064
1D 000	118 784	F00 000	15 728 640
1E 000	122 880	1 000 000	16 777 216
1F 000	126 976	2 000 000	33 554 432

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
010	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
020	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
030	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
040	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
050	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
060	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
070	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
080	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
090	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A0	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B0	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C0	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D0	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E0	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F0	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255

HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
100	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
110	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
120	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
130	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
140	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
150	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
160	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
170	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
180	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
190	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A0	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B0	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C0	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D0	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E0	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F0	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
200	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
210	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
220	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
230	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
240	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
250	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
260	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
270	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
280	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
290	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A0	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B0	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C0	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D0	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E0	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F0	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
300	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
310	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
320	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
330	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
340	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
350	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
360	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
370	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
380	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
390	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A0	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B0	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C0	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D0	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E0	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F0	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
400	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
410	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
420	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
430	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
440	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
450	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
460	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
470	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
480	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
490	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A0	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B0	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C0	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D0	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E0	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F0	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
500	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
510	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
520	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
530	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
540	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
550	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
560	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
570	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
580	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
590	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A0	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B0	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C0	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D0	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E0	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F0	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535
600	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
610	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
620	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
630	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
640	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
650	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
660	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
670	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
680	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
690	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A0	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B0	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C0	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D0	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E0	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F0	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791

HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
700	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
710	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
720	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
730	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
740	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
750	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
760	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
770	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
780	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
790	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A0	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B0	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C0	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D0	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E0	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F0	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
800	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
810	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
820	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
830	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
840	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
850	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
860	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
870	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
880	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
890	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A0	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B0	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C0	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D0	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E0	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F0	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
900	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
910	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
920	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
930	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
940	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
950	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
960	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
970	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
980	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
990	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A0	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B0	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C0	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D0	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E0	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F0	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559

HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A00	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A10	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A20	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A30	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A40	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A50	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A60	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A70	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A80	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A90	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA0	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB0	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC0	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD0	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE0	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF0	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B00	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B10	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B20	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B30	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B40	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B50	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B60	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B70	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B80	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B90	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA0	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB0	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC0	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD0	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE0	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF0	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071
C00	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C10	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C20	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C30	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C40	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C50	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C60	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C70	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C80	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C90	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA0	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB0	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC0	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD0	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE0	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF0	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327

HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
D00	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D10	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D20	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D30	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D40	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D50	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D60	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D70	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D80	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D90	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA0	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB0	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC0	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD0	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE0	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF0	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E00	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E10	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E20	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E30	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E40	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E50	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E60	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E70	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E80	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E90	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA0	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB0	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC0	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED0	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE0	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF0	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F00	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F10	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F20	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F30	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F40	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F50	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F60	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F70	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F80	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F90	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA0	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB0	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC0	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD0	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE0	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF0	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

APPENDIX "F"

-- ASCII TABLE --

The 8080 uses a seven-bit ASCII code, which is the normal 8 bit ASCII code with the parity (high order) bit always reset.

Graphic or Control	ASCII (Hexadecimal)
NULL	00
SOM	01
EOA	02
EOM	03
EOT	04
WRU	05
RU	06
BELL	07
FE	08
H.Tab	09
Line Feed	0A
V. Tab	0B
Form	0C
Return	0D
SO	0E
SI	0F
DCO	10
X-On	11
Tape Aux. On	12
X-Off	13
Tape Aux. Off	14
Error	15
Sync	16
LEM	17
S0	18
S1	19
S2	1A
S3	1B
S4	1C
S5	1D
S6	1E
S7	1F

Graphic or Control	ASCII Hexadecimal
ACK	7C
Alt. Mode	7D
Rubout	7F
!	21
"	22
#	23
$	24
%	25
&	26
'	27
(	28
)	29
*	2A
+	2B
,	2C
-	2D
.	2E
/	2F
:	3A
;	3B
<	3C
=	3D
>	3E
?	3F
[	5B
\	5C
]	5D
↑	5E
←	5F
@	40
blank	20
0	30
1	31
2	32
3	33
4	34
5	35
6	36
7	37
8	38
9	39

Graphic or Control	ASCII Hexadecimal
A	41
B	42
C	43
D	44
E	45
F	46
G	47
H	48
I	49
J	4A
K	4B
L	4C
M	4D
N	4E
O	4F
P	50
Q	51
R	52
S	53
T	54
U	55
V	56
W	57
X	58
Y	59
Z	5A

APPENDIX G
I/O PORT ASSIGNMENT AND SAMPLE DEVICE DRIVERS

The INTELLEC 8I I/O system assigns specific functions to certain of the I/O ports. Table G-1 gives these assignments. When reading device status or writing device commands, each bit of the 8 bit byte transmitted may have a specific meaning or function. These are given in Tables G-2, G-3, and G-4. For examples of how to use the I/O ports and commands, sample I/O driver programs which read and punch a character using the Teletype, and which read a character from the high-speed paper tape reader are given. However, these functions should normally be performed by calling the I/O subroutines as described in Section 3.3.2.

PORT	USE
INPUT PORT 0	Teletype data in
INPUT PORT 1	Teletype/PTR status in
INPUT PORT 2	PROM data in
INPUT PORT 3	PTR data in
INPUT PORT 4	CRT data in
INPUT PORT 5	CRT status in
OUTPUT PORT 0	Teletype data out
OUTPUT PORT 1	Teletype/PTR/PROM commands out
OUTPUT PORT 2	PROM address out
OUTPUT PORT 3	PROM data out
OUTPUT PORT 4	CRT data out

Table G-1. I/O Port Assignment

NOTE: Input ports 0 through 3, and output ports 0 through 3 are located on I/O board 1.
Input ports 4 and 5, and output port 4 are located on I/O board 2.

BIT NUMBER	NORMAL VALUE	USE
0	1	If = 0, teletype input is ready
1	1	Overrun error
2	0	If = 0, teletype output is ready
3	1	Framing error
4	1	Parity error
5	0	If = 1, paper tape reader has a character
6	1	If = 1, paper tape punch is ready
7	--	Unassigned

Table G-2. Input Port 1 Bit Assignments
(Bit 0 = least significant bit)

BIT NUMBER	NORMAL VALUE	USE
0	0	Teletype reader on/off
1	0	Paper tape punch on/off
2	0	Paper tape reader on/off
3	1	PROM enable/disable (disable = 1)
4	0	Data in is true/complement
5	0	Data out is true/complement
6	0	1702 PROM programmer on/off
7	0	1702a PROM programmer on/off

Table G-3. Output Port 1 Bit Assignments

BIT NUMBER	NORMAL VALUE	USE
0	1	If = 0, input is ready
1	1	Overrun error
2	0	If = 0, output is ready
3	1	Framing error
4	1	Parity error
5	--	Unassigned
6	--	Unassigned
7	--	Unassigned

Table G-4. Input Port 5 Bit Assignments

(1) READING TAPES UNDER PROGRAM CONTROL

Bit 0 of output port 1 is used as the ON/OFF control for the teletype reader, and is normally in the 0 state. To read one character from a paper tape, set this bit to 1 and immediately set it back to 0, causing the reader to advance one frame and stop on the next character.

When the character which has been read is ready for use by the program, bit 0 of input port 1 will be set to zero by the UART device (for a description of the UART, see the INTELLEC 8 Reference Manual). The character which has been read will then be available at input port 0.

The following program section will read one character from the TTY tape reader.

NOTE: The character is transmitted in its one's complement form, i.e., each bit is inverted.

```
LABEL       CODE        OPERAND     COMMENT

PTI         EQU         0           ; TTY INPUT DATA PORT
STI         EQU         1           ; TTY INPUT STATUS PORT
STO         EQU         1           ; TTY OUTPUT COMMAND PORT
;
            MVI         A,1         ; Set bit 0 of port 1=1, pulse reader on.
            OUT         STO
            MVI         A,0         ; Set bit 0 of port 1=0, pulse reader off.
            OUT         STO
; Character has been read and is being readied by UART
BACK:       IN          STI         ; Read UART status into accumulator
            ANI         01H         ; If bit 0=1, loop to BACK
            JNZ         BACK
            IN          PTI         ; Read character from input port 0
                                    ; into accumulator
            XRI         0FFH        ; Invert each bit of transmitted
                                    ; data to produce the character
; The character is now in the accumulator and ready for use
; by the program.
```

(2) PUNCHING TAPES UNDER PROGRAM CONTROL

Tapes may be punched under program control by merely outputting data to the Teletype when the tape punch has been turned on. If, however, printing operations are taking place which are not to be punched, provisions must be made to enable the operator to start and stop the tape punch.

A good procedure to use in this case is to have the program clear the accumulator, then loop until a character is input from the teletype keyboard. This allows the operator to turn off the tape punch and then cause program execution to continue by pressing any key on the teletype keyboard.

Example: The following program section will loop until a character is input from the TTY keyboard, then will punch the character contained in the B register.

```
LABEL       CODE        OPERAND     COMMENT

STI         EQU         1           ; TTY input status port
PTO         EQU         0           ; TTY output data port
PTI         EQU         0           ; TTY input data port
; Loop waiting for a character to be typed by user.
BACK:       IN          STI         ; Is character ready
            ANI         01H         ; to be input (bit 0=0)?
            JNZ         BACK        ; If not, loop to BACK
            IN          PTI         ; Read in character to clear
                                    ; the port
; Punch the character onto the tape
TO:         IN          STI         ; Check if TTY buffer is
                                    ; available
            ANI         04H
            JNZ         TO          ; Loop until available
            MOV         A,B         ; Move character from B
                                    ; register to accumulator for punching
            XRI         0FFH        ; Invert each bit of
                                    ; character
            OUT         PTI         ; Punch character
```

(3) Reading Paper Tape using the High-Speed Paper Tape Reader

Bit 2 of Output Port 1 is used as the Step control for the high-speed paper tape reader. In order to read one character from the reader, set this bit to one and immediately reset it to zero. This produces a pulse on the reader STEP control line; the reader will advance one frame, read the data in that frame, and stop.

When the character which has been read is ready for use in the program (available on Input Port 3), bit 5 of Input Port 1 will be set to zero by the reader. The character which has been read is transmitted in its one's complement form, i.e., each bit has been inverted.

The following program section will read and input one character from the high-speed paper tape reader:

```
LABEL           CODE        OPERAND         COMMENT

HSI             EQU         3               ;HSPTR INPUT DATA PORT
SHI             EQU         1               ;HSPTR INPUT STATUS PORT
SHO             EQU         1               ;HSPTR OUTPUT COMMAND PORT
;
                MVI         A,04H           ;BIT 2, PORT 1 = 1, PULSE READER ON
                OUT         SHO
                XRA         A               ;BIT 2, PORT 1 = 0, PULSE READER OFF
                OUT         SHO
;CHARACTER HAS BEEN READ AND IS TRANSMITTED TO INPUT PORT 3
;
BACK:           IN          SHI             ;READ HSPTR STATUS INTO ACC.
                ANI         20H             ;IF STATUS BIT = 1, LOOP TO BACK:.
                JNZ         BACK
                IN          HSI             ;READ CHARACTER FROM INPUT PORT 3
                                            ;TO ACCUMULATOR
                XRI         0FFH            ;;INVERT EACH BIT OF TRANSMITTED
                                            ;DATA TO PRODUCE THE CHARACTER
;THE CHARACTER IS NOW IN THE ACCUMULATOR AND IS READY FOR USE BY THE
;PROGRAM.
```

West: 1651 E. 4th St., Suite 228/(714)835-9642, TWX: 910-595-1114/Santa Ana, California 92701
Mid-America: 6350 L.B.J. Freeway, Suite 178/(214)661-8829/Dallas, Texas 75240
Great Lakes: 856 Union Road/(513)836-2808/Englewood, Ohio 45322
Northeast: 2 Militia Drive, Suite 4/(617)861-1136, TELEX: 92-3493/Lexington, Massachusetts 02173
Mid-Atlantic: 30 S. Valley Road., Suite 108/(215)647-2615, TWX: 510-668-7768/Paoli, Pennsylvania 19301
Europe: 216 Avenue Louise, B1050 Brussels, Belgium
Orient: Intel Japan Corp., Kasahara Bldg., 1-6-10, Uchikanda, Chiyoda-ku/03-354-8251, TELEX: 781-28426/Tokyo 101

© Intel 1974/Printed in U.S.A

MCS-186-0576/1.5K